Multimedia
Module No: CM0340

Part I

Introduction to Multimedia

About This Course

Aims of Module

To give students a broad grounding in issue surrounding multimedia, including the role of and design of multimedia Systems which incorporate digital audio, graphics and video, underlying concepts and representations of sound, pictures and video, data compression and transmission, integration of media, multimedia authoring, and delivery of multimedia.

Objectives of Module

Students should be able to:

- Understand the relevance and underlying infrastructure of the multimedia systems.
- Understand core multimedia technologies and standards (Digital Audio, Graphics, Video, VR, data transmission/compression)
- Be aware of factors involved in multimedia systems performance, integration and evaluation

Syllabus Outline

Topics in the module include the following:

1. Introduction: Multimedia applications and requirements (e.g., overview of multimedia systems, video-on-demand, interactive television, video conferencing, hypermedia courseware, groupware, World Wide Web, and digital libraries).
2. Audio/Video fundamentals including analog and digital representations, human perception, and audio/video equipment, applications.
3. Audio and video compression including perceptual transform coders for images/video (e.g., JPEG, MPEG, H.263, etc.), scalable coders (e.g., pyramid coders), and perceptual audio encoders. Application and performance

comparison of various coding algorithms including hardware/software trade-offs. Image and video processing applications and algorithms.

4. Multimedia Programming Frameworks: Java for Quicktime, Java Media Framework

Recommended Course Books

The following book is the core text for this module:

- Multimedia Communications: Applications, Networks, Protocols and Standards, Fred Halsall, Addison Wesley, 2000 (ISBN 0-201-39818-4)

  OR

- Networked Multimedia Systems, Raghavan and Tripathi, Prentice Hall, (ISBN 0-13-210642)

The following books are highly recommended reading:

- Hypermedia and the Web: An Engineering Approach, D. Lowe and W. Hall, J. Wiley and Sons, 1999 (ISBN 0-471-98312-8).
- Multimedia Systems, J.F.K, Buford, ACM Press, 1994 (ISBN 0-201-53258-1).
- Understanding Networked Multimedia, Fluckiger, Prentice Hall, (ISBN 0-13-190992-4)
- Design for Multimedia Learning, Boyle, Prentice Hall, (ISBN 0-13-242215-8)
- Distributed Multimedia:Technologies, Applications, and Opportunities in the Digital Information Industry (1st Edition) P.W. Agnew and A.S. Kellerman , Addison Wesley, 1996 (ISBN 0-201-76536-5)
- Multimedia Communication, Sloane, McGraw Hill, (ISBN 0-077092228)
- Virtual Reality Systems, J. Vince, Addison Wesley, 1995 (ISBN 0-201-87687-6)
- Encyclopedia of Graphics File Formats, Second Edition by James D. Murray and William vanRyper, O'Reilly & Associates, 1996 (ISBN: 1-56592-161-5)

The following provide good reference material for parts of the module:
Multimedia Systems

- Hyperwave:The Next Generation Web Solution, H. Maurer, Addison Wesley, 1996 (ISBn 0-201-40346).

Digital Audio

- A programmer's Guide to Sound, T. Kientzle, Addison Wesley, 1997 (ISBN 0-201-41972-6)
- Audio on the Web — The official IUMA Guide, Patterson and Melcher, Peachpit Press.
- The Art of Digital Audio, Watkinson, Focal/Butterworth-Heinmann.
- Synthesiser Basics, GPI Publications.
- Signal Processing: Principles and Applications, Brook and Wynne, Hodder and Stoughton.
- Digital Signal Processing, Oppenheim and Schafer, Prentice Hall.

Digital Imaging/Graphics/Video

- *Digital video processing*, A.M. Tekalp, Prentice Hall PTR, 1995.
- *Intro. to Computer Pictures, http://ac.dal.ca:80/ dong/image.htm* from Allison Zhang at the School of Library and Information Studies, Dalhousie University, Halifax, N.S., Canada
- *http://www.cica.indiana.edu/graphics/image.formats.html* contains a comprehensive list of various graphics/image file formats.
- *Encyclopedia of Graphics File Formats*, Second Edition by James D. Murray and William vanRyper, 1996, O'Reilly & Associates.

Data Compression

- *The Data Compression Book*, Mark Nelson,M&T Books, 1995.
- *Introduction to Data Compression*, Khalid Sayood, Morgan Kaufmann, 1996.
- G.K. Wallace, *The JPEG Still Picture Compression Standard*
- CCITT, *Recommendation H.261*
- D. Le Gall, *MPEG: A Video Compression Standard for Multimedia Applications*
- K. Patel, et. al., *Performance of a Software MPEG Video Decoder*
- P. Cosman, et. al., *Using Vector Quantization for Image Processing*

Animation

- Animation on the Web, S. Wagstaff, Peachpit Press, 1999 (ISBN 0-201-69687)

Multimedia Authoring

- Creating and Designing Multimedia with Director, P. Petrik and B. Dubrovsky, Prentice Hall, 1997 (ISBN 0-13-528985-8)

User Interface Design Issues

- Human Computer Interaction, A. Dix et al, Printice Hall, 1998 (ISBN 0-13-239864)
- Designing the User Interface , B. Schneiderman, Addison Wesley, 1998 (ISBN 0-201-694497)
- Human Computer Interaction, Preece et al, Addison Wesley, 1994, 0-201-62769-8)

Multimedia Databases

- Multimedia Database Management Systems, B Prbhakaran, Kluwer, 1997, (ISBN 0-7923-9784-3).

Internet/WWW related Books
Teach Yourself Web Publishing with HTML 3.2 in 14 Days, *Laura Lemay*, Sams.Net Publishing.

The Internet Unleashed, *J. Ellsworth, B. Baron, et al.*, Sams.Net Publishing
WWW, Internet in General

The Internet Complete Reference(2nd Ed.), *H. Hahn*, McGraw-Hill

Webmaster in a Nutshell, *S. Spainhour and V. Quercia*, O'Reilly and Associates Inc.

Every Student's Guide to the World Wide Web, *K. Pitter and R. Minato*, McGraw Hill

Every Student's Guide to the World Wide Web (Macintosh Version),*K. Pitter and R. Minato*, McGraw Hill

Internet — Theory and Practice

Demystifying TCP/IP, *E. Taylor*, Wordware Publishing Inc.

HTML BOOKS:

HTML Sourcebook, *I.S. Graham*, Wiley and Sons

HTML: The Definitive Guide, *C. Musciano and B. Kennedy*, O'Reilly and Associates Inc.

CGI Scripts / Perl Programming BOOKS:

Teach Yourself CGI Programming with Perl 5 in a Week, *E. Herrmann*, Sams.Net

CGI Developer's Guide, *E.E. Kim*, Sams.Net

CGI Programming on the World Wide Web, *S. Gundavaram*, O'Reilly and Associates Inc.

Learning Perl, *R.L. Schwartz*, O'Reilly and Associates Inc.

Programming Perl, *L. Wall, T. Christiansen and R.L. Schwartz*, O'Reilly and Associates Inc.

Perl 5 Desktop Reference, *J. Vromans*, O'Reilly and Associates Inc.

WWW Design Issues BOOKS:

Designing with Style Sheets, Tables, and Frames, *M.E. Holzschlag*, Sams.Net

HTML Style Sheets Quick Reference, *R. Falla*, Que

10 Minute Guide to HTML Style Sheets, *C. Zacker*, Que

Teach Yourself Great Web Design in a Week, *A. Vasquez-Peterson and P. Chow*, Sams.Net

Designing for the Web, *Jennifer Niederst*, O'Reilly and Associates

GIF Animation Studio: Animating Your Web Site, R. Koman, O'Reilly and Associates

A comprehensive guide to all Internet related books is available at
the *Unofficial Internet Book List* WWW site (URL: *http://www.savetz.com/booklist/*)

Chapter 1

Introduction

1.1 History of Multimedia Systems

Newspaper were perhaps the first mass communication medium to employ Multimedia — they used mostly text, graphics, and images.

In 1895, Gugliemo Marconi sent his first wireless radio transmission at Pontecchio, Italy. A few years later (in 1901) he detected radio waves beamed across the Atlantic. Initially invented for telegraph, radio is now a major medium for audio broadcasting.

Television was the new media for the 20th century. It brings the video and has since changed the world of mass communications.

Some of the important events in relation to Multimedia in Computing include:

- 1945 - Bush wrote about Memex
- 1967 - Negroponte formed the Architecture Machine Group at MIT
- 1969 - Nelson & Van Dam hypertext editor at Brown
- Birth of The Internet
- 1971 - Email
- 1976 - Architecture Machine Group proposal to DARPA: Multiple Media
- 1980 - Lippman & Mohl: Aspen Movie Map
- 1983 - Backer: Electronic Book
- 1985 - Negroponte, Wiesner: opened MIT Media Lab
- 1989 - Tim Berners-Lee proposed the World Wide Web to CERN (European Council for Nuclear Research)
- 1990 - K. Hooper Woolsey, Apple Multimedia Lab, 100 people, educ.

- 1991 - Apple Multimedia Lab: Visual Almanac, Classroom MM Kiosk
- 1992 - the first M-bone audio multicast on the Net
- 1993 - U. Illinois National Center for Supercomputing Applications: NCSA Mosaic
- 1994 - Jim Clark and Marc Andreesen: Netscape
- 1995 - JAVA for platform-independent application development. Duke is the first applet.
- 1996 - Microsoft, Internet Explorer.

1.2 Multimedia/Hypermedia

1.2.1 What is Multimedia?

Multimedia can have a many definitions these include:

Multimedia means that computer information can be represented through audio, video, and animation in addition to traditional media (i.e., text, graphics drawings, images).

A good general definition is:

Multimedia is the field concerned with the computer-controlled integration of text, graphics, drawings, still and moving images (Video), animation, audio, and any other media where every type of information can be represented, stored, transmitted and processed digitally.

A *Multimedia Application* is an Application which uses a collection of multiple media sources e.g. text, graphics, images, sound/audio, animation and/or video.

Hypermedia can be considered as one of the multimedia applications.

1.2.2 What is HyperText and HyperMedia?

Hypertext is a text which contains links to other texts. The term was invented by Ted Nelson around 1965.

Hypertext is therefore usually non-linear (as indicated below).

HyperMedia is not constrained to be text-based. It can include other media, e.g., graphics, images, and especially the continuous media – sound and video. Apparently, Ted Nelson was also the first to use this term.

The World Wide Web (WWW) is the best example of hypermedia applications.

1.3 Multimedia Systems

A *Multimedia System* is a system capable of processing multimedia data and applications.

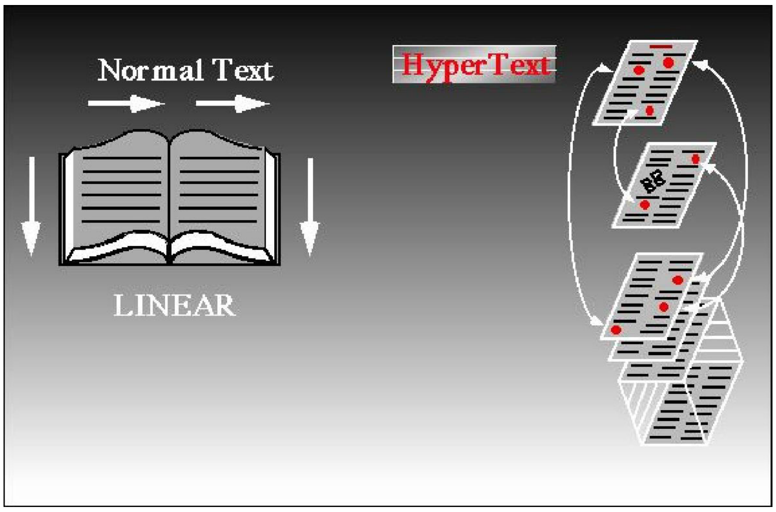


Figure 1.1: Illustration of Hypertext Links

A *Multimedia System* is characterised by the processing, storage, generation, manipulation and rendition of Multimedia information.

1.3.1 Characteristics of a Multimedia System

A Multimedia system has four basic characteristics:

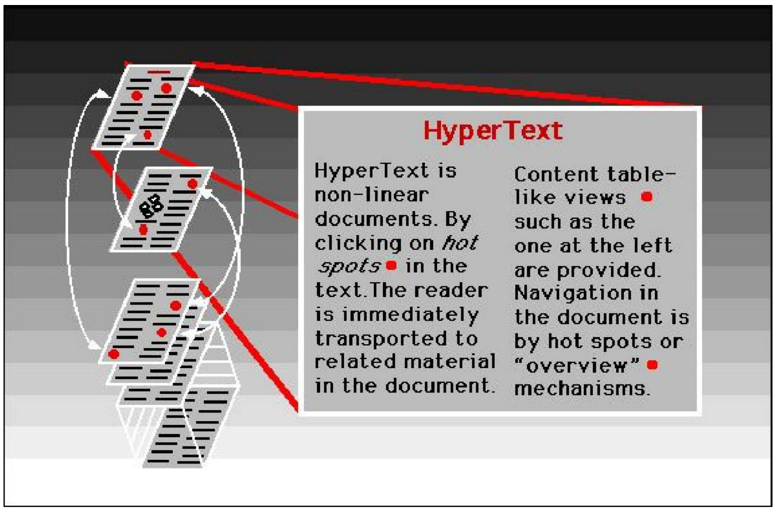
- Multimedia systems must be *computer controlled.*
- Multimedia systems are *integrated.*
- The information they handle must be represented *digitally.*
- The interface to the final presentation of media is usually *interactive.*

1.3.2 Challenges for Multimedia Systems

Supporting multimedia applications over a computer network renders the application *distributed.* This will involve many special computing techniques — discussed later.

Multimedia systems may have to render a variety of media at the same instant — a distinction from normal applications. There is a temporal relationship between many forms of media (*e.g.* Video and Audio. There 2 are forms of problems here

- Sequencing within the media — *playing frames in correct order/time frame in video*



HyperText

HyperText is non-linear documents. By clicking on *hot spots* ● in the text. The reader is immediately transported to related material in the document.

Content table-like views ● such as the one at the left are provided. Navigation in the document is by hot spots or "overview" ● mechanisms.

Figure 1.2: Definition of Hypertext

- *Synchronisation* — inter-media scheduling (*e.g.* Video and Audio). Lip synchronisation is clearly important for humans to watch playback of video and audio and even animation and audio. Ever tried watching an out of (lip) sync film for a long time?

The key issues multimedia systems need to deal with here are:

- How to represent and store temporal information.
- How to strictly maintain the temporal relationships on play back/retrieval
- What process are involved in the above.

Data has to represented *digitally* so many initial source of data needs to be *digitise* — translated from analog source to digital representation. The will involve scanning (graphics, still images), sampling (audio/video) although digital cameras now exist for direct scene to digital capture of images and video.

The data is *large* several Mb easily for audio and video — therefore storage, transfer (bandwidth) and processing overheads are high. Data compression techniques very common.

1.3.3 Desirable Features for a Multimedia System

Given the above challenges the following feature a desirable (if not a prerequisite) for a Multimedia System:

Very High Processing Power — needed to deal with large data processing and real time delivery of media. Special hardware commonplace.

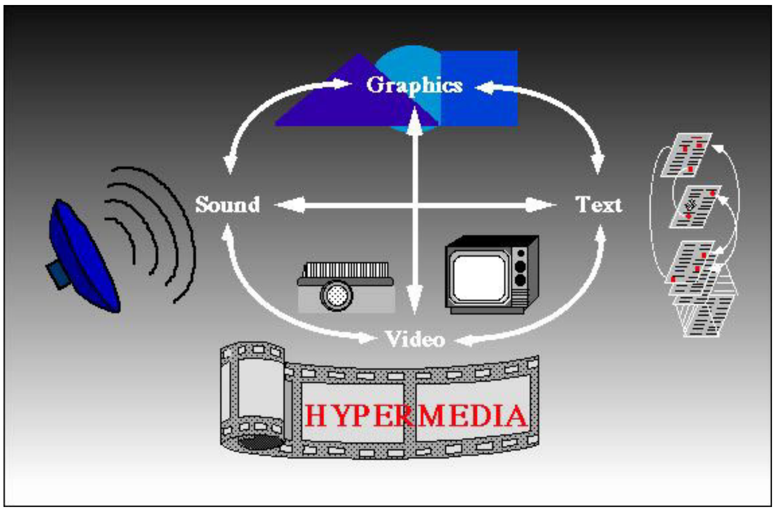


Figure 1.3: Definition of HyperMedia

Multimedia Capable File System — needed to deliver real-time media — *e.g.* Video/Audio Streaming. Special Hardware/Software needed *e.g* RAID technology.

Data Representations/File Formats that support multimedia — Data representations/file formats should be easy to handle yet allow for compression/decompression in real-time.

Efficient and High I/O — input and output to the file subsystem needs to be efficient and fast. Needs to allow for real-time recording as well as playback of data. *e.g.* Direct to Disk recording systems.

Special Operating System — to allow access to file system and process data efficiently and quickly. Needs to support direct transfers to disk, real-time scheduling, fast interrupt processing, I/O streaming *etc.*

Storage and Memory — large storage units (of the order of 50 -100 Gb or more) and large memory (50 -100 Mb or more). Large Caches also required and frequently of Level 2 and 3 hierarchy for efficient management.

Network Support — Client-server systems common as distributed systems common.

Software Tools — user friendly tools needed to handle media, design and develop applications, deliver media.

1.3.4 Components of a Multimedia System

Now let us consider the Components (Hardware and Software) required for a multimedia system:

Capture devices — Video Camera, Video Recorder, Audio Microphone, Keyboards, mice, graphics tablets, 3D input devices, tactile sensors, VR devices. Digitising/Sampling Hardware

Storage Devices — Hard disks, CD-ROMs, Jaz/Zip drives, DVD, *etc*

Communication Networks — Ethernet, Token Ring, FDDI, ATM, Intranets, Internets.

Computer Systems — Multimedia Desktop machines, Workstations, MPEG/VIDEO/DSP Hardware

Display Devices — CD-quality speakers, HDTV,SVGA, Hi-Res monitors, Colour printers *etc.*

1.4 Applications

Examples of Multimedia Applications include:

- World Wide Web
- Hypermedia courseware
- Video conferencing
- Video-on-demand
- Interactive TV
- Groupware
- Home shopping
- Games
- Virtual reality
- Digital video editing and production systems
- Multimedia Database systems

1.5 Trends in Multimedia

Current big applications areas in Multimedia include:

World Wide Web — Hypermedia systems — embrace nearly all multimedia technologies and application areas. Ever increasing popularity.

MBone — Multicast Backbone: Equivalent of conventional TV and Radio on the Internet.

Enabling Technologies — developing at a rapid rate to support ever increasing need for Multimedia. Carrier, Switching, Protocol, Application, Coding/Compression, Database, Processing, and System Integration Technologies at the forefront of this.

1.6 Further Reading/Exploration

Try some good sources for locating internet multimedia examples on the Internet For example:

- WebMuseum, Paris — *http://metalab.unc.edu/wm/*
- Audio Net — *http://www.audionet.com/*
- BBC Web Site — http://www.bbc.co.uk
- Index of Multimedia Information —- http://viswiz.gmd.de/MultimediaInfo/Sources

Part II

Multimedia Authoring

Chapter 2

Multimedia Authoring:Systems and Applications

2.1 What is an Authoring System?

An Authoring System is a program which has pre-programmed elements for the development of interactive multimedia software titles. Authoring systems vary widely in orientation, capabilities, and learning curve. There is no such thing (at this time) as a completely point-and-click automated authoring system; some knowledge of heuristic thinking and algorithm design is necessary. Whether you realize it or not, authoring is actually just a speeded-up form of programming; you don't need to know the intricacies of a programming language, or worse, an API, but you do need to understand how programs work.

2.1.1 Why should you use an authoring system?

It generally takes about 1/8th the time to develop an interactive multimedia project, such as a CBT (Computer Based Training) program, in an authoring system as opposed to programming it in compiled code. This means 1/8 the cost of programmer time and likely increased re-use of code (assuming that you pass this project's code to the next CBT project, and they use a similar or identical authoring system). However, the content creation (graphics, text, video, audio, animation, etc.) is not generally affected by the choice of an authoring system; any production time gains here result from accelerated prototyping, not from the choice of an authoring system over a compiled language.

2.2 Multimedia Authoring Paradigms

The *authoring paradigm*, or *authoring metaphor*, is the methodology by which the authoring system accomplishes its task.

There are various paradigms, including:

Scripting Language — the Scripting paradigm is the authoring method closest in form to traditional programming. The paradigm is that of a programming language, which specifies (by filename) multimedia elements, sequencing, hotspots, synchronization, etc. A powerful, object-oriented scripting language is usually the centerpiece of such a system; in-program editing of elements (still graphics, video, audio, etc.) tends to be minimal or non-existent. Scripting languages do vary; check out how much the language is object-based or object-oriented. The scripting paradigm tends to be longer in development time (it takes longer to code an individual interaction), but generally more powerful interactivity is possible. Since most Scripting languages are interpreted, instead of compiled, the runtime speed gains over other authoring methods are minimal. The media handling can vary widely; check out your system with your contributing package formats carefully.

The Apple's HyperTalk for HyperCard, Assymetrix's OpenScript for ToolBook and Lingo scripting language of Macromedia Director are examples of a Multimedia scripting language.

Here is an example lingo script to jump to a frame

```
global gNavSprite

on exitFrame
  go the frame
  play sprite gNavSprite
end
```

Iconic/Flow Control — This tends to be the speediest (in development time) authoring style; it is best suited for rapid prototyping and short-development time projects. Many of these tools are also optimized for developing Computer-Based Training (CBT). The core of the paradigm is the Icon Palette, containing the possible functions/interactions of a program, and the Flow Line, which shows the actual links between the icons. These programs tend to be the slowest runtimes, because each interaction carries with it all of its possible permutations; the higher end packages, such as Authorware (Fig. 2.1)or IconAuthor, are extremely powerful and suffer least from runtime speed problems.

Frame — The Frame paradigm is similar to the Iconic/Flow Control paradigm in that it usually incorporates an icon palette; however, the links drawn between icons are conceptual and do not always represent the actual flow

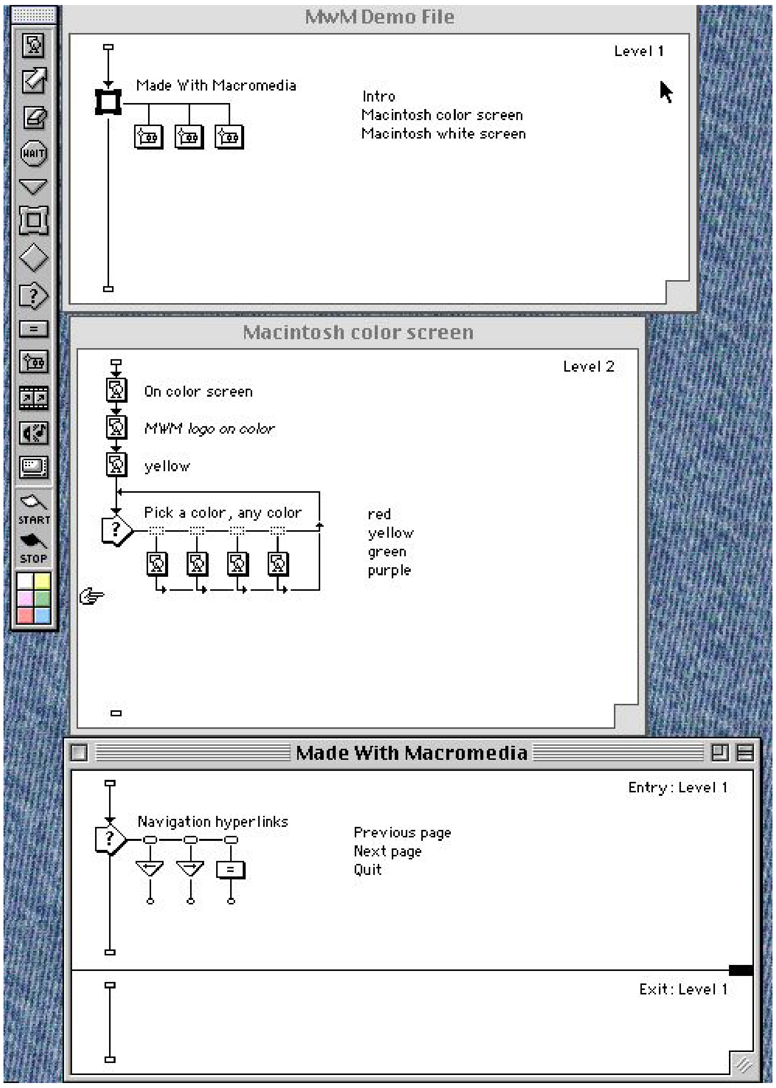


Figure 2.1: Macromedia Authorware Iconic/Flow Control Examples

of the program. This is a very fast development system, but requires a good auto-debugging function, as it is visually un-debuggable. The best of these have bundled compiled-language scripting, such as Quest (whose scripting language is C) or Apple Media Kit.

Card/Scripting — The Card/Scripting paradigm provides a great deal of power (via the incorporated scripting language) but suffers from the index-card structure. It is excellently suited for Hypertext applications, and supremely suited for navigation intensive (a la Cyan's "MYST" game) applications. Such programs are easily extensible via XCMDs and DLLs; they are widely used for shareware applications. The best applications allow all objects (including individual graphic elements) to be scripted; many entertainment applications are prototyped in a card/scripting system prior to compiled-language coding.

Cast/Score/Scripting — The Cast/Score/Scripting paradigm uses a music score as its primary authoring metaphor; the synchronous elements are shown in various horizontal *tracks* with simultaneity shown via the vertical columns. The true power of this metaphor lies in the ability to script the behavior of each of the cast members. The most popular member of this paradigm is Director, which is used in the creation of many commercial applications. These programs are best suited for animation-intensive or synchronized media applications; they are easily extensible to handle other functions (such as hypertext) via XOBJs, XCMDs, and DLLs.

Macromedia Director uses this method and examples can be found in Figs 2.2— 2.4

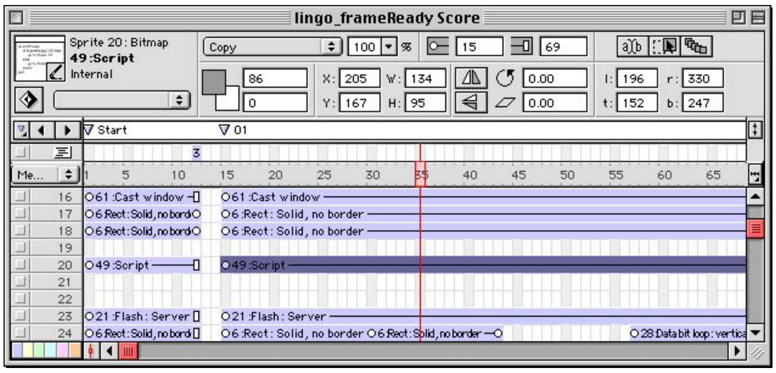


Figure 2.2: Macromedia Director Score Window

Hierarchical Object — The Hierarchical Object paradigm uses a object metaphor (like OOP) which is visually represented by embedded objects and iconic properties. Although the learning curve is non-trivial, the visual representation of objects can make very complicated constructions possible.

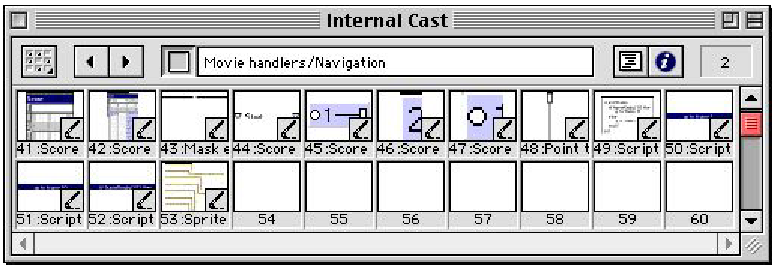
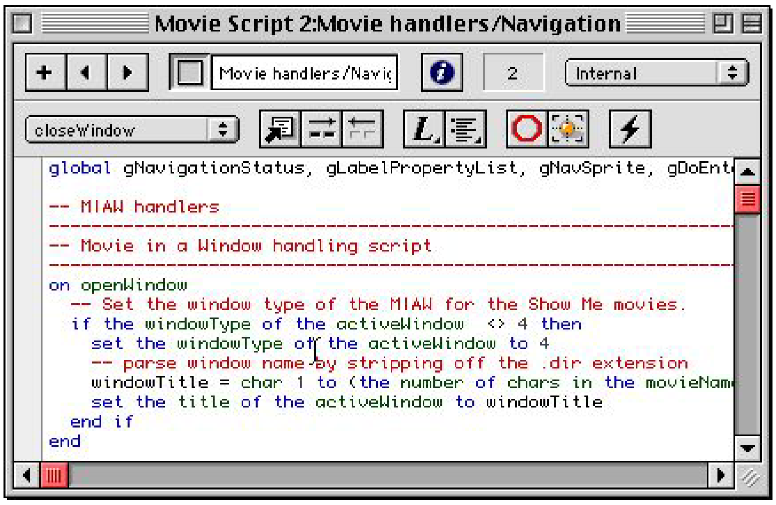


Figure 2.3: Macromedia Director Cast Window



```
global gNavigationStatus, gLabelPropertyList, gNavSprite, gDoEnt

-- MIAW handlers
------------------------------------------------------------------------
-- Movie in a Window handling script
------------------------------------------------------------------------
on openWindow
  -- Set the window type of the MIAW for the Show Me movies.
  if the windowType of the activeWindow  <> 4 then
    set the windowType of the activeWindow to 4
    -- parse window name by stripping off the .dir extension
    windowTitle = char 1 to (the number of chars in the movieNam
    set the title of the activeWindow to windowTitle
  end if
end
```

Figure 2.4: Macromedia Director Script Window

Hypermedia Linkage — The Hypermedia Linkage paradigm is similar to the Frame paradigm in that it shows conceptual links between elements; however, it lacks the Frame paradigm's visual linkage metaphor.

Tagging — The Tagging paradigm uses tags in text files (for instance, SGML/HTML, SMIL (Synchronised Media Integration Language), VRML, 3DML and WinHelp) to link pages, provide interactivity and integrate multimedia elements.

2.3 Multimedia Programming vs Multimedia Authoring

It should be noted that a distinction should be made between Programming and Authoring.

Authoring involves the assembly and bringing togther of Multimedia with possiby high level graphical interface design and some high level scripting.

Programming involves low level assembly and construction and control of Multimedia and involves real languages like C and Java.

Later in this course will will study Java programming in Quicktime and the Java Media Framework.

Quicktime may also be programmed in C.

2.4 Issues in Multimedia Applications Design

There are various issues in Multimedia authoring below we summarise issues involved in Multimedia content and technical design.

2.4.1 Content Design

Content design deals with:

- What to say, what vehicle to use.

"In multimedia, there are five ways to format and deliver your message. You can *write* it, *illustrate* it, *wiggle* it, *hear* it, and *interact* with it."

2.1.1 Scripting (*writing*)

Rules for good writing:

1. Understand your audience and correctly address them.
2. Keep your writing as simple as possible. (e.g., write out the full message(s) first, then shorten it.)
3. Make sure technologies used complement each other.

2.1.2 Graphics (*illustrating*)

- Make use of pictures to effectively deliver your messages.
- Create your own (draw, (color) scanner, PhotoCD, ...), or keep "copy files" of art works. – "Cavemen did it first."

Graphics Styles

- fonts
- colors
  - pastels
  - earth-colors
  - metallic
  - primary color
  - neon color

2.1.3 Animation (*wiggling*)

1. **Types of Animation**
   - Character Animation – humanise an object
     e.g., a toothbrush, a car, a coke bottle, etc.
     Factors in choosing a character
     - Emotion – Is it happy, sad, funny, sloppy, ...?
     - Movement – Is it fast, slow, bumpy, ...?
     - Visual style – Is its color/texture consistent with the rest?
     - Copyright – "Don't use Mickey before checking with Walt."
     - Adequacy – e.g., Does it provide various poses (can't make a broomstick sit!)
   - Highlights and Sparkles
     e.g., to pop a word in/out of the screen, to sparkle a logo
     –> to draw attention
   - Moving Text
     e.g., put up one character at a time like a typewriter e.g., "pulsing" – the word grows/shrinks (or changes color) a few times
     Note: Do not slowly move entire line of text, they are not readable. Instead, for example, slide the bullets in and out.
   - Video – live video or digitized video
     +: more powerful than still images
     +: often easier to obtain than graphics animation
     -: takes a lot of disk space
     -: sometimes needs special hardware

2. **When to Animate**

   "A leaf doesn't flutter if the wind doesn't blow." **Only animate when it has a specific purpose**

   - Enhance emotional impact

     e.g., dove softly flapping its wings –> peace

     e.g., air bag explosion + dummy movements –> car crash.
   - Make a point

     e.g., show insertion of a memory chip onto the motherboard (much better than a diagram) e.g., Microsoft Golf (instructional)
   - Improve information delivery

     e.g., "pulsing" words (in and out of screen) adds emphasis
   - Indicate passage of time

     e.g., clock/hourglass –> program still running e.g., animated text –> to prompt for interaction/response
   - Provide a transition to next subsection
     - Wipes – e.g., L-to-R, T-D, B-U, diagonal, iris round, center to edge, etc.
     - Dissolve – the current image distorts into an unrecognizable form before the next clear image appears, e.g., boxy dissolve, cross dissolve, etc.
     - Fade – a metaphor for a complete change of scene
     - Cut – immediate change to next image, e.g., for making story points using close-up ** Cuts are easy to detect in "video segmentation"

2.1.4 Audio (*hearing*)

Types of Audio in Multimedia Applications:

1. Music – set the mood of the presentation, enhance the emotion, illustrate points
2. Sound effects – to make specific points, e.g., squeaky doors, explosions, wind, ...
3. Narration – most direct message, often effective

2.1.5 Interactivity (*interacting*)

- interactive multimedia systems!
- people remember 70% of what they interact with (according to late 1980s study)

Types of Interactive Multimedia Applications:

1. Menu driven programs/presentations

   – often a hierarchical structure (main menu, sub-menus, ...)

2. Hypermedia

   +: less structured, cross-links between subsections of the same subject –> non-linear, quick access to information

   +: easier for introducing more multimedia features, e.g., more interesting "buttons"

   -: could sometimes get lost in navigating the hypermedia

3. Simulations / Performance-dependent Simulations

   – e.g., Games – SimCity, Flight Simulators

2.4.2 Technical Design

Technologicical factors may limit the ambition of your mutlimedia presentation:

- Technical parameters that affect the design and delivery of multimedia applications

```
Video Mode Resolution  Colors
----------    ----------------------
  CGA      320 x 200       4
  MCGA     320 x 200     256
  EGA      640 x 350      16
  VGA      640 x 480     256
  S-VGA        1,024 x 768  $>$= 256
  S-VGA        1,280 x 1,024     $>$= 256
         .
         .
         .
   16-bit color --$>$ 65536 colors
   24-bit color --$>$ 16.7 million colors
```

1. **Video Mode and Computer Platform**

   PC <–> Macintosh

   There are many "portable", "cross-platform" software and "run-time modules", but many of them lose quality/performance during the translation.

2. **Memory and Disk Space Requirement**

   Rapid progress in hardware alleviates the problem, but software is too "greedy", especially the multimedia ones.

3. **Delivery**

   - Live Presentation
     Short checking list for hardware/software requirements:
     - type of graphics card
     - video memory (1 MB, 2 MB, 4 MB, etc.)
     - access time of hard disk (important for real-time video)
     - type of sound card (support for General MIDI)
     - audio-video software
   - Delivery by diskette
     -: Small in size, slow to install
   - Delivery by CD-ROM
     +: Large capacity
     -: Access time of CD-ROM drives is longer than hard-disk drives
   - Electronic Delivery (ftp, www, etc.)
     – depends on baud rate, network connection, and monthly bill

2.4.3 Visual Design

Here we summarise factors that should be considers in the visual design of a multimedia presentation:

1. **Themes and Styles**

   – A multimedia presentation should have a consistent theme/style, it should not be disjointed and cluttered with multiple themes.

   – The choice of theme/style depends on the styles and emotions of your audience.

   Some Possible Themes:

   - Cartoon theme
     +: interesting / entertaining
     -: must be consistent with the character's personality
   - Traditional theme – straightforward
     +: simple, often informative
     -: not as interesting
   - High tech theme – contemporary computer art work (morphing, texture mapping, metal texture, explosions, ...)
     +: attractive, easy to animate
   - Technical theme – include blueprints, 3D models of the product, ... e.g., start with a drawing, then transformed into a rendered image.
     +: shows adequate technical information
     +: gives impression of solid design and construction

Color Schemes and Art Styles

- Natural and floral
  (outdoor scenes, e.g., mountains, lakes, ...) –> getting back to nature
- Oil paints, watercolours, colored pencils, pastels.
  – these art styles can be combined with e.g., cartoon or high tech themes

2. **Pace and Running length**

   A few guidelines:

   - Allow a block of text to be slowly read twice.
   - Transition time should be an indication of real-time.
     - dissolve – time delay, scene change
     - cut – two views of same scene at same time, or abrupt scene change
   - Running length
     - self running presentation: 2-3 minutes
     - limited interaction: 5-6 minutes
     - complete analytical, hands-on demo: < 15 minutes
     - with questions, discussions: > 30 minutes

     ** build in breaks for long presentations

3. **Basic Layout**

   (a) Title (b) Action area (c) Narration (d) Dialog (e) Interactive controls

   - make sure that the information delivery path in the layout is smooth, not irregular/jumpy
   - use headlines/subtitles, additional shapes, buttons, fonts, backgrounds and textures to enhance the visual appearance.

2.5 Storyboarding

The concept of storyboarding has been by animators and their like for many years.

Storyboarding is used to help plan the general organisation or content of a presentation by recording and organizing ideas on index cards, or placed on board/wall. The storyboard evolves as the media are collected and organised: new ideas and refinements to the presentation are

2.6 Overview of Multimedia Software Tools

2.6.1 Digital Audio

Macromedia Soundedit —- Edits a variety of different format audio files, apply a variety of effects (Fig 2.5)

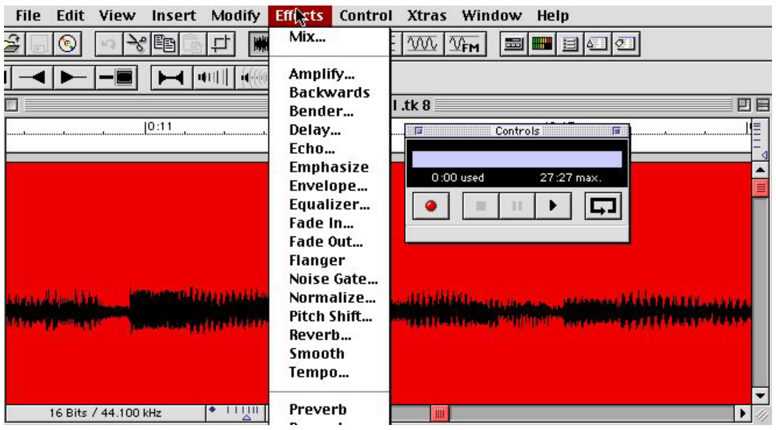


Figure 2.5: Macromedia Soundedit Main and Control Windows and Effects Menu

CoolEdit —- Edits a variety of different format audio files
Many Public domain tools on the Web.

2.6.2 Music Sequencing and Notation

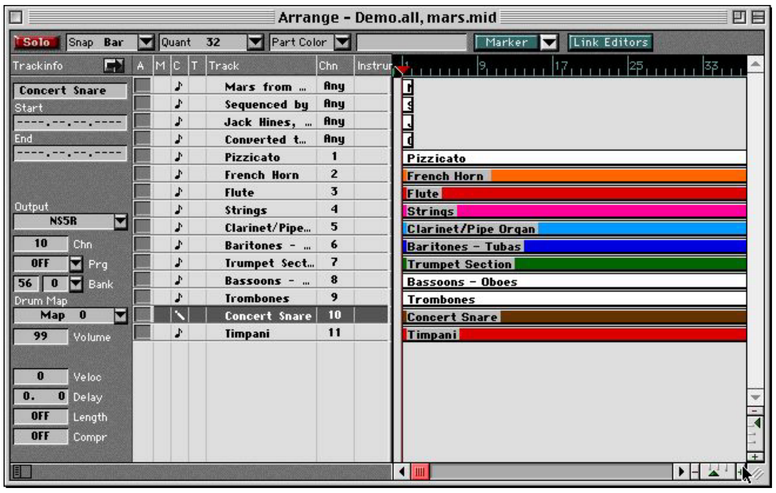
Cakewalk

- Supports General MIDI
- Provides several editing views (staff, piano roll, event list) and Virtual Piano
- Can insert WAV files and Windows MCI commands (animation and video) into tracks

Cubase

- A better software than Cakewalk Express
- Intuitive Interface to arrange and play Music (Figs 2.6 and 2.7)
- Wide Variety of editing tools including Audio (Figs 2.8 and 2.9
- Allows printing of notation sheets

Logic Audio


Arrange - Demo.all, mars.mid

Snap Bar
Quant 32

Concert Snare
Output NS5R
10 Chn
OFF Prg
56 0 Bank
Drum Map: Map 0
99 Volume
0 Veloc
0. 0 Delay
OFF Length
OFF Compr

Track	Chn
Mars from ...	Any
Sequenced by	Any
Jack Hines, ...	Any
Converted t...	Any
Pizzicato	1
French Horn	2
Flute	3
Strings	4
Clarinet/Pipe...	5
Baritones - ...	6
Trumpet Sect...	7
Bassoons - ...	8
Trombones	9
Concert Snare	10
Timpani	11

Figure 2.6: Cubase Arrange Window (Main)

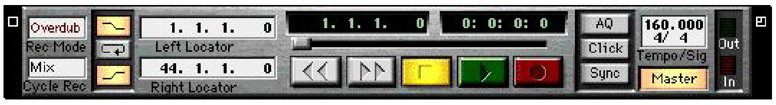
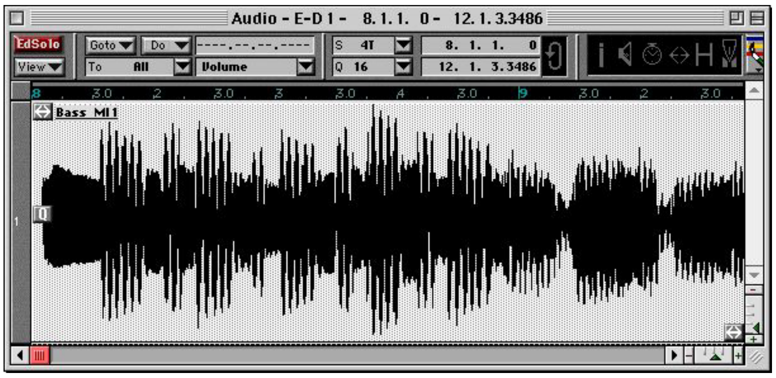

Overdub Rec Mode
Mix Cycle Rec
1. 1. 1. 0 Left Locator
44. 1. 1. 0 Right Locator
1. 1. 1. 0
0: 0: 0: 0
160.000 4/ 4 Tempo/Sig

Figure 2.7: Cubase Transport Bar Window — Emulates a Tape Recorder Interface


Audio - E-D 1 - 8. 1. 1. 0 - 12. 1. 3.3486

To All
Volume
S 4T
Q 16
8. 1. 1. 0
12. 1. 3.3486

Figure 2.8: Cubase Audio Window

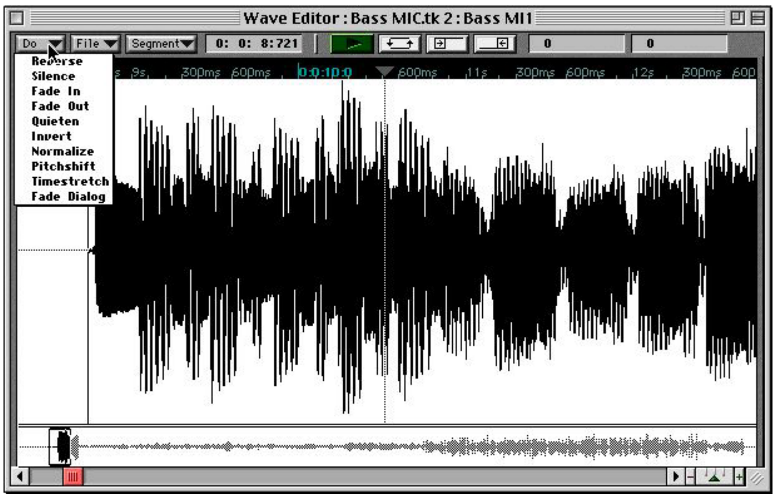


Figure 2.9: Cubase Audio Editing Window with Editing Functions

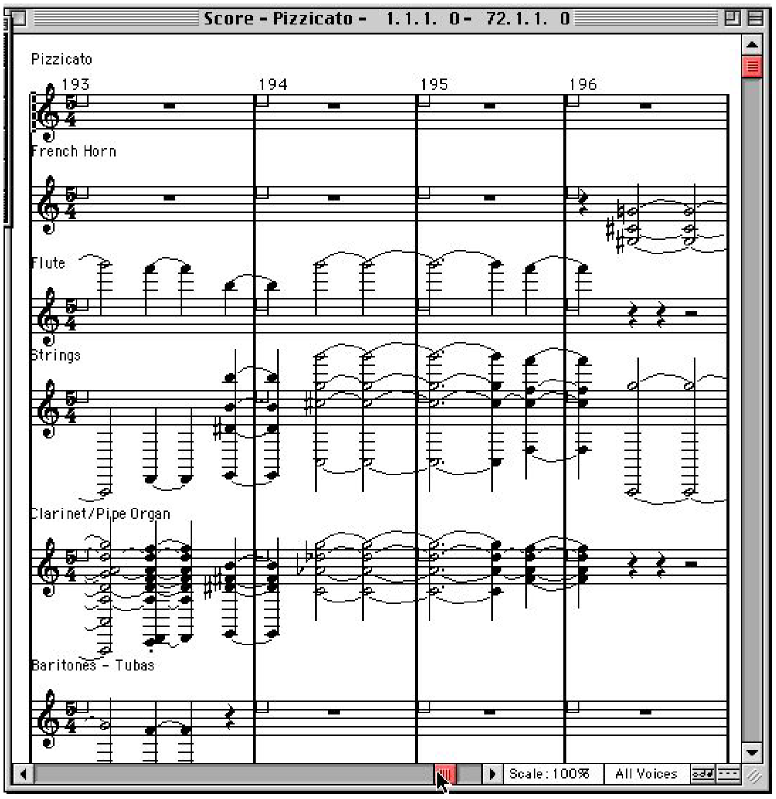


Figure 2.10: Cubase Score Editing Window

- Cubase Competitor, similar functionality

Marc of the Unicorn Performer

- Cubase/Logic Audio Competitor, similar functionality

2.6.3 Image/Graphics Editing

Adobe Photoshop

- Allows layers of images, graphics and text
- Includes many graphics drawing and painting tools
- Sophisticate lighting effects filter
- A good graphics, image processing and manipulation tool

Adobe Premiere

- Provides large number (up to 99) of video and audio tracks, superimpositions and virtual clips
- Supports various transitions, filters and motions for clips
- A reasonable desktop video editing tool

Macromedia Freehand

- Graphics drawing editing package

Many other editors in public domain and commercially

2.6.4 Image/Graphics Editing

Many commercial packages available

- Adobe Premier
- Videoshop
- Avid Cinema
- SGI MovieMaker

2.6.5 Animation

Many packages available including:

- Avid SoftImage
- Animated Gif building packages *e.g. GifBuilder*

2.6.6 Multimedia Authoring

– Tools for making a complete multimedia presentation where users usually have a lot of interactive controls.

Macromedia Director

- Movie metaphor (the cast includes bitmapped sprites, scripts, music, sounds, and palettes, etc.)
- Can accept almost any bitmapped file formats
- Lingo script language with own debugger allows more control including external devices, e.g., VCRs and video disk players
- Ready for building more interactivities (buttons, etc.)
- Currently in version 7.0, this popular general market product follows the cast/score/scripting paradigm, which makes it the tool of choice for animation content. Its roots as a cel- and sprite-animation program are unmistakable; and its inclusion of Lingo, its object-based scripting language, has made it the animation-capable program to beat. The AfterBurner compression Xtra creates Shockwave files, allowing Web playback.

Authorware

- Professional multimedia authoring tool
- Supports interactive applications with hyperlinks, drag-and-drop controls, and integrated animation
- Compatibility between files produced from PC version and MAC version

Other authoring tools include:

- Microcosm : Multicosm, Ltd. ; DOS, Windows Microcosm is a Hypermedia Linkage authoring system.
- Question Mark : Question Mark Computing Ltd ; DOS, Mac, Windows; WWW (via Perception) Question Mark is optimized for Electronic Assessment production.
- Emblaze Creator : Geo International ; JavaScript, Mac, Windows95, WWW.

  Emblaze Creator 2.5 is a cast/score/scripting tool which is designed for Web-based playback of interactive multimedia.
- Flash : Macromedia ; Mac, Windows95, NT, WWW (via Flash Player).

  Flash 3.0 is a cast/score/scripting tool, which primarily uses vector graphics (and can create vector graphics from imported bitmaps). It is optimized for Web delivery, and is especially common for banner adds and small interactive web deliverables.

- HyperCard : Apple Computer ; Mac, WWW (via LiveCard!).

  HyperCard is a card/scripting authoring system currently in version 2.4.1. It runs natively on both 68K and PowerMacintosh machines, and is widely used because of its easy availability at a low price. Its largest drawback is the lack of integrated color; current color implementation is via the ColorTools XCMD set (included) or via third-party XCMDs.

- HyperGASP : Caliban Mindwear.

  HyperGASP is a card/scripting authoring system currently in version 3.0; the newest version no longer requires HyperCard. Supports export to HTML for Web authoring.

- HyperStudio ; Roger Wagner Publishing ; Mac, Windows, WWW (via HyperStudio plug-in).

  HyperStudio is a card/scripting paradigm authoring system, optimized for and focussed on the educational market.

- IconAuthor : Asymetrix ; Windows, NT, Solaris, UNIX, WWW (via Windows).

  IconAuthor follows the iconic/flow control paradigm. It is notable for its SmartObject editor, which tags content files (still graphics, RTF text, etc.) for interactivity. It has the option to either embed content files or leave them external in specified directories. The biggest strength of this program is its included data handling, which makes it unparalleled for CBT data tracking. The latest version should also provide WWW porting of existing content. Avoid its internal "Move Object" path animation feature due to jerky response – use a .FLC or .AVI instead

2.7 Further Information

See Chapter 12 *Multimedia Presentation and Authoring* pages 285–303, Multimedia Systems, J.F.K, Buford, ACM Press, 1994 (ISBN 0-201-53258-1).

See also:

- Mutlimedia Frequently Asked Question
  (URL *http://www.tiac.net/users/jasiglar/MMASFAQ.HTML*)
- *http://hibp.ecse.rpi.edu/ connor/education/storyboard.html*— StoryBoarding URLs
- Director Web — *http://www.mcli.dist.maricopa.edu/director/*

Chapter 3

Multimedia Programming:Scripting (Lingo)

In the last chapter we covered many Multimedia Authoring paradigms. Some of these basically involve only graphical programming. However, there is always a limit to how much can be achieved this ways, and so we must resort to scripting or hard programming.

We will examine two of these programming paradigms a little further in the coming 2 chapters:

- Scripting — we will overview the Director Authoring systems and in particular it's *LINGO* scripting language
- Tagging — we will overview *SMIL* an extension of XML for synchronised media integration.

3.1 Director programming/Lingo Scripting

This section provides a very brief overview of Basic ideas of Director programming based on the *Cast/Score/Scripting* paradigm.

This section is essentially a precise of Director's own manual pages. You should consult:

- *Macromedia Director 7: Using Director* Manual — *In Library*
- *Macromedia Director 7: Lingo Dictionary* Manual — *In Library*
- *Macromedia Director: Application Help* — Select *Help* from within the Director application. This is very thorough resource of information.

- *Macromedia Director Guided tours* — see Help menu option.
- A variety of web sites contain director tutorials, hints and information:
  - Director Web: *http://www.mcli.dist.maricopa.edu/director/*
  - Macromedia Director Support Center: *http://www.macromedia.com/support/director/*
  - An excellent set of Basic Director tutorial may be found at: *http://www.fbe.unsw.edu.au/subjects/BENV/1043/tutorials.htm*
  - The book *Creating and Designing Multimedia with Director*, P. Petrik and B. Dubrovsky, Prentice Hall, 1997 (ISBN 0-13-528985-8) is also worth investigating.

3.2 Director Basics

3.3 Overview and Definitions

The Basic commodity in Director is the *Director Movie*:

Director movies are interactive multimedia pieces that can include animation, sound, text, digital video, and many other types of media. A movie can be as small and simple as an animated logo or as complex as an online chat room or game.

You're probably familiar with Director movies in the Shockwave movie format, which play in web browsers.

A movie may link to external media or be one of a series of movies that refer to each other. Director's interactivity lets the movie respond to events and change in specified ways.

Director divides lengths of time into a series of *frames*, similar to the frames in a celluloid movie.

When creating and editing movies, you typically work in the four key windows that make up Director's *work area*:

the Stage , the rectangular area where the movie plays(Fig. 3.1):

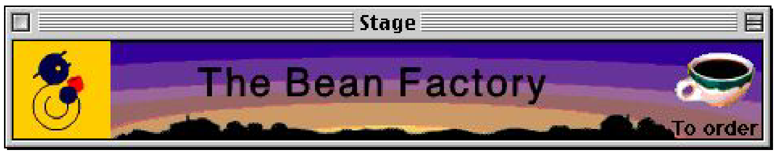


Figure 3.1: Macromedia Director Stage Window

the Score , where the movie is assembled(Fig. 3.2);

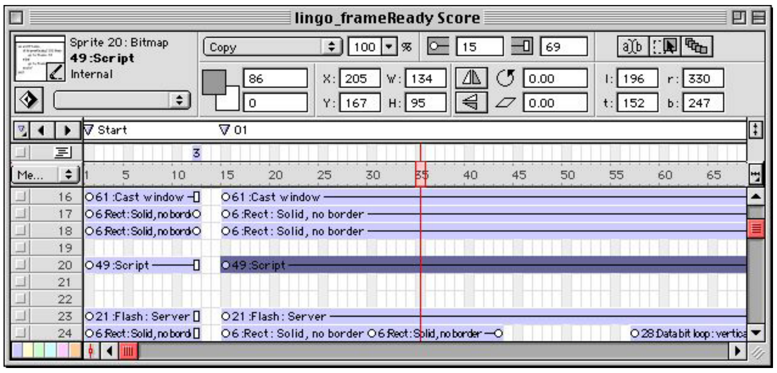


Figure 3.2: Macromedia Director Score Window

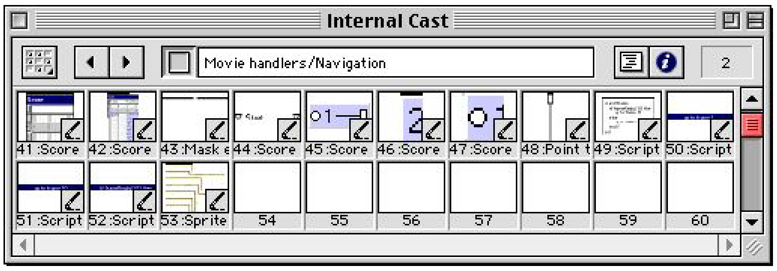


Figure 3.3: Macromedia Director Cast Window

one or more Cast windows , where the movie's media elements are assembled (Fig. 3.3);

and

the Control Panel , which controls how the movie plays back(Fig. 3.4).

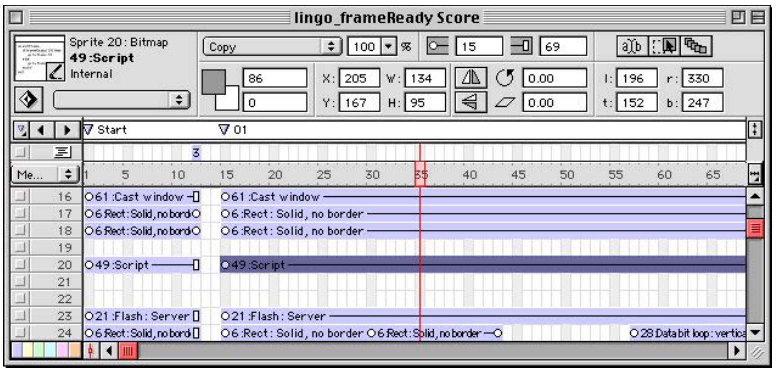

lingo_frameReady Score
Sprite 20: Bitmap
49 :Script
Internal
Copy
100 %
15
69
86
0
X: 205
Y: 167
W: 134
H: 95
0.00
0.00
l: 196
t: 152
r: 330
b: 247
Start
01
3
Me...
1 5 10 15 20 25 30 35 40 45 50 55 60 65
16 61 :Cast window 61 :Cast window
17 6 Rect: Solid, no bord 6 :Rect: Solid, no border
18 6 Rect: Solid, no bord 6 :Rect: Solid, no border
19
20 49 :Script 49 :Script
21
22
23 21 :Flash: Server 21 :Flash: Server
24 6 Rect: Solid, no bord 6 :Rect: Solid, no border 6 Rect: Solid, no border 28 Data bit loop: vertica

Figure 3.4: Macromedia Director Control Panel

To create a new movie:

- Choose File > New > Movie.

3.4 The Score and the Stage

The Score coordinates the movie's media, determining when images appear and sounds play. Special channels control the movie's tempo, sound, and color palettes. The Score also assigns scripts (Lingo instructions) that specify what the movie does when certain events occur in the movie (more on this shortly).

The Stage is the visible portion of a movie. Use the Stage to determine where media appears. Working together, the Score's settings and controls create a dynamic, high-quality interactive piece that plays in a web page or as a standalone application directly on the user's local computer.

Movie properties specify properties that affect the entire movie, such as how colors are defined, the size and location of the Stage, the number of channels in the Score, copyright information, and font mapping.

Use the Movie > Properties command to set these settings. These settings apply only to the current movie, whereas the settings you choose from File > Preferences apply to every movie. See manuals for exact properties and parameters *etc.*—

3.5 Using The Score

The Score organizes and controls a movie's content over time in channels. The most important components of the Score are *channels, frames*, and the *playback head*.

You can control the Score by zooming to reduce or magnify your view and by displaying multiple Score windows. You can also control the Score's appearance using preference settings.

To display the Score:

- Choose Window > Score.

3.6 The playback head

The playback head moves through the Score to show what frame is currently displayed on the Stage. The playback head moves to any frame you click in the Score. (See Fig 3.6)

3.7 Channels

Channels are the rows in the Score that contain sprites for controlling media. Sprite channels are numbered and contain the sprites that control all the visible media in the movie. Special effects channels at the top of Score contain behaviors as well as controls for the tempo, palettes, transitions, and sounds. The Score displays channels in the order shown in Fig. 3.5.

The channel at the very top of the Score contains markers that identify places in the Score, such as the beginning of a new scene. Markers are useful for making quick jumps to certain locations in a movie.

The Score can include up to 1000 channels. Most movies contain a much smaller number. To improve a movie's performance in the authoring environment and during playback, you should not use many more channels than necessary. Use Movie Properties to control the number of channels in the Score for the current movie.

Use the button at the left of any channel to hide its contents on the Stage or to disable the contents if they are not visible sprites. When you turn off a special effects channel, the channel's data has no effect on the movie. Turn Score channels off when testing performance or working on complex overlapping animations.

3.8 Frames

Frames are represented by the numbers listed horizontally in the sprite and special effects channels. A frame is a single step in the movie, like the frames in a traditional film. Setting the number of frames displayed per second sets the movie's playback speed;

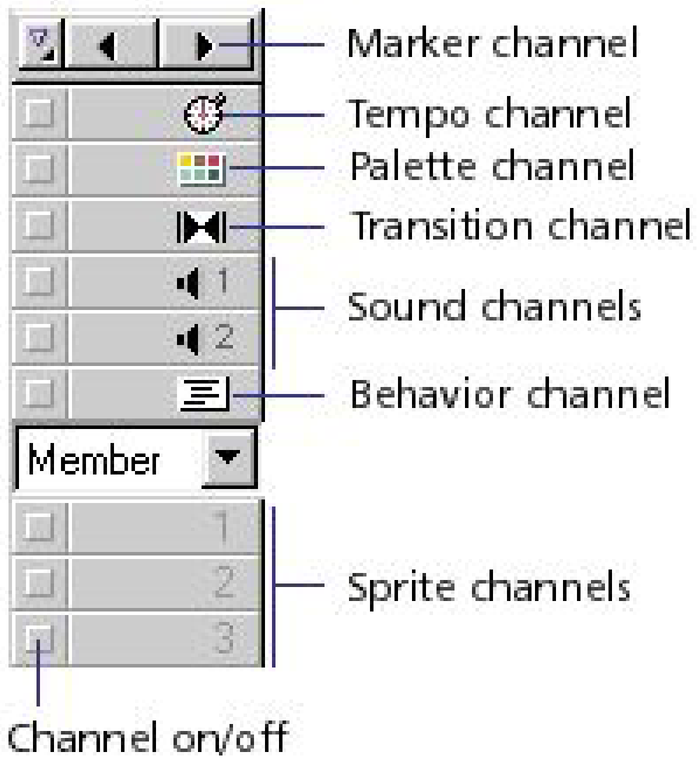


Figure 3.5: Macromedia Director Channel Display

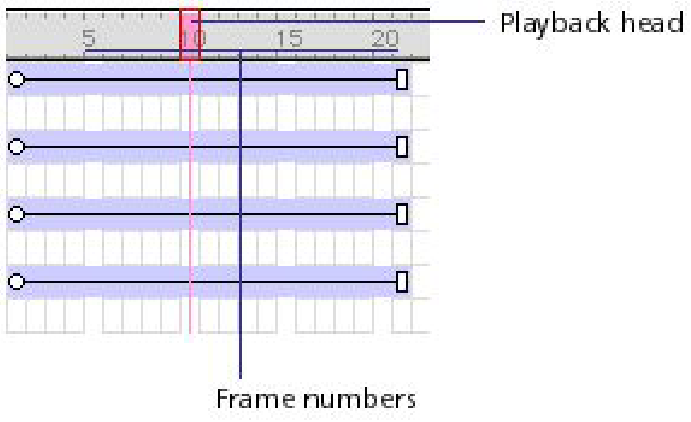


Figure 3.6: Macromedia Director Frame Playback

3.9 Sprites

Sprites are objects that control when, where, and how media appears in a movie. The media assigned to sprites are *cast members*.

Director organizes cast members in libraries called casts.

Creating a Director movie *consists* largely of defining where sprites appear on the Stage, when they appear in the movie, how they behave, and what their properties are. You work with sprites on the Stage and in the Score to change where and how cast members appear in the movie.

3.10 Cast members

Cast members are the media that make up a movie. They can include bitmap images, text, vector shapes, sounds, Flash movies, digital videos, and more.

You can create cast members directly in Director or import existing media.

3.11 Lingo

Lingo, Director's scripting language, adds interactivity to a movie. Often Lingo accomplishes the same tasks-such as moving sprites on the Stage or playing sounds-that you can accomplish using Director's interface.

Much of Lingo's usefulness, however, is in the flexibility it brings to a movie. Instead of playing a series of frames exactly as the Score dictates, Lingo can control the movie in response to specific conditions and events.

For example, whether a sprite moves can depend on whether the user clicks a specific button; when a sound plays can depend on how much of the sound has already streamed from the Internet.

Behaviors are pre-existing sets of Lingo instructions. Attaching behaviors to sprites and frames lets you add Lingo's interactivity without writing Lingo scripts yourself.

If you prefer writing scripts to using Director's interface and behaviors, Lingo provides an alternative way to implement common Director features; for example, you can use Lingo to create animation, stream movies from the web, perform navigation, format text, and respond to user actions with the keyboard and mouse.

Writing Lingo also lets you do some things that the Score alone can't do. For example, Lingo's lists let you create and manage data arrays, and Lingo operators let you perform mathematical operations and combine strings of text. You can also write your own behaviors that perform tasks beyond those possible with the behaviors that you already have available.

3.12 Markers

Markers identify fixed locations at a particular frame in a movie. Markers are vital to navigation in movies. Using Lingo or draggable behaviors, you can instantly move the playback head to any marker frame.

This is useful when jumping to new scenes from a menu or looping while cast members download from the web. Markers are also useful while authoring to advance quickly to the next scene.

Once you've marked a frame in the Score, you can use the marker name in your behaviors or scripts to refer to exact frames. Marker names remain constant no matter how you edit the Score. They are more reliable to use as navigation references than frame numbers, which can change if you insert or delete frames in the Score.

Use the Markers window to write comments associated with markers you set in the Score and to move the playback head to a particular marker.

To create a marker:

- Click the markers channel to create a marker. A text insertion point appears to the right of the marker.
- Type a short name for the marker.

To delete a marker:

- Drag the marker up or down and out of the markers channel.

To jump to markers while authoring:
There are a few ways to do this:

- Click the Next and Previous Marker buttons on the left side of the marker channel.
- Press the 4 and 6 keys on the numeric keypad to cycle backward and forward through markers.
- Choose the name of a marker from the Markers menu.

3.13 Editing Frames

You can select a range of frames in the Score and then copy, delete, or paste all the contents of the selected frames. When you select frames, any sprite within the range is selected, even if it extends beyond the range. You can add new frames to a movie at any time.

To move or delete all the contents of a range of frames:

- Double-click and drag in the frame channel to select frames.

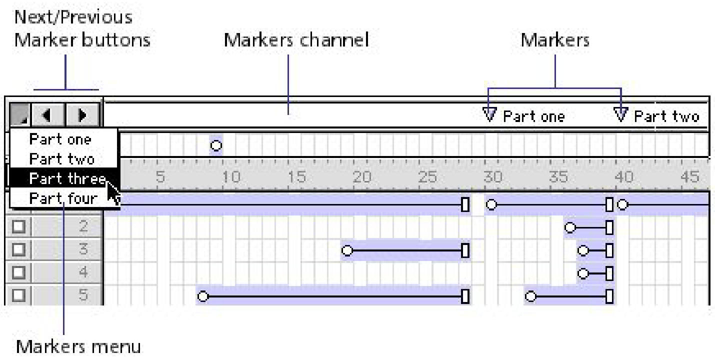


Figure 3.7: Macromedia Director Markers

- Choose Edit > Cut or Edit > Copy, or press Delete.

  If you cut or delete the selected frames, Director removes the frames and closes up the empty space.

- To paste copied frames, select any frame and choose Edit > Paste.

Note: To delete a single frame, choose Insert > Remove Frame.
To add new frames:

- Select a frame in the Score.
- Choose Insert > Frames.
- Enter the number of frames to insert.

  The new frames appear to the right of the selected frame. If there are sprites in the frames you select, they are tweened or extended.

3.14 Identifying Frames with Lingo

If you are writing scripts, use these Lingo terms to refer to frames in a movie:

- The function the frame refers to the current frame.
- The frame number or the frame marker label refers to a specific frame. For example, frame 60 indicates frame 60.
- The keyword `loop` refers to the marker at the beginning of the current segment. If the current frame has a marker, `loop` refers to the current frame; if not, `loop` refers to the first marker before the current frame.
- The word `next` or `previous` refers to the next marker or the marker before the current scene, respectively.

- The term `the frame` followed by a minus or plus sign and the number of frames before or after the current frame refers to a frame that's a specific number of frames before or after the current frame. For example, the frame - 20 refers to the frame 20 frames before the current frame.
- The function `marker()` with the number of markers used as the parameter refers to the marker that's a specific number of markers before or after the current frame. For example, `marker(-1)` gives the previous marker. If the frame is marked, `marker(0)` gives the current frame; if not, `marker(0)` gives the name of the previous marker.
- The word `movie` followed by the movie name refers to the beginning of another movie. For example,

  ```
  movie "Navigation"
  ```

  refers to the beginning of the Navigation movie.
- The word `frame` plus a frame `identifier`, the word of, the word movie, and the movie name refers to a specific frame in another movie; for example,

  ```
  frame 15 of movie "Navigation"
  ```

  refers to frame 15 of the Navigation movie.

3.15 Lingo Scripting

Lingo, Director's scripting language, adds interactivity to a movie. Use Lingo to control a movie in response to specific conditions and events. For example, Lingo can play a sound after a specified amount of the sound has streamed from the Internet.

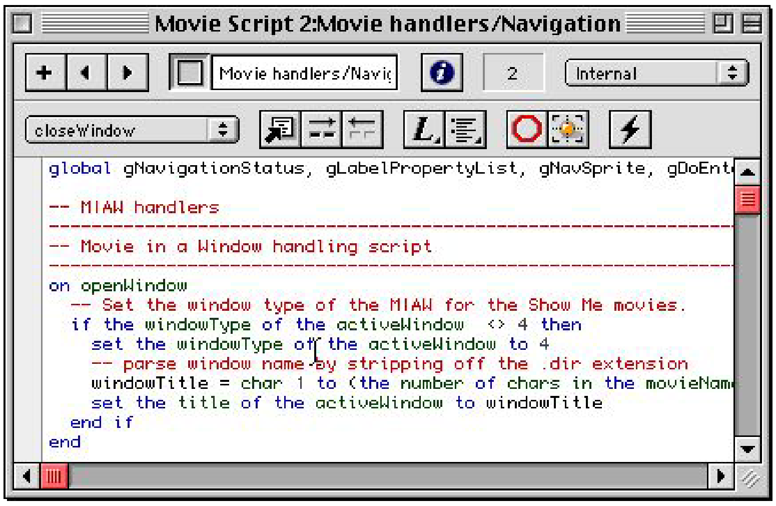
Use the Script window (Fig. 3.8) to write and edit scripts. (Window > Script or *Command-O* will display this window

For an a very good introduction to scripting, see the Lingo basics tutorial movie and online help..

3.16 When does Lingo run?

When an event occurs, Director generates a message that describes the event. For example, when the user types at the keyboard, a movie stops, a sprite starts, or the playback head enters a frame, these actions are events and generate event messages.

Handlers contain groups of Lingo statements that run when a specific event occurs in a movie. Each handler begins with the word on followed by the message that the handler is set to respond to. The last line of the handler is the word end; you can repeat the handler's name after end, but this is optional.



```
global gNavigationStatus, gLabelPropertyList, gNavSprite, gDoEnt

-- MIAW handlers
------------------------------------------------------------------
-- Movie in a Window handling script
------------------------------------------------------------------
on openWindow
  -- Set the window type of the MIAW for the Show Me movies.
  if the windowType of the activeWindow  <> 4 then
    set the windowType of the activeWindow to 4
    -- parse window name by stripping off the .dir extension
    windowTitle = char 1 to (the number of chars in the movieNam
    set the title of the activeWindow to windowTitle
  end if
end
```

Figure 3.8: Macromedia Director Script Window

For example, the `mouseDown` message indicates that the user clicked the mouse button. A handler that started with the line on `mouseDown` contains Lingo statements that run when the handler receives a `mouseDown` message. Whether the handler receives the message depends on which objects the handler is attached to in the movie.

Director contains handlers within scripts. Attach a set of handlers to an object by attaching the handlers' script to the object.

3.17 The Lingo language

Just like any scripting language, Lingo has certain elements that you use and rules that you follow.

Lingo terms fall into seven categories:

- commands — terms that instruct a movie to do something while the movie is playing. For example, go to sends the playback head to a specific frame, marker, or another movie.
- properties — attributes that define an object. For example`colorDepth`pth is a property of a bitmap cast member,
- functions — terms that return a value. For example, the date function returns the current date set in the computer. The key function returns the key that was pressed last. Parentheses occur at the end of a function,
- keywords — reserved words that have a special meaning. For example, end indicates the end of a handler,

- events,
- constants — elements that don't change. For example, the constants TAB, EMPTY, and RETURN always have the same meaning, and
- operators — terms that calculate a new value from one or more values. For example, the add operator (+) adds two or more values together to produce a new value.

A Lingo *statement* is any valid instruction that Director can execute.

An *expression* is any part of a statement, meant to be taken as a whole, that produces a value.

For example, `2 + 2` is an expression but is not a valid statement all by itself.

The line `go to frame 23` is a statement — `go to` is the command, and `frame 23` is the expression that produces the value that the command requires to execute the instruction.

Lingo supports a variety of data types:

- references to sprites and cast members,
- (Boolean) values: `TRUE` and `FALSE` ,
- strings,
- constants,
- integers, and
- floating-point numbers.

Scripts can use variables to store, update, and retrieve values as the movie plays. Use the equals operator (=) or the set command to assign values to variables or change the values of many properties.

Use `if...then, case`, and `repeat` loop structures to set up statements so that they run when specific conditions exist.

For example, you can create an `if...then` structure that tests whether text has finished downloading from the Internet and then attempts to format the text if it has.

Director always executes Lingo statements in a handler starting with the first statement and continuing in order until it reaches the final statement or a statement that instructs Lingo to go somewhere else.

Some statements that send Lingo to somewhere other than the next statement are repeat loops, `ifthenelse` structures, the exit command, the return function, and handler names placed within scripts. The order in which statements are executed affects the order in which you should place statements. For example, if you write a statement that requires some calculated value, you need to put the statement that calculates the value first. For instance, in the following example, the first statement adds two numbers, and the second assigns them to a field cast member to be displayed on the Stage:

```
x = 2 + 2
put x into member "The Answer"
```

3.18 Dot Syntax

Use dot syntax to express the properties or functions related to an object or to specify a chunk within a text object. An dot syntax expression begins with the name of the object, followed by a period (dot), and then the property, function, or chunk that you want to specify.

For example, the loc sprite property indicates a sprite's horizontal and vertical position on the Stage. The expression `sprite(15).loc` refers to the `loc` property of `sprite 15`. As another example, the number cast member property specifies a cast member's number.

The expression `member("Hot Button").number` refers to the cast member number of the Hot Button cast member. Expressing a function related to an object follows the same pattern. For example, the `pointInHyperLink` text sprite function reports whether a specific point is within a hyperlink in a text sprite.

In addition to the syntax demonstrated in the Lingo Dictionary, you can use the dot syntax `textSpriteObject.pointInHyperlink` to express this function.

For chunks of text, include terms after the dot to refer to more specific items within text. For example, the expression `member("News Items").paragraph(1)` refers to the first paragraph of the `text cast member News Items`.

The expression `member("News Items").paragraph(1).line(1)` refers to the first line in the first paragraph.

3.19 Parentheses

Functions that return values require parentheses. When you define functions in handlers, you need to include parentheses in the calling statement.

Use parentheses after the keywords sprite or member to identify the object's identifier:

for example, `member("Jason Jones-Hughes")`

. You can also use parentheses to override Lingo's order of precedence or to make your Lingo statements easier to read.

3.20 Character spaces

Words within expressions and statements are separated by spaces. Lingo ignores extra spaces. In strings of characters surrounded by quotation marks, spaces are treated as characters.

If you want spaces in a string, you must insert them explicitly.

Uppercase and lowercase letters Lingo is not case sensitive-you can use uppercase and lowercase letters however you want. For example, the following statements are equivalent:

```
member ("Cat").hilite = TRUE
Set the hiLite of member "cat" to True
set the hilite of member "Cat" to True
SET THE HILITE OF MEMBER "CAT" TO TRUE
Set The Hilite Of Member "Cat" To True
```

However, it's a good habit to follow script writing conventions, such as the ones that are used in this book, to make it is easier to identify names of handlers, variables, and cast members when reading Lingo code.

3.21 Comments

Comments in scripts are preceded by double hyphens (`--`). You can place a comment on its own line or after any statement. Lingo ignores any text following the double hyphen on the same line.

Comments can consist of anything you want, such as notes about a particular script or handler or notes about a statement whose purpose might not be obvious. Comments make it easier for you or someone else to understand a procedure after you've been away from it for a while. Use the Comment and Uncomment buttons in the Script window to enter and remove comments easily.

3.22 Optional keywords and abbreviated commands

You can abbreviate some Lingo statements. Abbreviated versions of a command are easier to enter but may be less readable than the longer versions. The go command is a good example. All the following statements are equivalent. The last one takes the fewest number of keystrokes.

```
go to frame "This Marker"
go to "This Marker"
go "This Marker"
```

It is good practice to use the same abbreviations throughout a movie.

3.23 Literal Values

A *literal value* is any part of a statement or expression that is to be used exactly as it is, rather than as a variable or a Lingo element.

Literal values that you encounter in Lingo are character strings, integers, decimal numbers, cast member names, cast member numbers, symbols, and constants.

Note: The value function can convert a string into a numerical value. The string function can convert a numerical value into a string.

Each type of literal value has its own rules.

Strings are characters that Lingo treats as characters instead of as variables. Strings must be enclosed in double quotation marks. For example, in the statement

```
member ("Greeting").text = "Hello"
```

"Hello" and "Greeting" are both strings. "Hello" is the actual string being put into a text cast member; "Greeting" is the actual name of the cast member.

Similarly, if you test a string, double quotation marks must surround each string, as in the following example:

```
if "Hello Mr. Jones" contains "Hello" then soundHandler
```

Lingo treats spaces at the beginning or end of a string as a literal part of the string. The following expression includes a space after the word to:

```
put "My thoughts amount to "
```

Director works with integers between -2,147,483,648 and +2,147,483,647. (For numbers outside of this range, use floating-point numbers.) Enter integers without using commas. Use a minus (-) sign for negative numbers. You can convert a decimal number to an integer by using the integer() function. For example, the statement

```
set theNumber = integer(3.9)
```

rounds off the decimal number 3.9 and converts it to the integer 4.

Some Lingo commands and functions require integers for their parameters. See the entry for the specific Lingo element for more information.

A *decimal number*, is what Lingo refers to as a floating-point number. The `floatPrecision` property controls the number of decimal places used to display these numbers. (However, Director always uses the complete number in calculations.) See the `floatPrecision` for information about setting the number of decimal places used for decimal numbers. You can also use exponential notation with decimal numbers: for example, -1.1234e-100 or 123.4e+9. You can convert an integer or string to a decimal number by using the float() function. For example, the statement

```
set theNumber = float(3)
```

stores the value 3.0 in the variable.

3.24 Identifying cast members and casts

Lingo refers to a cast member by using the term member followed by a cast member name or number in parentheses. If more than one cast member has the same name, Director uses the lowest numbered cast member in the lowest numbered cast. (Cast member names are strings and follow the same syntax rules as other strings.) An alternative syntax is the term `member` without parentheses, followed by the cast `member` name or number.

For example, the following all refer to cast `member 50`, which has the name `Jason_Jones-Hughes`:

```
member("Jason_Jones-Hughes")
member(50)
member "Jason_Jones-Hughes"
member 50
```

Use an optional second parameter to specify the cast member's cast. If more than one cast contains a cast member with the same name, you must also specify the cast. If you identify a cast member by its cast member number, you must also specify the cast. To specify a cast when using member without parentheses, include the term of `castLib` followed by the cast's name or number.

When the cast member's name is unique in the movie, the cast's name isn't required, but you can include it for clarity. For example, the following statements refer to cast member 50, which is named `Jason_Jones-Hughes`, in castLib 4, which is named `Wales_Player`:

```
member(50, 4)
member 50 of castLib 4
member("Jason_Jones-Hughes", 4)
member "Jason_Jones-Hughes" of castLib 4
member(50, "Wales_Player")
member 50 of castLib "Wales_Player"
member("Jason_Jones-Hughes", "Wales_Player")
member "Jason_Jones-Hughes" of castLib "Wales_Player"
```

If more than one cast member has the same name and you use the name in a script without specifying the cast or cast member number, Lingo uses the first (lowest numbered) cast member in the lowest numbered cast that has the specified name.

A *symbol* is a string or other value that begins with the pound sign (`#`). Symbols are user-defined constants. Comparisons using symbols can usually be performed very quickly, providing more efficient code. For example, the statement

```
userLevel = #novice
```

runs more quickly than the statement

```
userLevel = "novice"
```

Symbols can't contain spaces or punctuation. Convert a string to a symbol by using the symbol() function. Convert a symbol back to a string by using the string() function.

A *constant* is a named value whose content never changes. For example, TRUE, VOID, and EMPTY are constants because their values are always the same. The constants BACKSPACE, ENTER, QUOTE, RETURN, SPACE, and TAB refer to keyboard characters. For example, to test whether the user is pressing the Enter key, use the following statement:

```
if the key = ENTER then beep
```

3.25 Lingo operators

Operators are elements that tell Lingo how to combine, compare, or modify the values of an expression. They include:

- Arithmetic operators (such as +, -, /, and *)
- Comparison operators (for example, +, >, and >=), which compare two arguments
- Logical operators (not, and, or), which combine simple conditions into compound ones
- String operators (`&` and `&&`), which join strings of characters

3.26 Lingo Lists

In Lingo, Lists provide an efficient way to track and update an array of data such as a series of names or the values assigned to a set of variables.

Lists are basically a set of elements separated by commas. Lingo encloses the set of values in square brackets. A simple example of a list is a list of numbers such as `[1, 4, 2]`.

Lingo can create, retrieve, add to, reorder, sort, or substitute a list's contents. Director offers two types of lists:

Linear lists — In which each element is a single value. For example, this list is a simple set of numbers:

```
[100, 150, 300, 350]
```

Property lists — In which each element contains two values separated by a colon. The first value is a *property.* The second value is the *value associated* with that property.

For example, this list could be a sprite's Stage coordinates, with a value for each one:

```
[#left:100, #top:150, #right:300, #bottom:350]
```

Lingo has many functions that operate on lists. You can display lists (`put`), find list lengths (`count()`), find largest and smallest elements in the list (`max()`, `min()`), add and delete items in a list (`append, add, .....`), copy (`duplicate()`), and `sort` lists. See the Lingo dictionary for complete details..

3.27 Types of Scripts

Director uses four types of scripts.

Behaviors — Behaviors are attached to sprites or frames in the Score. Behaviors assigned to sprites are sprite behaviors. Behaviors assigned to a frame's behavior channel are frame behaviors. Director includes a set of behaviors that are already written. Using Lingo, you can create additional behaviors for your specific needs. Behaviors are script cast members that appear in a Cast window. The Cast window thumbnail for each behavior contains a behavior icon in the lower-right corner.

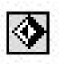


Figure 3.9: Behavior Icon

All behaviors appear in the Sprite Inspector's Behavior pop-up menu. (Other types of scripts don't appear in the Behavior pop-up menu.) Attach behaviors to sprites or frames in two ways:

- Drag a behavior from a cast to a sprite or frame in the Score or on the Stage.
- Select the sprites or frames that you're attaching the behavior to and then choose the behavior from the Behavior pop-up menu.

You can attach the same behavior to more than one location in the Score. When you edit a behavior, the edited version is applied everywhere the behavior is attached in the Score.

Movie scripts Movie scripts are available to the entire movie, regardless of which frame the movie is in or which sprites the user is interacting with. When a movie plays in a window or as a linked movie, a movie script is available only to its own movie. In addition to responding to events such as key presses and mouse clicks, movie scripts can control what happens when a movie starts, stops, or pauses. Handlers in a movie script can be called from other scripts in the movie as the movie plays. Movie scripts are cast members that appear in a Cast window. A movie script icon appears in the lower-right corner of the movie script's Cast window thumbnail.

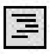


Figure 3.10: Movie script icon

Parent scripts Parent scripts are special scripts that contain Lingo used to create child objects. Parent scripts are cast members that appear in a Cast window. A parent script icon appears in the lower-right corner of the Cast window thumbnail:

Figure 3.11: Parent script icon

Scripts attached to cast members Scripts attached to cast members are attached directly to a cast member, independent of the Score. Whenever the cast member is assigned to a sprite, the cast member's script is available. Unlike movie scripts, parent scripts, and behaviors, cast member scripts don't appear in the Cast window. Open scripts attached to cast members by clicking Script in the cast member's Cast Member Properties dialog box or by selecting a cast member in the Cast window and then clicking the Script button. You can also open a cast member script from the Script window.

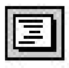


Figure 3.12: Script button

If Show Cast Member Script Icons is selected in the Cast Window Preferences dialog box, cast members that have a script attached display a small Script icon in the lower-left corner of their thumbnails in the Cast window.

3.28 Messages and Events

To run the appropriate set of Lingo statements at the right time, Director must determine what is occurring in the movie and which Lingo to run in response to specific events.

Director sends messages to indicate when specific events occur in a movie, such as when sprites are clicked, keyboard keys are pressed, a movie starts, the playback head enters or exits a frame, or a script returns a certain result.

Handlers contain instructions that run when a specific message is received. The handler's name begins with the word on followed by the message name. When an object receives a message that corresponds to a handler attached to the object, Director runs the Lingo statements within the handler. For example, a handler named on `enterFrame` that is attached to a frame runs when the playback head enters the frame.

Most common events that occur in a movie have built-in message names. See the following categories in the *Lingo Dictionary* for the built-in messages that describe events:

- Keyboard and mouse events.
- Frame events.
- Browser and Internet events.
- Sprite events.
- Movie in a window events.
- Movie events.
- Synchronizing media events.
- Idle events.
- Timeout events.
- Authoring behavior events.

You can also define your own messages and corresponding handler names. A *custom message* can call another script, another handler, or the statement's own handler. When the called handler stops executing, the handler that called it resumes. Director can send a custom message from any location. The message is first available to handlers in the script from which the message was sent. If no handler is found, the message is available to movie scripts. If more than one movie script contains a handler for the message, the handler in the movie script that has the lowest cast member number is executed. A custom handler name must:

- Start with a letter
- Include alphanumeric characters only (no special characters or punctuation)
- Consist of one word or multiple words connected by an underscore-no spaces are allowed
- Not be the same as the name of a predefined Lingo element

Using Lingo keywords for handler names can create confusion. Although it is possible to explicitly replace or extend the functionality of a Lingo element by using it as a handler name, this should be done only in certain advanced situations. When you have multiple handlers with similar functions, it is useful to give them names that have similar beginnings so they appear together in an alphabetical listing, such as the listing displayed by the Find Handler option in the Edit menu.

Director follows a definite order when sending messages about events that occur during the course of a movie. When the movie first starts, events occur in the following order:

- `prepareMovie`.
- `beginSprite`. This event occurs when the playback head enters a sprite span.
- `prepareFrame`. Immediately after the `prepareFrame` event, Director plays sounds, draws sprites, and performs any transitions or palette effects. This event occurs before the `enterFrame` event. An on `prepareFrame` handler is a good location for Lingo that you want to run before the frame draws.
- `startMovie`. This event occurs in the first frame that plays.

When Director plays a frame, events occur in this order:

- `beginSprite`. This event occurs only if new sprites begin in the frame.
- `stepFrame`.
- `prepareFrame`. Immediately after the `prepareFrame` event, Director plays sounds, draws sprites, and performs any transitions or palette effects. This event occurs before the `enterframe` event.
- `enterFrame`. After `enterFrame` and before `exitFrame`, Director handles any time delays required by the tempo setting, idle events, and keyboard and mouse events.
- `exitFrame`.
- `endSprite`. This event occurs only if the playback head exits a sprites in the frame.

When a movie stops, events occur in this order:

- `endSprite`. This event occurs only if sprites currently exist in the movie.
- `stopMovie`.

A movie can contain more than one handler for the same message. Director manages this situation by sending the message to objects in a definite order.

The general order in which messages are sent to objects is as follows:

- Messages are sent first to behaviors attached to a sprite involved in the event. If a sprite has more than one behavior attached to it, behaviors respond to the message in the order in which they were attached to the sprite.
- Messages are sent next to a script attached to the cast member assigned to the sprite.
- Messages are then sent to behaviors attached to the current frame.
- Messages are sent last to movie scripts.

When a message reaches a script that contains a handler corresponding to the message, Director executes the handler's instructions.

After a handler intercepts a message, the message doesn't automatically pass on to the remaining locations. (You can use the pass command to override this default rule and pass the message to other objects.) If no matching handler is found after the message passes to all possible locations, Director ignores the message.

The exact order of objects to which Director sends a message depends on the message. See the message's *Lingo Dictionary* entry for details about the sequence of objects to which Director sends specific messages.

You can place handlers in any type of script. However, the following are some useful guidelines for many common situations:

- *To set up a handler that affects a specific sprite or runs in response to an action on a specific sprite*:

  Put the handler in a behavior attached to the sprite.
- *To set up a handler that should be available any time that the movie is in that frame*:

  Put the handler in a frame script attached to the frame.

  For example, to have a handler respond to a mouse click while the playback head is in a frame, regardless of where the click occurs, place an `on mouseDown` or `on mouseUp` handler in the frame script rather than a sprite script.
- *To set up a handler that runs in response to an event that affects a cast member, regardless of which sprites use the cast member*:

  Put the handler in a cast member script.
- *To set up a handler that runs in response to messages about events anywhere in the movie*:

  Put the handler in a movie script.

  A script can contain multiple handlers. It's a good idea to group related handlers in a single place, though, for easier maintenance.

3.29 Director Example 1: Simple Animation

This example does not employ any scripting. It illustrates how easy it is to create a basic animation where a cast member (a ball graphic) is placed at different locations in the scene. We also introduce some features of Director that we have not yet met (these should be fairly straightforward to understand). **Note:** there are still many features of Director we will have not touched.

The following steps achieve a simple bouncing ball animation along a path:

1. Let us begin by creating a new movie and setting the Stage size:
   - Start a New movie: File > New > Movie (Shortcut = Command+N)
   - Choose Modify > Movie > Properties. In stage size, choose 640 x 480.
2. Now let us create a ball, using a the vector shape tool:
   - Choose Window > Vector Shape (Shortcut = Command+Shift+V)
   - Click the filled ellipse button.
   - Draw an ellipse (circle) about the size of the Vector Shape Window - don't worry about being exact, we will be changing the size of it later.
   - Click on the Gradient fill button. This fills the ellipse with the default colours, which happen to be a light grey to red (unless someone has changed it on your computer).
   - To change the colours, click the colour box on the left side of the Gradient colour control and choose a sky blue from the colour menu. You will notice the Fill colour chip changes too. Change the colour on the right side of the Gradient Colours to a dark blue.
   - Change the Gradient type pull-down menu at the top of your window from Linear to Radial.
   - Change the Stroke Colour to white - notice how the outline of the ellipse disappears.
3. Now let us change a few other properties of this ellipse - for us to compare these changes, we will make a copy of this cast member.
   - Close the Vector Shape window.
   - In the Cast Window, select the ellipse. Choose Edit > Duplicate (Shortcut = Command+D). A copy of the cast member is produced in the next available cast slot. Double click the new cast, which opens it in the Vector Shape Tool.
   - Change the Cycles to 3 and the Spread to 200. Click the Previous Cast button and compare the 2 ellipses. Experiment with different cycles and spreads to get an idea of what they mean.

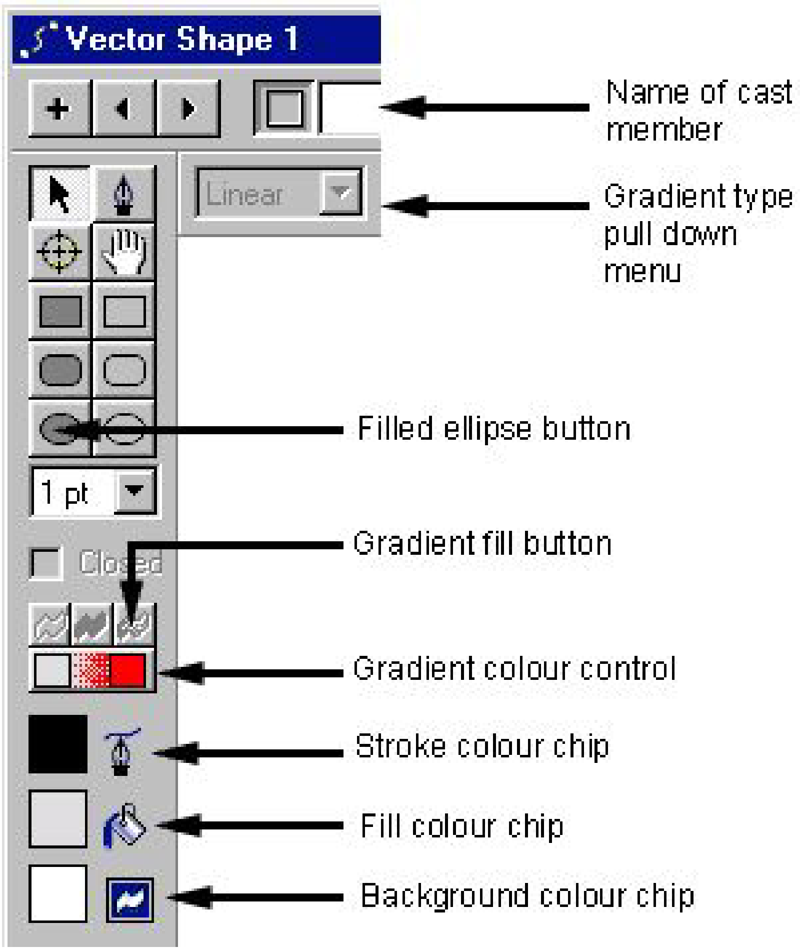


Figure 3.13: Macromedia Vector Shape Window

- Name the latest ellipse to 'bouncing ball' - this can either be done at in the Vector Shape window or the Cast Member Window.

4. Now we are going to animate the ball.

    - Drag 'bouncing ball' from the cast member window to the stage. You could drag and drop it straight into the score, which would automatically centre it on the stage.
    - You will notice the sprite (the object that appears in the score) is extended over 20 frames. This is a default setting that we can change. Drag the right end of the sprite to frame 40.
    - Click anywhere in the middle of the sprite to select it. We are now going to resize the ellipse. Click on the top of the stage window to make it active. Press Shift and while still holding down, click on a corner handle of the spite and drag it in to make it smaller. Holding down Shift lets us resize the object in proportion to its original dimensions. Resize the sprite to approximately the size shown in diagram 2, and move it to the left side of the stage.
    - Click on frame 40 in channel 1 (the end of the sprite), hold down Option and shift and drag the ellipse to the right end of the stage. (Holding down shift restricts the movement to 90 degrees). With Option selected You will notice a line being drawn on the stage - this is the animation path. Rewind and play the movie to see what you made.
    - To curve the path, we are going to insert keyframes within the sprite. Click on frame 10 of the sprite and choose Insert > Keyframe (Shortcut = Command+Option+K) Create keyframes at frame 20 and 30.
    - You will notice at each keyframe, a circle appears on the path shown on the stage. Click on the keyframe 10 circle and drag it up. Do the same with keyframe 30, producing a path similar to that shown below. Rewind and play the movie.

5. Save the movie as *example1.dir*.

Further Animation
1. Shrinking the ball

- (Optional) Click on keyframe 40 in the score and drag it to frame 60, notice how all the keyframes spread out proportionally.
- (Optional) Click on the keyframes in the score and adjust the path if you feel like it.
- While moving the keyframes, resize the balls so they slowly get smaller. Notice while you resize the balls, the path changes and you will need to edit the path again.

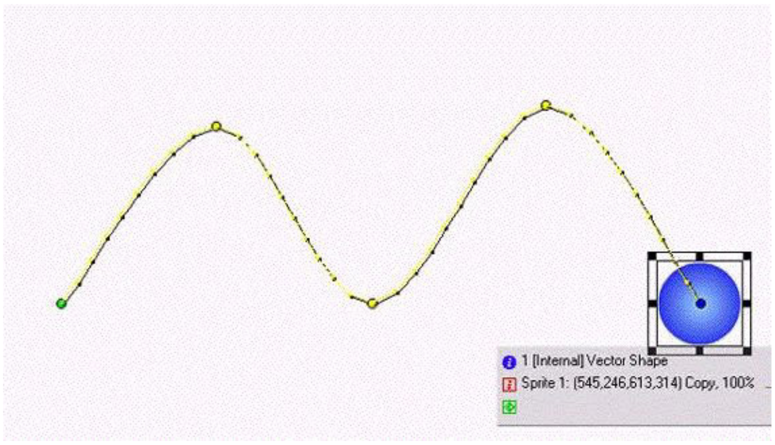

1 [Internal] Vector Shape
Sprite 1: (545,246,613,314) Copy, 100%

Figure 3.14: Curving the Ball's Path

- Rewind and play the movie.
- Save your movie as *example2.dir*.

2. Animating sprite colour

- Working still with *example1.dir*.
- Click on the keyframes in the score, and change the Foreground colour chip to different colours.
- Changing the foreground colour (refer to Fig 3.15) is like putting a coloured film over your object. The resulting colour is a mixture of the object's original colour and the 'film'. For this reason, light colours work better than dark colours for this effect..

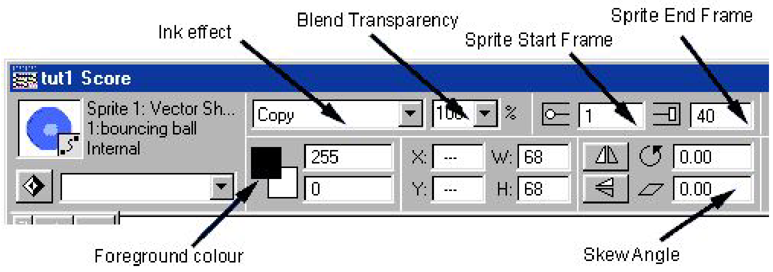


Figure 3.15: Macromedia Director Score Property Selection

- Rewind and play the movie.

- Save as *example3.dir.*

3. Animating sprite transparency — Making the Ball Disappear

- Open *example1.dir*
- Click on the keyframes in the score, and change the Blend Transparency (refer to Fig 3.15) to 100, 75, 50, 25, 0 for the consecutive keyframes.
- Rewind and play the movie.
- Save as *example4.dir.*

4. Animating sprite shape — Deforming The Ball

- Open *example2.dir*
- Click on the keyframes in the score, and change the Skew Angle (refer to Fig 3.15) to 0, 20, 40, 60 and 80 for the consecutive keyframes.
- Rewind and play the movie
- Save as *example5.dir.*

3.30 Director Example 2:Importing media

To import multimedia data there are two basic ways:

- Choose File > Import ...
- Drag and drop source media into a cast member location.

The first method is useful for importing batches of data (*e.g.* Several image sequences. The latter is clearly very intuitive.
Example: Simple Image import and Manipulation

- Drag an image into a spare cast member.
- Drag this cast member to the Score
- Manipulate as for a vector item above.
- Examples:
  - *ex_dave_roll.dir.*
    sets up some keyframes and alters the rotation of the image
  - *ex_dave_sq.dir.*
    alters the skew angle

Example: Falling Over Movie, ex_dave_movie.dir

- Several Gif images depicting sequence exist on disk.
- Choose File > Import
- Select items you wish to import by double-clicking or pressing the Add button (Fig. 3.16).

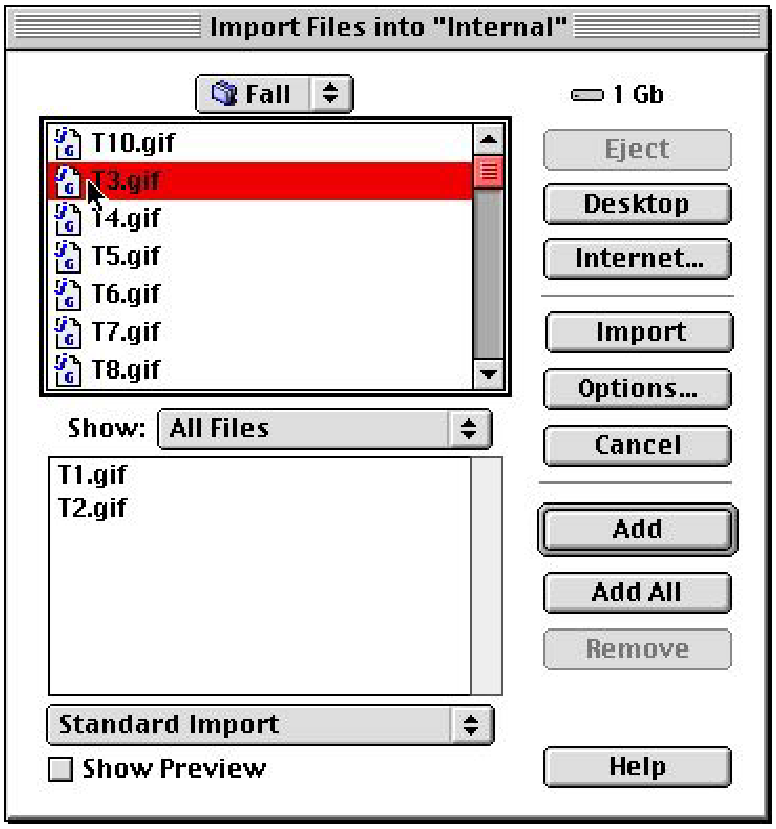


Figure 3.16: Macromedia Director Import Window

- Click on the Import Button
- Several new cast members should be added
- Set looping on and play

4. Example: Pinching Movie Movie, ex_dave_pinch.dir
To obtain other graphical effects external packages such as photoshop may be used.

- Photoshop has been used to set a pinch effect of varying degree for an image.

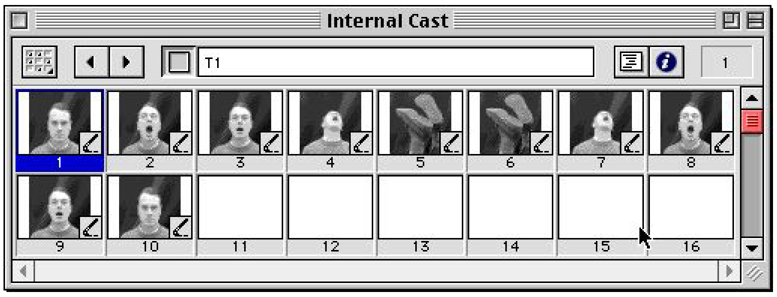


Figure 3.17: Falling Cast Members

- Import images as above
- We now need to reverse the image set to obtain a smooth back and forth animation:
  - Select the sprite sequence in the score by clicking anywhere in the middle of the sprite sequence- press Command+C (Copy), then click on the frame just after the sprite sequence and press Command+V (Paste).
  - Click on this second sprite sequence and choose Modify > Reverse Sequence.
  - Select the 2 sprites by pressing Shift and clicking on both. Choose Modify > Join Sprites.
- Rewind and play the movie.

3.31 Director Example 3:Simple Lingo Scripting

Here we illustrate the basic mechanism of scripting in Director by developing and extending a very basic example: Making a button beep and attaching a message to a button

Making the a button beep

- Open a new movie.
- Turn the looping on in the control panel.
- Open the tool palette.
- Click the push button icon.
- Draw a button on the stage, and type in button (a very original name).
- Press Ctrl+click the button in the cast window and choose cast member script.

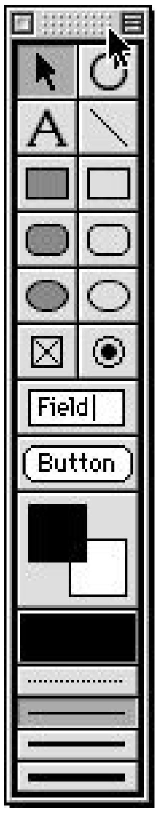
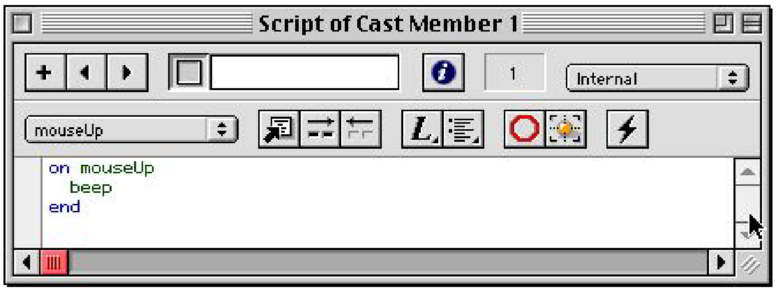


Figure 3.18: Macromedia Director Tool Palette



```
on mouseUp
  beep
end
```

Figure 3.19: Macromedia Director Script of Cast Member Window

- Director writes the first and last line for us, add a beep command so the script look like this:

```
on mouseUp
   beep
end
```

- Close the window.
- Rewind and play the movie.
- Click the button a few times.

To pop up a message box on button press (and still beep)

- Reopen the cast member script.
- Change the text so it now reads.

```
on mouseUp
  beep
  alert "Button Pressed"
end
```

- Close the window.
- Play the movie and click the button.

3.32 Director Example 4:Controlling Navigation with Lingo

We begin we a preassembled Director movie:

- Open *lingo_ex.3.1.dir*
- Play the movie and to see it does.

We are first going to create a *loop the frame* script:

- The scripting window appears. Change the text so it now reads.

```
on exitFrame
   go the frame
end
```

This frame script tells Director to keep playing the same frame.

- Pressing down Alt, drag the frame script to frame 24.

Now We will create some *markers*
As we have mentioned in above Director allows you to mark certain frames in your movie so they can be identified by buttons and links.

- Click in the marking channel in frame 1, as soon as you click a marker is created with text naming it as New Marker. Type in `scene1` over this text.
- Create markers at frame 10 and 20, naming them `scene2` and `scene3` respectively. You can delete a marker by clicking the triangle and dragging it below the marker channel.
- Create a cast member script for the next button which reads:

  ```
  on mouseUp
      go to next
  end
  ```

  The `go to next` command tells Director to go to the next consecutive marker in the score.
- Create a cast member script for the back button which reads:

  ```
  on mouseUp
      go to previous
  end
  ```

  The `go to previous` command tells Director to go to the previous marker in the score.
- Right click on the next button cast member, go to cast member properties, then check the Highlight when Clicked option. Do the same for the back button.
- Play the movie, click on the buttons and see how they work.
- Save the movie.

Now We will create some sprite scripts:
Sometimes a button will behave one way in one part of the movie and another way in a different part of the movie. That's when we use sprite scripts.

- Click on frame 10 of channel 6 (the next button) this sprite and choose Modify > Split Sprite. Do the same at frame 20.
- Select the first sprite sequence in channel 6 and select new behaviour from the behaviour pull-down in the score.

- A script window will pop up. Type `go to "scene2"` in the command area. This command tells Director to send the movie to marker `"scene2"`.

Using the *Behaviour Inspector* (Fig. 3.20):
Another way to write a sprite script - using the Behaviour Inspector.

- elect the second sprite sequence in channel 6.
- Open the Behaviour Inspector window - click the diamond shaped icon next to the behaviour pull-down menu.

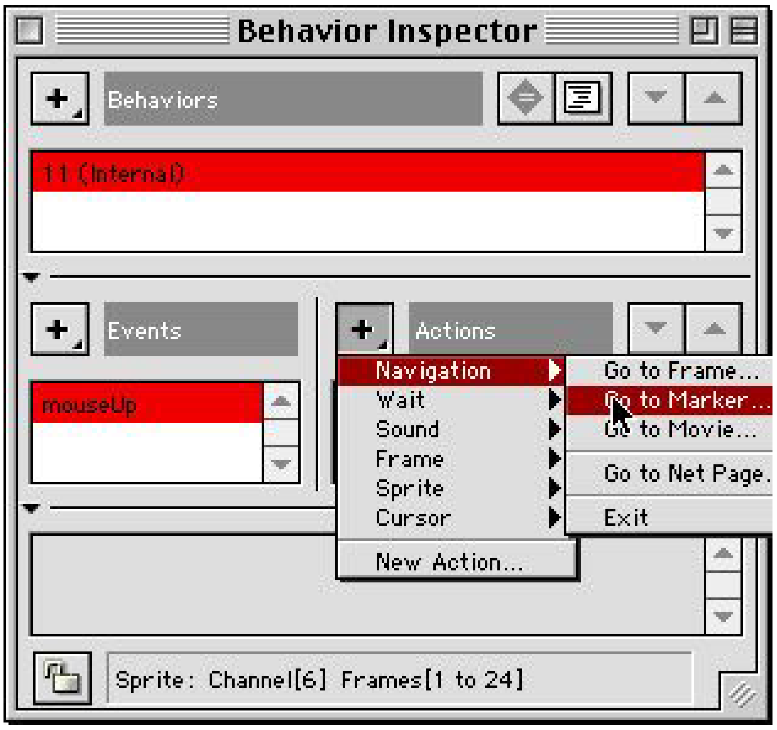


Figure 3.20: Macromedia Director Behaviour Inspector Window

- Expand the behaviour inspector window, so you can see events and actions. Click on the little triangle pointing to the right, when the window expands the triangle will point down.
- Click the + icon at the top left of the window and select new behaviour from the pull-down.
- Give the behaviour a name, I called mine `next2`.
- Under Events click the + icon and choose `mouseUp` from the menu.

- Under Actions click the + icon and choose Navigation > Go to marker then find scene3 on the list (Fig. 3.20)

You may realise that we now have 2 scripts attached to a single object - a cast member script as well as a sprite script. Sprite scripts take priority over cast member scripts so in our case the cast member script will be ignored.

Using Lingo `play`/ `play do` *to record actions*:

Sometimes you may want only part an image to be linked instead of the whole object. That's when we use invisible buttons. Invisible buttons aren't magical objects, they are just shape cast members with an invisible border.

- Click on frame 1 of channel 8.
- Open the Tool palette window.
- Click on the no line button.
- Click on the rectangle button and draw a rectangle on the stage around the 1 button.
- Attach a sprite script to this shape with the command play "scene1".
- Extend the sprite sequence so it covers frame 1 to 24.
- Drag the invisible shape from the cast to the score and place it over the 2 button.
- Attach a sprite script to this sprite with the command play "scene2".
- Extend the sprite sequence so it covers frame 1 to 24.
- Repeat the steps placing the sprite over the 3 button and adding the command play "scene3".

We used a different command for the above scripts. The `play` command is similar to the `go` to command but with play,

- Director *records* every time a play is initiated,
- keeping track of the users' path through the movie.
- You can move back on along this path by using the `play done` command.
- Select the sprite sequence in channel 5.
- Attach a sprite script reading

```
on mouseUp
   play done
end
```

- Rewind, play the movie, click all the 1, 2, 3 buttons in various orders, click the back button also and observe the
- Save your movie.

You can see the completed example in *lingo_ex3.2.dir*

Chapter 4

Multimedia Programming:Tagging (SMIL)

In the last chapter we looked at scripting programming paradigms for multimedia.

In this chapter we will study *Tagging*

- We will overview *SMIL* an extension of XML for synchronised media integration.

4.1 What it is SMIL?

Synchronized Multimedia Integration Language (SMIL) is to synchronized multimedia what HTML is to hyperlinked text. Pronounced smile, SMIL is a simple, vendor-neutral markup language designed to let Web builders of all skill levels schedule audio, video, text, and graphics files across a timeline without having to master development tools or complex programming languages.

Whilst the SMIL language is a powerful tool for creating synchronized multimedia presentations on the web over low bandwidth connections. It is mainly meant to work with linear presentations where several types of media can be synchronized to one timeline. It does not work well with non-linear presentations and its ability to skip around in the timeline is buggy at best. However, for slideshow style mixed media presentations it the best the web has to offer.

For instance, with just a text editor and a few lines of HTML-like tags, SMIL lets Web builders specify such actions as

- *play audio file A five seconds after video file B starts and then show image file C.*

SMIL marks a significant step toward making it easy to create low-bandwidth, TV-like content on the Web. It offers a new level of control over synchronized multimedia by allowing individual components of a presentation to be choreographed across a timeline in relation to each other. It also lets you control the layout, appearance, and exit time of each file.

What makes SMIL different from other multimedia presentation tools is that instead of forcing each component into a single video file, the text-based SMIL file merely references each file by its URL. Since the media files exist outside of the SMIL file, they retain their individual file sizes; there's no file-size bloat to slow download times.

SMIL's text-based format also makes multimedia presentations easy to edit. If you want to change when an audio component within a complex presentation begins, you can just edit the SMIL file. You don't have to rebuild the entire presentation from scratch.

As an application of XML, SMIL supports hyperlinks, which makes it the first Web-specific multimedia language to offer true interactivity.

4.1.1 SMIL support

The W3C recommended SMIL in June 1998, but that doesn't mean it will be universally supported across the Web. Quicktime 4.0 supports SMIL (1999)

All emerging standards have opponents, and SMIL is no exception.

No Web browser currently supports SMIL, and Microsoft and Netscape have no plans to support the standard anytime soon. In fact, even though Microsoft helped develop the SMIL 1.0 specification, Microsoft now says SMIL is not mature enough to support, claiming it overlaps with several other technologies, such as Dynamic HTML and the Document Object Model. (Microsoft's streaming media player NetShow does not support SMIL either).

Probably the real reason Microsoft stopped SMILing was the fact that Microsoft, along with Macromedia and Compaq, submitted (March 1998) its own synchronized multimedia proposal to the W3C, called HTML+TIME (Timed Interactive Multimedia Extensions for HTML). This proposal claims to address SMIL's limitations:

- SMIL is a data interchange format for media authoring tools and players
- SMIL **does not include** a means to apply the ideas to HTML and Web browsers.

However, SMIL does have an active following on the Web. RealNetworks' RealPlayer, the most common streaming media player on the Web with 85 percent of the market, supports SMIL in its G2 player.

Major media companies such as CNN Interactive, Fox News, the History Channel, and dozens of others have already begun to offer SMIL-based content via its RealPlayer G2.

Many other SMIL-compliant players, authoring tools, and servers are also becoming available.

4.2 Running SMIL Applications

For this course there are basically three ways to run SMIL applications (two use the a Java Applet) so there are basically two SMIL supported mediums:

Quicktime — supported since Quicktime Version 4.0.

RealPlayer G2 — integrated SMIL support

Web Browser — use the SOJA SMIL applet viewer with html wrapper

Applet Viewer — use the SOJA SMIL applet viewer with html wrapper

You will need to use both as RealPlayer and SOJA support different media (see below and Section 4.4.3).

Using Quicktime

Load the SMIL file into a Quicktime plug-in (confgure Browser helper app or mime type) or the Quicktime movie player.

Using RealPlayer G2

The RealPlayer G2 is installed on the applications HD in the *RealPlayer* folder.

Real player supports lots of file format and can use plugins. The main supported formats are:

- Real formats: RealText, RealAudio, etc...
- Images: GIF, JPEG
- Audio: AU, WAV, MIDI, etc...

To run SMIL files

Real Player uses streaming to render presentations.

The player works better when calling a SMIL file given by a Real Server, rather than from an HTTP one.

You can also open a local SMIL file or drag a SMIL file onto the RealPlayer G2 Application

4.2.1 Using the SOJA applet

SOJA stands for SMIL Output in Java Applet. It was written by HELIO in 1998.

HELIO is a French association based in Melun, France. The player is free and can be downloaded from

www.helio.org.

SOJA is an applet that render SMIL in a web page or in a separate window. It supports the following formats:

- Images: GIF, JPEG
- Audio: AU and AUZ (AU zipped) — SUN Audio files
- Text: plain text

SOJA *does not* use streaming. HELIO chose to store media before rendering them. You have to wait until each of them has properly been loaded but the presentation will never be stopped.

Running SOJA

To run SMIL through an applet you have to

- call the applet from an HTML file:

  ```
  <APPLET CODE="org.helio.soja.SojaApplet.class"
          ARCHIVE="soja.jar" CODEBASE="../"
  WIDTH="600" HEIGHT="300">
      <PARAM NAME="source" VALUE="cardiff_eg.smil">
      <PARAM NAME="bgcolor" VALUE="#000066">
     </APPLET>
  ```

  .

- the SOJA (`soja.jar`) archive is located in the SMIL folder on the Macintoshes.
- You may need to alter the `CODEBASE` attribute for your own applications
- The `PARAM NAME="source" VALUE="MY_SMILFILE.smil"` is how the file is called.

There are plenty of HTML SMIL/applet wrapper example files in the SMIL demos folder.

RUNNING APPLETS

This should be easy to do

- Run the html file through a java enabled browser
- Use Apple Applet Runner
  - uses MAC OS Runtime Java (Java 1.2)
  - less fat for SMIL applications (we do really need Web connection for our examples)
  - Efficient JAVA and MAC OS run.
  - Located in *Apple Extras:Mac OS Runtime For Java* folder
  - *TO RUN*: Drag files on to application, **OR**
  - *TO RUN*: Open file from within application

4.2.2 another SMILE viewer:GRINS

This viewer is NOT available for this course but you may wish to investigate it.
GRINS stands for Graphical Interface for SMIL. It can be found at *www.cwi.nl*.

4.3 Let us begin to SMIL — SMIL Authoring

The notes here are essentially summaries of the SMIL Specification and tutorials available at the WWW3 consortium web site:

- http://WDVL.com/Authoring/Languages/XML/SMIL/Intro/smil.html
- http://www.w3.org/TR/REC-smil (see handout)

A SMIL document is a special kind of XML 1.0 document:

Extensible Markup Language (XML) is a human-readable, machine-understandable, general syntax for describing hierarchical data, applicable to a wide range of applications (databases, e-commerce, Java, web development, searching, etc.). XML is an ISO compliant subset of SGML (Standard Generalized Markup Language). XML is extensible because it is a metalanguage, which enables someone to write a Document Type Definition (DTD) like HTML 4.0 and define the rules of the language so the document can be interpreted by the document receiver. The purpose of XML is to provide an easy to use subset of SGML that allows for custom tags to be processed. Custom tags will enable the definition, transmission and interpretation of data structures between organizations.

Further information on XML may be found at:

- *Extensible Markup Language (XML) 1.0*, T. Bray, J. Paoli, C.M. Sperberg-McQueen Available at http://www.w3.org/TR/REC-xml/

4.4 SMIL Syntax Overview

SMIL files are usually named with `.smi` or `.smil` extensions

4.4.1 Basic Layout

The basic Layout of a SMIL Documents is as follows:

```
<smil>
 <head>
  <meta name="copyright" content="Your Name" />
  <layout>
   <!-- layout tags -->
  </layout>
 </head>
 <body>
  <!-- media and synchronization tags -->
 </body>
</smil>
```

A source begins with `<smil>` and ends with `</smil>`.

Note that SMIL, like XML *but* unlike HTML, is *case sensitive*

```
<smil>
[...]
</smil>
```

SMIL documents have two parts: head and body. Each of them must have `<smil>` as a parent.

```
<smil>
 <head>
  [...]
 </head>
 <body>
  [...]
 </body>
</smil>
```

Some tags, such as meta can have a slash at their end:

```
[...]
 <head>
  <meta name="copyright" content="Your Name" />
 </head>
[...]
```

This is because SMIL is XML-based.

Some tags are written:

- `<tag> ... </tag>`
- `<tag />`

4.4.2 SMIL Layout

Everything concerning layout (including window settings) is stored between the `<layout>` and the `</layout>` tags in the *header* as shown in the above section.

A variety of Layout Tags define the presentation layout:

```
 <smil>
 <head>
<layout>
   <!-- layout tags -->
 </layout>
......
```

Window settings

You can set width and height for the window in which your presentation will be rendered with `<root-layout>`.

The following source will create a window with a 300x200 pixels dimension and also sets the background to be white.

```
<layout>
 <root-layout width="300" height="200"
           background-color="white" />
</layout>
```

4.4.3 Positioning Media

It is really easy to position media with SMIL.

You can position media in 2 ways:

Absolute Positioning — Media are located with offsets from the origin — the upper left corner of the window.

Relative Positioning — Media are located relative to the window's dimensions.

We define position with a `<region>` tag

The Region tag

—

To insert a media within our presentation we use the `<region>` tag. we must specify the region (the place) where it will be displayed. Let's say we want to insert the `Cardiff` icon (533x250 pixels) at 30 pixels from the left border and at 25 pixels from the top border.

We must also assign an `id` that identifies the region.

The header becomes:

```
<smil>
 <head>
  <layout>
   <root-layout width="600" height="300"
        background-color="white" />
   <region id="cardiff_icon" left="30" top="25"
        width="533" height="250" />
  </layout>
 </head>
 ......
```

The img tag

To insert the Cardiff icon in the region called "cardiff_icon", we use the `<img>` tag as shown in the source below.

Note that the region attribute is a pointer to the `<region>` tag.

```
<smil>
 <head>
  <layout>
   <root-layout width="600" height="300"
        background-color="white" />
   <region id="cardiff_icon" left="30" top="25"
        width="533" height="250" />
  </layout>
 </head>
 <body>
  <img src="cardiff.gif" alt="The Cardiff icon"
        region="cardiff_icon" />
 </body>
</smil>
```

This produces the following output:

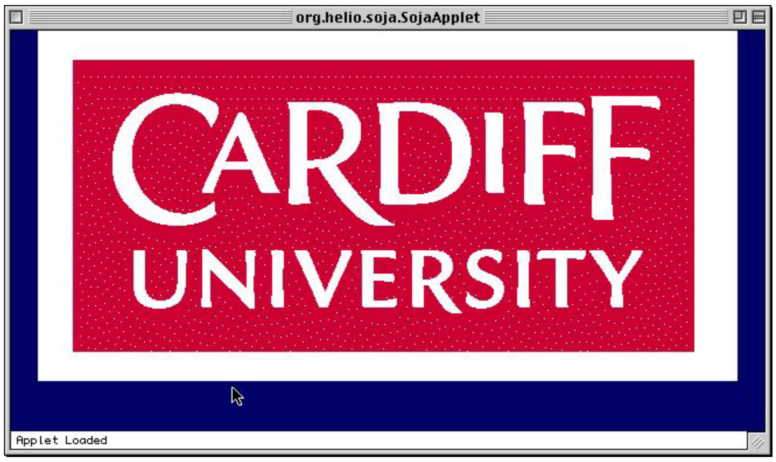


Figure 4.1: Simple Cardiff Image Placement in SMIL

Relative Position Example

if you wish to display the Cardiff icon at 10% from the `left` border and at 5% from the `top` border, modify the previous source and replace the left and top attributes.

```
<smil>
 <head>
  <layout>
   <root-layout width="600" height="600"
        background-color="white" />
   <region id="cardiff_icon" left="10%" top="5%"
       width="533" height="250" />
  </layout>
 </head>
 <body>
  <img src="cardiff.gif" alt="The Cardiff icon"
        region="cardiff_icon" />
 </body>
</smil>
```

Overlaying Regions

We have just seen how to position a media along x and y axes (left and top).
What if two regions overlap?

- Which one should be displayed on top ?

The following code points out the problem:

```
<smil>
 <head>
  <layout>
   <root-layout width="300" height="200" background-color="white" />
   <region id="region_1" left="50" top="50" width="150" height="125" />
   <region id="region_2" left="25" top="25" width="100" height="100" />
  </layout>
 </head>
 <body>
  <par>
   <text src="text1.txt" region="region_1" />
   <text src="text2.txt" region="region_2" />
  </par>
 </body>
</smil>
```

To ensure that one region is over the other, add `z-index` attribute to `<region>`.
When two region overlay:

- the one with the greater z-index is on top.
- If both regions have the same z-index, the first rendered one is below the other.

In the following code, we add `z-index` to `region_1` and `region_2`:

```
<smil>
 <head>
  <layout>
   <root-layout width="300" height="200" background-color="white" />
   <region id="region_1" left="50" top="50" width="150"
           height="125" z-index="2"/>
   <region id="region_2" left="25" top="25" width="100"
           height="100" z-index="1"/>
  </layout>
 </head>
 <body>
  <par>
   <text src="text1.txt" region="region_1" />
   <text src="text2.txt" region="region_2" />
  </par>
 </body>
</smil>
```

Support for Other Media

We have used images and text media in previous presentations but other media can be introduced within SMIL.

Given the non-standard support for SMIL media support depends on the player implementation. The table below sum-up what can be seen and the corresponding tag:

Media	SMIL Tag	RealPlayer G2	GRiNS	Soja
GIF	img	OK	OK	OK
JPEG	img	OK	OK	OK
Wav	audio	OK	OK	-
.au Audio	audio	OK	OK	OK
.auz Audio Zipped	audio	-	-	OK
MP3	audio	OK	-	-
Plain text	text	OK	OK	OK
Real text	textstream	OK	-	-
Real movie	video	OK	-	-
AVI	video	OK	OK	-
MPEG	video	OK	OK	-
MOV	video	OK	-	-

4.5 `fitting` media to regions

You can set the `fit` attribute of the `<region>` tag to force media to be resized *etc.*

The following values are valid for `fit`:

- `fill` — make media grow and fill the area.
- `meet` — make media grow (without any distortion) until it meets the region frontier.
- `slice` — media grows (without distortion) and fill entirely its region.
- `scroll` — if media is bigger than its region area gets scrolled.
- `hidden` — don't show media

Obviously you set the value like this:

```
<region id="region_1"  .....
     fit="fill" />
```

4.5.1 Synchronisation

There are two basic ways in which we may want to play media:

- play several media one after the other,
- how to play several media in parallel.

In order to do this we need to add *synchronisation*:

- we will need to add time parameter to media elements,

Adding a duration of time to media — `dur`

To add a duration of time to a media element simply specify a `dur` attribute parameter in an appropriate media tag:

```
.....
<body>
  <img src="cardiff.gif" alt="The Cardiff icon"
        region="cardiff_icon" dur="6s" />
</body>
.....
```

Delaying Media — the `begin` tag

To specify a delay *i.e* when to `begin` set the `begin` attribute parameter in an appropriate media tag:

If you add begin=”2s” in the cardiff image tag, you will see that the Cardiff icon will appear 2 seconds after the document began and will remain during 6 other seconds. Have a look at the source:

```
.....
<body>
 <body>
  <img src="cardiff.gif" alt="The Cardiff icon"
        region="cardiff_icon" dur="6s" begin="2s" />
</body>
.....
```

Sequencing Media — the `seq` tag

Now that we have some basic control over individual media let’s see how we play them together.

The `<seq>` tag is used to define a sequence of media.

- The media are executed one after each other:

```
.....
<seq>
   <img src="img1.gif"
        region="reg1" dur="6s" />
   <img src="img2.gif"
        region="reg2" dur="4s" begin="1s" />
</seq>
.....
```

So the setting `1s` makes the `img2.gif` icon appear 1 second after `img1.gif`.

Parallel Media — the `par` tag

We use the `<par>` to play media at the same time:

```
<par>
   <img src="cardiff.gif" alt="The cardiff icon"
        region="cardiff_icon" dur="6s" />
   <audio src="music.au" alt="Some Music"
         dur="6s" />
</par>
```

This will display an image and play some music along with it.

Synchronisation Example 1: Planets Soundtrack

The following SMIL code plays on long soundtrack along with as series of images.
Essentially:

- The audio file and
- image sequences are played in `par`allel
- The Images are run in `sequence` with no break (`begin = 0s`)

The files are stored on the MACINTOSHES in the Multimedia Lab (in the SMIL folder) as follows:

- `planets.html` — call smil source (below) with the SOJA applet. This demo uses zipped (SUN) audio files (`.auz`) which are not supported by RealPlayer.
- `planets.smil` — smil source (listed below),

```
<smil>
 <head>
  <layout>
   <root-layout height="400" width="600" background-color="#000000" title="Dreaming out Loud"/>
   <region id="satfam" width="564" height="400" top="0" left="0" background-color="#000000" z-index="2" />
   <region id="jupfam" width="349" height="400" top="0" left="251" background-color="#000000" z-index="2" />
   <region id="redsun" width="400" height="400" top="0" left="100" background-color="#000000" z-index="2" />
   <region id="ngc3918" width="484" height="400" top="0" left="58" background-color="#000000" z-index="2" />
   <region id="lagoon1" width="394" height="396" top="2" left="103" background-color="#000000" z-index="2" />
   <region id="lagoon2" width="436" height="308" top="46" left="82" background-color="#000000" z-index="2" />
   <region id="m33" width="371" height="400" top="0" left="114" background-color="#000000" z-index="2" />
   <region id="orion" width="371" height="400" top="0" left="114" background-color="#000000" z-index="2" />
   <region id="hubble5" width="455" height="400" top="0" left="72" background-color="#000000" z-index="2" />

   <region id="pillars" width="409" height="400" top="0" left="0" background-color="#000000" z-index="2" />

   <region id="blank" width="191" height="400" top="0" left="409" background-color="#ffffff" z-index="2" />
   <region id="music" width="100" height="25" top="30" left="453" background-color="#ffffff" z-index="3" />
   <region id="dreamland" width="150" height="25" top="55" left="453" background-color="#ffffff" z-index="3"/
   <region id="by1" width="100" height="25" top="80" left="453" background-color="#ffffff" z-index="3" />
   <region id="don" width="100" height="25" top="105" left="453" background-color="#ffffff" z-index="3" />

   <region id="images" width="100" height="25" top="140" left="453" background-color="#ffffff" z-index="3" />
   <region id="nasa" width="100" height="25" top="165" left="453" background-color="#ffffff" z-index="3" />

   <region id="smil" width="100" height="25" top="200" left="453" background-color="#ffffff" z-index="3" />
   <region id="by2" width="100" height="25" top="225" left="453" background-color="#ffffff" z-index="3" />
   <region id="me" width="100" height="25" top="250" left="453" background-color="#ffffff" z-index="3" />
   <region id="jose" width="100" height="25" top="250" left="453" background-color="#ffffff" z-index="3" />

   <region id="title" width="125" height="25" top="40" left="237" background-color="#ffffff" z-index="2" />
   </layout>
  </head>

  <body>
   <par>
    <audio src="media/dreamworldb.auz" dur="61.90s" begin="3.00s" system-bitrate="14000" />
     <seq>
      <img src="media/satfam1a.jpg" region="satfam" begin="1.00s" dur="4.50s" />
      <img src="media/jupfam1a.jpg" region="jupfam" begin="1.50s" dur="4.50s" />
      <img src="media/redsun.jpg" region="redsun" begin="1.00s" dur="4.50s" />
      <img src="media/ngc3918a.jpg" region="ngc3918" begin="1.00s" dur="4.50s" />
      <img src="media/lagoon1c.jpg" region="lagoon1" begin="1.00s" dur="4.50s" />
      <img src="media/lagoon2b.jpg" region="lagoon2" begin="1.00s" dur="4.50s" />
      <img src="media/m33c.jpg" region="m33" begin="1.00s" dur="4.50s" />
      <img src="media/hubble5a.jpg" region="hubble5" begin="1.00s" dur="4.50s" />
      <img src="media/orion.jpg" region="orion" begin="1.00s" dur="4.50s" />
       <par>
        <img src="media/pillarsb.jpg" region="pillars" begin="1.00s" end="50s" />

<img src="media/blank.gif" region="blank" begin="2.00s" end="50.00s" />
        <text src="media/music.txt" region="music" begin="3.00s" end="50.00s" />
        <text src="media/dreamland.txt" region="dreamland" begin="4.00s" end="50.00s" />
        <text src="media/by.txt" region="by1" begin="7.00s" end="50.00s" />
        <text src="media/don.txt" region="don" begin="8.00s" end="50.00s" />

        <text src="media/images.txt" region="images" begin="14.00s" end="50.00s" />
        <text src="media/nasa.txt" region="nasa" begin="15.00s" end="50.00s" />

        <text src="media/smil.txt" region="smil" begin="18.00s" end="50.00s" />
        <text src="media/by.txt" region="by2" begin="19.00s" end="50.00s" />
        <text src="media/me.txt" region="me" begin="20.00s" dur="3.00s" />
        <text src="media/jose.txt" region="jose" begin="23.00s" end="50.00s" />
       </par>
      <text src="media/title.txt" region="title" begin="3.00s" end="25.00s" />
     </seq>
    </par>
   </body>
 </smil>
```

Synchronisation Example 2: Slides 'N' Sound

Dr John Rosbottom of Plymouth Univ has come up with a novel way of giving lectures.

This has

- one long sequence of
- parallel pairs of images and audio files

Try out the online version:
http://www.dis.port.ac.uk/ johnr/lal2/start.htm
You may examine the files on the Macintoshes in the Multimedia lab (in SMIL folder)

- `slides_n_sound.smil` — smil source (listed below), play with RealPlayer G2. **NOTE:** This demo uses real audio files which are not supported by SOJA:

```
<smil>
<head>
<layout>
<root-layout height="400" width="600" background-color="#000000" title="Slides and Sound"/>
</layout>
</head>
  <body>



  <seq>
    <par>
     <audio src="audio/leconlec.rm" dur="24s" title="slide 1"/>
     <img src="slides/img001.GIF" dur="24s"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="24s" clip-end="51s" dur="27s" title="slide 2"/>
<img src="slides/img002.GIF" dur="27s"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="51s" clip-end="67s" dur="16s" title="slide 3"/>
<img src="slides/img003.GIF" dur="16s"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="67s" clip-end="116s" dur="49s" title="slide 4"/>
<img src="slides/img004.GIF" dur="49s"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="116s" clip-end="186s" dur="70s" title="slide 5"/>
<img src="slides/img005.GIF" clip-begin="116s" clip-end="186s" dur="70s" title="slide 5"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="186s" clip-end="290s" dur="104s" title="slide 6"/>
<img src="slides/img006.GIF" clip-begin="186s" clip-end="290s" dur="104s" title="slide 6"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="290s" clip-end="357s" dur="67s" title="The Second Reason"/>
<img src="slides/img007.GIF" clip-begin="290s" clip-end="357s" dur="67s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="357s" clip-end="373s" dur="16s" title="The Second Reason"/>
<img src="slides/img008.GIF" clip-begin="357s" clip-end="373s" dur="16s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="373s" clip-end="466s" dur="93s" title="The Second Reason"/>
<img src="slides/img009.GIF" clip-begin="373s" clip-end="466s" dur="93s" title="The Second Reason"/>
    </par>

    <par>
```

```
       <audio src="audio/leconlec.rm" clip-begin="466s" clip-end="497s" dur="31s" title="The Second Reason"/>
<img src="slides/img010.GIF" clip-begin="466s" clip-end="497s" dur="31s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="497s" clip-end="518s" dur="21s" title="The Second Reason"/>
<img src="slides/img011.GIF" clip-begin="497s" clip-end="518s" dur="21s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="518s" clip-end="536s" dur="18s" title="The Second Reason"/>
<img src="slides/img012.GIF" clip-begin="518s" clip-end="536s" dur="18s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="536s" clip-end="553s" dur="17s" title="The Second Reason"/>
<img src="slides/img013.GIF" clip-begin="536s" clip-end="553s" dur="17s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="553s" clip-end="562s" dur="9s" title="The Second Reason"/>
<img src="slides/img014.GIF" clip-begin="553s" clip-end="562s" dur="9s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="562s" clip-end="568s" dur="6s" title="The Second Reason"/>
<img src="slides/img015.GIF" clip-begin="562s" clip-end="568s" dur="6s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="568s" clip-end="578s" dur="10s" title="The Second Reason"/>
<img src="slides/img016.GIF" clip-begin="568s" clip-end="578s" dur="10s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="578s" clip-end="610s" dur="32s" title="The Second Reason"/>
<img src="slides/img017.GIF" clip-begin="578s" clip-end="610s" dur="32s" title="The Second Reason"/>
    </par>

    <par>
       <audio src="audio/leconlec.rm" clip-begin="610s" clip-end="634s" dur="24s" title="The Second Reason"/>
<img src="slides/img018.GIF" clip-begin="610s" clip-end="634s" dur="24s" title="The Second Reason"/>
    </par>

   <par>
       <audio src="audio/leconlec.rm" clip-begin="634s" clip-end="673s" dur="39s" title="Slide 19"/>
<img src="slides/img019.GIF" clip-begin="634s" clip-end="673s" dur="39s" title="Slide 19"/>
    </par>

    <img src="slides/img006.GIF" fill="freeze" title="And finally..."
      author="Abbas Mavani (dis80047@port.ac.uk)"
     copyright="Everything is so copyright protected (c)1999"/>

    <!-- kept this in to remind me that you can have single things
    <audio src="audio/AbbasTest.rm" dur="50.5s"/>
    -->
  </seq>
  </body>
</smil>
```

4.5.2 SMIL Events

Smiles supports event based synchronisation:

`begin` events

When a media begins, it sends a `begin` event. If another media waits for this event, it catches it. To make a media wait to an event, one of its synchronisation attributes (begin or end) should be written as follows:

```
<!-- if you want tag to start when
     another tag begins -->
<tag begin="id(specifiedId)(begin)" />

<!-- if you want tag to start 3s after
     another tag begins -->
<tag begin="id(specifiedId)(3s)" />

<!-- if you want tag to start when
     another tag ends -->
<tag begin="id(specifiedId)(end)" />
```

For example:

```
 <body>
  <par>

   <img src="cardiff.gif" region="cardiff"
id="cf" begin="4s" />

   <img src="next.gif" region="next"
        begin="id(cf)(2s)" />

  </par>
 </body>
```

will make the `next.gif` image begin 2s after `cardiff.gif` begins.

The `switch` Tag

The syntax for the `switch` tag is:

```
<switch>
 <!-- child1 testAttributes1 -->
 <!-- child2 testAttributes2 -->
 <!-- child3 testAttributes3 -->
</switch>
```

The rule is:

- The first of the `<switch>` tag children whose test attributes are all evaluated to TRUE is executed.
- A tag with no test attributes is evaluated to TRUE.
- See SMIL reference for list of valid test attributes

For example you may wish to provide presentations in english or welsh:

```
 <body>
  <switch>

   <!-- English only -->
   <par system-language="en">
       <img src="cardiff.gif" region="cardiff"/>
       < audio src ="english.au" />
   </par>

  <!-- Welsh only -->
   <par system-language="cy">
       <img src="caerdydd.gif" region="cardiff"/>
       <audio src ="cymraeg.au" />
   </par>
```

somewhere in code you will (or it will be set) set the `system-language`

4.6 SMIL ON

Please refer to online sources and the SMIL reference handout for further SMIL features.

Some good online SMIL resources include:

- *http://www.w3.org/AudioVideo/* — Synchronized Multimedia
- *http://www.real.com/devzone/library/creating/* — Content Creation and Programmer's Guide
- *http://www.cwi.nl/SMIL/* — The CWI SMIL Page
- *http://indy.cs.concordia.ca/smil/* — SMIL Web Pages

- *http://www.ruleweb.com/dhtml/smil.html* — Jeff Rule's Dynamic HTML and SMIL Site
- *http://dejavu.cs.vu.nl/ symm/validator/* — SMIL syntax validator
- *http://www.w3.org/TR/REC-smil/* — Synchronized Multimedia Integration Language
- *http://www.builder.com/Authoring/Standards/ss01.html*— CNET Builder.com - Web Authoring - Emerging Web standards - Synchronized Multimedia Integration Language (SMIL): multimedia made easy
- *http://www.helio.org/products/smil/tutorial/* — SMIL Tutorial
- *http://smw.internet.com/smil/news/* — Just SMIL: News

Part III

Multimedia Technology

Chapter 5

Multimedia Systems Technology

5.1 Discrete v continuous media

Multimedia systems deal with the generation, manipulation, storage, presentation, and communication of information in digital form.

The data may be in a variety of formats: text, graphics, images, audio, video.

A majority of this data is large and the different media may need synchronisation — the data may have temporal relationships as an integral property.

Some media is time independent or *static* or *discrete* media: normal data, text, single images, graphics are examples.

Video, animation and audio are examples of *continuous* media.

5.2 Analog and Digital Signals

We will discuss the mechanism and issues involved in transforming signals from analog-digital in Chapter 6. Here we will introduce some basic definitions before overviewing the technology required to perform such tasks.

The world we sense is full of analog signal, electrical sensors such as transducers, thermocouples, microphones convert the medium they sense into electrical signals. These are usually continuous and still analog. These analog signals must be converted or *digitised* into discrete digital signals that computer can readily deal with.

Special hardware devices called *Analog-to-Digital* converters perform this task.

For playback *Digital-to-Analog* must perform a converse operation.

Note that Text, Graphics and some images are generated directly by computer and *do not* require digitising: they are generated directly in binary format.

Handwritten text would have to digitised either by electronic pen sensing of scanning of paper based form.

5.3 Input Devices and Storage

Let us now consider each media in turn and summarise how it may be input into a Multimedia system. We also briefly analyse the basic storage requirements for each type of data. We do not yet consider any effect of compression on the files. Note that storage requirements a large for many forms of media.

5.3.1 Text and Static Data

The sources of this media are the keyboard, floppies, disks and tapes. Text files are usually stored and input character by character. Files may contain raw text or formatted text *e.g* HTML, Rich Text Format (RTF) or a program language source (C, Pascal, *etc.*.

Even though to data medium does not include any temporal constraints there may be an natural implied sequence *e.g.* HTML format sequence, Sequence of C program statements.

The basic storage of text is 1 byte per character (text or format character). For other forms of data *e.g.* Spreadsheet files some formats may store format as text (with formatting) others may use binary encoding.

Even the the storage requirements of this data is never high when data is stored on disk small files may take larger disk storage requirements due to block and sector sizes of disk partitions.

5.3.2 Graphics

Graphics are usually constructed by the composition of primitive objects such as lines, polygons, circles, curves and arcs. Graphics are usually generated by a graphics editor program (*e.g.* Freehand) or automatically by a program (*e.g.* Postscript usually generated this way). Graphics are usually editable or revisable (unlike Images).

Graphics input devices include: keyboard (for text and cursor control), mouse, trackball or graphics tablet.

Graphics files may adhere to a graphics standard (OpenGL, PHIGS, GKS) Text may need to stored also. Graphics files usually store the primitive assembly and do not take up a very high overhead.

5.3.3 Images

Images are still pictures which (uncompressed) are represented as a bitmap (a grid of pixels).

Images may be generated by programs similar to graphics or animation programs. But images may be scanned for photographs or pictures using a digital scanner or from a digital camera. Some Video cameras allow for still image capture also. Analog sources will require digitising.

Images may be stored at 1 bit per pixel (Black and White), 8 Bits per pixel (Grey Scale, Colour Map) or 24 Bits per pixel (True Colour) (See Chapter 6).

Thus a 512x512 Grey scale image takes up 1/4 Mb, a 512x512 24 bit image takes 3/4 Mb with no compression. This overhead soon increases with image size so compression is commonly applied (See Chapter 7)

5.3.4 Audio

Audio signals are continuous analog signals. They are first captured by a microphones and then digitised and store — usually compressed as CD quality audio requires 16-bit sampling at 44.1 KHz (There are other audio sampling rates — Chapter 6). So 1 Minute of Mono CD quality audio requires $60 * 44100 * 2$ Bytes which is approximately 5 Mb.

5.3.5 Video

Analog Video is usually captured by a video camera and then digitised. There are a variety of video (analog and digital) formats (Chapter 6)

Raw video can be regarded as being a series of single images. There are typically 25, 30 or 50 frames per second. Therefore a 512x512 size monochrome video images take 25*0.25 = 6.25Mb for a minute to store uncompressed. Digital video clearly needs to be compressed.

5.4 Output Devices

The output devices for a basic multimedia system include

- A High Resolution Colour Monitor
- CD Quality Audio Output
- Colour Printer
- Video Output to save Multimedia presentations to (Analog) Video Tape, CD-ROM DVD.
- Audio Recorder (DAT, DVD, CD-ROM, (Analog) Cassette)
- Storage Medium (Hard Disk, Removable Drives, CD-ROM) — see next section.

5.5 Storage Media

Let us first recap the major problems that affect storage media:

- Large volume of date
- Real time delivery
- Data format
- Storage Medium
- Retrieval mechanisms

First two factors are the real issues that storage media have to deal and we have discussed these factors already. Due to the volume of data the Data format will include compression (see Chapters 6 and 7).

The type of storage medium and underlying retrieval mechanism will affect how the media is stored and delivered. Ultimately any system will have to deliver high performance I/O. We discuss this issue next before going on to discuss actual Multimedia storage devices.

5.5.1 High performance I/O

There are four factors that influence I/O performance:

Data — Data is high volume, maybe continuous and may require contiguous storage. Direct relationship between size of data and how long it takes to handle. Compression and also distributed storage (See RAID technology (Section 5.5.3 below).

Data Storage — The strategy for data storage depends of the storage hardware and the nature of the data. The following storage parameters affect how data is stored:

- Storage Capacity
- Read and Write Operations of hardware
- Unit of transfer of Read and Write
- Physical organisation of storage units
- Read/Write heads, Cylinders per disk, Tracks per cylinder, Sectors per Track
- Read time
- Seek time

Data Transfer — Depend how data generated and written to disk, and in what sequence it needs to retrieved. Writing/Generation of Multimedia data is usually sequential *e.g.* streaming digital audio/video direct to disk. Individual data (*e.g.* audio/video file) usually streamed.

RAID architecture can be employed to accomplish high I/O rates by exploiting parallel disk access (Section 5.5.3)

Operating System Support — Scheduling of processes when I/O is initiated. Time critical operations can adopt special procedures. Direct disk transfer operations free up CPU/Operating system space.

5.5.2 Basic Storage

Basic storage units have problems dealing with large multimedia data

- Single Hard Drives — SCSI/IDE Drives. So called *AV* (Audio-Visual) drives, which avoid thermal recalibration between read/writes, are suitable for desktop multimedia. New drives are fast enough for direct to disk audio and video capture. But not adequate for commercial/professional Multimedia. Employed in RAID architectures (Section 5.5.3)
- Removable Media — Jaz/Zip Drives, CD-ROM, DVD. Conventional (dying out?) floppies not adequate due 1.4 Mb capacity. Other media usually ok for backup but usually suffer from worse performance than single hard drives.

5.5.3 RAID — Redundant Array of Inexpensive Disks

This concept of RAID has been developed to fulfill the needs of current multimedia and other data hungry application programs, and which require fault tolerance to be built into the storage device. Further, techniques of parallel processing are also suitable to exploiting the benefits of such an arrangement of hard disks.

Raid technology offers some significant advantages as a storage medium:

- Affordable alternative to mass storage
- High throughput and reliability

The cost per megabyte of a disk has been constantly dropping, with smaller drives playing a larger role in this improvement. Although larger disks can store more data, it is generally more power effective to use small diameter disks (as less power consumption is needed to spin the smaller disks). Also, as smaller drives have fewer cylinders, seek distances are correspondingly lower. Following this general trend, a new candidate for mass storage has appeared on the market, based on the same technology as magnetic disks, but with a new organisation. These are arrays of small and inexpensive disks placed together, based on the idea that disk throughput can be increased by having many disk drives with many heads operating in parallel. The distribution of data over multiple disks automatically forces access to several disks at one time improving throughput. Disk arrays are therefore obtained by placing small disks together to obtain the performance of more expensive high end disks.

The key components of a RAID System are:

- Set of disk drives, disk arrays, viewed by user as one or more logical drives.

- Data may be distributed across drives
- Redundancy added in order to allow for disk failure

Disk arrays can be used to store large amounts of data, have high I/O rates and take less power per megabyte (when compared to high end disks) due to their size, but they have very poor reliability[1].

What do you think is the reason of this low reliability?

As more devices are added, reliability deteriorates (N devices generally have $\frac{1}{N}$ the reliability of a single device).

Files stored on arrays may be *striped* across multiple spindles. Since a high capacity is available due to the availability of more disks, it is possible to create redundancy within the system, so that if a disk fails the contents of a file may be reconstructed from the redundant information. This off course leads to a penalty in capacity (when storing redundant information) and in bandwidth (to update the disk). Four main techniques are available to overcome the lack of reliability of arrays:

- *Mirroring* or shadowing of the contents of disk, which can be a *capacity kill* approach to the problem. Each disk within the array is mirrored and a write operation performs a write on two disks - resulting in a 100% capacity overhead. Reads to disks may however be optimised. This solution is aimed at *high bandwidth, high availability environments.*
- *Horizontal Hamming Codes*: A special means to reconstruct information using an error correction encoding technique. This may be an overkill, as it is complex to compute over a number of disks.
- *Parity and Reed-Soloman Codes*: Also an error correction coding mechanism. Parity may be computed in a number of ways, either horizontally across disks or through the use of an interleaved parity block. Parity information also has to be stored on disk, leading to a 33% capacity cost for parity. Use of wider arrays reduces the capacity cost, but leads to a decrease

  in the expected availability and increased reconstruction times. This approach is generally aimed at high bandwidth scientific applications (such as image processing).
- *Failure Prediction*: There is no capacity overhead in this technique, though it is controversial in nature, as its use cannot be justified if all errors or failures can be forecast correctly.

[1]Reliability here is defined in terms of availability of a disk, and corruption of data during transfer from the disk to CPU.

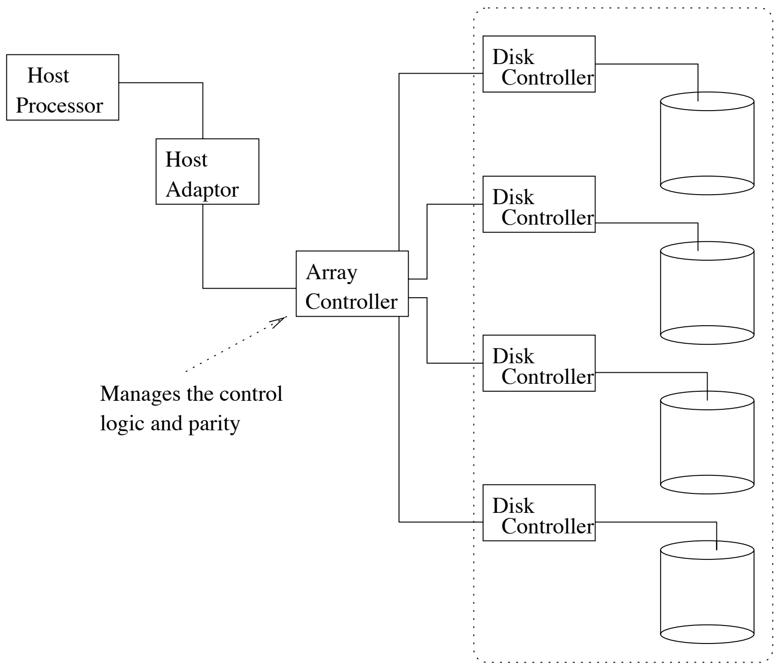


Figure 5.1: A disk array link to the host processor

Each disk within the array needs to have its own I/O controller, but interaction with a host computer may be mediated through an *array controller* as shown in figure 5.1.

It may also be possible to combine the disks together to produce a collection of devices, where each vertical array is now the unit of data redundancy. Such an arrangement is called an *orthogonal RAID* and shown in figure 5.2; other arrangements of disks are also possible.

Figure 5.3 identifies disk performance for a number of machines and operating systems. The Convex supercomputer seems to provide the best performance in terms of throughput (or megabytes transferred per second) based on a 32KB read operation due to the use of a RAID disk. The figure also illustrates transfer rate dependencies between hardware and the particular operating system being used.

There are now 8 levels of RAID technology, with each level providing a greater amount of resilience then the lower levels:

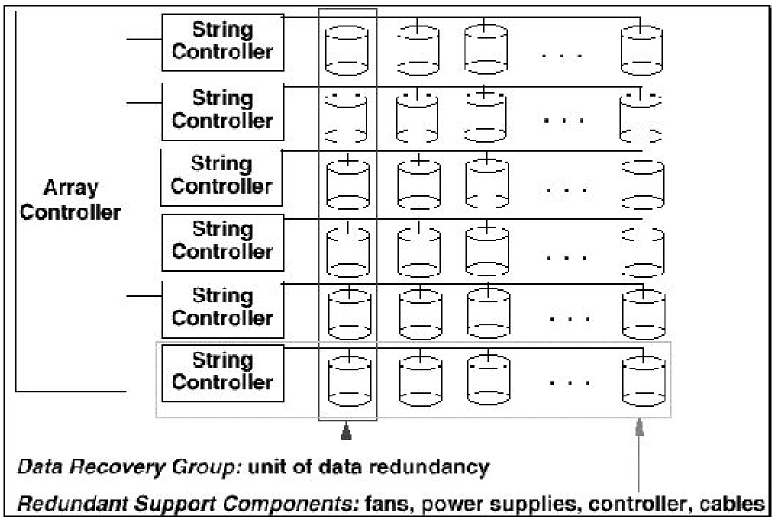


Figure 5.2: Orthogonal RAID

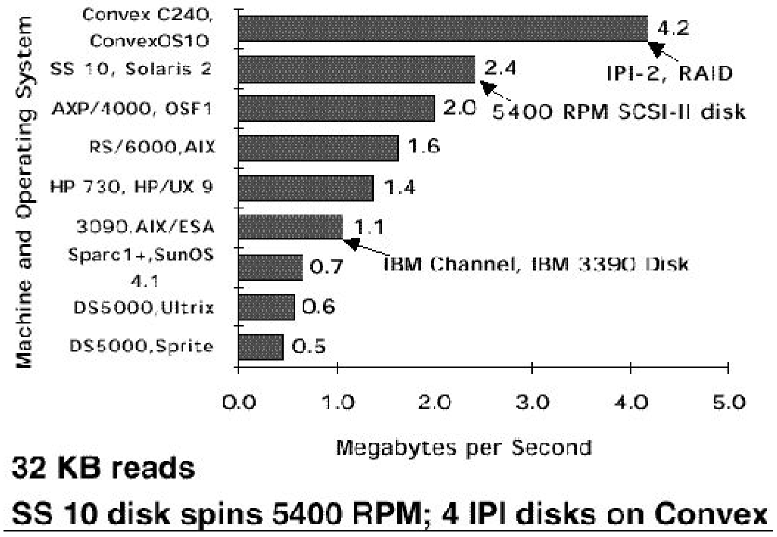


Figure 5.3: *Comparison of disk performance*

Level 0: Disk Striping — distributing data across multiple drives. Level 0 is an independent array without parity redundancy that accesses data across all drives in the array in a block format. To accomplish this, the first data block is read/written from/to the first disk in the array, the second block from/to the second disk, and so on. RAID 0 only addresses improved data throughput, disk capacity and disk performance. In RAID 0 data is striped across the various drives present. Although striping can improve performance on request rates, it does not provide fault tolerance. If a single disk fails, the entire system fails. This would therefore be equivalent to storing the complete set of data on a single drive, though with a lower access rate.

Level 1: Disk Mirroring — Level 1 focusses on *fault tolerancing* and involves a second duplicate write to a mirror

disk each time a write request is made. The write is performed automatically and is transparent to the user, application and system. The mirror disk contains an exact replica of the data on the *actual disk*. Data is recoverable if a drive fails and may be recoverable if both drives fail. The biggest disadvantage is that only half of the disk capacity is available for storage. Also, capacity can only be expanded in pairs of drives.

Of the RAID levels, level 1 provides the highest data availability since two complete copies of all information are maintained. In addition, read performance may be enhanced if the array controller allows simultaneous reads from both members of a mirrored pair. During writes, there will be a minor performance penalty when compared to writing to a single disk. Higher availability will be achieved if both disks in a mirror pair are on separate I/O busses.

Level 2: Bit Interleaving and HEC Parity — Level 2 stripes data to a group of disks using a bite stripe. A Hamming code symbol for each data stripe is stored on the check disk. This code can correct as well as detect data errors and permits data recovery without complete duplication of data. This RAID level is also sometimes referred to as Level 0+1. It combines the benefits of both striping and Level 1 - with both high availability and high performance. It can be tuned for either a request rate intensive or transfer rate intensive environment. Level 2 arrays *sector-stripe* data across groups of drives, with some drives being dedicated to storing Error Checking and Correction (ECC) information within each sector. However, since most disk drives today embed ECC information within each sector as standard, Level 2 offers no significant advantages over Level 3 architecture. At the present time there are no manufacturers of Level 2 arrays.

Level 3: Bit Interleaving with XOR Parity — Level 3 is a striped parallel array where data is distributed by bit or byte. One drive in the array provides data protection by storing a parity check byte for each data stripe.

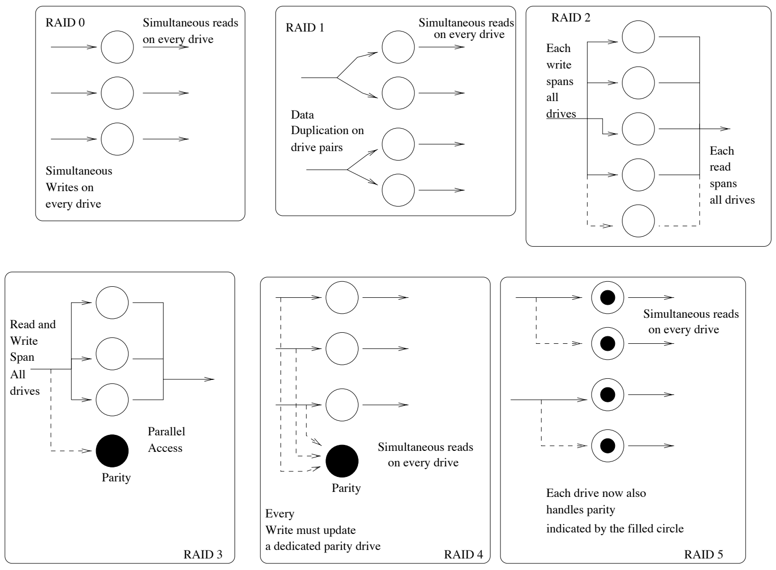

RAID 0
Simultaneous reads on every drive
Simultaneous Writes on every drive
RAID 1
Simultaneous reads on every drive
Data Duplication on drive pairs
RAID 2
Each write spans all drives
Each read spans all drives
Read and Write Span All drives
Parallel Access
Parity
RAID 3
Simultaneous reads on every drive
Parity
Every Write must update a dedicated parity drive
RAID 4
Simultaneous reads on every drive
Each drive now also handles parity indicated by the filled circle
RAID 5

Figure 5.4: *RAID levels*

Level 3 has the advantage over lower RAID levels in that the ratio of check disk capacity to data disk capacity decreases as the number of data drives increases. It has parallel data paths and therefore offers high transfer rate performance for applications that transfer large files. Array capacity can be expanded in single drive or group increments. With Level 3, data chunks are much smaller than the average I/O size and the disk spindles are synchronised to enhance throughput in transfer rate intensive environments. Level 3 is well suited for CAD/CAM or imaging type applications.

Level 4: Block Interleaving with XOR Parity — In Level 4 parity is interleaved at the sector or transfer level. As with Level 3, a single drive is used to store redundant data using a parity check byte for each data stripe.

Level 4 offers high read performance and good write performance. Level 4 is a general solution, especially where the ratio of reads to writes is high. This makes Level 4 a good choice for small block transfers, which are typical for transaction processing applications. Write performance is low because the parity drive has to be written for each data write. Thus the parity drive becomes a performance bottleneck when multiple parity write I/Os are required. In this instance, Level 5 is a better solution because parity information is spiralled across all available disk drives. Level 4 systems are almost never implemented mainly because it offers no significant advantages over Level 5.

Level 5: Block Interleaving with Parity Distribution — Level 5 combines the throughput of block interleaved data

striping of Level 0 with the parity reconstruction mechanism of Level 3 without requiring an extra parity drive. In Level 5, both parity and data are striped across a set of disks. Data chunks are much larger than the average I/O size. Disks are able to satisfy requests independently which provides high read performance in a request-rate intensive environment. Since parity information is used, a Level 5 stripe can withstand a single disk failure without losing data or access to data.

Level 5's strength lies in handling large numbers of small files. It allows improved I/O transfer performance because the parity drive bottleneck of Level 4 is eliminated. While Level 5 is more cost effective because a separate parity drive is not used, write performance suffers. In graphic art and imaging applications, the weakness of Level 5 versus Level 3 is the write penalty from the striped parity information. In Level 3 there is no write penalty. Level 5 is usually seen in applications with large numbers of small read/write calls. Level 5 offers higher capacity utilisation when the array has less than 7 drives. With a full array, utilisation is about equal between Level 3 and 5.

Level 6: Fault Tolerant System — additional error recovery. This is an improvement of Level 5. Disks are considered to be in a matrix formation and parity is generated for each row and column of the matrix. Multidimensional parity is then computed and distributed among the disks in the matrix.

Level 6 became a common feature in many systems but the advent of Level 7 has led to the abandonment of Level 6 in many cases.

Level 7: Heterogeneous System — Fast access across whole system. Level 7 allows each individual drive to access data as fast as possible by incorporating a few crucial features:

- Each I/O drive is connected to high speed data bus which posses a central cache store capable of supporting multiple host I/O paths.
- A real time process-oriented OS is embedded into the disk array architecture — frees up drives, allowing independent drive head operation. Substantial improvement.
- All parity checking and check/microprocessor/bus/cache control logic embedded in this OS.
- OS designed to support multiple host interfaces — other RAID levels support only one.
- Additional ability to reconstruct data in the event of dependent drive failure increased due to separate cache/device control, and secondary, tertiary and beyond parity calculation — up to four simultaneous disk failures supported.

- *Dynamic Mapping* used. In conventional storage a block of data, once created, is written to fixed memory location. All operations then rewrite data back to this location. In Dynamic Memory this constraint is freed and new write locations logged and mapped. This frees additional disk accesses and the potential for a bottleneck.

These First 6 RAID levels are illustrated in Figure 5.4, where each circle represents a single disk drive, and arrows represent data flows.

5.5.4 Optical Storage

Optical storage has been the most popular storage medium in the multimedia context due its compact size, high density recording, easy handling and low cost per MB.

CD is the most common and we discuss this below. Laser disc and recently DVD are also popular.

5.5.5 CD Storage

There a now various formats of CD:

In the beginning, there was CD-DA (Compact Disc-Digital Audio), or standard music CDs. CD-DA moved onto CD-ROM when people realized that you could store a whole bunch of computer data on a 12cm optical disk (650mb). CD-ROM drives are simply another kind of digital storage media for computers, albeit read-only. They are peripherals just like hard disks and floppy drives. (Incidentally, the convention is that when referring to magnetic media, it is spelled *disc*. Optical media like CDs, LaserDisc, and all the other formats I'm about to explain are spelled disc.)

CD-I (Compact Disc-Interactive) came next. This is a consumer electronics format that uses the optical disk in combination with a computer to provide a home entertainment system that delivers music, graphics, text, animation, and video in the living room. Unlike a CD-ROM drive, a CD-I player is a standalone system that requires no external computer. It plugs directly into a TV and stereo system and comes with a remote control to allow the user to interact with software programs sold on disks. It looks and feels much like a CD player except that you get images as well as music out of it and you can actively control what happens. In fact, it *is* a CD-DA player and all of your standard music CDs will play on a CD-I player; there is just no video in that case.

Next came CD-ROM/XA (eXtended Architecture). Now we go back to computer peripherals - a CD-ROM drive but with some of the compressed audio capabilities found in a CD-I player (called ADPCM). This allows interleaving of audio and other data so that an XA drive can play audio and display pictures (or other things) simultaneously. There is special hardware in an XA drive controller to handle the audio playback. This format came from a desire to inject some of the features of CD-I back into the professional market.

Now, along comes the idea from Kodak for Photo CD - digital pictures on compact disk. They teamed up with Philips to develop the standard for Photo CD disks. At this point, a new problem enters the picture, if you'll pardon the expression. All of the disk formats mentioned so far are read-only; there is no way for anyone but the producer of one of these disks to store his/her own content on the disk - that is, to write to it. But there already existed a technology called WORM (Write Once Read Many). This is an optical disk that can be written to, but exactly once. You can *burn* data on it, but once burned the data can not be erased, although it can then be used like a CD-ROM disk and read forever. (Depending on your definition of forever, of course.)

CD-ROM, CD-ROM/XA, and CD-I disks are normally *mastered*, as opposed to burned. That means that one master copy is made and then hundreds, or thousands, or millions (if you're lucky enough to need that many) of replicates are pressed from the master. This process is much cheaper than burning for quantities above a few dozen or so. Generally, disk pressing plants can handle all of these formats as the underlying technology is the same; the only difference is in the data and disk format.

The reason that WORM technology was critical for Photo CD is obvious - the content of these disks is not determined by the manufacturer or publisher. For Photo CD, each disk will be different - a roll or few rolls of film per disk from a customer.

Kodak and Philips wanted Photo CD disks to be playable on both computer peripherals for desktop publishing uses AND on a consumer device for home viewing. For the former, CD-ROM/XA was chosen as a carrier and for the latter CD-I, which was already designed as a consumer electronics device, and dedicated Photo CD players. This desire for a hybrid disk, or one with multi-platform compatibility, led to the development of the CD-I Bridge disk format. A Bridge disk is one that is readable on both a CD-I player and a CD-ROM/XA drive.

This Bridge format is the reason there is so much confusion about CD-ROM drives for Photo CD. A drive that supports Photo CD must be a CD-ROM/XA drive that is also Bridge-compatible. (The technical description of Bridge disks calls for supporting certain kinds of sectors identified by *form* and *mode* bits, which is what you usually hear instead of the *Bridge* disk label.) That almost completes the picture, except for the concept of sessions.

Although a WORM disk can only be written to once, it is not necessary to write, or burn, the entire disk all at once. You can burn the disk initially with, say, a few hundred megabytes of data, and then go back later and burn some more data onto it. Of course, each burn must be to a virgin part of the disk; once a spot on the disk is burned, it can not be re-burned. Each burn operation is referred to as a *session*, and a drive or disk that supports this multiple burning operation is called *multisession*.

Originally, all WORMs were single session only. That is, you could not go back and add data to a WORM disk once it was burned, even if it was not full. For Photo CD, they wanted the consumer to be able to add more pictures to an existing disk as additional rolls of film were processed. So the extension of

WORM technology to multisession was developed and adopted for the Bridge disk format. This required hardware changes to CD-ROM/XA drives and that is why there are a fair number of single session XA drives on the market and multisession ones appearing more and more.

A single session drive can read a multisession disk, but it can only read the contents of the first session that was burned. Incidentally, all Philips CD-I players are multisession, although all current CD-I disks have only a single session on them.

The capacity of a CD-ROM is 620-700 Mbs depending on CD material, Drives that read and write the CD-ROMS. 650 Mb (74 Mins) is a typical write once CD-ROM size.

CD Standards

There are several CD standard for different types of media:

Red Book — Digital Audio: Most Music CDs.

Yellow Book — CD-ROM: Model 1 – computer data, Model 2 – compress audio/video data.

Green Book — CD-I

Orange Book — write once CDs

Blue Book — LaserDisc

5.5.6 DVD

DVD, which stands for *Digital Video Disc*, *Digital Versatile Disc*, or nothing, depending on whom you ask, is the next generation of optical disc storage technology.

DVD has become a major new medium for a whole host of multimedia system:

It's essentially a bigger, faster CD that can hold video as well as audio and computer data. DVD aims to encompass home entertainment, computers, and business information with a single digital format, eventually replacing audio CD, videotape, laserdisc, CD-ROM, and perhaps even video game cartridges. DVD has widespread support from all major electronics companies, all major computer hardware companies, and most major movie and music studios, which is unprecedented and says much for its chances of success (or, pessimistically, the likelihood of it being forced down our throats).

It's important to understand the difference between DVD-Video and DVD-ROM. DVD-Video (often simply called DVD) holds video programs and is played in a DVD player hooked up to a TV. DVD-ROM holds computer data and is read by a DVD-ROM drive hooked up to a computer. The difference is similar to that between Audio CD and CD-ROM. DVD-ROM also includes future variations that are recordable one time (DVD-R) or many times (DVD-RAM). Most people expect DVD-ROM to be initially much more successful than DVD-Video. Most new computers with DVD-ROM drives will also be able to play DVD-Videos.

There's also a DVD-Audio format. The technical specifications for DVD-Audio are not yet determined.

What are the features of DVD-Video?

The main features of DVD include:

- Over 2 hours of high-quality digital video (over 8 on a double-sided, dual-layer disc).
- Support for widescreen movies on standard or widescreen TVs (4:3 and 16:9 aspect ratios).
- Up to 8 tracks of digital audio (for multiple languages), each with as many as 8 channels.
- Up to 32 subtitle/karaoke tracks.
- Automatic *seamless* branching of video (for multiple story lines or ratings on one disc).
- Up to 9 camera angles (different viewpoints can be selected during playback).
- Menus and simple interactive features (for games, quizzes, etc.).
- Multilingual identifying text for title name, album name, song name, cast, crew, etc.
- *Instant* rewind and fast forward, including search to title, chapter, track, and timecode.
- Durability (no wear from playing, only from physical damage).
- Not susceptible to magnetic fields. Resistant to heat.
- Compact size (easy to handle and store, players can be portable, replication is cheaper).

Note: Most discs do not contain all features (multiple audio/subtitle tracks, seamless branching, parental control, etc.). Some discs may not allow searching or skipping.

Most players support a standard set of features:

- Language choice (for automatic selection of video scenes, audio tracks, subtitle tracks, and menus) which must be supported by additional content on the disc. Some players include additional features:

- Special effects playback: freeze, step, slow, fast, and scan (no reverse play or reverse step).
- Parental lock (for denying playback of discs or scenes with objectionable material) which again must be supported by additional content on the disc. Some players include additional features
- Programmability (playback of selected sections in a desired sequence).
- Random play and repeat play.
- Digital audio output (PCM stereo and Dolby-Digital).
- Compatibility with audio CDs.
- Component (YUV or RGB) output for highest-quality picture.
- Compatibility with Video CDs.
- Six-channel analog output from internal audio decoder.
- Compatibility with laserdiscs and CDVs.
- Reverse single frame stepping.
- RF output (for TVs with no direct video input).
- Multilingual on-screen display.

Quality of DVD-Video

DVD has the capability to produce near-studio-quality video and better-than-CD-quality audio. DVD is vastly superior to videotape and generally better than laserdisc. However, quality depends on many production factors. Until compression experience and technology improves we will occasionally see DVDs that are inferior to laserdiscs. Also, since large amounts of video have already been encoded for Video CD using MPEG-1, a few low-budget DVDs will use that format (which is no better than VHS) instead of higher-quality MPEG-2 (See Chapters 6 and 7)

DVD video is compressed from digital studio master tapes to MPEG-2 format. This is a *lossy* compression (see Chapter 7) which removes redundant information (such as sections of the picture that don't change) and information that's not readily perceptible by the human eye. The resulting video, especially when it is complex or changing quickly, may sometimes contain *artifacts* such as blockiness, fuzziness, and video noise depending on the processing quality and amount of compression (Chapter 7). At average rates of 3.5 Mbps (million bits/second), artifacts may be occasionally noticeable. Higher data rates result in higher quality, with almost no perceptible difference from the original master at rates above 6 Mbps. As MPEG compression technology improves, better quality is being achieved at lower rates.

Some DVD demos have visible artifacts such as color banding, blurriness, shimmering, missing detail, and even effects such as a face which *floats* behind the rest of the moving picture. This is sometimes caused by poor MPEG encoding, but is just as often caused by a poorly adjusted TV or by sloppy digital noise reduction prior to encoding. The Free Willy and Twister excerpts on the Panasonic demo disc are good examples of this. In any case, bad demos are not an indication that DVD quality is bad, since other demos show no artifacts or other problems. Bad demos are simply an indication of how bad DVD can be if not properly processed and correctly reproduced. Early demos were shown on prototype players based on prerelease hardware and firmware. Many demo discs were rushed through the encoding process in order to be distributed as quickly as possible. Contrary to popular opinion, and as stupid as it may seem, these demos are not carefully *tweaked* to show DVD at its best. Also, most salespeople are incapable of properly adjusting a television set. Most TVs have the sharpness set too high for the clarity of DVD. This exaggerates high-frequency video and causes distortion, just as the treble control set too high for a CD causes it to sound harsh. DVD video has exceptional color fidelity, so muddy or washed-out colors are almost always a problem in the display, not in the DVD player or disc.

DVD audio quality is excellent. One of DVD's audio formats is LPCM (linear pulse code modulation) with sampling sizes and rates higher than audio CD. Alternately, audio for most movies is stored as discrete multi-channel surround sound using Dolby Digital or MPEG-2 audio compression similar to the surround sound formats used in theaters. As with video, audio quality depends on how well the encoding was done. Most audio on DVD will be in Dolby Digital format, which is close to CD quality.

The final assessment of DVD quality is in the hands of consumers. Most initial reports consistently rate it better than laserdisc. No one can guarantee the quality of DVD, just as no one should dismiss it based on demos or hearsay. In the end it's a matter of individual perception.

What are the disadvantages of DVD?

Despite several positive attributes mentioned above there are some potential disadvantages of DVD:

- It will take years for movies and software to become widely available.
- It can't record (yet).
- It has built-in copy protection and regional lockout.
- It uses digital compression. Poorly compressed audio or video may be blocky, fuzzy, harsh,
- or vague.
- The audio downmix process for stereo/Dolby Surround can reduce dynamic range.

- It doesn't fully support HDTV.
- Some DVD players and drives may not be able to read CD-Rs.
- First-generation DVD players and drives can't read DVD-RAM discs.
- Current players can't play in reverse at normal speed.

Compatibility of DVD

DVD is compatible with most other optical media storage (but there is a distinction between DVD and DVD-ROM below:

- CD audio (CD-DA) —- All DVD players and drives will read audio CDs (Red Book). This is not actually required by the DVD spec, but so far all manufacturers have stated that their DVD hardware will read CDs. On the other hand, you can't play a DVD in a CD player. (The pits are smaller, the tracks are closer together, the data layer is a different distance from the surface, the modulation is different, the error correction coding is new, etc.)
- CD-ROM is compatible with DVD-ROM — All DVD-ROM drives will read CD-ROMs (Yellow Book). However, DVD-ROMs are not readable by CD-ROM drives.
- CD-R maybe compatible with DVD-ROM — The problem is that CD-Rs (Orange Book Part II) are *invisible* to DVD laser wavelength because the dye used in CD-Rs doesn't reflect the beam. This problem is being addressed in many ways. Sony has developed a twin-laser pickup in which one laser is used for reading DVDs and the other for reading CDs and CD-Rs. Samsung has also announced dual-laser using a holographic annular masked lens. These solutions provide complete backwards compatibility with millions of CD-R discs. Philips has also stated that its DVD-ROM drives will read CD-Rs. In addition, new CD-R Type II blanks that will work with CD-ROM and DVD are supposedly in development. In the meantime, some first-generation DVD-ROM drives and many first-generation DVD-Video players will not read CD-R media.
- Is CD-RW is compatible with DVD —- CD-Rewritable (Orange Book Part III) discs can not be read by existing CD-ROM drives and CD players. CD-RW has a lower reflectivity difference, requiring automatic-gain-control (AGC) circuitry. The new *MultiRead* standard addresses this and some DVD manufacturers have already suggested they will support it. Supposedly the optical circuitry of DVD-ROM drives and DVD players is good enough to read CD-RW. CD-RW does not have the *invisibility* problem of CD-R .

- Video CD maybe compatible with DVD —- It's not required by the DVD spec, but it's trivial to support the White Book standard since any MPEG-2 decoder can also decode MPEG-1 from a Video CD. Panasonic, RCA, Samsung, and Sony models play Video CDs. Japanese Pioneer models play Video CDs but American models don't. Toshiba players don't play Video CDs

  VCD resolution is 352x288 for PAL and 352x240 for NTSC. The way most DVD players and Video CD players deal with the difference is to chop off the extra lines or add blank lines. When playing PAL VCDs, the Panasonic and RCA NTSC players apparently cut the entire 48 lines (17bottom and none off the top. The Sony looks better. [Does anyone know if it cuts 24 lines off the top and the bottom, or if it scales the full picture to fit?]

  Most DVD-ROM computers will be able to play Video CDs (with the right software), since its already possible with current-model CD-ROM computers.

  Note: Many Asian VCDs achieve *two* soundtracks by putting one language on the left channel and another on the right. They will be mixed together into babel on a stereo system unless you adjust the balance to get only one channel.

- Photo CD is **NOT** compatible with DVD — DVD players could support Photo CD with a few extra chips and a license from Kodak. No one has announced such a player. Most DVD-ROM drives will read Photo CDs (if they read CD-Rs) since it's trivial to support the XA and Orange Book multisession standards. The more important question is, "Does the OS or application support Photo CD"

- CD-I **NOT** compatible with DVD — Most DVD players will not play CD-I (Green Book) discs. However, Philips has announced that it will make a DVD player that supports CD-I. Some people expect Philips to create a *DVD-I* format in attempt to breathe a little more life into CD-I (and recover a bit more of the billion or so dollars they've invested in it).

- Enhanced CD is compatible with DVD — DVD players will play music from Enhanced Music CDs (Blue Book, CD Plus, CD Extra), and DVD-ROM drives will play music and read data from Enhanced CDs. Older ECD formats such as mixed mode and track zero (pregap, hidden track) should also be compatible, but there may be a problem with DVD-ROM drivers skipping track zero (as has been the case with some new CD-ROM drivers).

- Laserdisc is **NOT** compatible with DVD — Standard DVD players will not play laserdiscs, and you can't play a DVD disc on any standard laserdisc player. (Laserdisc uses analog video, DVD uses digital video; they are very different formats.)

However, Pioneer and Samsung have announced combo players that will play laserdiscs and DVDs (and also CDVs and audio CDs). Denon is rumored to have an LD/DVD player in the works also.

Sizes and capacities of DVD

There are many variations on the DVD theme. There are two physical sizes: 12 cm (4.7 inches) and 8 cm (3.1 inches), both 1.2 mm thick. These are the same form factors as CD. A DVD disc can be single-sided or double-sided. Each side can have one or two layers of data. The amount of video a disc can hold depends on how much audio accompanies it and how heavily the video and audio are compressed. The oft-quoted figure of 133 minutes is apocryphal: a DVD with only one audio track easily holds over 160 minutes, and a single layer can actually hold up to 9 hours of video and audio if it's compressed to VHS quality.

At a rough average rate of 4.7 Mbps (3.5 Mbps for video, 1.2 Mbps for three 5.1-channel soundtracks), a single-layer DVD holds around 135 minutes. A two-hour movie with three soundtracks can average 5.2 Mbps. A dual-layer disc can hold a two-hour movie at an average of 9.5 Mbps (very close to the 10.08 Mbps limit).

Capacities of DVD:

For reference, a CD-ROM holds about 650 MB (megabytes), which is 0.64 GB (gigabytes) or 0.68 G bytes (billion bytes). In the list below, SS/DS means single-/double-sided, SL/DL means single-/dual-layer, GB means gigabytes ($2^{3}0$), G means billions of bytes (10^{9}).

```
DVD-5 (12cm, SS/SL): 4.38 GB (4.7 G) of data, over 2 hours of video
DVD-9(12cm, SS/DL): 7.95 GB (8.5 G), about 4 hours
DVD-10 (12cm, DS/SL): 8.75 GB (9.4 G), about 4.5 hours
DVD-18 (12cm, DS/DL): 15.90 GB (17 G), over 8 hours
DVD-1? (8cm, SS/SL): 1.36 (1.4 G), about half an hour
DVD-2? (8cm, SS/DL): 2.48 GB (2.7 G), about 1.3 hours
DVD-3? (8cm, DS/SL): 2.72 GB (2.9 G), about 1.4 hours
DVD-4? (8cm, DS/DL): 4.95 GB (5.3 G), about 2.5 hours
DVD-R (12cm, SS/SL): 3.68 GB (3.95 G)
DVD-R (12cm, DS/SL): 7.38 GB (7.9 G)
DVD-R (8cm, SS/SL): 1.15 GB (1.23 G)
DVD-R (8cm, DS/SL): 2.29 GB (2.46 G)
DVD-RAM (12cm, SS/SL): 2.40 GB (2.58 G)
DVD-RAM (12cm, DS/SL): 4.80 GB (5.16 G)
```

Note: It takes about two gigabytes to store one hour of average video (Chapter 6).

The increase in capacity from CD-ROM is due to:

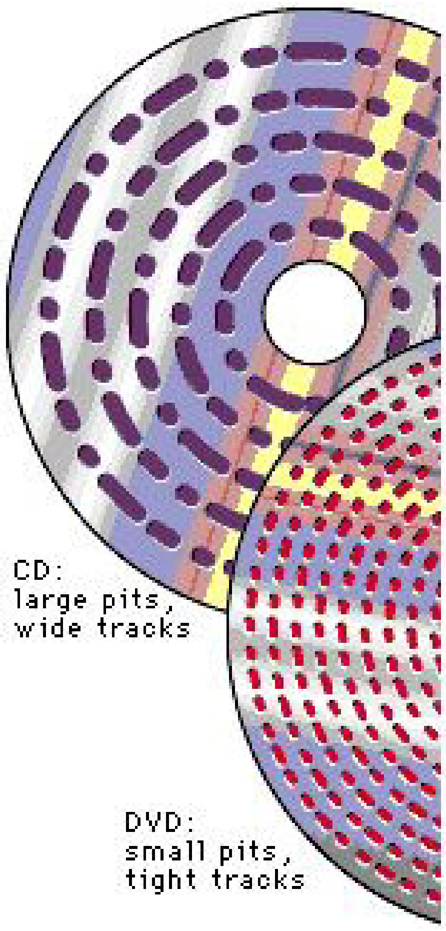


Figure 5.5: *DVD vs CD-ROM Pit Length*

- smaller pit length (∼2.08x) (Fig.(5.5),
- tighter tracks (∼2.16x),
- slightly larger data area (∼1.02x),
- slightly larger data area (∼1.02x),
- discs single or double sided
- another data layer added to each side creating a potential for four layers of data per disc (Fig. 5.6)
- more efficient channel bit modulation (∼1.06x),
- more efficient error correction (∼1.32x),

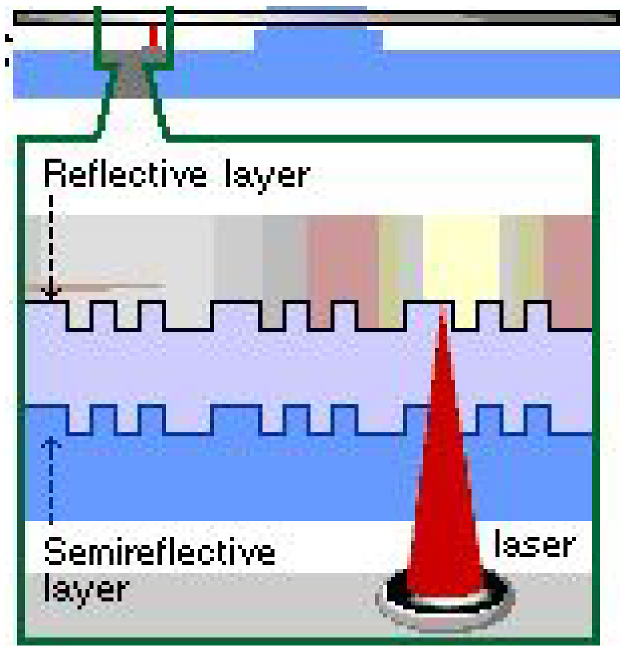


Figure 5.6: *DVD layers*

- less sector overhead (1.06x). Total increase for a single layer is about 7 times a standard CD-ROM. There's a slightly different explanation at http://www.mpeg.org/MPEG/DVD/General/Gain.html

The capacity of a dual-layer disc is slightly less than double that of a single-layer disc. The laser has to read through the outer layer to the inner layer (a distance of 20 to 70 microns). To reduce inter-layer crosstalk, the minimum pit length of both layers is increased from .4 um to .44 um. In addition, the reference scanning velocity is slightly faster – 3.84 m/s, as opposed to 3.49 m/s for single layer discs. Bigger pits, spaced farther apart, are easier to read correctly and are less susceptible to jitter. Bigger pits and fewer of them mean reduced capacity per layer.

DVD video details

We will look at the details of video and audio in general in Chapter 6 and compression (MPEG formats) in Chapter 7. These concepts are mentioned below and will be explained in further detail in these later chapters.

DVD-Video is an application of DVD-ROM. DVD-Video is also an application of MPEG-2. This means the DVD format defines subsets of these standards to be applied in practice as DVD-Video. DVD-ROM can contain any desired digital information, but DVD-Video is limited to certain data types designed for television reproduction.

A disc has one track (stream) of MPEG-2 constant bit rate (CBR) or variable bit rate (VBR) compressed digital video. A limited version of MPEG-2 Main Profile at Main Level (MP@ML) is used. MPEG-1 CBR and VBR video is also allowed. 525/60 (NTSC, 29.97 interlaced frames/sec) and 625/50 (PAL, 25 interlaced frames/sec) video systems are supported. Coded frame rates of 24 fps progressive or interlaced from film, 25 fps interlaced from PAL video, and 29.97 fps interlaced from NTSC video are supported. In the case of 24 fps source, the encoder embeds MPEG-2 repeat_first_field flags into the video stream to make the decoder either perform 3-2 pulldown for 60 (59.94) Hz displays or 2-2 pulldown (with 4displays. In other words, the player doesn't really know what the encoded rate is, it simply follows the MPEG-2 encoder's instructions to arrive at the predetermined display rate of 25 fps or 29.97 fps. (No current players convert from PAL to NTSC or NTSC to PAL.) See the Chapter 7 for more information on MPEG-2 video.

Picture dimensions are max 720x480 (29.97 frames/sec) or 720x576 (25 frames/sec). Pictures are subsampled from 4:2:2 ITU-R 601 down to 4:2:0, allocating an average of 12 bits/pixel. (Color depth is still 24 bits, since color samples are shared across 4 pixels.) The uncompressed source is 124.416 Mbps for video source (720x480x12x30 or 720x576x12x25), or either 99.533 or 119.439 Mbps for film source (720x480x12x24 or 720x576x12x24). Using the traditional (and rather subjective) television measurement of *lines of horizontal resolution* DVD can have 540 lines on a standard TV (720/(4/3)) and 405 on a widescreen TV (720/(16/9)). In practice, most DVD players provide about 500 lines because of filtering. VHS has about 230 (172 w/s) lines and laserdisc has about 425 (318 w/s).

Different players use different numbers of bits for the video digital-to-analog converter. (Sony and Toshiba use 10 bits, Pioneer and Panasonic use 9 bits.) This has nothing to do with the MPEG decoding process. It provides more *headroom* and more analog signal levels which supposedly give a better picture.

Maximum video bitrate is 9.8 Mbps. The *average* bitrate is 3.5 but depends entirely on the length, quality, amount of audio, etc. This is a 36:1 reduction from uncompressed 124 Mbps (or a 28:1 reduction from 100 Mbps film source). Raw channel data is read off the disc at a constant 26.16 Mbps. After 8/16 demodulation it's down to 13.08 Mbps. After error correction the user data stream goes into the track buffer at a constant 11.08 Mbps. The track buffer feeds system stream data out at a variable rate of up to 10.08 Mbps. After system overhead, the maximum rate of combined elementary streams (audio + video + subpicture) is 10.08. MPEG-1 video rate is limited to 1.856 Mbps with a typical rate of 1.15 Mbps.

Still frames (encoded as MPEG-2 I-frames) are supported and can be displayed for a specific amount of time or indefinitely. These are generally used for menus. Still frames can be accompanied by audio.

A disc also can have up to 32 subpicture streams that overlay the video for subtitles, captions for the hard of hearing, captions for children, karaoke, menus, simple animation, etc. These are full-screen, run-length-encoded bitmaps limited to four pixel types. For each group of subpictures, four colors are selected from a palette of 16 (from the YCrCb gamut), and four contrast values are selected out of 16 levels from transparent to opaque. Subpicture display command sequences can be used to create effects such as scroll, move, color/highlight, and fade. The maximum subpicture data rate is 3.36 Mbps, with a maximum size per frame of 53220 bytes.

Video can be stored on a DVD in 4:3 format (standard TV shape) or 16:9 (widescreen). The 16:9 format is *anamorphic*, meaning the picture is squeezed horizontally to fit a 4:3 rectangle then unsqueezed during playback. DVD players can output video in four different ways:

- full frame (4:3 video for 4:3 display)
- letterbox (16:9 video for 4:3 display)
- pan and scan (16:9 video for 4:3 display)
- widescreen (16:9 video for 16:9 display)

Video stored in 4:3 format is not changed by the player. It will appear normally on a standard 4:3 display. Widescreen systems will either enlarge it or add black bars to the sides. 4:3 video may have been formatted in various ways before being transferred to DVD. For example, it may have been letterboxed to hold video with a wider shape. Or it may have been panned and scanned from film composed for a wider theatrical presentation. All formatting done to the video prior to it being stored on the disc is transparent to the player. It merely reproduces the signal in standard form.

For automatic letterbox mode, the player creates black bars at the top and the bottom of the picture (60 lines each for NTSC, 72 for PAL). This leaves 3/4 of the height remaining, creating a shorter but wider rectangle. In order to fit this shorter rectangle, the picture is squeezed vertically using a *letterbox filter* that combines every 4 lines into 3. This compensates for the original horizontal squeezing, resulting in the movie being shown in its full width. The vertical resolution is reduced from 480 lines to 360.

For automatic pan and scan mode, the video is unsqueezed to 16:9 and a portion of the image is shown at full height on a 4:3 screen by following a 'center of interest' offset that's encoded in the video stream according to the preferences of the people who transferred the film to video. The pan and scan *window* is 75% of the full width, which reduces the horizontal pixels from 720 to 540. The pan and scan window can only travel laterally. This does not duplicate a true pan and scan process in which the window can also travel up and down and zoom in and out. Therefore, most DVD producers choose to put a separate pan and scan version on the disc in addition to the widescreen version.

For widescreen mode, the anamorphic video is stretched back out by widescreen equipment to its original width. If anamorphic video is shown on a standard 4:3 display, people will look like they have been on a crash diet. Widescreen mode is complicated because most movies today are shot with a *soft matte*. (The cinematographer has two sets of frame marks in her viewfinder, one for 1.33 (4:3) and one for 1.85, so she can allow for both formats). A few movies are even wider, such as the 2.35 ratio of Panavision. Since most movies are wider than 1.78 (16:9), one of at least 4 methods must be used during transfer to make it fit the 1.78 rectangle: 1) add additional thin black bars to the top and bottom; 2) include a small amount of extra picture at the top and bottom from the soft matte area; 3) crop the sides; 4) pan and scan with a 1.78 window. With the first two methods, the difference between 1.85 and 1.78 is so small that the letterbox bars or extra picture are hidden in the overscan area of most televisions. Nevertheless, and especially with 2.35 movies, many DVD producers put 16:9 source on one side (or layer) of the disc and 4:3 source on the other. This way the full-frame version of the film can be used for a horizontal and vertical pan and scan, and zoom process with no letterbox bars and no reduction in resolution.

Anamorphosis causes no problems with line doublers, which simply double the lines before they are stretched out by the widescreen display.

For anamorphic video, the pixels are fatter. Different pixel aspect ratios (none of them square) are used for each aspect ratio and resolution. 720-pixel and 704-pixel sizes have the same aspect ratio because the first includes overscan. Note that conventional values of 1.0950 and 0.9157 are for height/width (and are tweaked to match scanning rates). The table below uses less-confusing width/height values (y/x * h/w).

	720x480	720x576		
	704x480	704x486	352x480	352x576
4:3	0.909	1.091	1.818	2.182
16:9	1.212	1.455	2.424	2.909

Playback of widescreen material can be restricted. Programs can be marked for the following display modes:

- 4:3 full frame
- 4:3 LB (for automatically setting letterbox expand mode on widescreen TV)
- 16:9 LB only (player not allowed to pan and scan on 4:3 TV)
- 16:9 PS only (player not allowed to letterbox on 4:3 TV)
- 16:9 LB or PS (viewer can select pan and scan or letterbox on 4:3 TV)

DVD audio

The DVD-Audio format is not yet specified. The International Steering Committee announced it expects to have a final draft specification by December 1997. This means DVD-Audio products may show up around 1999.

The following details are for audio tracks on DVD-Video. Some DVD manufacturers such as Pioneer are developing audio-only players using the DVD-Video format.

A disc can have up to 8 audio tracks (streams). Each track can be in one of three formats:

- Dolby Digital (formerly AC-3): 1 to 5.1 channels
- MPEG-2 audio: 1 to 5.1 or 7.1 channels
- PCM: 1 to 8 channels.

Two additional optional formats are supported: DTS and SDDS. Both require external decoders.

The *.1* refers to a low-frequency effects (LFE) channel that connects to a subwoofer. This channel carries an emphasized bass audio signal.

All five audio formats support karaoke mode, which has two channels for stereo (L and R) plus an optional guide melody channel (M) and two optional vocal channels (V1 and V2).

Discs containing 525/60 (NTSC) video must use PCM or Dolby Digital on at least one track. Discs containing 625/50 (PAL/SECAM) video must use PCM or MPEG audio on at least one track. Additional tracks may be in any format. The DVD Forum has clarified that only stereo MPEG audio is mandatory for 625/50 discs, while multichannel MPEG-2 audio is recommended. Since multichannel MPEG-2 decoders are not yet available, most 625/50 discs include Dolby Digital audio.

For stereo output (analog or digital), all NTSC players and all PAL players (so far) have a built-in Dolby Digital decoder which downmixes from 5.1 channels (if present on the disc) to Dolby Surround stereo (i.e., 5 channels are matrixed into 2 channels to be decoded to 4 by an external Dolby Pro Logic processor). Both Dolby Digital and MPEG-2 support 2-channel Dolby Surround as the source in cases where the disc producer can't or doesn't want to remix the original onto discrete channels. This means that a DVD labelled as having Dolby Digital sound may only use the L/R channels for surround or *plain* stereo. Even movies with old monophonic soundtracks may use Dolby Digital – but only 1 or 2 channels.

The downmix process does not include the LFE channel and may compress the dynamic range in order to improve dialog audibility and keep the sound from becoming *muddy* on average home audio systems. This can result in reduced sound quality on high-end audio systems. Some players have the option to turn off the dynamic range compression. The downmix is auditioned when the disc is prepared, and if the result is not acceptable the audio may be tweaked or a separate L/R Dolby Surround track may be added. Experience has shown that minor tweaking is sometimes required to make the dialog more audible within the limited dynamic range of a home stereo system, but that a separate track is not usually necessary. If surround audio is important to you, you will hear significantly better results from multichannel discs if you have a Dolby Digital system.

Linear PCM is uncompressed (lossless) digital audio, the same format used on CDs. It can be sampled at 48 or 96 kHz with 16, 20, or 24 bits/sample. (Audio CD is limited to 44.1 kHz at 16 bits.) There can be from 1 to 8 channels. The maximum bitrate is 6.144 Mbps, which limits sample rates and bit sizes with 5 or more channels. It's generally felt that the 96 dB dynamic range of 16 bits or even the 120 dB range of 20 bits combined with a frequency response of up to 22,000 Hz from 48 kHz sampling is adequate for high-fidelity sound reproduction. However, additional bits and higher sampling rates are useful in studio work, noise shaping, advanced digital processing, and three-dimensional sound field reproduction. DVD players are required to support all the variations of LPCM, but some of them may subsample 96 kHz down to 48 kHz, and some may not use all 20 or 24 bits. The signal provided on the digital output for external digital-to-analog converters may be limited to less than 96 kHz or less than 24 bits.

Dolby Digital is multi-channel digital audio, compressed using AC-3 coding technology from original PCM with a sample rate of 48 kHz at 16 bits. The bitrate is 64 kbps to 448 kbps, with 384 being the normal rate for 5.1 channels and 192 being the normal rate for stereo (with or without surround encoding). The channel combinations are (front/surround): 1/0, 1+1/0 (dual mono), 2/0, 3/0, 2/1, 3/1, 2/2, and 3/2. The LFE channel is optional with all 8 combinations.

MPEG audio is multi-channel digital audio, compressed from original PCM format with sample rate of 48 kHz at 16 bits. Both MPEG-1 and MPEG-2 formats are supported. The variable bitrate is 32 kbps to 912 kbps, with 384 being the normal average rate. MPEG-1 is limited to 384 kbps. Channel combinations are (front/surround): 1/0, 2/0, 2/1, 2/2, 3/0, 3/1, 3/2, and 5/2. The LFE channel is optional with all combinations. The 7.1 channel format adds left-center and right-center channels, but will probably be rare for home use. MPEG-2 surround channels are in an extension stream matrixed onto the MPEG-1 stereo channels, which makes MPEG-2 audio backwards compatible with MPEG-1 hardware (an MPEG-1 system will only see the two stereo channels.)

DTS is an optional multi-channel (5.1) digital audio format, compressed from PCM at 48 kHz. The data rate is from 64 kbps to 1536 kbps. Channel combinations are (front/surround): 1/0, 2/0, 3/0, 2/1, 2/2, 3/2. The LFE channel is optional with all 6 combinations.

SDDS is an optional multi-channel (5.1 or 7.1) digital audio format, compressed from PCM at 48 kHz. The data rate can go up to 1280 kbps.

A DVD-5 with only one surround stereo audio stream (at 192 kbps) can hold over 55 hours of audio. A DVD-18 can hold over 200 hours.

Interactive DVD features

DVD-Video players (and software DVD-Video navigators) support a command set that provides rudimentary interactivity. The main feature is menus, which are present on almost all discs to allow content selection and feature control. Each menu has a still-frame graphic and up to 36 highlightable, rectangular buttons (only 12 if widescreen, letterbox, and pan and scan modes are used). Remote control units have four arrow keys for selecting onscreen buttons, plus numeric keys, select key, menu key, and return key. Additional remote functions may include freeze, step, slow, fast, scan, next, previous, audio select, subtitle select, camera angle select, play mode select, search to program, search to part of title (chapter), search to time, and search to camera angle. Any of these features can be disabled by the producer of the disc.

Additional features of the command set include simple math (add, subtract, multiply, divide, modulo, random), bitwise and, bitwise or, bitwise xor, plus comparisons (equal, greater than, etc.), and register loading, moving, and swapping. There are 24 system registers for information such as language code, audio and subpicture settings, and parental level. There are 16 general registers for command use. A countdown timer is also provided. Commands can branch or jump to other commands. Commands can also control player settings, jump to different parts of the disc, and control presentation of audio, video, subpicture, camera angles, etc.

DVD-V content is broken into *titles* (movies or albums), and *parts of titles* (chapters or songs). Titles are made up of *cells* linked together by one or more *program chains* (PGC). A PGC can be defined as sequential play, random play (may repeat), or shuffle play (random order but no repeats). Individual cells may be used by more than one PGC, which is how parental management and seamless branching are accomplished: different PGCs define different sequences through mostly the same material.

Additional material for camera angles and seamless branching is interleaved together in small chunks. The player jumps from chunk to chunk, skipping over unused angles or branches, to stitch together the seamless video. Since angles are stored separately, they have no direct effect on the bitrate but they do affect the playing time. Adding 1 camera angle for a program roughly doubles the amount of space it requires (and cuts the playing time in half).

5.5.7 DVD and computers

SO far we have focussed on the media representation and standard DVD players. DVD and DVD-ROM in particular is beginning to have a huge impact on computers.

For a computer to employ DVD it must have the following features:

In addition to a DVD-ROM drive, you must have extra hardware to decode MPEG-2 video and Dolby Digital/MPEG-2/PCM audio. The computer operating system or playback system must support regional codes and be licensed to decrypt copy-protected movies. You may also need software that can read the MicroUDF format used to store DVD data files and interpret the DVD control codes. It's estimated that 10-30% of new computers with DVD-ROM drives will include decoder hardware, and that most of the remaining DVD-ROM computers will include movie playback software.

Some DVD-Videos and many DVD-ROMs will use video encoded using MPEG-1 instead of MPEG-2. Many existing computers have MPEG-1 hardware built in or are able to decode MPEG-1 with software.

CompCore Multimedia and Mediamatics make software to play DVD-Video movies (SoftDVD, DVD Express). Both require at least a 233 MHz Pentium MMX with AGP and an IDE/SCSI DVD-ROM drive with bus mastering DMA support to achieve about 20 frame/sec film rates (or better than 300 MHz for 30 frame/sec video), and can decrypt copy-protected movies. Oak's software requires hardware support. The software *navigators* support most DVD-Video features (menus, subpictures, etc.) and can emulate a DVD-Video remote control.

CompCore, Mediamatics, and Oak Technology have defined standards to allow certain MPEG decoding tasks to be performed by hardware on a video card and the remainder by software. Video graphics controllers with this feature are being called *DVD MPEG-2 accelerated.* (The Mediamatics standard is called MVCCA.)

If you have at least a 433 MHz Alpha workstation you'll be able to play DVD movies at full 30 fps in software.

DVD-ROM Drives:

Most DVD-ROM drives have a seek time of 150-200 ms, access time of 200-250 ms, and data transfer rate of 1.3 MB/s ($11.08 * 10^6/8/2^{20}$) with burst transfer rates of up to 12 MB/s or higher. The data transfer rate from DVD-ROM discs is roughly equivalent to a 9x CD-ROM drive. DVD spin rate is about 3 times faster than CD, so when reading CD-ROMs, some DVD-ROM drives transfer data at 3x speed while others are faster. 2x and 3x DVD-ROM drives are already in the works. Hitachi is shipping samples of a 2x DVD-ROM drive which also reads CDs at 20x.

Connectivity is similar to that of CD-ROM drives: EIDE (ATAPI), SCSI-2, etc. All DVD-ROM drives have audio connections for playing audio CDs. No DVD-ROM drives have been announced with DVD audio or video outputs (which would require internal audio/video decoding hardware).

DVD-ROMs use a MicroUDF/ISO 9660 bridge file system. The OSTA UDF file system will eventually replace the ISO 9660 system of CD-ROMs, but the bridge format provides backwards compatibility until operating systems support UDF.

Recordable DVD-ROM: DVD-R and DVD-RAM:

There are two recordable versions of DVD-ROM: DVD-R (record once) and DVD-RAM (erase and record many times), with capacities of 3.95 and 2.58 G bytes. Both specifications have been published. DVD-R and DVD-RAM are not currently usable for home video recording.

DVD-R uses organic dye polymer technology like CD-R and is compatible with almost all DVD drives. The technology will improve to support 4.7 G bytes in 1 to 2 years, which is crucial for desktop DVD-ROM and DVD-Video production.

Further Information

Further information on DVD can be obtained from:

http://www.dvddigital.com/
In fact this section is heavily based on the FAQs section at
http://www.dvddigital.com/facts

Part IV

Multimedia Data

Chapter 6

Multimedia Data Representations

In this chapter we focus on the underlying representations of common forms of media Audio, Graphics and Video. In the next chapter we focus on compression techniques as we shortly understand why these data can be large in storage size (even when compressed).

The topics we consider here are specifically:

- Digital Audio
- Sampling/Digitisation
- Compression (Details of Compression algorithms Next Chapter)
- Graphics/Image Formats
- Digital Video (Basics of Video end of this chapter but more on Didgiatl Video in next Chapter as it so closely entwined with Compression)

6.1 Basics of Digital Audio

6.1.1 Application of Digital Audio — Selected Examples

Music Production

- Hard Disk Recording
- Sound Synthesis
- Samplers
- Effects Processing

Video – Audio Important Element: Music and Effects

Web — Many uses on Web

- Spice up Web Pages
- Listen to Cds
- Listen to Web Radio

Many More Uses — try and think of some?

6.1.2 Digitization of Sound

Let us first analyse what a sound actually is:

- Sound is a continuous wave that travels through the air
- The wave is made up of pressure differences. Sound is detected by measuring the pressure level at a location.
- Sound waves have normal wave properties (reflection, refraction, diffraction, etc.).

A variety of sound sources:

Source — Generates Sound

- Air Pressure changes
- *Electrical* — Loud Speaker
- *Acoustic* — Direct Pressure Variations

The destination receives (sensed the sound wave pressure changes) and has to deal with accordingly:

Destination — Receives Sound

- *Electrical* — Microphone produces electric signal
- *Ears* — Responds to pressure **hear** sound

Sound is required input into a computer: it needs to sampled or digitised:

- Microphones, video cameras produce *analog signals* (continuous-valued voltages) as illustrated in Fig 6.1

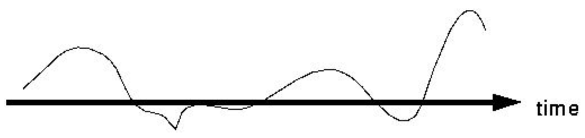


Figure 6.1: Continuous Analog Waveform

- To get audio or video into a computer, we have to *digitize* it (convert it into a stream of numbers) **Need to convert Analog-to-Digital** — Specialised Hardware
- So, we have to understand *discrete sampling* (both time and voltage)
- *Sampling* – divide the horizontal axis (the time dimension) into discrete pieces. Uniform sampling is ubiquitous.
- *Quantization* – divide the vertical axis (signal strength) into pieces. Sometimes, a non-linear function is applied.
    - 8 bit quantization divides the vertical axis into 256 levels. 16 bit gives you 65536 levels.

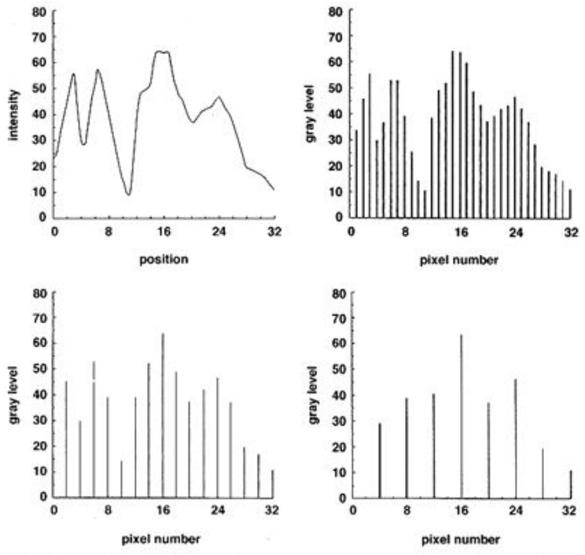


Figure 6.2: Sampling a Waveform at different discrete frequencies affects the perceived waveform

6.1.3 Digitizing Audio

That is the basic idea of digitizing a sound unfortunately things are (practically speaking) not so simple.

- Questions for producing digital audio (Analog-to-Digital Conversion):
    1. How often do you need to sample the signal?
    2. How good is the signal?
    3. How is audio data formatted?

6.1.4 Computer Manipulation of Sound

Once Digitised processing the digital sound is essentially straightforward although it depends on the processing you wish to do (*e.g.* volume is easier to code than accuarte reverb)

Essentially they all operate on the 1-D array of digitised samples, typical examples include:

- Volume
- Cross-Fading
- Looping
- Echo/Reverb/Delay
- Filtering
- Signal Analysis

Soundedit Demos

- Volume
- Cross-Fading
- Looping
- Echo/Reverb/Delay
- Filtering

6.1.5 Sample Rates and Bit Size

How do we store each sample value (*Quantisation*)?

8 Bit Value (0-255)

16 Bit Value (Integer) (0-65535)

How many Samples to take?

11.025 KHz — Speech (Telephone 8KHz)

22.05 KHz — Low Grade Audio
(WWW Audio, AM Radio)

44.1 KHz — CD Quality

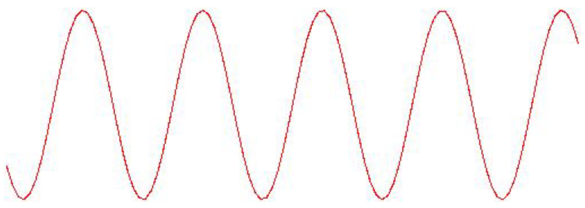


Figure 6.3: A Sine Wave

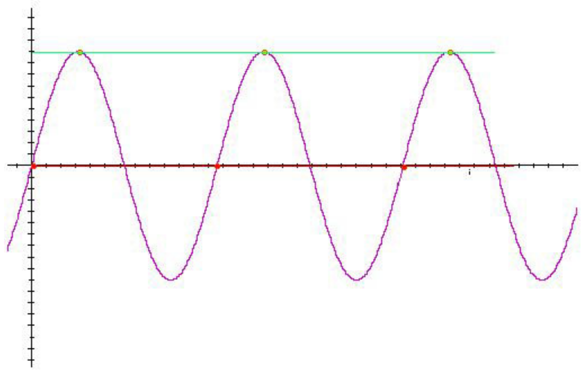


Figure 6.4: Sampling at 1 time per cycle

6.1.6 Nyquist's Sampling Theorem

- Suppose we are sampling a sine wave (Fig 6.3. How often do we need to sample it to figure out its frequency?
- If we sample at 1 time per cycle, we can think it's a constant (Fig 6.4)
- If we sample at 1.5 times per cycle, we can think it's a lower frequency sine wave (Fig. 6.6)

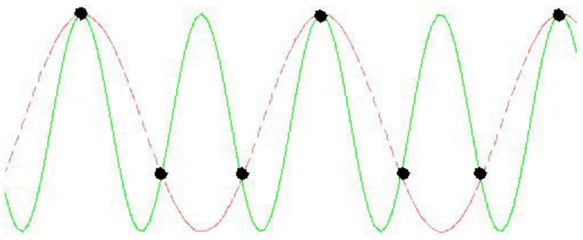


Figure 6.5: Sampling at 1.5 times per cycle

- Now if we sample at twice the sample frequency, i.e Nyquist Rate, we start to make some progress. An alternative way of viewing thr waveform (re)genereation is to think of straight lines joining up the peaks of the samples. In this case (at these sample points) we see we get a sawtooth wave that begins to start crudely approximating a sine wave

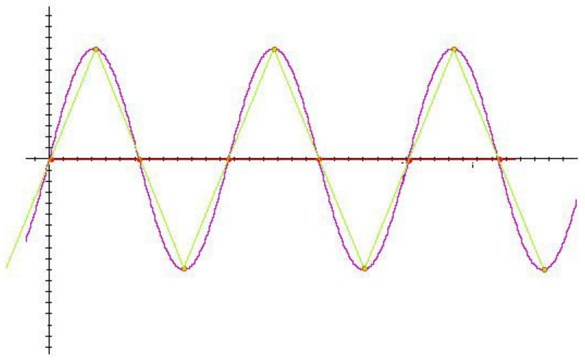


Figure 6.6: Sampling at the Nyquist Rate (Twice the Frequency)

- **Nyquist rate** — For lossless digitization, the sampling rate should be *at least twice* the maximum frequency responses. Indeed many times more the better.

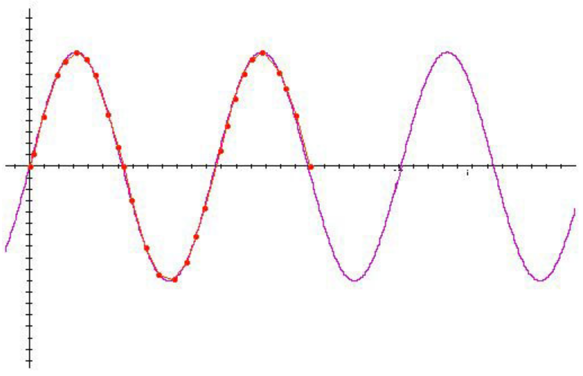


Figure 6.7: Sampling at many times per cycle

6.1.7 Implications of Sample Rate and Bit Size

Affects Quality of Audio

- Ears do not respond to sound in a linear fashion
- Decibel (**dB**) a logarithmic measurement of sound
- 16-Bit has a signal-to-noise ratio of 98 dB — virtually inaudible
- 8-bit has a signal-to-noise ratio of 50 dB
- Therefore, 8-bit is roughly 8 times as noisy
  - 6 dB increment is twice as loud

Signal to Noise Ratio (SNR)

- In any analog system, some of the voltage is what you want to measure (*signal*), and some of it is random fluctuations (*noise*).
- Ratio of the power of the two is called the *signal to noise ratio* (*SNR*). SNR is a measure of the quality of the signal.
- SNR is usually measured in decibels (*dB*).

$$SNR = 10 \log \frac{V_{signal}^2}{V_{noise}^2} = 20 \log \frac{V_{signal}}{V_{noise}}$$

- Typically 8 bits or 16 bits.
- Each bit adds about 6 dB of resolution, so 16 bits => 96 dB.

File Type	*44.1 KHz*	*22.05 KHz*	*11.025 KHz*
16 Bit Stereo	10.1 Mb	5.05 Mb	2.52 Mb
16 Bit Mono	5.05 Mb	2.52 Mb	1.26 Mb
8 Bit Mono	2.52 Mb	1.26 Mb	630 Kb

Memory Required for 1 Minute of Digital Audio

- Samples are typically stored as raw numbers (*linear format*), or as logarithms (*u-law* (or *A-law* in Europe)).
  - Logarithmic representation approximates *perceptual uniformity.*

Affects Size of Data

There is therfore is a trade off between *Audio Quality vs. Data Rate*
Some typical applications of sample bit size and sample rate are listed below:

Quality	Sample Rate (KHz)	Bits per Sample	Mono/ Stereo	Data Rate (Uncompressed)	Frequency Band
Telephone	8	8	Mono	8 KBytes/sec	200-3,400 Hz
AM Radio	11.025	8	Mono	11.0 KBytes/sec	
FM Radio	22.050	16	Stereo	88.2 KBytes/sec	
CD	44.1	16	Stereo	176.4 KBytes/sec	20-20,000 Hz
DAT	48	16	Stereo	192.0 KBytes/sec	20-20,000 Hz

- Telephone uses *u-law* encoding, others use linear. So the dynamic range of digital telephone signals is effectively 13 bits rather than 8 bits.
- CD quality stereo sound –> 10.6 MB / min.

Practical Implications of Nyquist Sampling Theory

- Must (low pass) filter signal before sampling:

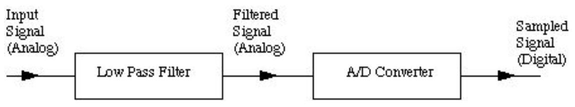


Figure 6.8: Practical Implication of Nyquist;Must Low Pass Filter Signal before Sampling

- Otherwise strange artifacts from high frequency signals appear and are audible.

We'll finish off with a question: *Why are CD Sample Rates 44.1 KHz?*
The answer should be obvious if you have paid attention to the above notes (Answer in Lecture).

6.1.8 Typical Audio Formats

- Popular audio file formats include .au (Unix workstations), .aiff (MAC, SGI), .wav (PC, DEC workstations)
- A simple and widely used audio compression method is Adaptive Delta Pulse Code Modulation (ADPCM). Based on past samples, it predicts the next sample and encodes the difference between the actual value and the predicted value.

6.1.9 Delivering Audio over a Network

- **Trade off between desired fidelity and file size**
- Bandwidth Considerations for Web and other media.
- Compress Files:
  - Could affect live transmission on Web

Streaming Audio

- Buffered Data:
  - Trick get data to destination before it's needed
  - Temporarily store in memory (Buffer)
  - Server keeps feeding the buffer
  - Client Application reads buffer
- Needs Reliable Connection, moderately fast too.
- Specialised client, Steaming Audio Protocol (PNM for real audio).

6.2 Synthetic Sounds

- FM (Frequency Modulation) Synthesis – used in low-end Sound Blaster cards, OPL-4 chip, Yamaha DX Synthesiser range popular in Early 1980's.
- Wavetable synthesis – wavetable generated from sound waves of real instruments
- Modern Synthesiser use a mixture of sample and synthesis.

6.3 Introduction to MIDI (Musical Instrument Digital Interface)

Definition of MIDI: a protocol that enables computer, synthesizers, keyboards, and other musical device to communicate with each other.

6.3.1 Components of a MIDI System

Synthesizer:

- It is a sound generator (various pitch, loudness, tone colour).
- A good (musician's) synthesizer often has a microprocessor, keyboard, control panels, memory, etc.

Sequencer:

- It can be a stand-alone unit or a software program for a personal computer. (It used to be a storage server for MIDI data. Nowadays it is more a software *music editor* on the computer.
- It has one or more MIDI INs and MIDI OUTs.

Track:

- Track in sequencer is used to organize the recordings.
- Tracks can be turned on or off on recording or playing back.

Channel:

- MIDI channels are used to separate information in a MIDI system.
- There are 16 MIDI channels in one cable.
- Channel numbers are coded into each MIDI message.

Timbre:

- The quality of the sound, e.g., flute sound, cello sound, etc.
- Multitimbral – capable of playing many different sounds at the same time (e.g., piano, brass, drums, etc.)

Pitch:

- musical note that the instrument plays

Voice:

- Voice is the portion of the synthesizer that produces sound.
- Synthesizers can have many (12, 20, 24, 36, etc.) voices.
- Each voice works independently and simultaneously to produce sounds of different timbre and pitch.

Patch:

- the control settings that define a particular timbre.

6.3.2 Hardware Aspects of MIDI

MIDI connectors:

– three 5-pin ports found on the back of every MIDI unit

- MIDI IN: the connector via which the device receives all MIDI data.
- MIDI OUT: the connector through which the device transmits all the MIDI data it generates itself.
- MIDI THROUGH: the connector by which the device echoes the data receives from MIDI IN.

Note: It is only the MIDI IN data that is echoed by MIDI through. All the data generated by device itself is sent through MIDI OUT.

Figure 6.9 illustrates a typical setup where:

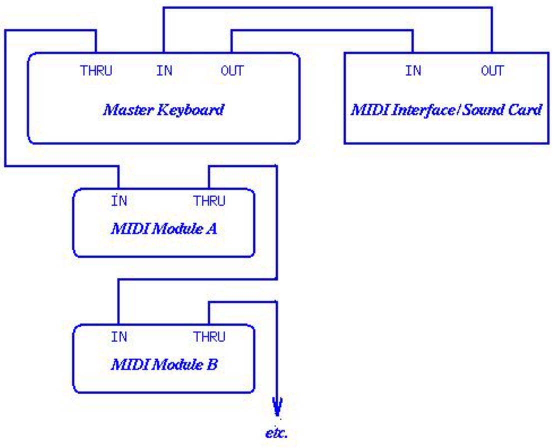


Figure 6.9: A Typical MIDI Sequencer Setup

- MIDI OUT of synthesizer is connected to MIDI IN of sequencer.
- MIDI OUT of sequencer is connected to MIDI IN of synthesizer and through to each of the additional sound modules.
- During recording, the keyboard-equipped synthesizer is used to send MIDI message to the sequencer, which records them.
- During play back: messages are send out from the sequencer to the sound modules and the synthesizer which will play back the music.

6.3.3 MIDI Messages

MIDI messages are used by MIDI devices to communicate with each other.

Structure of MIDI messages:

- MIDI message includes a status byte and up to two data bytes.
- Status byte
  - The most significant bit of status byte is set to 1.
  - The 4 low-order bits identify which channel it belongs to (four bits produce 16 possible channels).
  - The 3 remaining bits identify the message.
- The most significant bit of data byte is set to 0.

Classification of MIDI messages:

```
                                             ----- voice messages
                  ---- channel messages -----|
                 |                           ----- mode messages
                 |
MIDI messages ----|
                 |                           ---- common messages
                  ----- system messages -----|---- real-time messages
                                             ---- exclusive messages
```

A. Channel messages:

– messages that are transmitted on individual channels rather that globally to all devices in the MIDI network.

A.1. Channel voice messages:

- Instruct the receiving instrument to assign particular sounds to its voice
- Turn notes on and off
- Alter the sound of the currently active note or notes

Voice Message	Status Byte	Data Byte1	Data Byte2
Note off	8x	Key number	Note Off velocity
Note on	9x	Key number	Note on velocity
Polyphonic Key Pressure	Ax	Key number	Amount of pressure
Control Change	Bx	Controller number	Controller value
Program Change	Cx	Program number	None
Channel Pressure	Dx	Pressure value	None
Pitch Bend	Ex	MSB	LSB

Notes: ‘x’ in status byte hex value stands for a channel number.

Example: a Note On message is followed by two bytes, one to identify the note, and on to specify the velocity.

To play note number 80 with maximum velocity on channel 13, the MIDI device would send these three hexadecimal byte values: 9C 50 7F

A.2. Channel mode messages: – Channel mode messages are a special case of the Control Change message (Bx or 1011nnnn). The difference between a Control message and a Channel Mode message, which share the same status byte value, is in the first data byte. Data byte values 121 through 127 have been reserved in the Control Change message for the channel mode messages.

- Channel mode messages determine how an instrument will process MIDI voice messages.

1st Data Byte	Description	Meaning of 2nd Data Byte
79	Reset all controllers	None; set to 0
7A	Local control	0 = off; 127 = on
7B	All notes off	None; set to 0
7C	Omni mode off	None; set to 0
7D	Omni mode on	None; set to 0
7E	Mono mode on (Poly mode off)	**
7F	Poly mode on (Mono mode off)	None; set to 0

** if value = 0 then the number of channels used is determined by the receiver; all other values set a specific number of channels, beginning with the current basic channel.

B. System Messages:

- System messages carry information that is not channel specific, such as timing signal for synchronization, positioning information in pre-recorded MIDI sequences, and detailed setup information for the destination device.

B.1. System real-time messages:

- messages related to synchronization

System Real-Time Message	Status Byte
Timing Clock	F8
Start Sequence	FA
Continue Sequence	FB
Stop Sequence	FC
Active Sensing	FE
System Reset	FF

B.2. System common messages:

- contain the following unrelated messages

System Common Message	Status Byte	Number of Data Bytes
MIDI Timing Code	F1	1
Song Position Pointer	F2	2
Song Select	F3	1
Tune Request	F6	None

B.3. System exclusive message:

- (a) Messages related to things that cannot be standardized, (b) addition to the original MIDI specification.
- It is just a stream of bytes, all with their high bits set to 0, bracketed by a pair of system exclusive start and end messages (F0 and F7).

6.3.4 General MIDI

- MIDI + Instrument Patch Map + Percussion Key Map –> a piece of MIDI music sounds the same anywhere it is played
  - Instrument patch map is a standard program list consisting of 128 patch types.
  - Percussion map specifies 47 percussion sounds.
  - Key-based percussion is always transmitted on MIDI channel 10.
- Requirements for General MIDI Compatibility:
  - Support all 16 channels.
  - Each channel can play a different instrument/program (multitimbral).
  - Each channel can play many voices (polyphony).
  - Minimum of 24 fully dynamically allocated voices.

Additional MIDI Specifications

General MIDI Instrument Patch Map

Prog No.	Instrument	Prog No.	Instrument
(1-8	PIANO)	(9-16	CHROM PERCUSSION)
1	Acoustic Grand	9	Celesta
2	Bright Acoustic	10	Glockenspiel
3	Electric Grand	11	Music Box
4	Honky-Tonk	12	Vibraphone
5	Electric Piano 1	13	Marimba

6	Electric Piano 2	14	Xylophone
7	Harpsichord	15	Tubular Bells
8	Clav	16	Dulcimer
	(17-24 ORGAN)		(25-32 GUITAR)
17	Drawbar Organ	25	Acoustic Guitar(nylon)
18	Percussive Organ	26	Acoustic Guitar(steel)
19	Rock Organ	27	Electric Guitar(jazz)
20	Church Organ	28	Electric Guitar(clean)
21	Reed Organ	29	Electric Guitar(muted)
22	Accoridan	30	Overdriven Guitar
23	Harmonica	31	Distortion Guitar
24	Tango Accordian	32	Guitar Harmonics
	(33-40 BASS)		(41-48 STRINGS)
33	Acoustic Bass	41	Violin
34	Electric Bass(finger)	42	Viola
35	Electric Bass(pick)	43	Cello
36	Fretless Bass	44	Contrabass
37	Slap Bass 1	45	Tremolo Strings
38	Slap Bass 2	46	Pizzicato Strings
39	Synth Bass 1	47	Orchestral Strings
40	Synth Bass 2	48	Timpani
	(49-56 ENSEMBLE)		(57-64 BRASS)
49	String Ensemble 1	57	Trumpet
50	String Ensemble 2	58	Trombone
51	SynthStrings 1	59	Tuba
52	SynthStrings 2	60	Muted Trumpet
53	Choir Aahs	61	French Horn
54	Voice Oohs	62	Brass Section
55	Synth Voice	63	SynthBrass 1
56	Orchestra Hit	64	SynthBrass 2
	(65-72 REED)		(73-80 PIPE)
65	Soprano Sax	73	Piccolo
66	Alto Sax	74	Flute
67	Tenor Sax	75	Recorder
68	Baritone Sax	76	Pan Flute
69	Oboe	77	Blown Bottle
70	English Horn	78	Skakuhachi
71	Bassoon	79	Whistle
72	Clarinet	80	Ocarina
	(81-88 SYNTH LEAD)		(89-96 SYNTH PAD)
81	Lead 1 (square)	89	Pad 1 (new age)
82	Lead 2 (sawtooth)	90	Pad 2 (warm)

83	Lead 3 (calliope)	91	Pad 3 (polysynth)
84	Lead 4 (chiff)	92	Pad 4 (choir)
85	Lead 5 (charang)	93	Pad 5 (bowed)
86	Lead 6 (voice)	94	Pad 6 (metallic)
87	Lead 7 (fifths)	95	Pad 7 (halo)
88	Lead 8 (bass+lead)	96	Pad 8 (sweep)
(97-104	SYNTH EFFECTS)	(105-112	ETHNIC)
97	FX 1 (rain)	105	Sitar
98	FX 2 (soundtrack)	106	Banjo
99	FX 3 (crystal)	107	Shamisen
100	FX 4 (atmosphere)	108	Koto
101	FX 5 (brightness)	109	Kalimba
102	FX 6 (goblins)	110	Bagpipe
103	FX 7 (echoes)	111	Fiddle
104	FX 8 (sci-fi)	112	Shanai
(113-120	PERCUSSIVE)	(121-128	SOUND EFFECTS)
113	Tinkle Bell	121	Guitar Fret Noise
114	Agogo	122	Breath Noise
115	Steel Drums	123	Seashore
116	Woodblock	124	Bird Tweet
117	Taiko Drum	125	Telephone Ring
118	Melodic Tom	126	Helicopter
119	Synth Drum	127	Applause
120	Reverse Cymbal	128	Gunshot

General MIDI Percussion Key Map

MIDI Key	Drum Sound	MIDI Key	Drum Sound
35	Acoustic Bass Drum	59	Ride Cymbal 2
36	Bass Drum 1	60	Hi Bongo
37	Side Stick	61	Low Bongo
38	Acoustic Snare	62	Mute Hi Conga
39	Hand Clap	63	Open Hi Conga
40	Electric Snare	64	Low Conga
41	Low Floor Tom	65	High Timbale
42	Closed Hi-Hat	66	Low Timbale
43	High Floor Tom	67	High Agogo
44	Pedal Hi-Hat	68	Low Agogo
45	Low Tom	69	Cabasa
46	Open Hi-Hat	70	Maracas
47	Low-Mid Tom	71	Short Whistle
48	Hi-Mid Tom	72	Long Whistle

49	Crash Cymbal 1	73	Short Guiro
50	High Tom	74	Long Guiro
51	Ride Cymbal 1	75	Claves
52	Chinese Cymbal	76	Hi Wood Block
53	Ride Bell	77	Low Wood Block
54	Tambourine	78	Mute Cuica
55	Splash Cymbal	79	Open Cuica
56	Cowbell	80	Mute Triangle
57	Crash Cymbal 2	81	Open Triangle
58	Vibraslap		

6.3.5 Digital Audio and MIDI

There are many application os DIgital Audio and Midi being used together:

- Modern Recording Studio — Hard Disk Recording and MIDI
  - Analog Sounds (Live Vocals, Guitar, Sax etc) — DISK
  - Keyboards, Drums, Samples, Loops Effects — MIDI
- Sound Generators: use a mix of
  - Synthesis
  - Samples
- Samplers — Digitise (Sample) Sound then
  - Playback
  - Loop (beats)
  - Simulate Musical Instruments

6.3.6 Digital Audio, Synthesis, Midi and Compression — MPEG 4 Structured Audio

- We have seen the need for compression already in Digital Audio — Large Data Files
- Basic Ideas of compression (see next Chapter) used as integral part of audio format — MP3, real audio *etc.*
- Mpeg-4 audio — actually combines compression synthesis and midi to have a massive impact on compression.
- Midi, Synthesis encode what note to play and how to play it with a small number of parameters — Much greater reduction than simply having some encoded bits of audio.
- Responsibility to create audio delegated to generation side.

6.3.7 MPEG-4 Structured Audio

MPEG-4 covers the the whole range of digital audio:

- from very low bit rate speech
- to full bandwidth high quality audio
- built in anti-piracy measures
- *Structured Audio*

Structured Audio Tools
MPEG-4 comprises of 6 *Structured Audio tools* are:

SAOL , the Structured Audio Orchestra Language

SASL , the Structured Audio Score Language

SASBF , the Structured Audio Sample Bank Format

a set of MIDI semantics which describes how to control SAOL with MIDI

a scheduler , which describes how to take the above parts and create sound

the AudioBIFS part of BIFS, which lets you make audio soundtracks in MPEG-4 using a variety of tools and effects-processing techniques

Very briefly each of the above tools have a specific function
SAOL (Structured Audio Orchestra Language)
SAOL is pronounced like the English word "sail" and is the central part of the Structured Audio toolset. It is a new software-synthesis language; it was specifically designed it for use in MPEG-4. You can think of SAOL as a language for describing synthesizers; a program, or instrument, in SAOL corresponds to the circuits on the inside of a particular hardware synthesizer.

SAOL is not based on any particular method of synthesis. It is general and flexible enough that any known method of synthesis can be described in SAOL. Examples of FM synthesis, physical-modeling synthesis, sampling synthesis, granular synthesis, subtractive synthesis, FOF synthesis, and hybrids of all of these in SAOL.

There's a page on using MIDI to control SAOL available at
http://sound.media.mit.edu/mpeg4-old/
SASL (Structured Audio Score Language)
SASL is a very simple language that was created for MPEG-4 to control the synthesizers specified by SAOL instruments. A SASL program, or score, contains instructions that tell SAOL what notes to play, how loud to play them, what tempo to play them at, how long they last, and how to control them (vary them while they're playing).

SASL is like MIDI in some ways, but doesn't suffer from MIDI's restrictions on temporal resolution or bandwidth. It also has a more sophisticated controller structure than MIDI; since in SAOL, you can write controllers to do anything, you need to be able to flexibly control them in SASL.

SASL is simpler (or more ”lightweight”) than many other score protocols. It doesn’t have any facilities for looping, sections, repeats, expression evaluation, or some other things. Most SASL scores will be created by automatic tools, and so it’s easy to make those tools map from the intent of the composer (”repeat this block”) to the particular arrangement of events that implement the intent.

SASBF (Structured Audio Sample Bank Format)

SASBF (pronounces ”sazz-biff”!!!) is a format for efficiently transmitting banks of sound samples to be used in wavetable, or sampling, synthesis. The format is being re-examined right now in hopes of making it at least partly compatible with the MIDI Downloaded Sounds (DLS) format.

The most active participants in this activity are E-Mu Systems and the MIDI Manufacturers Association (MMA).

MIDI Semantics

As well as controlling synthesis with SASL scripts, it can be controlled with MIDI files and scores in MPEG-4. MIDI is today’s most commonly used representation for music score data, and many sophisticated authoring tools (such as sequencers) work with MIDI.

The MIDI syntax is external to the MPEG-4 Structured Audio standard; only references to the MIDI Manufacturers Association’s definition in the standard. But in order to make the MIDI controls work right in the MPEG context, some semantics (what the instructions ”mean”) have been redefined in MPEG-4. The new semantics are carefully defined as part of the MPEG-4 specification.

Scheduler

The scheduler is the ”guts” of the Structured Audio definition. It’s a set of carefully defined and somewhat complicated instructions that specify how SAOL is used to create sound when it is driven by MIDI or SASL. It’s in the style of ”when this instruction arrives, you have to remember this, then execute this program, then do this other thing”.

This component of Structured Audio is crucial but very dull unless you’re a developer who wants to implement a SAOL system.

AudioBIFS

BIFS is the MPEG-4 *Binary Format for Scene Description.* It’s the component of MPEG-4 Systems which is used to describe how the different ”objects” in a structured media scene fit together. To explain this a little more: in MPEG-4, the video clips, sounds, animations, and other pieces each have special formats to describe them. But to have something to show, we need to put the pieces together – the background goes in the back, this video clip attaches to the side of this ”virtual TV” object, the sound should sound like it’s coming from the speaker over there. BIFS lets you describe how to put the pieces together.

AudioBIFS is a major piece of MPEG-4 that has designed for specifying the mixing and post-production of audio scenes as they’re played back. Using AudioBIFS, we can specify how the voice-track is mixed with the background music, and that it fades out after 10 seconds and this other music comes in and has a nice reverb on it.

BIFS is generally based on the Virtual Reality Modeling Language (VRML) (See Later in Course), but has extended capabilities for streaming and mixing audio and video data into a virtual-reality scene. The AudioBIFS functions are very advanced compared to VRML's sound model, which is rather simple, and are being tentatively considered for use in a future version of VRML.

In MPEG-4, AudioBIFS allows you to describe a sound as the combination of a number of sound objects. These sound objects may be coded using different coders (for example, CELP-coded voice and synthetic background music), and combined together in many ways. We can mix sounds together, or apply special filters and other processing functions written in SAOL.

Like the rest of BIFS, AudioBIFS is based on a scene graph. However, unlike in visual BIFS, the nodes in the AudioBIFS scene graph don't represent a bunch of objects which are presented to the user. Each AudioBIFS sound subgraph represents one sound object which is created by mixing and processing the elementary sound streams on which it is based.

For example, Fig 6.10 audio subgraph which shows how a simple sound is created from three elementary sound streams:

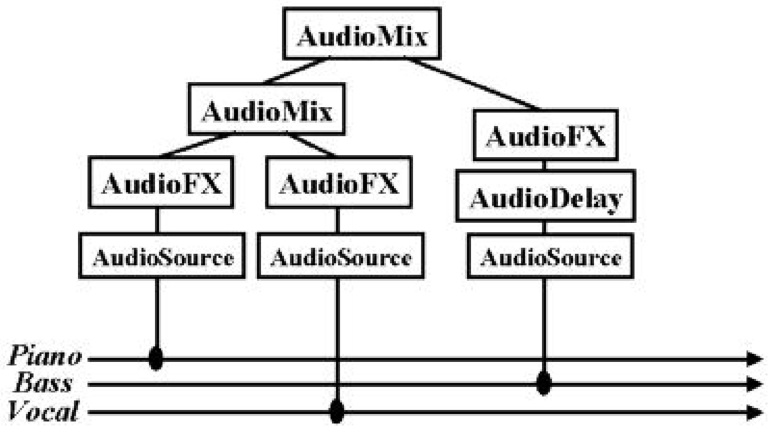


Figure 6.10: AudioBIFS Subgraph

Each of the rectangles show a node in the audio scene subgraph. Each node has a certain function, like mixing some sounds together, or delaying a sound, or doing some effects-processing. The arrows along the bottom represent the three elementary sound streams which make up the sound object. Each sound stream can be coded a different way. For example, we might code the piano sound with the Structured Audio decoder, the bass sound with the MPEG-4 Parametric HILN coder, and the vocal track with the MPEG-4 CELP coder.

These three sound streams are just like a "multitrack" recording of the final music sound object. The sound of each instrument is represented separately, then the scene graph mixes them all together. The processing in the audio subgraph is like a "data-flow" diagram. The sounds flow from the streams at the bottom, up through the nodes, and turn into a single sound at the top.

This single, final sound can be put into an audiovisual scene: it can be given a 3-D spatial location, moved around, and so on.

There's a page on AudioBIFS available at
http://www.risc.rockwell.com/349/343/MPEG4/

6.3.8 Further Reading/Information for Digital Audio and Midi

Some good texts on these areas include:

- A programmer's Guide to Sound, T. Kientzle, Addison Wesley, 1997 (ISBN 0-201-41972-6)
- Audio on the Web — The official IUMA Guide, Patterson and Melcher, Peachpit Press.
- The Art of Digital Audio, Watkinson, Focal/Butterworth-Heinmann.
- Synthesiser Basics, GPI Publications.
- Signal Processing: Principles and Applications, Brook and Wynne, Hodder and Stoughton.
- Digital Signal Processing, Oppenheim and Schafer, Prentice Hall.
- E. D. Scheirer, "Structured audio and effects processing in the MPEG-4 multimedia standard", ACM Multimedia Systems, in press.
- B. L. Vercoe, W. G. Gardner, E. D. Scheirer, "Structured Audio: Creation, transmission, and rendering of parametric sound representations", Proc. IEEE, in press.

Try some good sources for locating Digital Audio/internet sound/music materials at

- Digital Audio on the Web —
  http://www.interactive-media.com/presentations/AUDWEB2/ Audio file formats,
- Audio File Formats — *http://esewww.essex.ac.uk/ djmrob/filetypes.html*
- *http://www.harmony-central.com/MIDI/* — Harmony Central: MIDI Tools and Resources. Excellent resource. Full documentation of the MIDI specification, guides to making a MIDI interface, links to development tools, keyboard specific resources and more.
- *http://www.midifarm.com/* — MIDI Farm. Hub for MIDI on the Web. Includes their own HTML newsletter.

- *http://nuinfo.nwu.edu/musicschool/links/projects/midi/expmidiindex.html* — Exploring Midi: excellnt information resource on Midi and Audio.
- *http://www.midiweb.com/* — MIDIWeb. Resources for the MIDI community.
- *http://midiworld.com* — MIDIWorld
- *http://www.student.wau.nl/ olivier/midi/index.htm* — Best of MIDI, dedicated to the definitions and standards of MIDI and of course, MIDI files.
- *http://www.cabelov.com/midi/midi.shtml* — Charles Belov's MIDI Tips.
- *http://www2.iicm.edu/Cpub* — Gunter Nagler's MIDI Utilities. MIDI converters with C++ source code, executable for PC (MSDOS and Windows95) midi format conversion, error check, file repairs, etc.
- *http://www.eeb.ele.tue.nl/midi/intro.html* — Introduction to MIDI.
- *http://www.aitech.ac.jp./ ckelly/mmuig.html* — Macintosh MIDI User's Internet Guide¡
- *http://www.cs.ruu.nl/pub/MIDI/* — MIDI Archive,documentation, synth patches, MIDI programmes and utilities, information on MIDI and much more.
- *http://www.serve.com/papisi/* — MIDI Connection, links and mailing lists concerning sounds and information files for synths and samplers.
- *http://www.quicknet.se/home/q-112005/dan.htm* — MIDI Music Web Site. Files, glossary, explanation of MIDI, tools, shareware, MPEG 3 information and much more.
- *http://www.borg.com/ jglatt/* — MIDI Technical Fanatic's Brainwashing Center. Tutorials about MIDI and audio, technical/programming info, files, software, etc.
- *http://www.cabelov.com/midi/miditipw.shtml* — MIDI Tips for Webmeisters: explains the realities of embedding MIDI files in a web page.
- *http://www.midizone.com* — MIDI Zone: Home of MidiSeek, a midi search engine. Also includes lot of Information and midi files.
- *http://user.chollian.net/ moolly/midi00.htm* — Moolly's MIDI Page: Explains what MIDI is, techniques, connecting MIDI to a PC, and more.

MPEG 4 Pages:

- *http://sound.media.mit.edu/mpeg4/audio/* — Main MPEG 4 Audio Page
- *http://sound.media.mit.edu/ eds/mpeg4/* — MPEG 4 Home Page

- *http://sound.media.mit.edu/mpeg4-old/* — MPEG FAQ's
- *http://sound.media.mit.edu/mpeg4/links.html* — MPEG 4 Useful Links
- *http://153.96.172.2/amm/techinf/mpeg4/scalable.html* — MPEG 4 Page

6.4 Graphic/Image File Formats

This section introduces some of the most common graphics and image file formats. Some of them are restricted to particular hardware/operating system platforms, others are *cross-platform* independent formats. While not all formats are cross-platform, there are conversion applications that will recognize and translate formats from other systems.

The document (*http://www.cica.indiana.edu/graphics/image.formats.html*) by CICA at Indiana Univ. provides a fairly comprehensive listing of various formats.

Most image formats incorporate some variation of a *compression* technique due to the large storage size of image files. Compression techniques can be classified into either **lossless** or **lossy**. We will study various video and audio compression techniques in the Next Chapter.

6.4.1 Graphic/Image Data Structures

A digital image consists of many picture elements, termed **pixels**. The number of pixels that compose a monitor image determine the quality of the image (**resolution**). Higher resolution always yields better quality.

A *bit-map* representation stores the graphic/image data in the same manner that the computer monitor contents are stored in video memory.

Monochrome/Bit-Map Images

An example 1 bit monochrome image is illustrated in Fig. 6.11 where:

- Each pixel is stored as a single bit (0 or 1)
- A 640 x 480 monochrome image requires 37.5 KB of storage.
- *Dithering* is often used for displaying monochrome images

Gray-scale Images

An example gray-scale image is illustrated in Fig. 6.12 where:

- Each pixel is usually stored as a byte (value between 0 to 255)
- A 640 x 480 greyscale image requires over 300 KB of storage.

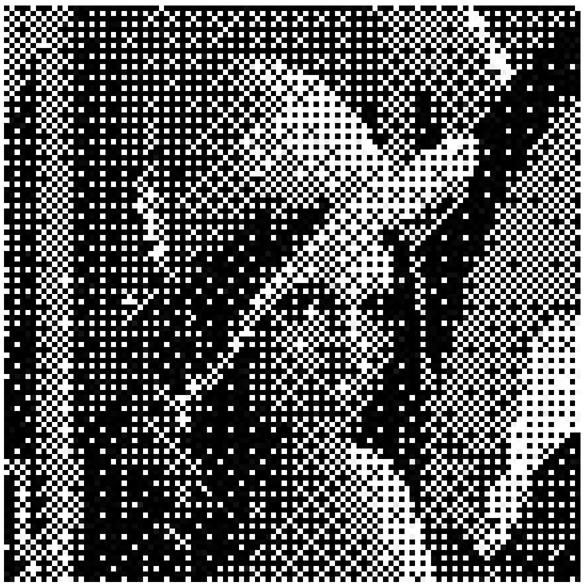


Figure 6.11: Sample Monochrome Bit-Map Image

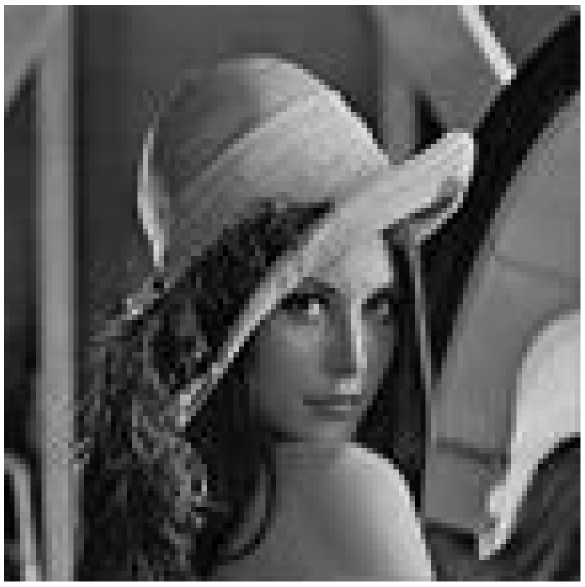


Figure 6.12: Example of a Gray-scale Bit-map Image

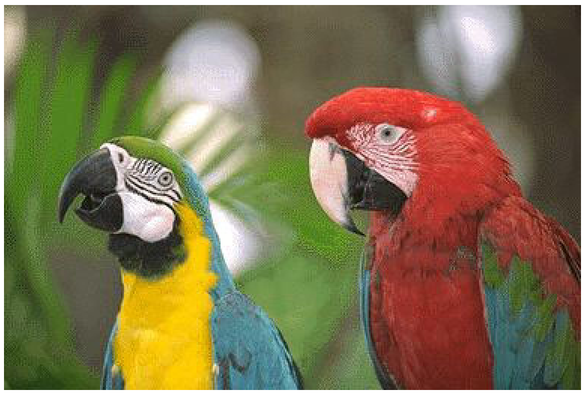


Figure 6.13: Example of 8-Bit Colour Image

8-bit Colour Images

An example 8-bit colour image is illustrated in Fig. 6.13 where:

- One byte for each pixel
- Supports 256 out of the millions s possible, acceptable colour quality
- Requires Colour Look-Up Tables (LUTs)
- A 640 x 480 8-bit colour image requires 307.2 KB of storage (the same as 8-bit greyscale)

24-bit Colour Images

An example 24-bit colour image is illustrated in Fig. 6.14 where:

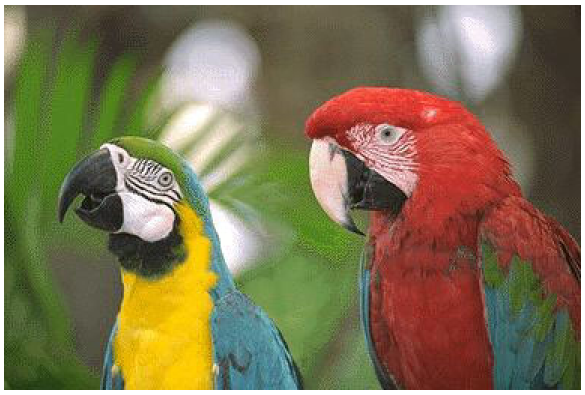


Figure 6.14: Example of 24-Bit Colour Image

- Each pixel is represented by three bytes (e.g., RGB)
- Supports 256 x 256 x 256 possible combined colours (16,777,216)
- A 640 x 480 24-bit colour image would require 921.6 KB of storage
- Most 24-bit images are 32-bit images, the extra byte of data for each pixel is used to store an *alpha* value representing special effect information

6.4.2 Standard System Independent Formats

The following brief format descriptions are the most commonly used formats. Follow some of the document links for more descriptions.

GIF (GIF87a, GIF89a)

- Graphics Interchange Format (GIF) devised by the UNISYS Corp. and Compuserve, initially for transmitting graphical images over phone lines via modems
- Uses the Lempel-Ziv Welch algorithm (a form of Huffman Coding), modified slightly for image scan line packets (line grouping of pixels)
- Limited to only 8-bit (256) colour images, suitable for images with few distinctive colours (e.g., graphics drawing)
- Supports *interlacing*

JPEG

- A standard for photographic image compression created by the Joint Photographics Experts Group
- Takes advantage of limitations in the human vision system to achieve high rates of compression
- Lossy compression which allows user to set the desired level of quality/compression
- Detailed discussions in next chapter on compression.

TIFF

- Tagged Image File Format (TIFF), stores many different types of images (e.g., monochrome, greyscale, 8-bit & 24-bit RGB, etc.) –> tagged
- Developed by the Aldus Corp. in the 1980's and later supported by the Microsoft
- TIFF is a lossless format (when not utilizing the new JPEG tag which allows for JPEG compression)

- It does not provide any major advantages over JPEG and is not as user-controllable it appears to be declining in popularity

Graphics Animation Files

- FLC – main animation or moving picture file format, originally created by Animation Pro
- FLI – similar to FLC
- GL – better quality moving pictures, usually large file sizes

Postscript/Encapsulated Postscript

- A typesetting language which includes text as well as vector/structured graphics and bit-mapped images
- Used in several popular graphics programs (Illustrator, FreeHand)
- Does not provide compression, files are often large

6.4.3 System Dependent Formats

Many graphical/imaging applications create their own file format particular to the systems they are executed upon. The following are a few popular system dependent formats:

Microsoft Windows: BMP

- A system standard graphics file format for Microsoft Windows
- Used in PC Paintbrush and other programs
- It is capable of storing 24-bit bitmap images

Macintosh: PAINT and PICT

- PAINT was originally used in MacPaint program, initially only for 1-bit monochrome images.
- PICT format is used in MacDraw (a vector based drawing program) for storing structured graphics

X-windows: XBM

- Primary graphics format for the X Window system
- Supports 24-bit colour bitmap
- Many public domain graphic editors, e.g., *xv*
- Used in X Windows for storing icons, pixmaps, backdrops, etc.

6.4.4 Further Reading/Information

Intro. to Computer Pictures, http://ac.dal.ca:80/ dong/image.htm from Allison Zhang at the School of Library and Information Studies, Dalhousie University, Halifax, N.S., Canada

http://www.cica.indiana.edu/graphics/image.formats.html contains a comprehensive list of various graphics/image file formats.

An complete reference text on the topic is the *Encyclopedia of Graphics File Formats*, Second Edition by James D. Murray and William vanRyper, 1996, O'Reilly & Associates.

6.5 Colour in Image and Video

6.5.1 Basics of Colour

Light and Spectra

- Visible light is an electromagnetic wave in the 400nm - 700 nm range.
- Most light we see is not one wavelength, it's a combination of many wavelengths (Fig. 6.15).

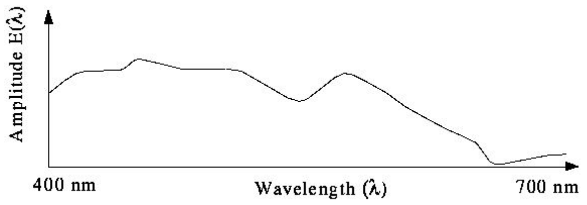


Figure 6.15: Light Wavelengths

- The profile above is called a *spectra.*

The Human Retina

- The eye is basically just a camera
- Each neuron is either a *rod* or a *cone.* Rods are not sensitive to colour.

Cones and Perception

- Cones come in 3 types: red, green and blue. Each responds differently to various frequencies of light. The following figure shows the spectral-response functions of the cones and the luminous-efficiency function of the human eye (Fig. 6.16.
- The profile above is called a *spectra.*

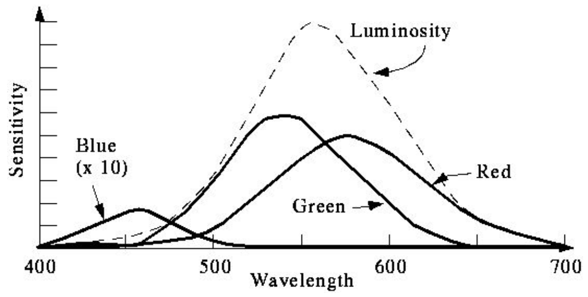


Figure 6.16: Cones and Luminous-efficiency Function of the Human Eye

- The colour signal to the brain comes from the response of the 3 cones to the spectra being observed (Fig 6.17). That is, the signal consists of 3 numbers:

$$R = \int E(\lambda)S_R(\lambda)d\lambda$$
$$G = \int E(\lambda)S_G(\lambda)d\lambda$$
$$B = \int E(\lambda)S_B(\lambda)d\lambda$$

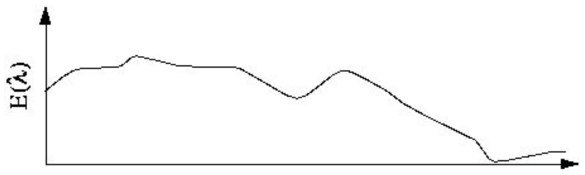


Figure 6.17: Spectra Response

where E is the light and S are the sensitivity functions.

- A colour can be specified as the sum of three colours. So colours form a 3 dimensional vector space.
- The following figure shows the amounts of three primaries needed to match all the wavelengths of the visible spectrum (Fig. refspectrum).
- The negative value indicates that some colours cannot be exactly produced by adding up the primaries.

6.5.2 CIE Chromaticity Diagram

Does a set of primaries exist that span the space with only positive coefficients?

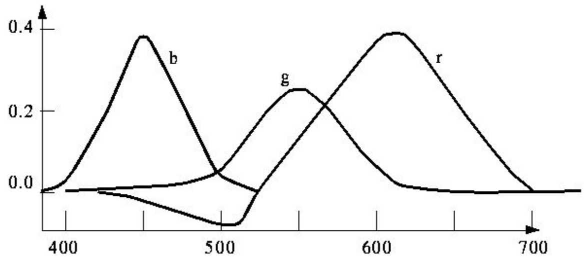


Figure 6.18: Wavelengths of the Visible Spectrum

- Yes, but no pure colours.
- In 1931, the CIE defined three standard primaries **(X, Y, Z)** . The **Y** primary was intentionally chosen to be identical to the luminous-efficiency function of human eyes.

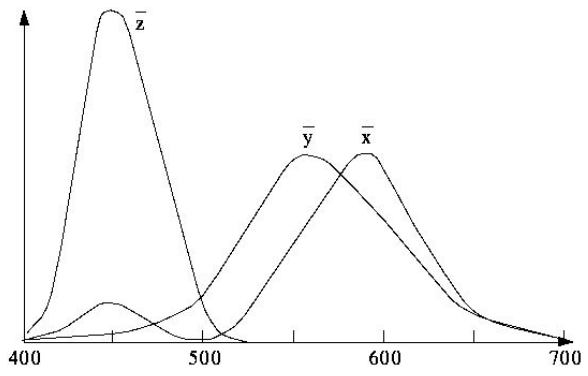


Figure 6.19: Reproducing Visible Colour

- Figure 6.19 shows the amounts of X, Y, Z needed to exactly reproduce any visible colour via the formulae:

$$X = \int E(\lambda)\overline{x}(\lambda)d\lambda$$
$$Y = \int E(\lambda)\overline{y}(\lambda)d\lambda$$
$$Z = \int E(\lambda)\overline{z}(\lambda)d\lambda$$

- All visible colours are in a *horseshoe* shaped cone in the X-Y-Z space. Consider the plane *X+Y+Z=1* and project it onto the X-Y plane, we get the *CIE chromaticity diagram* as shown in Fig. 6.20.

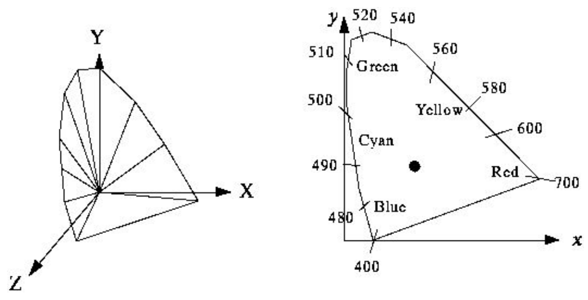


Figure 6.20: CIE Chromaticity Diagram

- The edges represent the *pure* colours (sine waves at the appropriate frequency)
- White (a blackbody radiating at 6447 kelvin) is at the *dot*
- When added, any two colours (points on the CIE diagram) produce a point on the line between them.

L*a*b (Lab) Colour Model

- A refined CIE model, named CIE L*a*b in 1976
- Luminance: L Chrominance: a – ranges from green to red, b – ranges from blue to yellow (Fig, 6.21)
- Used by *Photoshop*

CRT Displays

- CRT displays have three phosphors (RGB) which produce a combination of wavelengths when excited with electrons (Fig. 6.22).
- The *gamut* of colours is all colours that can be reproduced using the three primaries
- The gamut of an colour monitor is smaller than the CIE (LAB) colour gamut on the CIE diagram.

6.5.3 Colour Image and Video Representations

- A black and white image is a 2-D array of integers.
- A colour image is a 2-D array of (R,G,B) integer triplets. These triplets encode how much the corresponding phosphor should be excited in devices such as a monitor.
- Example is shown in Fig 6.23.

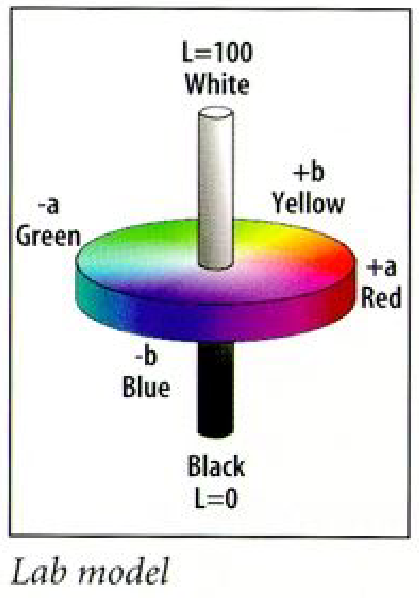


Figure 6.21: Lab Colour Model

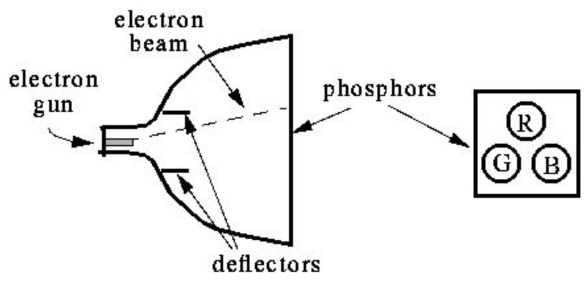


Figure 6.22: RGB Colour Display

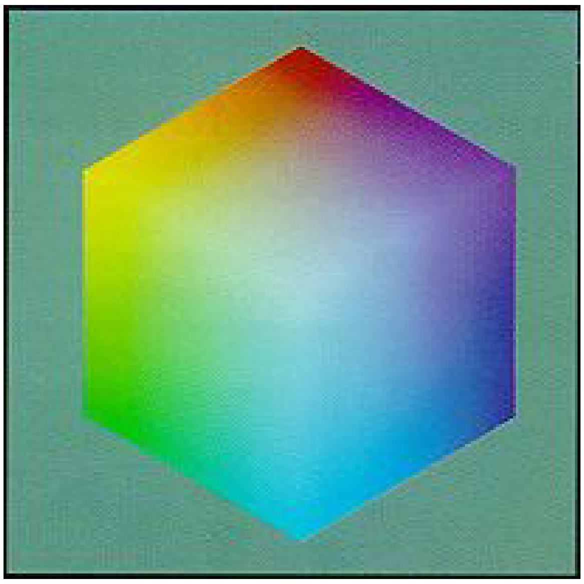


Figure 6.23: Display of a Colour Cube

Beside the RGB representation, YIQ and YUV are the two commonly used in video.

YIQ Colour Space

- YIQ is used in colour TV broadcasting, it is downward compatible with B/W TV.
- Y (luminance) is the CIE Y primary.

  Y = 0.299R + 0.587G + 0.114B
- the other two vectors:

  I = 0.596R - 0.275G - 0.321B Q = 0.212R - 0.528G + 0.311B
- The YIQ transform:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & -0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- I is red-orange axis, Q is roughly orthogonal to I.
- Eye is most sensitive to Y, next to I, next to Q. In NTSC, 4 MHz is allocated to Y, 1.5 MHz to I, 0.6 MHz to Q.

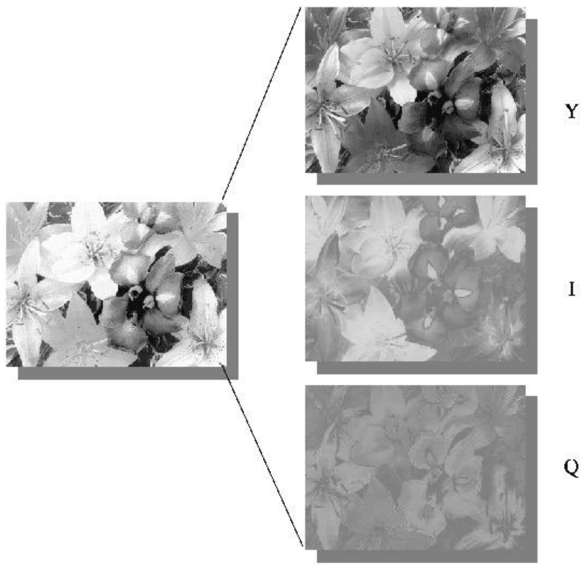


Figure 6.24: Example YIQ Decomposition

- An Example YIQ Decomposition is shown in Fig. 6.24.

CCIR 601 (YUV)

- Established in 1982 to build digital video standard
- Video is represented by a sequence of fields (odd and even lines). Two fields make a frame.
- Works in PAL (50 fields/sec) or NTSC (60 fields/sec)
- Uses the Y, Cr, Cb colour space (also called YUV) *Y = 0.299R + 0.587G + 0.114B Cr = R - Y Cb = B - Y*
- The YCrCb (YUV) Transform:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- CCIR 601 also defines other image parameters, e.g. for NTSC, Luminance (Y) image size = 720 x 243 at 60 fields per second Chrominance image size = 360 x 243 at 60 fields per second
- An example YCrCb Decomposition is shown in Fig. 6.25

The CMY Colour Model

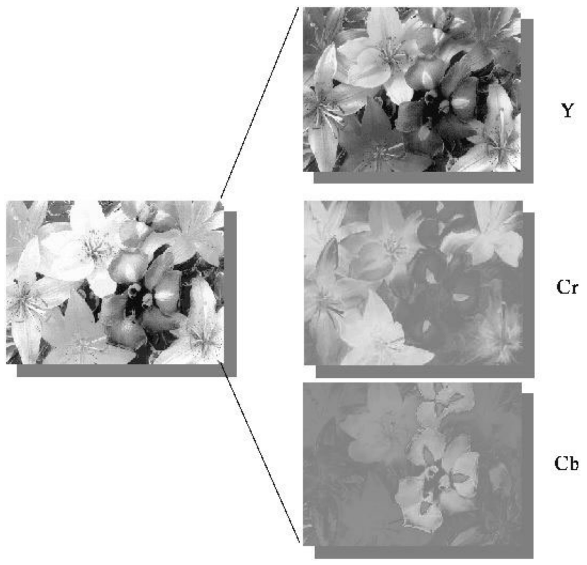


Figure 6.25: YCrCb Decomposition of a colour image

- Cyan, Magenta, and Yellow (CMY) are complementary colours of RGB (Fig. 6.26). They can be used as *Subtractive Primaries.*
- CMY model is mostly used in printing devices where the colour pigments on the paper absorb certain colours (e.g., no red light reflected from cyan ink).

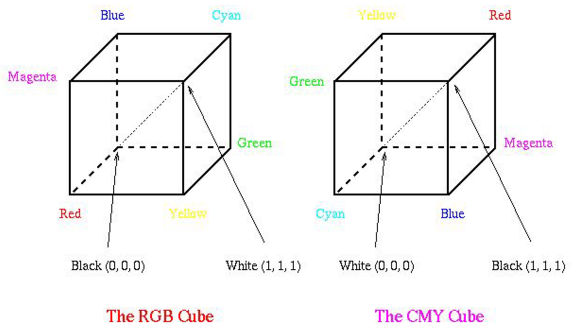


Figure 6.26: The RGB and CMY Cubes

Conversion between RGB and CMY: – e.g., convert **White** from (1, 1, 1) in RGB to (0, 0, 0) in CMY.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

- Sometimes, an alternative CMYK model (K stands for *Black*) is used in colour printing (e.g., to produce darker black than simply mixing CMY). where

$$\begin{aligned} K &= min(C, M, Y), \\ C &= C - K, \\ M &= M - K, \\ Y &= Y - K. \end{aligned}$$

6.5.4 Summary of Colour

- Colour images are encoded as triplets of values.
- Three common systems of encoding in video are RGB, YIQ, and YCrCb.
- Besides the hardware-oriented colour models (i.e., RGB, CMY, YIQ, YUV), HSB (Hue, Saturation, and Brightness, e.g., used in Photoshop) and HLS (Hue, Lightness, and Saturation) are also commonly used.
- YIQ uses properties of the human eye to prioritize information. Y is the black and white (luminance) image, I and Q are the colour (chrominance) images. YUV uses similar idea.
- CCIR 601 is a standard for digital video that specifies image size, and decimates the chrominance images (for 4:2:2 video).

6.6 Basics of Video

6.6.1 Types of Colour Video Signals

- **Component video** – each primary is sent as a separate video signal.
  - The primaries can either be RGB or a luminance-chrominance transformation of them (e.g., YIQ, YUV).
  - Best colour reproduction
  - Requires more bandwidth and good synchronization of the three components

- **Composite video** – colour (chrominance) and luminance signals are mixed into a single carrier wave. Some interference between the two signals is inevitable.
- **S-Video** (Separated video, e.g., in S-VHS) – a compromise between component analog video and the composite video. It uses two lines, one for luminance and another for composite chrominance signal.

6.6.2 Analog Video

The following figures (Fig. 6.27 and 6.28) are from A.M. Tekalp, *Digital video processing*, Prentice Hall PTR, 1995.

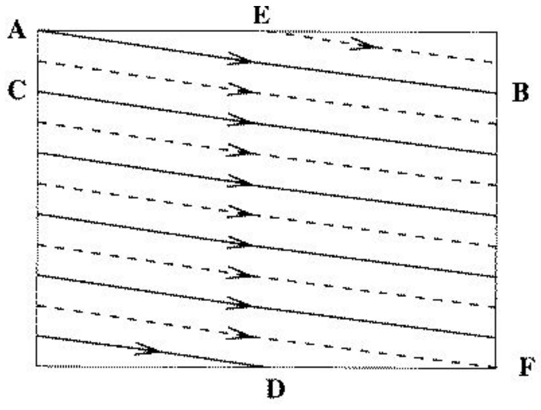


Figure 6.27: Raster Scanning

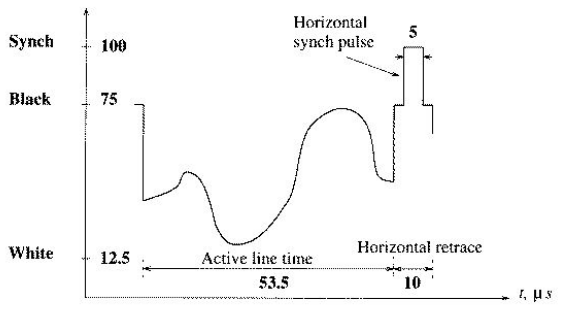


Figure 6.28: NTSC Signal

NTSC Video

- 525 scan lines per frame, 30 frames per second (or be exact, 29.97 fps, 33.37 msec/frame)

- Aspect ratio 4:3
- Interlaced, each frame is divided into 2 fields, 262.5 lines/field
- 20 lines reserved for control information at the beginning of each field (Fig. 6.29)
  - So a maximum of 485 lines of visible data
  - Laserdisc and S-VHS have actual resolution of $\tilde{4}$20 lines
  - Ordinary TV – $\tilde{3}$20 lines
- Each line takes 63.5 microseconds to scan. Horizontal retrace takes 10 microseconds (with 5 microseconds horizontal synch pulse embedded), so the active line time is 53.5 microseconds.

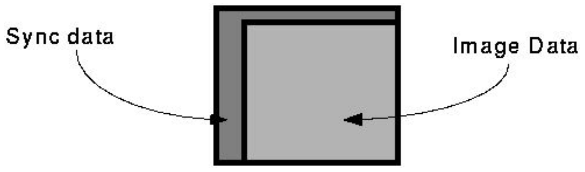


Figure 6.29: Digital Video Rasters

- Colour representation:
  - NTSC uses YIQ colour model.
  - composite = Y + I *cos*(Fsc t) + Q *sin*(Fsc t), where Fsc is the frequency of colour subcarrier
  - Eye is most sensitive to Y, next to I, next to Q. In NTSC, 4 MHz is allocated to Y, 1.5 MHz to I, 0.6 MHz to Q.

PAL Video

- 625 scan lines per frame, 25 frames per second (40 msec/frame)
- Aspect ratio 4:3
- Interlaced, each frame is divided into 2 fields, 312.5 lines/field
- Colour representation:
  - PAL uses YUV (YCbCr) colour model
  - composite = Y + 0.492 x U *sin*(Fsc t) + 0.877 x V *cos*(Fsc t)
  - In component analog video, U and V signals are lowpass filtered to about half the bandwidth of Y.

6.6.3 Digital Video

- **Advantages:**
  - Direct random access –> good for nonlinear video editing
  - No problem for repeated recording
  - No need for blanking and sync pulse
- Almost all digital video uses component video

6.6.4 Chroma Subsampling

- How to decimate for chrominance (Fig. 6.30)?

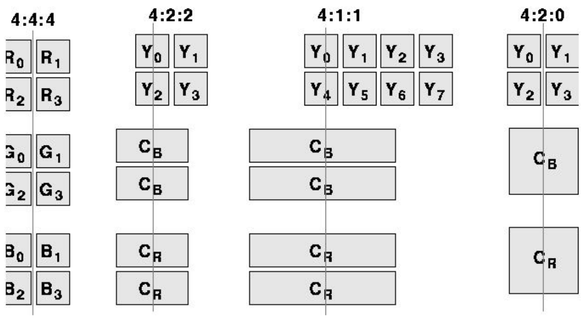


Figure 6.30: Chroma Subsampling

- 4:2:2 –> Horizontally subsampled colour signals by a factor of 2. Each pixel is two bytes, e.g., (Cb0, Y0)(Cr0, Y1)(Cb2, Y2)(Cr2, Y3)(Cb4, Y4) ...
- 4:1:1 –> Horizontally subsampled by a factor of 4
- 4:2:0 –> Subsampled in both the horizontal and vertical axes by a factor of 2 between pixels as shown in the Fig. 6.30.
- 4:1:1 and 4:2:0 are mostly used in JPEG and MPEG (see Chapter 4).

CCIR Standards for Digital Video

(CCIR – Consultative Committee for International Radio)

	CCIR 601 525/60 NTSC	CCIR 601 625/50 PAL/SECAM	CIF NTSC	QCIF
Luminance resolution	720 x 485	720 x 576	352 x 240	176 x 120
Chrominance resolut.	360 x 485	360 x 576	176 x 120	88 x 60

Colour Subsampling	4:2:2	4:2:2		
Fields/sec	60	50	30	30
Interlacing	Yes	Yes	No	No

- CCIR 601 uses interlaced scan, so each field only has half as much vertical resolution (e.g., 243 lines in NTSC). The CCIR 601 (NTSC) data rate is 1̃65 Mbps.
- CIF (Common Intermediate Format) is introduced to as an acceptable temporary standard. It delivers about the VHS quality. CIF uses progressive (non-interlaced) scan.

ATSC Digital Television Standard

(ATSC – Advanced Television Systems Committee) The **ATSC Digital Television Standard** was recommended to be adopted as the Advanced TV broadcasting standard by the FCC Advisory Committee on Advanced Television Service on November 28, 1995. It covers the standard for **HDTV** (High Definition TV).

Video Format

The video scanning formats supported by the ATSC Digital Television Standard are shown in the following table.

Vertical Lines	Horizontal Pixels	Aspect Ratio	Picture Rate
1080	920	16:9	60I 30P 24P
720	1280	16:9	60P 30P 24P
480	704	16:9 and 4:3	60I 60P 30P 24P
480	640	4:3	60I 60P 30P 24P

- The aspect ratio for HDTV is 16:9 as opposed to 4:3 in NTSC, PAL, and SECAM. (A 33% increase in horizontal dimension.)
- In the picture rate column, the &quot;I&quot; means interlaced scan, and the &quot;P&quot; means progressive (non-interlaced) scan.
- Both NTSC rates and integer rates are supported (i.e., 60.00, 59.94, 30.00, 29.97, 24.00, and 23.98).

6.6.5 Further Reading/Information

A good text on this area is:

Digital video processing, A.M. Tekalp, Prentice Hall PTR, 1995.

Interesting Web sites include:

http://www.atsc.org/ — Homepage of the Advanced Television Systems Committee (ATSC)

http://www.HDTV.net — HDTV Web Page.

http://www.prz.tu-berlin.de/ joe/mheg/mheg_mon/index.htm — MHEG Home Page

Chapter 7

Compression I: Basic Compression Algorithms

Video and Audio files are very large beasts. Unless we develop and maintain very high bandwidth networks (Gigabytes per second or more) we have to compress to data.

Relying on higher bandwidths is not a good option — M25 Syndrome: Traffic needs ever increases and will adapt to swamp current limit whatever this is.

As we will compression becomes part of the representation or *coding* scheme which have become popular audio, image and video formats.

We will first study basic compression algorithms and then go on to study some actual coding formats for Images (JPEG), Audio and Video (MPEG etc.) in following Chapters.

7.1 Classifying Compression Algorithms

We can classify compression by the why it employs redundancy or by the method it compresses the data.

7.1.1 What is Compression?

Compression basically employs redundancy in the data:

- Temporal — in 1D data, 1D signals, Audio etc.
- Spatial — correlation between neighbouring pixels or data items
- Spectral — correlation between colour or luminescence components. This uses the frequency domain to exploit relationships between frequency of change in data.
- psycho-visual — exploit perceptual properties of the human visual system.

Compression can be categorised in two broad ways:

Lossless Compression — where data is compressed and can be reconstituted (uncompressed) without loss of detail or information. These are referred to as bit-preserving or reversible compression systems also.

Lossy Compression — where the aim is to obtain the best possible *fidelity* for a given bit-rate or minimizing the bit-rate to achieve a given fidelity measure. Video and audio compression techniques are most suited to this form of compression.

If an image is compressed it clearly needs to uncompressed (decoded) before it can viewed/listened to. Some processing of data may be possible in encoded form however.

Lossless compression frequently involves some form of *entropy encoding* and are based in information theoretic techniques (Fig. 7.1)

Lossy compression use source encoding techniques that may involve transform encoding, differential encoding or vector quantisation (Fig. 7.1).

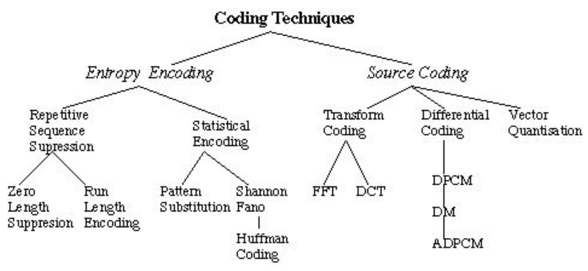


Figure 7.1: Classification of Coding Techniques

We now address common coding methods of each type in turn:

7.2 Lossless Compression Algorithms (Repetitive Sequence Suppression)

These methods are fairly straight forward to understand and implement. Their simplicity is their downfall in terms of attaining the best compression ratios. However, the methods have their applications, as mentioned below:

7.2.1 Simple Repetition Suppression

If in a sequence a series on n successive tokens appears we can replace these with a token and a count number of occurrences. We usually need to have a special *flag* to denote when the repeated token appears

For Example

```
894000000000000000000000000000000
```

we can replace with
`894f32`
where `f` is the flag for zero.
Compression savings depend on the content of the data.
Applications of this simple compression technique include:

- Suppression of zero's in a file (*Zero Length Suppression*)
    - Silence in audio data, Pauses in conversation *etc.*
    - Bitmaps
    - Blanks in text or program source files
    - Backgrounds in images
- other regular image or data tokens

7.2.2 Run-length Encoding

This encoding method is frequently applied to images (or pixels in a scan line). It is a small compression component used in JPEG compression (Section 8).

In this instance, sequences of image elements $X_1, X_2, \ldots, X_n$ are mapped to pairs $(c_1, l_1), (c_2, l_2), \ldots, (c_n, l_n)$ where c_i represent image intensity or colour and l_i the length of the ith run of pixels (Not dissimilar to zero length suppression above).

For example:
Original Sequence:
`111122233333311112222`
can be encoded as:
`(1,4),(2,3),(3,6),(1,4),(2,4)`

The savings are dependent on the data. In the worst case (Random Noise) encoding is more heavy than original file: 2*integer rather 1* integer if data is represented as integers.

7.3 Lossless Compression Algorithms (Pattern Substitution)

This is a simple form of statistical encoding.

Here we substitute a frequently repeating pattern(s) with a code. The code is shorter than than pattern giving us compression.

A simple Pattern Substitution scheme could employ predefined code (for example replace all occurrences of 'The' with the code '&').

More typically tokens are assigned to according to frequency of occurrence of patterns:

- Count occurrence of tokens

- Sort in Descending order
- Assign some symbols to highest count tokens

A predefined symbol table may used *i.e.* assign code i to token i.

However, it is more usual to dynamically assign codes to tokens. The entropy encoding schemes below basically attempt to decide the optimum assignment of codes to achieve the best compression.

7.4 Lossless Compression Algorithms (Entropy Encoding)

Lossless compression frequently involves some form of *entropy encoding* and are based in information theoretic techniques, Shannon is father of information theory and we briefly summarise information theory below before looking at specific entropy encoding methods.

7.4.1 Basics of Information Theory

According to Shannon, the entropy of an information source S is defined as:

$$H(S) = \eta = \sum_i \; p_i \log_2 \frac{1}{p_i}$$

where p_i is the probability that symbol S_i in S will occur.

- $\log_2 \frac{1}{p_i}$ indicates the amount of information contained in S_i, i.e., the number of bits needed to code S_i.
- For example, in an image with uniform distribution of gray-level intensity, i.e. $p_i = 1/256$, then the number of bits needed to code each gray level is 8 bits. The entropy of this image is 8.
- Q: How about an image in which half of the pixels are white (I = 220) and half are black (I = 10)?

7.4.2 The Shannon-Fano Algorithm

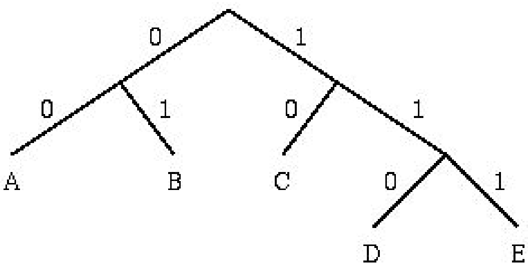
This is a basic information theoretic algorithm. A simple example will be used to illustrate the algorithm:

```
Symbol     A    B    C    D    E
----------------------------------
Count      15   7    6    6    5
```

Encoding for the Shannon-Fano Algorithm:

- A top-down approach

1. Sort symbols according to their frequencies/probabilities, e.g., ABCDE.
2. Recursively divide into two parts, each with approx. same number of counts.



Symbol	Count	log(1/p)	Code	Subtotal (# of bits)
A	15	1.38	00	30
B	7	2.48	01	14
C	6	2.70	10	12
D	6	2.70	110	18
E	5	2.96	111	15
			TOTAL (# of bits):	89

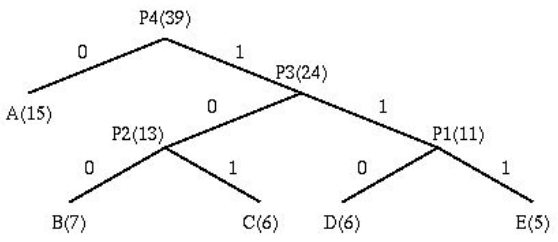
7.4.3 Huffman Coding

Huffman coding is based on the frequency of occurrence of a data item (pixel in images). The principle is to use a lower number of bits to encode the data that occurs more frequently. Codes are stored in a *Code Book* which may be constructed for each image or a set of images. In all cases the code book plus encoded data must be transmitted to enable decoding.

The Huffman algorithm is now briefly summarised:

- A bottom-up approach

1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times (e.g., ABCDE).
2. Repeat until the OPEN list has only one node left:
(a) From OPEN pick two nodes having the lowest frequencies/probabilities, create a parent node of them.
(b) Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.
(c) Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.



Symbol	Count	log(1/p)	Code	Subtotal (# of bits)
A	15	1.38	0	15
B	7	2.48	100	21
C	6	2.70	101	18
D	6	2.70	110	18
E	5	2.96	111	15
			TOTAL (# of bits):	87

The following points are worth noting about the above algorithm:

- Decoding for the above two algorithms is trivial as long as the coding table (the statistics) is sent before the data. (There is a bit overhead for sending this, negligible if the data file is big.)
- **Unique Prefix Property**: no code is a prefix to any other code (all symbols are at the leaf nodes) –> great for decoder, unambiguous.
- If prior statistics are available and accurate, then Huffman coding is very good.

In the above example:

$$\begin{aligned} \text{entropy} &= (15*1.38+7*2.48+6*2.7 \\ &\quad +6*2.7+5*2.96)/39 \\ &= 85.26/39 \\ &= 2.19 \end{aligned}$$

Number of bits needed for Huffman Coding is: 87/39 = 2.23

7.4.4 Huffman Coding of Images

In order to encode images:

- Divide image up into 8x8 blocks
- Each block is a symbol to be coded
- compute Huffman codes for set of block
- Encode blocks accordingly

7.4.5 Adaptive Huffman Coding

The basic Huffman algorithm has been extended, for the following reasons:

(a) The previous algorithms require the statistical knowledge which is often not available (e.g., live audio, video).

(b) Even when it is available, it could be a heavy overhead especially when many tables had to be sent when a non-order0 model is used, i.e. taking into account the impact of the previous symbol to the probability of the current symbol (e.g., "qu" often come together, ...).

The solution is to use adaptive algorithms. As an example, the Adaptive Huffman Coding is examined below. The idea is however applicable to other adaptive compression algorithms.

```
ENCODER                             DECODER
-------                             -------

Initialize_model();                 Initialize_model();
while ((c = getc (input)) != eof)   while ((c = decode (input)) != eof)
  {                                   {
    encode (c, output);                 putc (c, output);
    update_model (c);                   update_model (c);
  }                                   }

}
```

- The key is to have both encoder and decoder to use exactly the same *initialization* and *update_model* routines.
- *update_model* does two things: (a) increment the count, (b) update the Huffman tree (Fig 7.2).
  - During the updates, the Huffman tree will be maintained its *sibling property*, i.e. the nodes (internal and leaf) are arranged in order of increasing weights (see figure).
  - When *swapping* is necessary, the farthest node with weight W is swapped with the node whose weight has just been increased to W+1. **Note:** If the node with weight W has a subtree beneath it, then the subtree will go with it.
  - The Huffman tree could look very different after node swapping (Fig 7.2), e.g., in the third tree, node A is again swapped and becomes the #5 node. It is now encoded using only 2 bits.

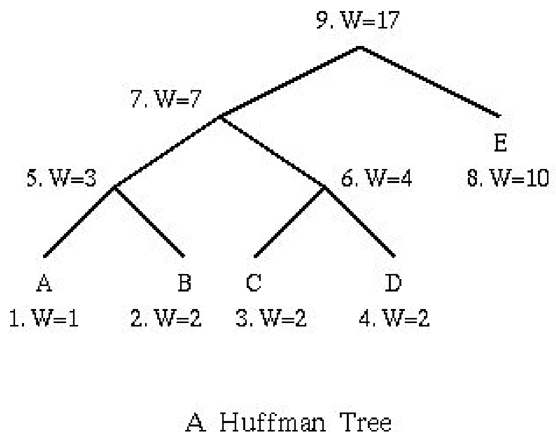
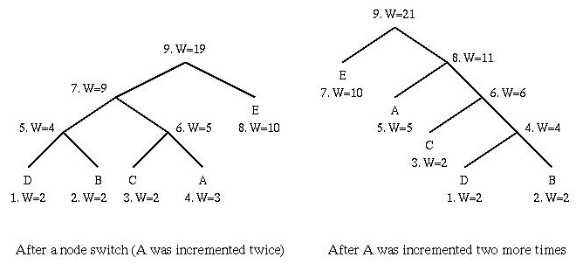


Figure 7.2: Huffman Tree



Note: Code for a particular symbol changes during the adaptive coding process.

7.4.6 Arithmetic Coding

Arithmetic coding, is a widely used entropy coder (see JPEG, Chapter 8). The only problem is it's speed due possibly complex computations due to large symbol tables, but compression tends to be better than Huffman can achieve, with entropy around the Shannon value.

Huffman coding and the like use an integer number (k) of bits for each symbol, hence k is never less than 1. Sometimes, e.g., when sending a 1-bit image, compression becomes impossible.

Here we will just discuss the basic method of arithmetic coding. Some more efficient algorithms exist.

Decimal Static Arithmetic Coding

We will first discuss the basic approach in relation to **decimal coding** and also to the basic static coding mode of operation.

The idea behind arithmetic coding is

- to have a probability line, 0–1, and
- assign to every symbol a range in this line based on its probability,
- the higher the probability, the higher range which assigns to it.

Once we have defined the ranges and the probability line,

- start to encode symbols,
- every symbol defines where the output floating point number lands within the range.

Lets consider an example to explain how basic arithmetic coding works.
Assume we have the following token symbol stream
`BACA`
Then the symbol A occurs twice and B and C once. Therefore A occurs with probability 0.5, and B and C with probabilities 0.25.
So start by assigning each symbol to the probability range 0–1.

- Sort symbols highest probability first

Symbol	Range
`A`	[0.0, 0.5)
`B`	[0.5, 0.75)
`C`	[0.75, 1.0)

The square bracket, "[", means that the number is also included, so all the numbers from 0.0 to 0.499999... belong to `A`.
The first symbol in our stream is `B` we now know that the code will be in the range 0.5 to 0.74999....
What need to do now is narrow down the range to give us a unique code.
To do this we have to subdivide the range for the first token given the probabilities of the second token then the third etc.
To recompute the range we simply subdivide the range as follows:
For all the symbols

- Range = high - low
- High = low + range * high_range of the symbol being coded
- Low = low + range * low_range of the symbol being coded

Where:

- Range, keeps track of where the next range should be.
- High and low, specify the output number.
- Initially High = 1.0, Low = 0.0

So for the second symbols we have (now Range = 0.25, Low = 0.5, High = 0.75):

Symbol	Range
B**A**	[0.5, 0.625)
BB	[0.625, 0.6875)
BC	[0.6875, 0.75)

The second symbol in our sequence is `A` so we now know that the code is in the range 0.5 to 0.6249....

We now reapply the subdivision of our scale again to get for our third symbol (Range = 0.125, Low = 0.5, High = 0.625):

Symbol	Range
BAA	[0.5, 0.5625)
BAB	[0.5625, 0.59375)
BA**C**	[0.59375, 0.625)

The third symbol in our sequence is `C` so we now know that the code is in the range 0.59375 to 0.6249...

Subdivide again (Range = 0.03125, Low = 0.59375, High = 0.625):

Symbol	Range
BAC**A**	[0.59375, 0.60937)
BACB	[0.609375, 0.6171875)
BACC	[0.6171875, 0.625)

So the output code for `BACA` is any number in the range [0.59375, 0.60937).

To **decode** is essentially the opposite

- We compile the table for the sequence given probabilities.
- Find the range of number within which the code number lies and carry on

Binary static algorithmic coding

This is very similar except we us binary fractions.

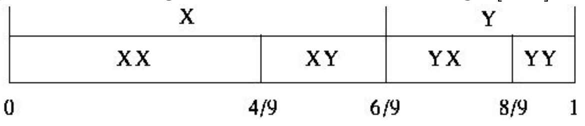
Binary fractions are simply an extension of the binary systems into fractions much like decimal fractions. That is to say:

0.1 decimal = $\frac{1}{10^1}$ = 1/10
0.01 decimal = $\frac{1}{10^2}$ = 1/100
0.11 decimal = $\frac{1}{10^1} + \frac{1}{10^2}$ = 11/100
So in binary we get
0.1 binary = $\frac{1}{2^1}$ = 1/2 decimal
0.01 binary = $\frac{1}{2^2}$ = 1/4 decimal
0.11 binary = $\frac{1}{2^1} + \frac{1}{2^2}$ = 3/4 decimal
For example:

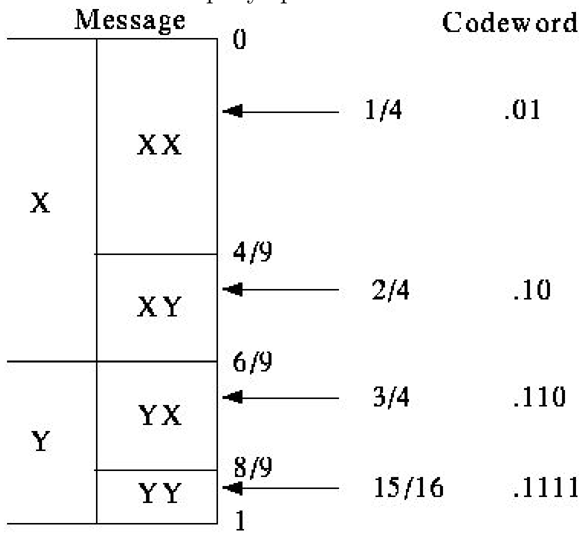
- Suppose alphabet was
$$X, Y$$
and

```
prob(X) = 2/3
prob(Y) = 1/3
```

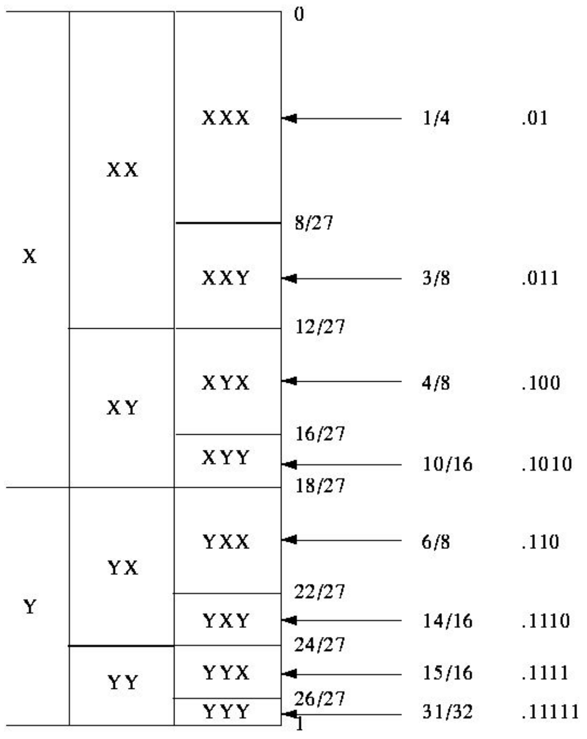
- If we are only concerned with encoding length 2 messages, then we can map all possible messages to intervals in the range [0..1] as before:



- With binary coding the trick is to encode message to send enough bits of a binary fraction that uniquely specifies the interval.



- Similarly, we can map all possible length 3 messages to intervals in the range [0..1]:



Implementation Issues

FPU Precision

In practice the resolution of the of the number we represent is limited by the precision of the computers FPU. The binary coding is one extreme where start to have to round, the full decimal coding is the other extreme.

Some FPUs may us up to 80 bits but let us consider working with 16 bit resolution.

We now encode the range 0–1 into 65535 segments:

0.000	0.250	0.500	0,750	1.000
0000h	4000h	8000h	C000h	FFFFh

If we take a number and divide it by the maximum (FFFFh) we will clearly see this:

```
0000h: 0/65535 = 0.0
4000h: 16384/65535 = 0.25
8000h: 32768/65535 = 0.5
C000h: 49152/65535 = 0.75
FFFFh: 65535/65535 = 1.0
```

The operation of coding is similar to what we have seen with the binary coding:

- adjust the probabilities so the bits needed for operating with the number aren't above 16 bits.
- define a new interval
- The way to deal with the infinite number is
  - to have only loaded the 16 first bits, and when needed shift more onto it:
    `1100 0110 0001 000 0011 0100 0100 ...`
  - Work only with those bytes
  - as new bits are needed they'll be shifted.

Memory Problems
What about an alphabet with 26 symbols, or 256 symbols, ...?

- In general, number of bits is determined by the size of the interval.
- In general, (from entropy) need $-\log p$ bits to represent interval of size p.
- can be memory and CPU intensive

Estimating Probabilities - Dynamic Arithmetic Coding?
How to determine probabilities?

- If we have a static stream we simply count the tokens.

Could use *a priori* information for static or dynamic if scenario familiar.
But for Dynamic Data?

- Simple idea is to use adaptive model: Start with guess of symbol frequencies. Update frequency with each new symbol.
- Another idea is to take account of intersymbol probabilities, e.g., Prediction by Partial Matching.

7.4.7 Lempel-Ziv-Welch (LZW) Algorithm

The LZW algorithm is a very common compression technique.

Suppose we want to encode the Oxford Concise English dictionary which contains about 159,000 entries. Why not just transmit each word as an 18 bit number?

Problems:

- Too many bits,
- everyone needs a dictionary,
- only works for English text.

- **Solution**: Find a way to build the dictionary adaptively.
- Original methods due to Ziv and Lempel in 1977 and 1978. Terry Welch improved the scheme in 1984 (called LZW compression).
- It is used in UNIX *compress* — 1D token stream (similar to below)
- It used in GIF comprerssion — 2D window tokens (treat image as with Huffman Coding Above).

Reference: Terry A. Welch, "A Technique for High Performance Data Compression", IEEE Computer, Vol. 17, No. 6, 1984, pp. 8-19.

The LZW Compression Algorithm can summarised as follows:

```
w = NIL;
while ( read a character k )
    {
      if wk exists in the dictionary
       w = wk;
      else
        add wk to the dictionary;
        output the code for w;
        w = k;
    }
```

- Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.

Example:

`Input string is "^WED^WE^WEE^WEB^WET".`

w	k	output	index	symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W

W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

- A 19-symbol input has been reduced to 7-symbol plus 5-code output. Each code/symbol will need more than 8 bits, say 9 bits.
- Usually, compression doesn't start until a large number of bytes (e.g., > 100) are read in.

The LZW Decompression Algorithm is as follows:

```
read a character k;
  output k;
  w = k;
  while ( read a character k )
 /* k could be a character or a code. */
      {
        entry = dictionary entry for k;
        output entry;
        add w + entry[0] to dictionary;
        w = entry;
      }
```

Example (continued):

`Input string is "^WED<256>E<260><261><257>B<260>T".`

w	k	output	index	symbol
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET

- Problem: What if we run out of dictionary space?
  - Solution 1: Keep track of unused entries and use LRU
  - Solution 2: Monitor compression performance and flush dictionary when performance is poor.
- Implementation Note: LZW can be made *really* fast; it grabs a fixed number of bits from input stream, so bit parsing is very easy. Table lookup is automatic.

7.4.8 Entropy Encoding Summary

- Huffman maps fixed length symbols to variable length codes. Optimal only when symbol probabilities are powers of 2.
- Arithmetic maps entire message to real number range based on statistics. Theoretically optimal for long messages, but optimality depends on data model. Also can be CPU/memory intensive.
- Lempel-Ziv-Welch is a dictionary-based compression method. It maps a variable number of symbols to a fixed length code.
- Adaptive algorithms do not need a priori estimation of probabilities, they are more useful in real applications.

7.4.9 Further Reading/Information

Two good text books:

- *The Data Compression Book*, Mark Nelson,M&T Books, 1995.
- *Introduction to Data Compression*, Khalid Sayood, Morgan Kaufmann, 1996.

7.5 Source Coding Techniques

Source coding is based on the content of the original signal is also called *semantic-based coding*

High compression rates may be high but a price of loss of information. Good compression rates make be achieved with source encoding with *lossless* or little loss of information.

There are three broad methods that exist:

7.5.1 Transform Coding

A simple transform coding example

A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

1. Take top left pixel as the base value for the block, pixel A.
2. Calculate three other transformed values by taking the difference between these (respective) pixels and pixel A, i.e. B-A, C-A, D-A.
3. Store the base pixel and the differences as the values of the transform.

Given the above we can easily for the forward transform:

$$\begin{aligned} X_0 &= A \\ X_1 &= B - A \\ X_2 &= C - A \\ X_3 &= D - A \end{aligned}$$

and the inverse transform is:

$$\begin{aligned} A_n &= X_0 \\ B_n &= X_1 + X_0 \\ C_n &= X_2 + X_0 \\ D_n &= X_3 + X_0 \end{aligned}$$

The above transform scheme may be used to compress data by exploiting redundancy in the data:

Any Redundancy in the data has been transformed to values, X_i. So We can compress the data by using fewer bits to represent the differences. I.e if we use 8 bits per pixel then the 2x2 block uses 32 bits/ If we keep 8 bits for the base pixel, X0, and assign 4 bits for each difference then we only use 20 bits. Which is better than an average 5 bits/pixel

Example

Consider the following 4x4 image block:

120	130
125	120

then we get:

$$\begin{aligned} X_0 &= 120 \\ X_1 &= 10 \\ X_2 &= 5 \\ X_3 &= 0 \end{aligned}$$

We can then compress these values by taking less bits to represent the data.

However for practical purposes such a simple scheme as outlined above is not sufficient for compression:

- It is **Too Simple**
- Needs to operate on larger blocks (typically 8x8 min)
- Calculation is also too simple and from above we see that simple encoding of differences for large values will result in loss of information — v poor losses possible here 4 bits per pixel = values 0-15 unsigned, -7 – 7 signed so either quantise in multiples of 255/max value or massive overflow!!

However, More advance transform encoding techniques are very common (See JPEG/MPEG below). Frequency Domain methods such as Fourier Transform and (more commonly) Discrete Cosine Transforms (DCT) compression techniques fall into this category. We no consider these methods in general and then specifically.

7.5.2 Frequency Domain Methods

Frequency domains can be obtained through the transformation from one (Time or Spatial) domain to the other (Frequency) via

- Discrete Cosine Transform (DCT),
- Discrete Fourier Transform (DFT)

As we see later, the DCT at the heart of image (JPEG, Chapter 8) and video (MPEG, Chapter 9) compression methods, it may also be a component of MPEG audio compression. The DFT is used in MPEG audio compression (Section 9.6)

1D Example

Lets consider a 1D (e.g. Audio) example to see what the different domains mean:

Consider a complicated sound such as the noise of a car horn. We can describe this sound in two related ways:

- sample the amplitude of the sound many times a second, which gives an approximation to the sound as a function of time.
- analyse the sound in terms of the pitches of the notes, or frequencies, which make the sound up, recording the amplitude of each frequency.

In the example below (Fig **??**) we have a signal that consists of a sinusoidal wave at 8 Hz. 8Hz means that wave is completing 8 cycles in 1 second and is the frequency of that wave. From the frequency domain we can see that the composition of our signal is one wave (one peak) occurring with a frequency of 8Hz with a magnitude/fraction of 1.0 i.e. it is the whole signal.

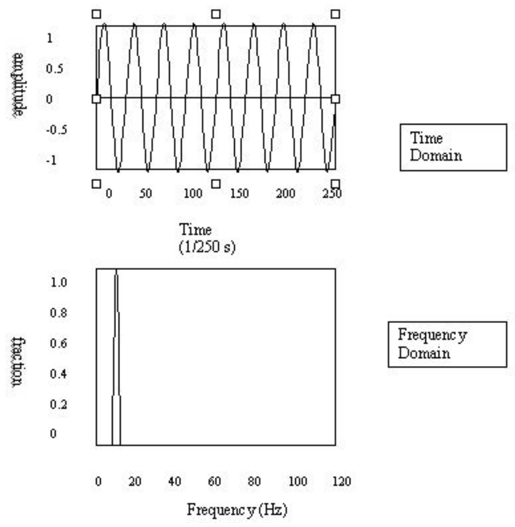


Figure 7.3: Relationship between Time and Frequency Domain

2D (Image) Example

Now images are no more complex really:

Similarly brightness along a line can be recorded as a set of values measured at equally spaced distances apart, or equivalently, at a set of spatial frequency values.

Each of these frequency values is referred to as a *frequency component.*

An image is a two-dimensional array of pixel measurements on a uniform grid.

This information be described in terms of a two-dimensional grid of spatial frequencies.

A given frequency component now specifies what contribution is made by data which is changing with specified x and y direction spatial frequencies.

What do frequencies mean in an image?

If an image has large values at *high* frequency components then the data is changing rapidly on a short distance scale. *e.g.* a page of text

If the image has large *low* frequency components then the large scale features of the picture are more important. *e.g.* a single fairly simple object which occupies most of the image.

For colour images, The measure (now a 2D matrix) of the frequency content is with regard to colour/chrominance: this shows if values are changing rapidly or slowly. Where the fraction, or value in the frequency matrix is low, the colour is changing gradually. Now the human eye is insensitive to gradual changes in colour and sensitive to intensity. So we can ignore gradual changes in colour and throw away data without the human eye noticing, we hope.

How can transforms into the Frequency Domain Help?

Any function (signal) can be decomposed into purely sinusoidal components (sine waves of different size/shape) which when added together make up our original signal.

In the example below (Fig 7.4) we have a square wave signal that has been decomposed by the Fourier Transform to render its sinusoidal components. Only the first few sine wave components are shown here. You can see that a the Square wave form will be roughly approximated if you add up the sinusoidal components.

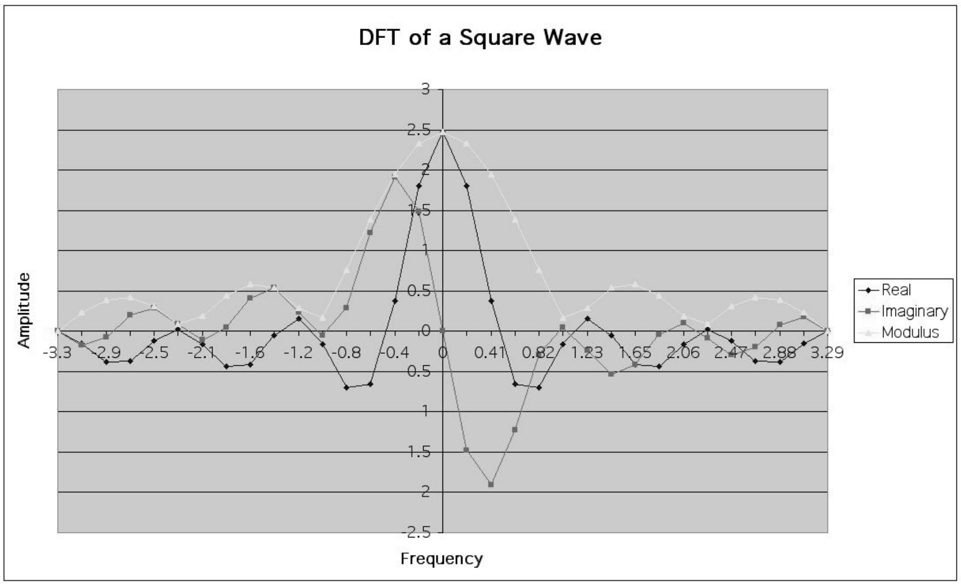


Figure 7.4: DFT of a Square Wave

Thus Transforming a signal into the frequency domain allows us to see what sine waves make up our signal e.g. One part sinusoidal wave at 50 Hz and two parts sinusoidal waves at 200 Hz.

More complex signals will give more complex graphs but the idea is exactly the same. The graph of the frequency domain is called the frequency spectrum.

An easy way to visualise what is happening is to think of a graphic equaliser on a stereo (Fig 9.15).

The bars on the left are the frequency spectrum of the sound that you are listening to. The bars go up and down depending on the type of sound that you are listening to. It is pretty obvious that the accumulation of these make up the whole. The bars on the right are used to increase and decrease the sound

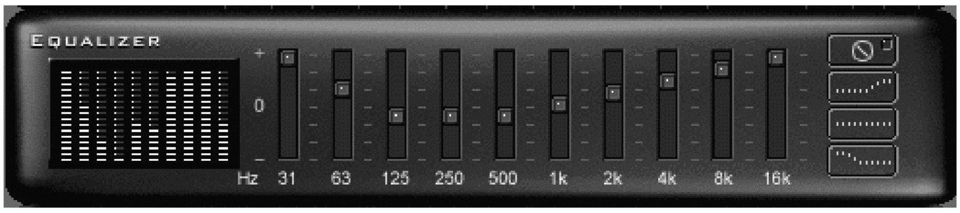


Figure 7.5: A Graphic Equaliser

at particular frequencies, denoted by the numbers (Hz). The lower frequencies, on the left, are for bass and the higher frequencies on the right are treble.

This is directly related to our example before. The bars show how much of the signal is made up of sinusoidal waves at that frequency. When all the waves are added together in their correct proportions that original sound is regenerated.

7.5.3 Fourier Theory

In order to fully comprehend the DCT will do a basic study of the Fourier theory and the Fourier transform first.

Whilst the DCT is ultimately used in multimedia compression it is easier to perhaps comprehend how such compression methods work by studying Fourier theory, from which the DCT is actually derived.

The tool which converts a spatial (real space) description of an image into one in terms of its frequency components is called the **Fourier transform**

The new version is usually referred to as the **Fourier space description** of the image.

The corresponding *inverse* transformation which turns a Fourier space description back into a real space one is called the **inverse Fourier transform**.

1D Case

Considering a continuous function $f(x)$ of a single variable x representing distance.

The Fourier transform of that function is denoted $F(u)$, where u represents spatial frequency is defined by

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-2\pi ixu}\,dx. \tag{7.1}$$

Note: In general $F(u)$ will be a complex quantity *even though* the original data is purely **real**.

The meaning of this is that not only is the magnitude of each frequency present important, but that its phase relationship is too.

The inverse Fourier transform for regenerating $f(x)$ from $F(u)$ is given by

$$f(x) = \int_{-\infty}^{\infty} F(u)e^{2\pi ixu}\,du, \tag{7.2}$$

which is rather similar, except that the exponential term has the opposite sign.

Let's see how we compute a Fourier Transform: consider a particular function $f(x)$ defined as

$$f(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{7.3}$$

shown in Fig. 7.6.

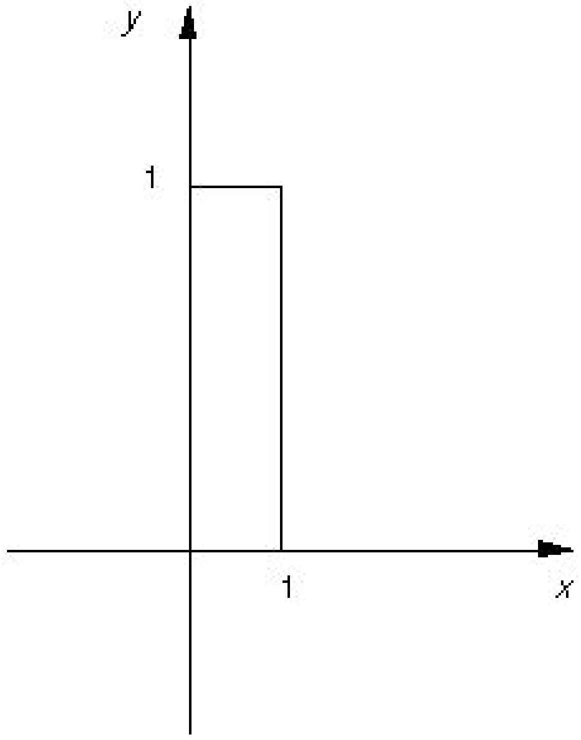


Figure 7.6: A top hat function

So its Fourier transform is:

$$\begin{aligned} F(u) &= \int_{-\infty}^{\infty} f(x)e^{-2\pi ixu}\,dx \\ &= \int_{-1}^{1} 1 \times e^{-2\pi ixu}\,dx \\ &= \frac{-1}{2\pi iu}(e^{2\pi iu} - e^{-2\pi iu}) \end{aligned}$$

$$= \frac{\sin 2\pi u}{\pi u}. \tag{7.4}$$

In this case $F(u)$ is purely real, which is a consequence of the original data being symmetric in x and $-x$. A graph of $F(u)$ is shown in Fig. 7.7. This function is often referred to as the Sinc function.

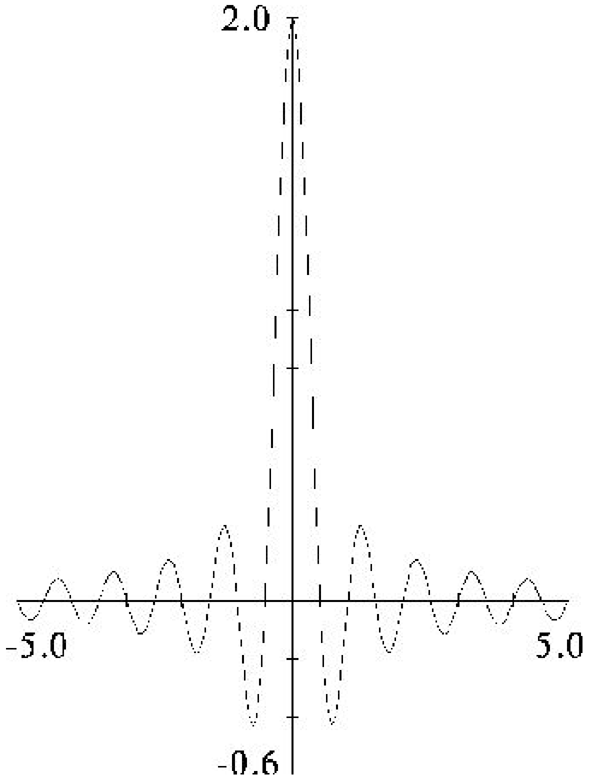


Figure 7.7: Fourier transform of a top hat function

2D Case

If $f(x, y)$ is a function, for example the brightness in an image, its Fourier transform is given by

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-2\pi i(xu+yv)}\, dx\, dy, \tag{7.5}$$

and the inverse transform, as might be expected, is

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{2\pi i(xu+yv)}\, du\, dv. \tag{7.6}$$

The Discrete Fourier Transform (DFT)

Images and Digital Audio are digitised !!

Thus, we need a *discrete* formulation of the Fourier transform, which takes such regularly spaced data values, and returns the value of the Fourier transform for a set of values in frequency space which are equally spaced.

This is done quite naturally by replacing the integral by a summation, to give the *discrete Fourier transform* or DFT for short.

In 1D it is convenient now to assume that x goes up in steps of 1, and that there are N samples, at values of x from 0 to $N-1$.

So the DFT takes the form

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-2\pi i x u/N}, \tag{7.7}$$

while the inverse DFT is

$$f(x) = \sum_{x=0}^{N-1} F(u) e^{2\pi i x u/N}. \tag{7.8}$$

NOTE: Minor changes from the continuous case are a factor of $1/N$ in the exponential terms, and also the factor $1/N$ in front of the forward transform which does not appear in the inverse transform.

The 2D DFT works is similar. So for an $N \times M$ grid in x and y we have

$$F(u,v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) e^{-2\pi i (xu/N + yv/M)}, \tag{7.9}$$

and

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u,v) e^{2\pi i (xu/N + yv/M)}. \tag{7.10}$$

Often $N = M$, and it is then it is more convenient to redefine $F(u,v)$ by multiplying it by a factor of N, so that the forward and inverse transforms are more symmetrical:

$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-2\pi i (xu + yv)/N}, \tag{7.11}$$

and

$$f(x,y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) e^{2\pi i (xu + yv)/N}. \tag{7.12}$$

Compression

How do we achieve compression:

- Low pass filter — ignore high frequency noise components
- Only store lower frequency components
- High Pass Filter — Spot Gradual Changes
- If changes to low Eye does not respond so ignore?

Where do put threshold to cut off?

Relationship between DCT and FFT

DCT (Discrete Cosine Transform) is actually a *cut-down* version of the FFT:

- Only the **real** part of FFT
- Computationally simpler than FFT
- DCT — Effective for Multimedia Compression
- DCT **MUCH** more commonly used.

7.5.4 The Discrete Cosine Transform (DCT)

The discrete cosine transform (DCT) helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality). The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain (Fig 7.8).

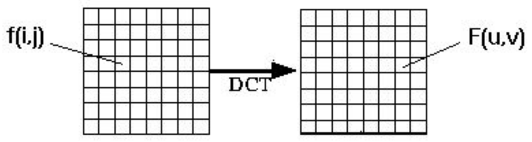


Figure 7.8: DCT Encoding

The general equation for a 1D (N data items) DCT is defined by the following equation:

$$F(u) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \Lambda(i).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] f(i)$$

and the corresponding *inverse* 1D DCT transform is simple $F^{-1}(u)$, i.e.:

$$\begin{aligned} f(i) &= F^{-1}(u) \\ &= \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \Lambda(i).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] F(i) \end{aligned}$$

where

$$\Lambda(i) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

The general equation for a 2D (N by M image) DCT is defined by the following equation:

$$F(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \left(\frac{2}{M}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(i).\Lambda(j).cos\left[\frac{\pi.u}{2.N}(2i+1)\right] cos\left[\frac{\pi.v}{2.M}(2j+1)\right].f(i,j)$$

and the corresponding *inverse* 2D DCT transform is simple $F^{-1}(u,v)$, i.e.:

$$\begin{aligned} f(i) &= F^{-1}(u,v) \\ &= \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(i)..\Lambda(j).cos\left[\frac{\pi.u}{2.N}(2i+1)\right].cos\left[\frac{\pi.v}{2.M}(2j+1)\right].F(i,j) \end{aligned}$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for} \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

The basic operation of the DCT is as follows:

- The input image is N by M;
- f(i,j) is the intensity of the pixel in row i and column j;
- F(u,v) is the DCT coefficient in row k1 and column k2 of the DCT matrix.
- For most images, much of the signal energy lies at low frequencies; these appear in the upper left corner of the DCT.
- Compression is achieved since the lower right values represent higher frequencies, and are often small - small enough to be neglected with little visible distortion.
- The DCT input is an 8 by 8 array of integers. This array contains each pixel's gray scale level;
- 8 bit pixels have levels from 0 to 255.

- Therefore an 8 point DCT would be:

$$F(u,v) = \frac{1}{4}\sum_{i,j}\Lambda(i).\Lambda(j).cos\left[\frac{\pi.u}{16}(2i+1)\right].cos\left[\frac{\pi.u}{16}(2i+1)\right]f(i,j)$$

  where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for}\xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

  Question: What is F[0,0]?

  answer: They define DC and AC components.

- The output array of DCT coefficients contains integers; these can range from -1024 to 1023.

- It is computationally easier to implement and more efficient to regard the DCT as a set of **basis functions** which given a known input array size (8 x 8) can be precomputed and stored. This involves simply computing values for a convolution mask (8 x8 window) that get applied (sum values x pixel the window overlap with image apply window across all rows/columns of image). The values as simply calculated from the DCT formula. The 64 (8 x 8) DCT basis functions are illustrated in Fig 7.9.

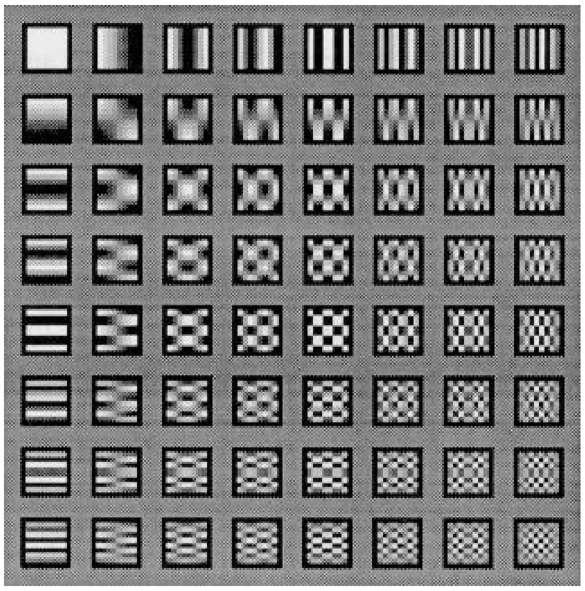


Figure 7.9: DCT basis functions

- Why DCT not FFT?

  DCT is similar to the Fast Fourier Transform (FFT), but can approximate lines well with fewer coefficients (Fig 7.10)

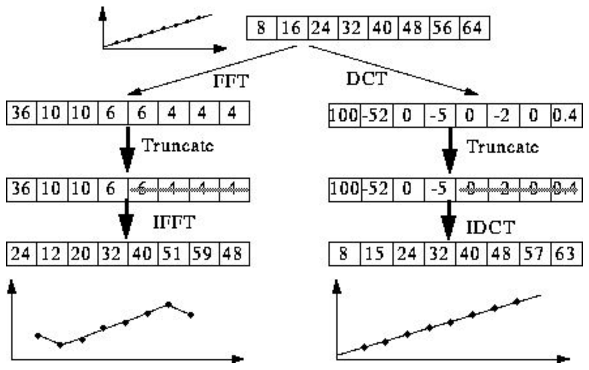


Figure 7.10: DCT/FFT Comparison

- Computing the 2D DCT
  - Factoring reduces problem to a series of 1D DCTs (Fig 7.11):
    * apply 1D DCT (Vertically) to Columns
    * apply 1D DCT (Horizontally) to resultant Vertical DCT above.
    * or alternatively Horizontal to Vertical.

    The equations are given by:

$$
\begin{aligned}
G(i,v) &= \frac{1}{2}\sum_i \Lambda(u).cos\left[\frac{\pi.v}{16}(2j+1)\right] f(i,j) \\
F(u,v) &= \frac{1}{2}\sum_i \Lambda(u).cos\left[\frac{\pi.u}{16}(2i+1)\right] G(i,v)
\end{aligned}
$$

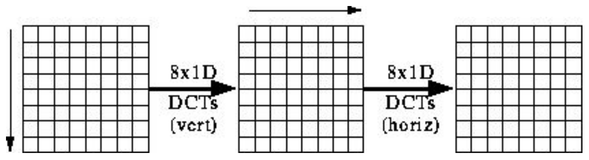


Figure 7.11: 2x1D Factored 2D DCT Computation

  - Most software implementations use fixed point arithmetic. Some fast implementations approximate coefficients so all multiplies are shifts and adds.

  - World record is 11 multiplies and 29 adds. (C. Loeffler, A. Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications", Proc. Int'l. Conf. on Acoustics, Speech, and Signal Processing 1989 (ICASSP '89), pp. 988-991)

7.5.5 Differential Encoding

Simple example of transform coding mentioned earlier and instance of this approach.

Here:

- The difference between the actual value of a sample and a prediction of that values is encoded.
- Also known as *predictive encoding.*
- Example of technique include: differential pulse code modulation, delta modulation and adaptive pulse code modulation — differ in prediction part.
- Suitable where successive signal samples do not differ much, but are not zero. *E.g.* Video — difference between frames, some audio signals.
- *Differential pulse code modulation* (DPCM) simple prediction:

  $$f_{predict}(t_i) = f_{actual}(t_{i-1})$$

  i.e. a simple Markov model where current value is the predict next value.

  So we simply need to encode:

  $$\Delta f(t_i) = f_{actual}(t_i) - f_{actual}(t_{i-1})$$

  If successive sample are close to each other we only need to encode first sample with a large number of bits:

  Actual Data: 9 10 7 6

  Predicted Data: 0 9 10 7

  $\Delta f(t)$: +9, +1, -3, -1.

- *Delta modulation* is a special case of DPCM: Same predictor function, coding error is a single bit or digit that indicates the current sample should be increased or decreased by a step.

  Not Suitable for rapidly changing signals.

- *Adaptive pulse code modulation* — Fuller Markov model: data is extracted from a function of a series of previous values: *E.g.* Average of last n samples. Characteristics of sample better preserved.

7.5.6 Vector Quantisation

The basic outline of this approach is:

- Data stream divided into (1D or 2D square) blocks — *vectors*
- A table or *code book* is used to find a pattern for each block.
- Code book can be dynamically constructed or predefined.
- Each pattern for block encoded as a look value in table
- Compression achieved as data is effectively subsampled and coded at this level.

Chapter 8

Compression II: Images (JPEG)

What is JPEG?

- "Joint Photographic Expert Group" — an international standard in 1992.
- Works with colour and greyscale images, Many applications e.g., satellite, medical, ...

8.1 Basic JPEG Compression Pipeline

JPEG compression does not involve one compression algorithm *per se*. It uses (or may use depending on comprssion level/quality desired) most of the underlying compression algorithms we have previously studied.

JPEG compression involves the following:

- Encoding (Fig 8.1)
- Decoding – Reverse the order for encoding

The Major algorithms used at various steps in JPEG Coding are:

- DCT (Discrete Cosine Transformation) - The heart of JPEG Compression.
- Quantization
- Zigzag Scan
- DPCM on DC component
- RLE on AC Components
- Entropy Coding

The DCT has already been introduced (Section 7.5.4) we now summarise the other steps.

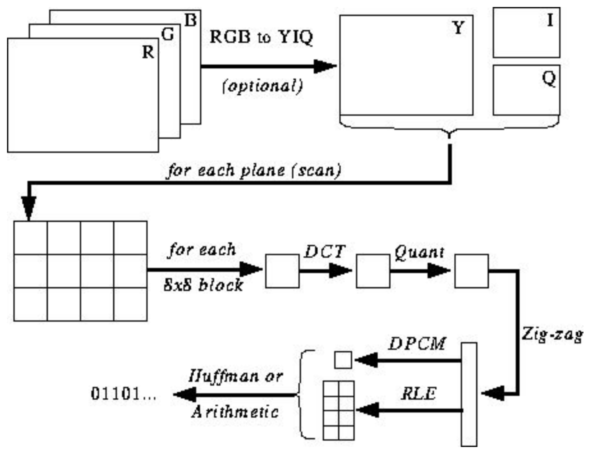


Figure 8.1: JPEG Encoding

8.1.1 Quantization

Why do we need to quantise:

- To throw out bits
- *Example*: 101101 = 45 (6 bits).
  Truncate to 4 bits: 1011 = 11.
  Truncate to 3 bits: 101 = 5.
- Quantization error is the main source of the Lossy Compression.

Uniform quantization

- Divide by constant N and round result ($N = 4$ or 8 in examples above).
- Non powers-of-two gives fine control (e.g., $N = 6$ loses 2.5 bits)

Quantization Tables

- In JPEG, each F[u,v] is divided by a constant q(u,v).
- Table of q(u,v) is called *quantization table*.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77

24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

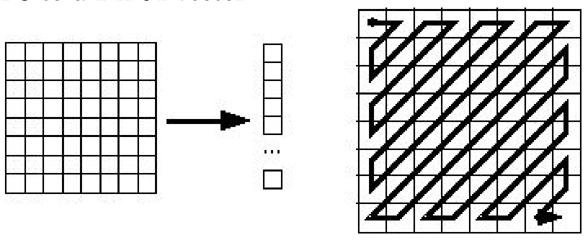
- Eye is most sensitive to low frequencies (upper left corner), less sensitive to high frequencies (lower right corner)
- Standard defines 2 default quantization tables, one for luminance (above), one for chrominance.
- Q: How would changing the numbers affect the picture (e.g., if I doubled them all)?

  Quality factor in most implementations is the scaling factor for default quantization tables.
- Custom quantization tables can be put in image/scan header.

8.1.2 Zig-zag Scan

What is the purpose of the Zig-zag Scan:

- to group low frequency coefficients in top of vector.
- Maps 8 x 8 to a 1 x 64 vector



8.1.3 Differential Pulse Code Modulation (DPCM) on DC component

Here we see that besides DCT another encoding method is employed: DPCM on the DC component at least. Why is this strategy adopted:

- DC component is large and varied, but often close to previous value (like lossless JPEG).
- Encode the difference from previous 8x8 blocks – DPCM

8.1.4 Run Length Encode (RLE) on AC components

Yet another simple compression technique is applied to the AC component:

- 1x64 vector has lots of zeros in it
- Encode as (*skip, value*) pairs, where *skip* is the number of zeros and *value* is the next non-zero component.
- Send (0,0) as end-of-block sentinel value.

8.1.5 Entropy Coding

DC and AC components finally need to be represented by a smaller number of bits:

- Categorize DC values into SSS (number of bits needed to represent) and actual bits.

```
--------------------
   Value       SSS
     0          0
    -1,1        1
 -3,-2,2,3      2
-7..-4,4..7     3
--------------------
```

- *Example*: if DC value is 4, 3 bits are needed.

  Send off SSS as Huffman symbol, followed by actual 3 bits.
- For AC components (*skip, value*), encode the composite symbol (*skip,SSS*) using the Huffman coding.
- Huffman Tables can be custom (sent in header) or default.

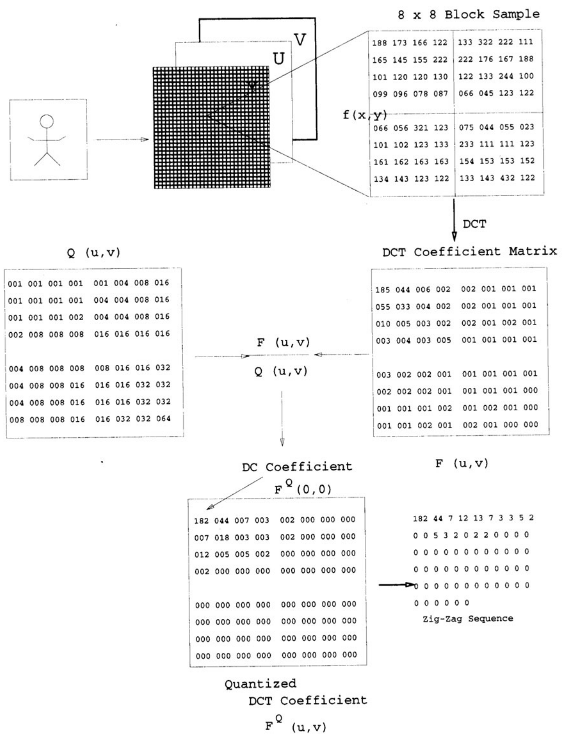
8.1.6 Example JPEG Compression

In Figure 8.2 below we see an example of the whole JPEG compression pipeline being applied to a single 8x8 block.

8.1.7 Summary of the JPEG bitstream

Figure 8.1 and the above JPEG components have described how compression is achieved at several stages. Let us conclude by summarising the overall compression process:

- A "Frame" is a picture, a "scan" is a pass through the pixels (e.g., the red component), a "segment" is a group of blocks, a "block" is an 8x8 group of pixels.



188	173	166	122	133	322	222	111
165	145	155	222	222	176	167	188
101	120	120	130	122	133	244	100
099	096	078	087	066	045	123	122
066	056	321	123	075	044	055	023
101	102	123	133	233	111	111	123
161	162	163	163	154	153	153	152
134	143	123	122	133	143	432	122

185	044	006	002	002	001	001	001
055	033	004	002	002	001	001	001
010	005	003	002	002	001	002	001
003	004	003	005	001	001	001	001
003	002	002	001	001	001	001	001
002	002	002	001	001	001	001	000
001	001	001	002	001	002	001	000
001	001	002	001	002	001	000	000

001	001	001	001	001	004	008	016
001	001	001	001	004	004	008	016
001	001	001	002	004	004	008	016
002	008	008	008	016	016	016	016
004	008	008	008	008	016	016	032
004	008	008	016	016	016	032	032
004	008	008	016	016	016	032	032
008	008	008	016	016	032	032	064

182	044	007	003	002	000	000	000
007	018	003	003	002	000	000	000
012	005	005	002	000	000	000	000
002	000	000	000	000	000	000	000
000	000	000	000	000	000	000	000
000	000	000	000	000	000	000	000
000	000	000	000	000	000	000	000
000	000	000	000	000	000	000	000

```
182 44 7 12 13 7 3 3 5 2
0 0 5 3 2 0 2 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
```

Figure 8.2: Example JPEG Compression (Figure taken from Networked Multimedia Systems, Raghavan and Tripathi, Prentice Hall)

- Frame header: sample precision (width, height) of image number of components unique ID (for each component) horizontal/vertical sampling factors (for each component) quantization table to use (for each component)
- Scan header Number of components in scan component ID (for each component) Huffman table for each component (for each component)
- Misc. (can occur between headers) Quantization tables Huffman Tables Arithmetic Coding Tables Comments Application Data

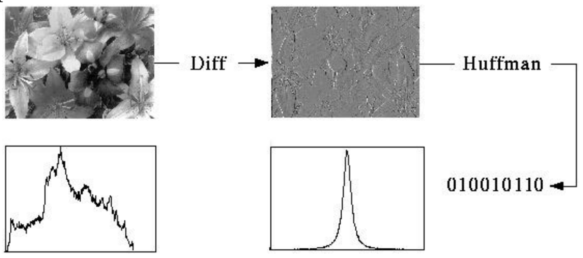
8.2 Practical JPEG Compression

JPEG compression algorithms may fall in to one of several categories depending on how the compression is actually performed:

- Baseline/Sequential – the one that we described in detail
- Lossless
- Progressive
- Hierarchical
- "Motion JPEG" – Baseline JPEG applied to each image in a video.

Briefly, this is how each above approach is encoded:

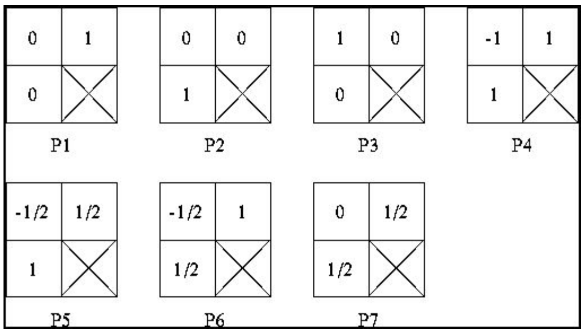
1. Lossless Mode
   - A special case of the JPEG where indeed there is no loss



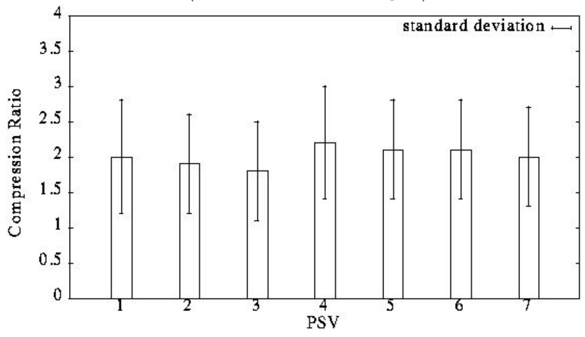
   - Take difference from previous pixels (not blocks as in the Baseline mode) as a "predictor".

     Predictor uses linear combination of previously encoded neighbors.

     It can be one of seven different predictor based on pixels neighbors



- Since it uses only previously encoded neighbors, first row always uses P2, first column always uses P1.
- Effect of Predictor (test with 20 images)



Note: "2D" predictors (4-7) always do better than "1D" predictors.

Comparison with Other Lossless Compression Programs (compression ratio):

Compression Program	Lena	football	F-18	flowers
lossless JPEG	1.45	1.54	2.29	1.26
optimal lossless JPEG	1.49	1.67	2.71	1.33
compress (LZW)	0.86	1.24	2.21	0.87
gzip (Lempel-Ziv)	1.08	1.36	3.10	1.05
gzip -9 (optimal Lempel-Ziv)	1.08	1.36	3.13	1.05
pack (Huffman coding)	1.02	1.12	1.19	1.00

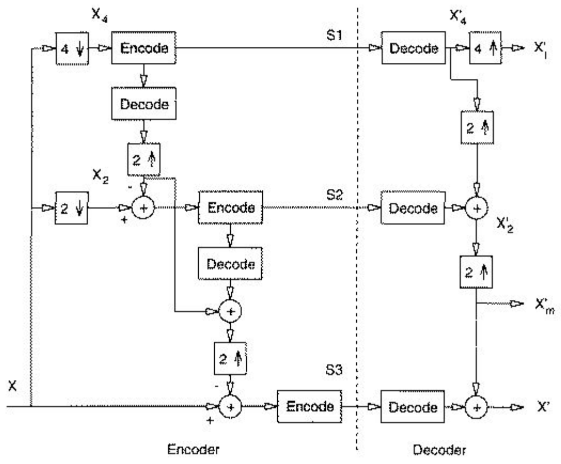
2. Progressive Mode
   - Goal: display low quality image and successively improve.
   - Two ways to successively improve image:

   (a) *Spectral selection*: Send DC component, then first few AC, some more AC, etc.
   (b) *Successive approximation*: send DCT coefficients MSB (most significant bit) to LSB (least significant bit).

3. Hierarchical Mode

   A Three-level Hierarchical JPEG Encoder

   (From *V. Bhaskaran and K. Konstantinides, "Image and Video Compression Standards: Algorithms and Architectures", Kluwer Academic Publishers, 1995.*)



   - Down-sample by factors of 2 in each direction.
     Example: map 640x480 to 320x240
   - Code smaller image using another method (Progressive, Baseline, or Lossless).
   - Decode and up-sample encoded image
   - Encode difference between the up-sampled and the original using Progressive, Baseline, or Lossless.
   - Can be repeated multiple times.
   - Good for viewing high resolution image on low resolution display.

4. JPEG-2

   - Big change was to use *adaptive quantization*

8.3 JPEG 2000

A new version of JPEG has been defined and was released in 2002.

The JPEG 2000 standard is based on discrete wavelet transform (DWT) (instead of DCT), scalar quantization, context modeling, arithmetic coding and post-compression rate allocation. The standard lends itself to a variety of uses, ranging from digital photography to medical imaging to advanced digital scanning and printing. For more information on the JPEG 2000 standard for still image coding, refer to

http://www.jpeg.org/JPEG2000.htm

Most notably, JPEG 2000 provides high compression efficiencyin many cases, visually lossless compression at 1 bit per pixel or better.

8.3.1 Further Reading

A good tutorial on JPEG may be found at:

http://www.ece.purdue.edu/ ace/jpeg-tut/jpegtut1.html

The main JPEG web page is:

http://www.jpeg.org/

Futher information on JPEG 2000 may be found at:

http://www.jpeg.org/JPEG2000.htm

Chapter 9

Compression III: Video and Audio Compression

We have studied the theory of encoding now let us see how this is applied in practice to video (MPEG and others) and then we revisit audio.

We need to compress video (and audio) in practice since:

1. Uncompressed video (and audio) data are huge. In HDTV, the bit rate easily exceeds 1 Gbps. — big problems for storage and network communications.

   For example:

   One of the formats defined for HDTV broadcasting within the United States is 1920 pixels horizontally by 1080 lines vertically, at 30 frames per second. If these numbers are all multiplied together, along with 8 bits for each of the three primary colors, the total data rate required would be approximately 1.5 Gb/sec. Because of the 6 MHz. channel bandwidth allocated, each channel will only support a data rate of 19.2 Mb/sec, which is further reduced to 18 Mb/sec by the fact that the channel must also support audio, transport, and ancillary data information. As can be seen, this restriction in data rate means that the original signal must be compressed by a figure of approximately 83:1. This number seems all the more impressive when it is realized that the intent is to deliver very high quality video to the end user, with as few visible artifacts as possible.

2. Lossy methods have to employed since the *compression ratio* of lossless methods (e.g., Huffman, Arithmetic, LZW) is not high enough for image and video compression, especially when distribution of pixel values is relatively flat.

The following compression types are commonly used in Video compression:

- Spatial Redundancy Removal – Intraframe coding (JPEG)

- Spatial and Temporal Redundancy Removal – Intraframe and Interframe coding (H.261, MPEG)

These are discussed in the following sections. We only concentrate on some basic principles of video compression (which evolved from the earlier H.261 and MPEG 1/2 standards).

9.1 Compression Standards

Image, Video and Audio Compression standards have been specifies and released by two main groups since 1985:

ISO - International Standards Organisation: JPEG, MEPG.

ITU - International Telecommunications Union: H.261 - 264.

Whilst in many cases one of the groups have specified the standards there is some crossover between the groups, For example:

- JPEG issued by ISO in 1989 but adopted by ITU as ITU T.81)
- MPEG 1 released by ISO in 1991,
- H.261 released by ITU in 1993 (based on CCITT 1990 draft). CCITT stands for *Comité Consultatif International Téléphonique et Télégraphique* whose parent company is ITU.
- H.262 is alternatively better known as MPEG-2 released in 1994.
- H.263 released in 1996 extended as H.263+, H.263++.
- MPEG 4 release in 1998.
- H.264 releases in 2002 for DVD quality and is now part of MPEG 4 (Part 10). Quicktime 6 supports this.

9.2 Basic Idea of Video Compression: Motion Estimation/Compensation

Video embraces two basic forms of means of compression the data:

- Spatial — Each frame can be regarded as an image and can be compressed very much like JPEG. As we shall see this is termed *Intraframe* Coding
- Temporal — Greater compression (than just JPEG coding each frame) can be achieved by noting the temporal coherence/incoherence over frames. Essentially we note the difference between frames.

- Furthermore, spatial and temporal information can used together so that if we can estimate the motion of parts of the picture (blocks) then we can represent each block (Compressed) and track it across frames and reuse the block in subsequent frame (frames). As we shall see this is termed *Interframe* Coding This is also called *Motion estimation/compensation.*

Lets us look at a very simple example to explain this (things are much more complex in practice of course).

How can be represent the compressed data? Simply based on Differential Pulse Code Modulation (DPCM).

Consider a simple image (block) of a moving circle (Fig 9.1)

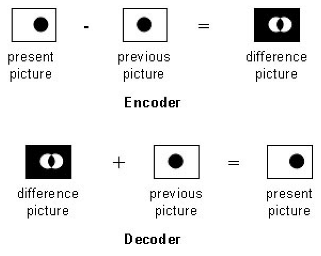


Figure 9.1: DPCM without motion compensation.

The example in Figures 9.3-?? explains the concept of motion estimation/compensation. We will examine methods of estimating motion vectors in due course.

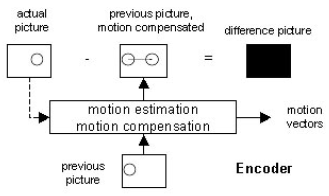


Figure 9.2: Motion estimation/compensation (encoding)

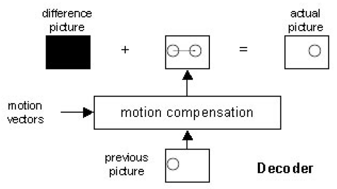


Figure 9.3: Motion estimation/compensation (decoding)

As we will shortly see motion estimation in MPEG-1/H.261 is done by using block matching techniques, i.e. for a certain area of pixels in a picture, one tries to find a good estimate of this area in a previous (or in a future) frame, within a specified search area. Motion compensation uses the motion vectors to compensate the picture. This means that parts of a previous (or future) picture can be reused in a subsequent picture.

In summary: Motion estimation/compensation techniques reduces the video bitrate significantly but also introduces complexity and delay, due to the need of buffering reference pictures.

9.3 H. 261 Compression

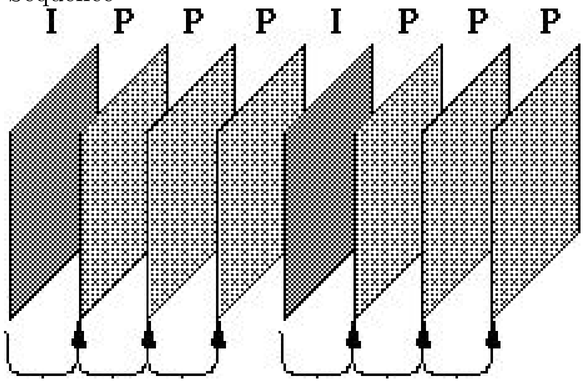
H. 261 Compression has been specifically designed for video telecommunication applications:

- Developed by CCITT in 1988-1990
- Meant for videoconferencing, videotelephone applications over ISDN telephone lines.
- Baseline ISDN is 64 kbits/sec, and integral multiples (px64)

9.3.1 Overview of H.261

The basic approach to H. 261 Compression is summarised as follows:

- Decoded Sequence

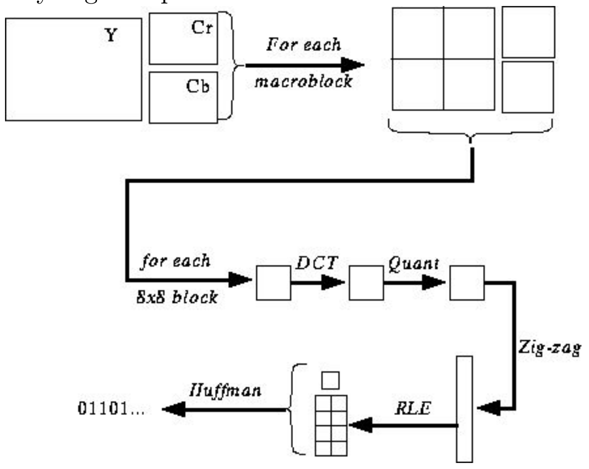


- Frame types are CCIR 601 CIF (352x288) and QCIF (176x144) images with 4:2:0 subsampling.
- Two frame types: Intraframes (*I-frames*) and Interframes (*P-frames*)
- I-frames use basically JPEG
- P-frames use *pseudo-differences* from previous frame (predicted), so frames depend on each other.

- I-frame provide us with an accessing point.

9.3.2 Intra Frame Coding

The term *intra frame coding* refers to the fact that the various lossless and lossy compression techniques are performed relative to information that is contained only within the current frame, and not relative to any other frame in the video sequence. In other words, no temporal processing is performed outside of the current picture or frame. This mode will be described first because it is simpler, and because non-intra coding techniques are extensions to these basics. Figure 1 shows a block diagram of a basic video encoder for intra frames only. It turns out that this block diagram is very similar to that of a JPEG still image video encoder, with only slight implementation detail differences.



The potential ramifications of this similarity will be discussed later. The basic processing blocks shown are the video filter, discrete cosine transform, DCT coefficient quantizer, and run-length amplitude/variable length coder. These blocks are described individually in the sections below or have already been described in JPEG Compression.

This is a basic Intra Frame Coding Scheme is as follows:

- Macroblocks are 16x16 pixel areas on Y plane of original image.

  A *macroblock* usually consists of 4 Y blocks, 1 Cr block, and 1 Cb block.

  In the example HDTV data rate calculation shown previously, the pixels were represented as 8-bit values for each of the primary colors red, green, and blue. It turns out that while this may be good for high performance computer generated graphics, it is wasteful in most video compression applications. Research into the Human Visual System (HVS) has shown that the eye is most sensitive to changes in luminance, and less sensitive to variations in chrominance. Since absolute compression is the name of the game, it makes sense that MPEG should operate on a color space that can effectively take advantage of the eyes different sensitivity to luminance and chrominance information. As such, H/261 (and MPEG) uses the YCbCr color space to represent the data values instead of RGB, where Y is the luminance signal, Cb is the blue color difference signal, and Cr is the red color difference signal.

A macroblock can be represented in several different manners when referring to the YCbCr color space. Figure 9.4 below shows 3 formats known as 4:4:4, 4:2:2, and 4:2:0 video. 4:4:4 is full bandwidth YCbCr video, and each macroblock consists of 4 Y blocks, 4 Cb blocks, and 4 Cr blocks. Being full bandwidth, this format contains as much information as the data would if it were in the RGB color space. 4:2:2 contains half as much chrominance information as 4:4:4, and 4:2:0 contains one quarter of the chrominance information. Although MPEG-2 has provisions to handle the higher chrominance formats for professional applications, most consumer level products will use the normal 4:2:0 mode.

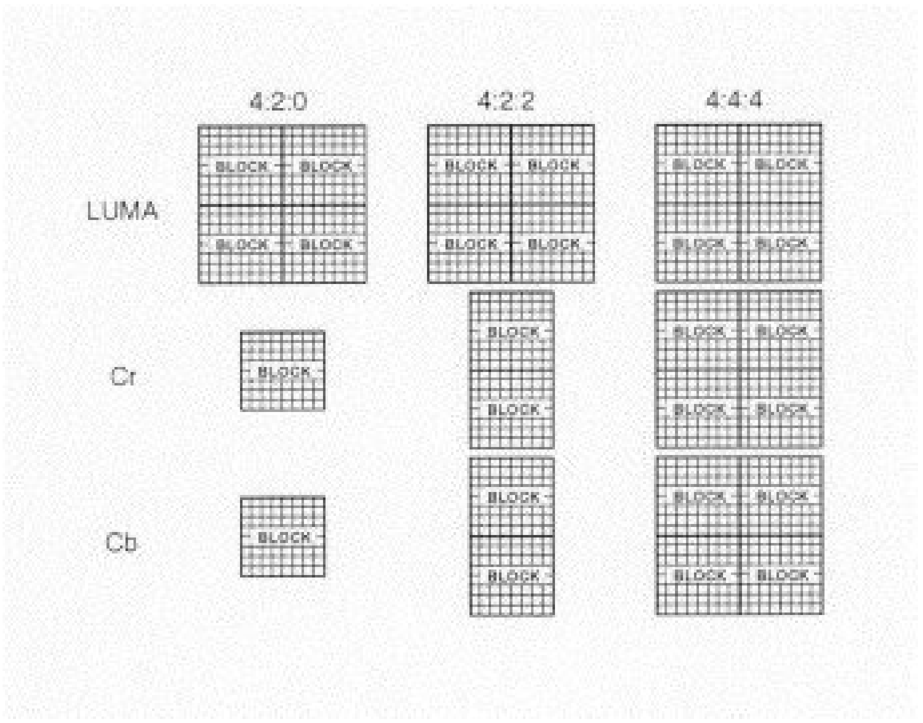


Figure 9.4: Macroblock Video Formats

Because of the efficient manner of luminance and chrominance representation, the 4:2:0 representation allows an immediate data reduction from 12 blocks/macroblock to 6 blocks/macroblock, or 2:1 compared to full bandwidth representations such as 4:4:4 or RGB. To generate this format without generating color aliases or artifacts requires that the chrominance signals be filtered.

The Macroblock is coded as follows:

Addr	Type	Quant	Vector	CBP	b0	b1	•••	b5

- Many macroblocks will be exact matches (or close enough). So send address of each block in image –> *Addr*
- Sometimes no good match can be found, so send INTRA block –> *Type*
- Will want to vary the quantization to fine tune compression, so send quantization value –> *Quant*
- Motion vector –> *vector*
- Some blocks in macroblock will match well, others match poorly. So send bitmask indicating which blocks are present (Coded Block Pattern, or *CBP*).
- Send the blocks (4 Y, 1 Cr, 1 Cb) as in JPEG.

• Quantization is by constant value for all DCT coefficients (i.e., no quantization table as in JPEG).

9.3.3 Inter-frame (P-frame) Coding

The previously discussed intra frame coding techniques were limited to processing the video signal on a spatial basis, relative only to information within the current video frame. Considerably more compression efficiency can be obtained however, if the inherent temporal, or time-based redundancies, are exploited as well. Anyone who has ever taken a reel of the old-style super-8 movie film and held it up to a light can certainly remember seeing that most consecutive frames within a sequence are very similar to the frames both before and after the frame of interest. Temporal processing to exploit this redundancy uses a technique known as block-based motion compensated prediction, using motion estimation. A block diagram of the basic encoder with extensions for non-intra frame coding techniques is given in Figure 9.5. Of course, this encoder can also support intra frame coding as a subset.

Starting with an intra, or I frame, the encoder can forward predict a future frame. This is commonly referred to as a P frame, and it may also be predicted from other P frames, although only in a forward time manner. As an example, consider a group of pictures that lasts for 6 frames. In this case, the frame ordering is given as I,P,P,P,P,P,I,P,P,P,P,

Each P frame in this sequence is predicted from the frame immediately preceding it, whether it is an I frame or a P frame. As a reminder, I frames are coded spatially with no reference to any other frame in the sequence.

P-coding can be summarised as follows:

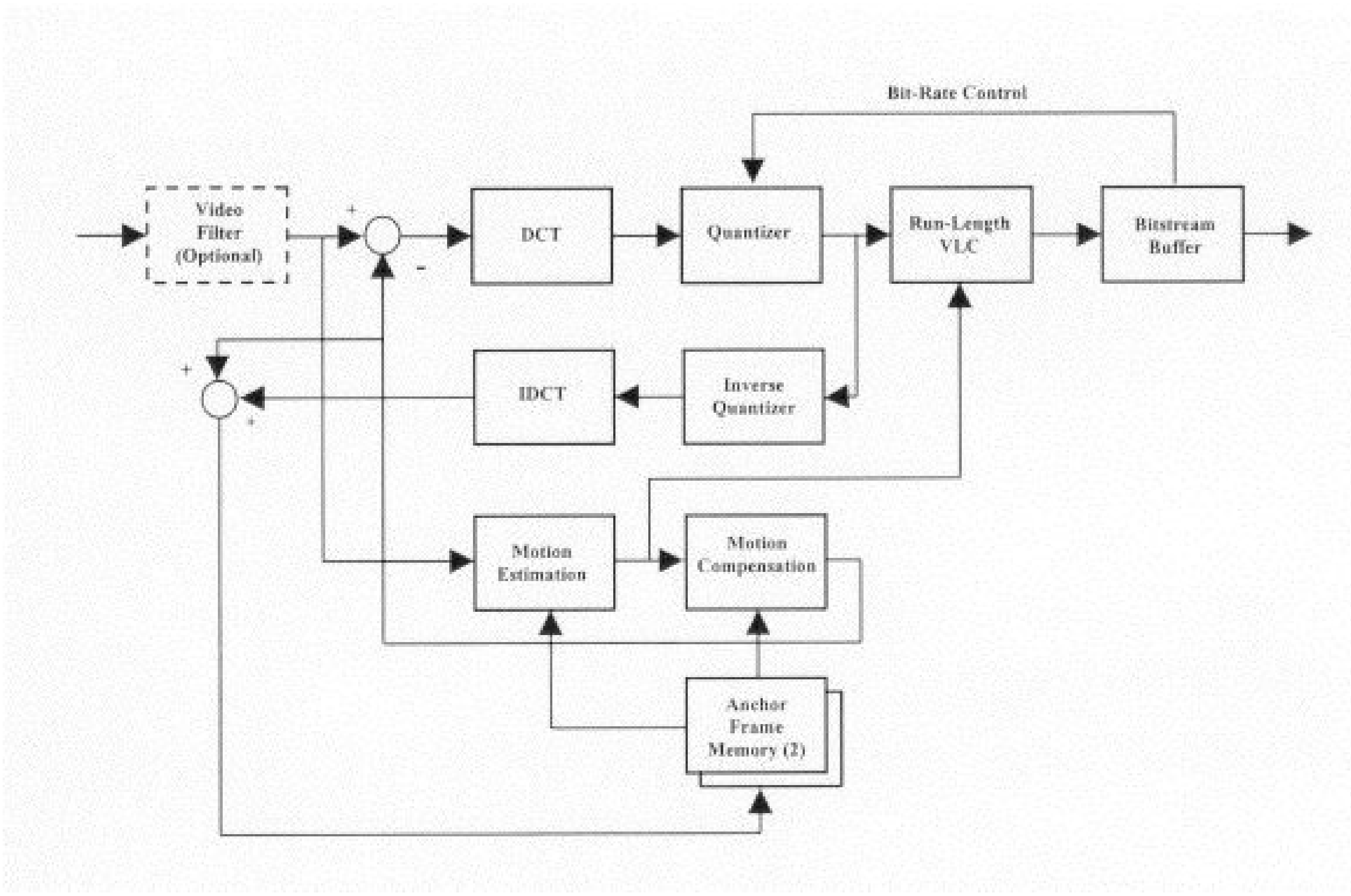
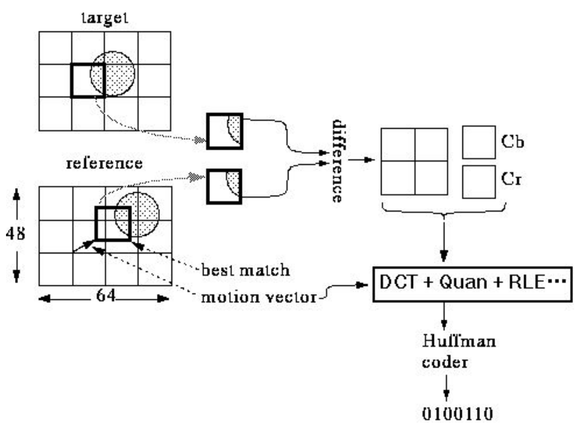


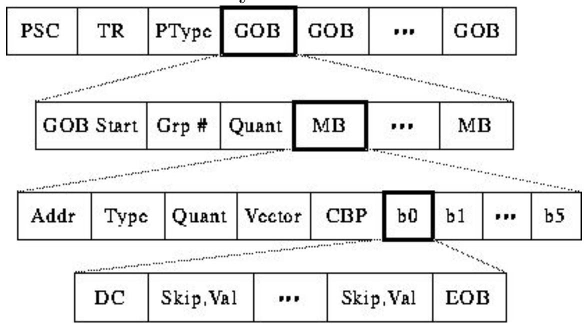
Figure 9.5: P-Frame Coding



- An Coding Example (P-frame)
- Previous image is called *reference image.*
- Image to code is called *target image.*
- Actually, the difference is encoded.
- Subtle points:
  1. Need to use decoded image as reference image, *not* original. Why?
  2. We're using "Mean Absolute Difference" (MAD) to decide best block. Can also use "Mean Squared Error" (MSE) = sum(E*E)

9.3.4 The H.261 Bitstream Structure

The H.261 Bitstream structure may be summarised as follows:



- Need to delineate boundaries between pictures, so send Picture Start Code –> *PSC*
- Need timestamp for picture (used later for audio synchronization), so send Temporal Reference –> *TR*

- Is this a P-frame or an I-frame? Send Picture Type –> *PType*
- Picture is divided into regions of 11x3 macroblocks called Groups of Blocks –> *GOB*
- Might want to skip whole groups, so send Group Number (*Grp #*)
- Might want to use one quantization value for whole group, so send Group Quantization Value –> *GQuant*
- Overall, bitstream is designed so we can skip data whenever possible while still unambiguous.

The overall H.261 Codec is summarised in Fig 9.3.4.

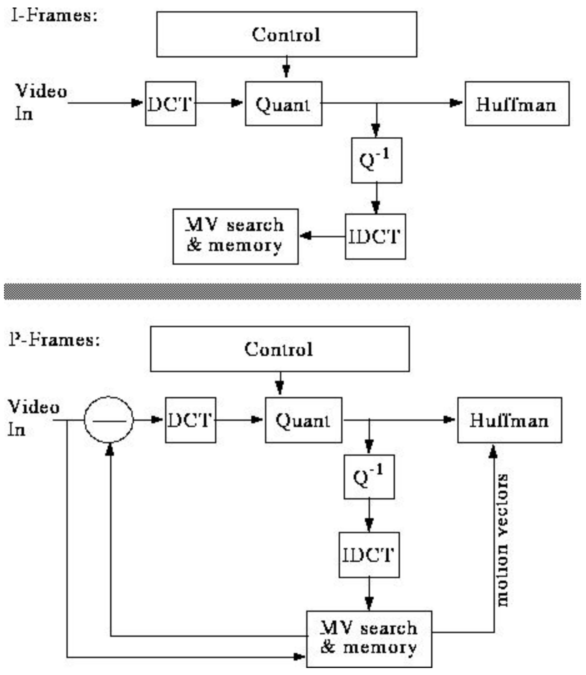


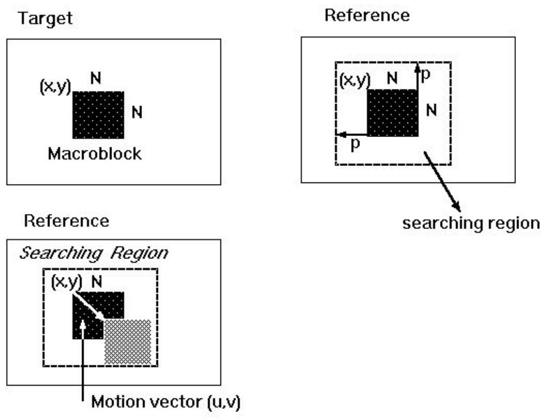
Figure 9.6: H.261 Codec

9.3.5 Hard Problems in H.261

There are however a few difficult problems in H.261:

- Motion vector search
- Propagation of Errors
- Bit-rate Control

Motion Vector Search



- $C(x+k, y+i)$ – pixels in the macro block with upper left corner (x, y) in the Target.

  $R(X+i+k, y+j+l)$ – pixels in the macro block with upper left corner $(x+i, y+j)$ in the Reference.

  Cost function is:

$$\mathrm{MAE}(i,j) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} \left| C(x+k, y+l) - R(x+i+k, y+j+l) \right|$$

  Where MAE stands for *Mean Absolute Error.*

- Goal is to find a vector (u, v) such that MAE (u, v) is minimum

- **Full Search Method:**

  1. Search the whole $[-p, p]$ searching region.
  2. Cost is:

$$\frac{IJF}{N^2} (2p+1)^2 \times N^2 \times 3$$

     operations,

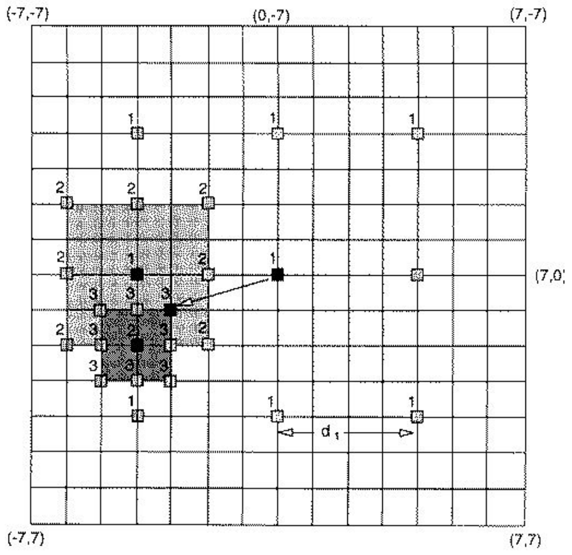
     assuming that each pixel comparison needs 3 operations (Subtraction, Absolute value, Addition).

- **Two-Dimensional Logarithmic Search:**

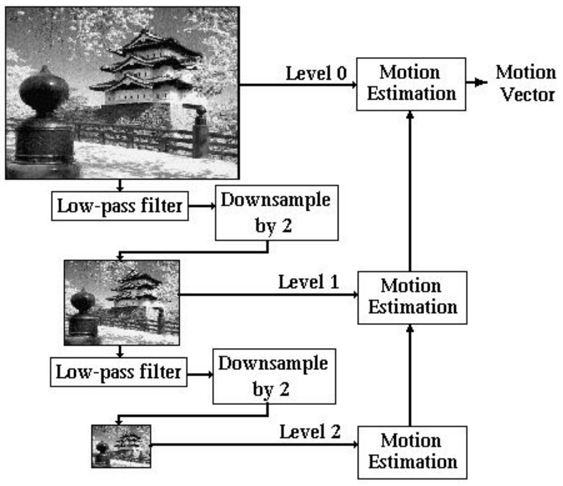
  Similar to binary search. MAE function is initially computed within a window of $[-p/2, p/2]$ at nine locations as shown in the figure.

  Repeat until the size of the search region is one pixel wide:

1. Find one of the nine locations that yields the minimum MAE.
2. Form a new searching region with half of the previous size and centered at the location found in step 1.



- **Hierarchical Motion Estimation:**



1. Form several low resolution version of the target and reference pictures
2. Find the best match motion vector in the lowest resolution version.
3. Modify the motion vector level by level when going up

- Performance comparison:

```
          ----------------------------------------------------------------
     Search Method         Operation for 720x480 at 30 fps
      p = 15                p=7
         ----------------------------------------------------------------
           Full Search                29.89 GOPS            6.99 GOPS
           Logarithmic                 1.02 GOPS          777.60 MOPS
           Hierarchical              507.38 MOPS          398.52 MOPS
         ----------------------------------------------------------------
```

Propagation of Errors

- Send an I-frame every once in a while
- Make sure you use decoded frame for comparison

Bit-rate Control

- Simple feedback loop based on "buffer fullness"

  If buffer is too full, increase the quantization scale factor to reduce the data.

9.4 MPEG Compression

The acronym MPEG stands for Moving Picture Expert Group, which worked to generate the specifications under ISO, the International Organization for Standardization and IEC, the International Electrotechnical Commission. What is commonly referred to as "MPEG video" actually consists at the present time of two finalized standards, MPEG-11 and MPEG-22, with a third standard, MPEG-4, was finalized in 1998 for *Very Low Bitrate Audio-Visual Coding*. The MPEG-1 and MPEG-2 standards are similar in basic concepts. They both are based on motion compensated block-based transform coding techniques, while MPEG-4 deviates from these more traditional approaches in its usage of software image construct descriptors, for target bit-rates in the very low range, < 64Kb/sec. Because MPEG-1 and MPEG-2 are finalized standards and are both presently being utilized in a large number of applications, this paper concentrates on compression techniques relating only to these two standards. Note that there is no reference to MPEG-3. This is because it was originally anticipated that this standard would refer to HDTV applications, but it was found that minor extensions to the MPEG-2 standard would suffice for this higher bit-rate, higher resolution application, so work on a separate MPEG-3 standard was abandoned.

The current thrust of is on Multimedia content and frameworks. MPEG-7 "Multimedia Content Description Interface" was defined in 2001. Work on the new standard MPEG-21 "Multimedia Framework" has started in June 2000 and has already produced a Draft Technical Report and two Calls for Proposals.

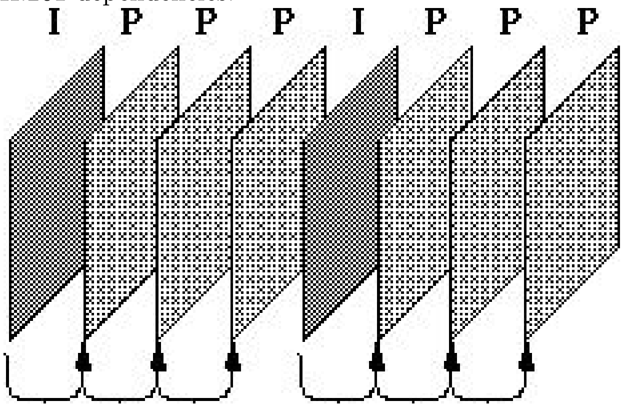
MPEG-1 was finalized in 1991, and was originally optimized to work at video resolutions of 352x240 pixels at 30 frames/sec (NTSC based) or 352x288 pixels at 25 frames/sec (PAL based), commonly referred to as Source Input Format (SIF) video. It is often mistakenly thought that the MPEG-1 resolution is limited to the above sizes, but it in fact may go as high as 4095x4095 at 60 frames/sec. The bit-rate is optimized for applications of around 1.5 Mb/sec, but again can be used at higher rates if required. MPEG-1 is defined for progressive frames only, and has no direct provision for interlaced video applications, such as in broadcast television applications.

MPEG-2 was finalized in 1994, and addressed issues directly related to digital television broadcasting, such as the efficient coding of field-interlaced video and scalability. Also, the target bit-rate was raised to between 4 and 9 Mb/sec, resulting in potentially very high quality video. MPEG-2 consists of profiles and levels. The profile defines the bitstream scalability and the colorspace resolution, while the level defines the image resolution and the maximum bit-rate per profile. Probably the most common descriptor in use currently is Main Profile, Main Level (MP@ML) which refers to 720x480 resolution video at 30 frames/sec, at bit-rates up to 15 Mb/sec for NTSC video. Another example is the HDTV resolution of 1920x1080 pixels at 30 frame/sec, at a bit-rate of up to 80 Mb/sec. This is an example of the Main Profile, High Level (MP@HL) descriptor.

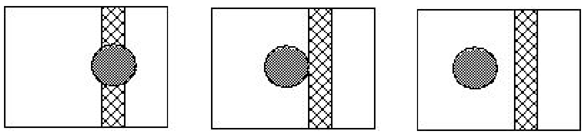
9.4.1 MPEG Video

MPEG compression is essentially a attempts to over come some shortcomings of H.261 and JPEG:

- Recall H.261 dependencies:

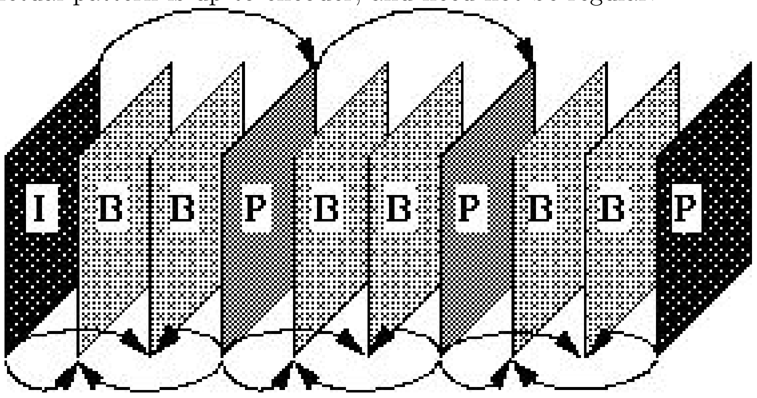


- The Problem here is that many macroblocks need information is **not** in the reference frame.
- For example:



- The *MPEG solution* is to add a third frame type which is a bidirectional frame, or *B-frame*
- B-frames search for macroblock in *past* and *future* frames.
- Typical pattern is IBBPBBPBB IBBPBBPBB IBBPBBPBB

  Actual pattern is up to encoder, and need not be regular.



MPEG Video Layers

MPEG video is broken up into a hierarchy of layers to help with error handling, random search and editing, and synchronization, for example with an audio bitstream. From the top level, the first layer is known as the video sequence layer, and is any self-contained bitstream, for example a coded movie or advertisement. The second layer down is the group of pictures, which is composed of 1 or more groups of intra (I) frames and/or non-intra (P and/or B) pictures that will be defined later. Of course the third layer down is the picture layer itself, and the next layer beneath it is called the slice layer. Each slice is a contiguous sequence of raster ordered macroblocks, most often on a row basis in typical video applications, but not limited to this by the specification. Each slice consists of macroblocks, which are 16x16 arrays of luminance pixels, or picture data elements, with 2 8x8 arrays of associated chrominance pixels. The macroblocks can be further divided into distinct 8x8 blocks, for further processing such as transform coding. Each of these layers has its own unique 32 bit start code defined in the syntax to consist of 23 zero bits followed by a one, then followed by 8 bits for the actual start code. These start codes may have as many zero bits as desired preceding them.

B-Frames

The MPEG encoder also has the option of using forward/backward interpolated prediction. These frames are commonly referred to as bi-directional interpolated prediction frames, or B frames for short. As an example of the usage of I, P, and B frames, consider a group of pictures that lasts for 6 frames, and is given as I,B,P,B,P,B,I,B,P,B,P,B, As in the previous I and P only example, I frames are coded spatially only and the P frames are forward predicted based on previous I and P frames. The B frames however, are coded based on a forward prediction from a previous I or P frame, as well as a backward prediction from a succeeding I or P frame. As such, the example sequence is processed by the encoder such that the first B frame is predicted from the first I frame and first P frame, the second B frame is predicted from the second and third P frames, and the third B frame is predicted from the third P frame and the first I frame of the next group of pictures. From this example, it can be seen that backward prediction requires that the future frames that are to be used for backward prediction be encoded and transmitted first, out of order. This process is summarized in Figure 9.7. There is no defined limit to the number of consecutive B frames that may be used in a group of pictures, and of course the optimal number is application dependent. Most broadcast quality applications however, have tended to use 2 consecutive B frames (I,B,B,P,B,B,P,) as the ideal trade-off between compression efficiency and video quality.

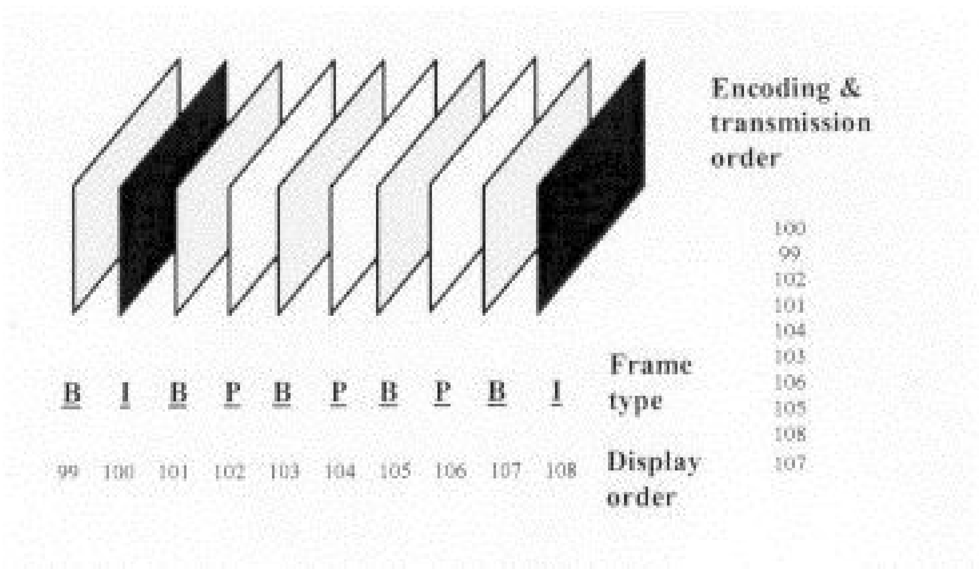


Figure 9.7: B-Frame Encoding

The main advantage of the usage of B frames is coding efficiency. In most cases, B frames will result in less bits being coded overall. Quality can also be improved in the case of moving objects that reveal hidden areas within a video sequence. Backward prediction in this case allows the encoder to make more intelligent decisions on how to encode the video within these areas. Also, since B frames are not used to predict future frames, errors generated will not be propagated further within the sequence.

One disadvantage is that the frame reconstruction memory buffers within the encoder and decoder must be doubled in size to accommodate the 2 anchor frames. This is almost never an issue for the relatively expensive encoder, and in these days of inexpensive DRAM it has become much less of an issue for the decoder as well. Another disadvantage is that there will necessarily be a delay throughout the system as the frames are delivered out of order as was shown in Figure **??**. Most one-way systems can tolerate these delays, as they are more objectionable in applications such as video conferencing systems.

Motion Estimation

The temporal prediction technique used in MPEG video is based on motion estimation. The basic premise of motion estimation is that in most cases, consecutive video frames will be similar except for changes induced by objects moving within the frames. In the trivial case of zero motion between frames (and no other differences caused by noise, etc.), it is easy for the encoder to efficiently predict the current frame as a duplicate of the prediction frame. When this is done, the only information necessary to transmit to the decoder becomes the syntactic overhead necessary to reconstruct the picture from the original reference frame. When there is motion in the images, the situation is not as simple.

Figure 9.8 shows an example of a frame with 2 stick figures and a tree. The second half of this figure is an example of a possible next frame, where panning has resulted in the tree moving down and to the right, and the figures have moved farther to the right because of their own movement outside of the panning. The problem for motion estimation to solve is how to adequately represent the changes, or differences, between these two video frames.

The way that motion estimation goes about solving this problem is that a comprehensive 2-dimensional spatial search is performed for each luminance macroblock. Motion estimation is not applied directly to chrominance in MPEG video, as it is assumed that the color motion can be adequately represented with the same motion information as the luminance. It should be noted at this point that MPEG does not define how this search should be performed. This is a detail that the system designer can choose to implement in one of many possible ways. This is similar to the bit-rate control algorithms discussed previously, in the respect that complexity vs. quality issues need to be addressed relative to the individual application. It is well known that a full, exhaustive search over a wide 2-dimensional area yields the best matching results in most cases, but this performance comes at an extreme computational cost to the encoder. As motion estimation usually is the most computationally expensive portion of the video encoder, some lower cost encoders might choose to limit the pixel search

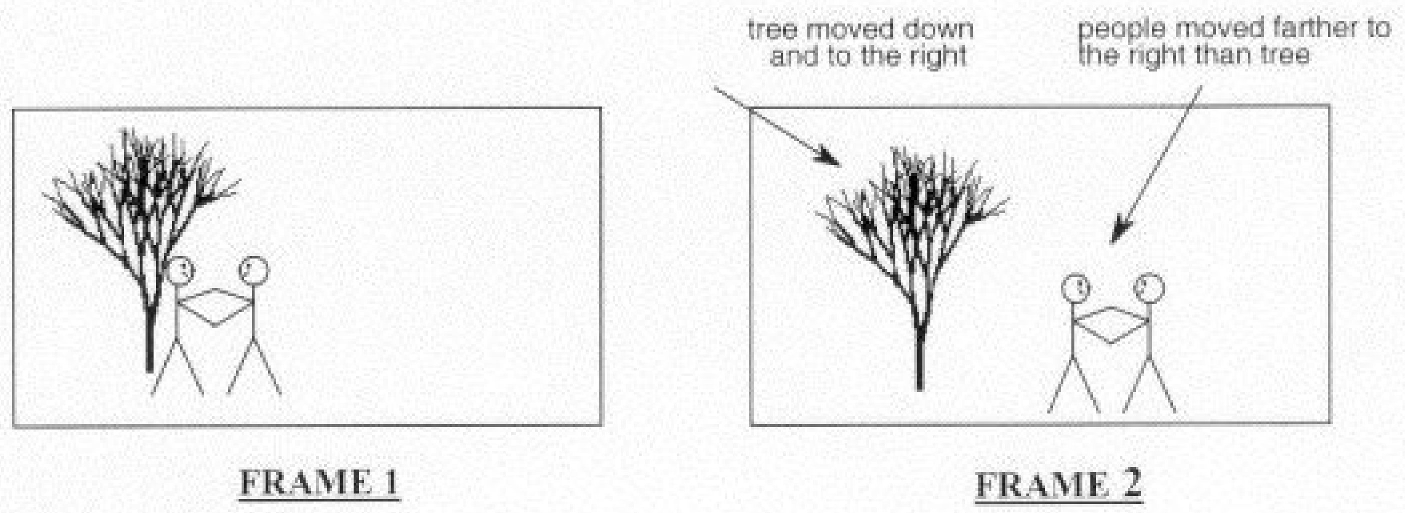

tree moved down and to the right
people moved farther to the right than tree
FRAME 1
FRAME 2

Figure 9.8: Motion Estimation Example

range, or use other techniques such as telescopic searches, usually at some cost to the video quality.

Figure 9.9 shows an example of a particular macroblock from Frame 2 of Figure 9.8, relative to various macroblocks of Frame 1. As can be seen, the top frame has a bad match with the macroblock to be coded. The middle frame has a fair match, as there is some commonality between the 2 macroblocks. The bottom frame has the best match, with only a slight error between the 2 macroblocks. Because a relatively good match has been found, the encoder assigns motion vectors to the macroblock, which indicate how far horizontally and vertically the macroblock must be moved so that a match is made. As such, each forward and backward predicted macroblock may contain 2 motion vectors, so true bidirectionally predicted macroblocks will utilize 4 motion vectors.

Figure 9.10 shows how a potential predicted Frame 2 can be generated from Frame 1 by using motion estimation. In this figure, the predicted frame is subtracted from the desired frame, leaving a (hopefully) less complicated residual error frame that can then be encoded much more efficiently than before motion estimation. It can be seen that the more accurate the motion is estimated and matched, the more likely it will be that the residual error will approach zero, and the coding efficiency will be highest. Further coding efficiency is accomplished by taking advantage of the fact that motion vectors tend to be highly correlated between macroblocks. Because of this, the horizontal component is

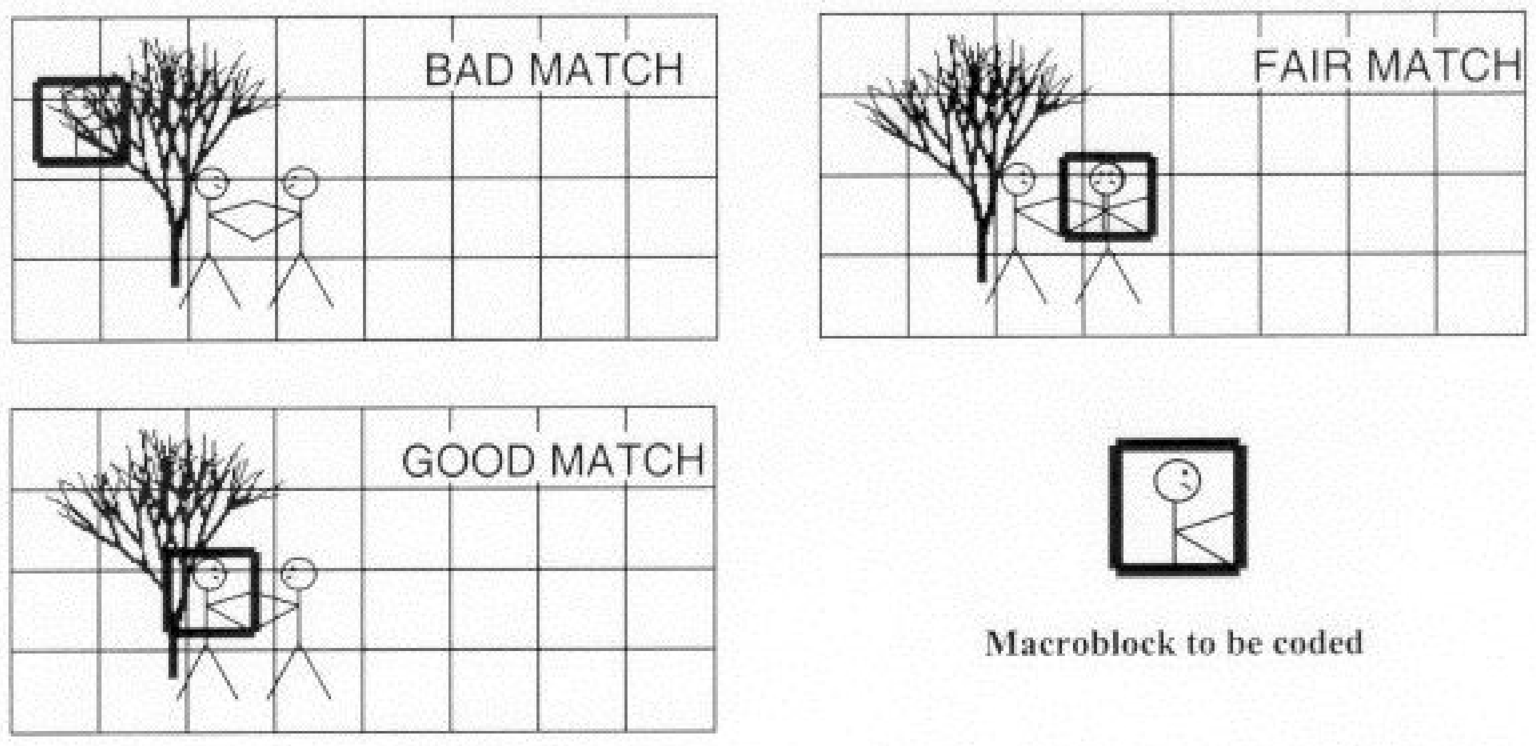


Figure 9.9: Motion Estimation Macroblock Example

compared to the previously valid horizontal motion vector and only the difference is coded. This same difference is calculated for the vertical component before coding. These difference codes are then described with a variable length code for maximum compression efficiency.

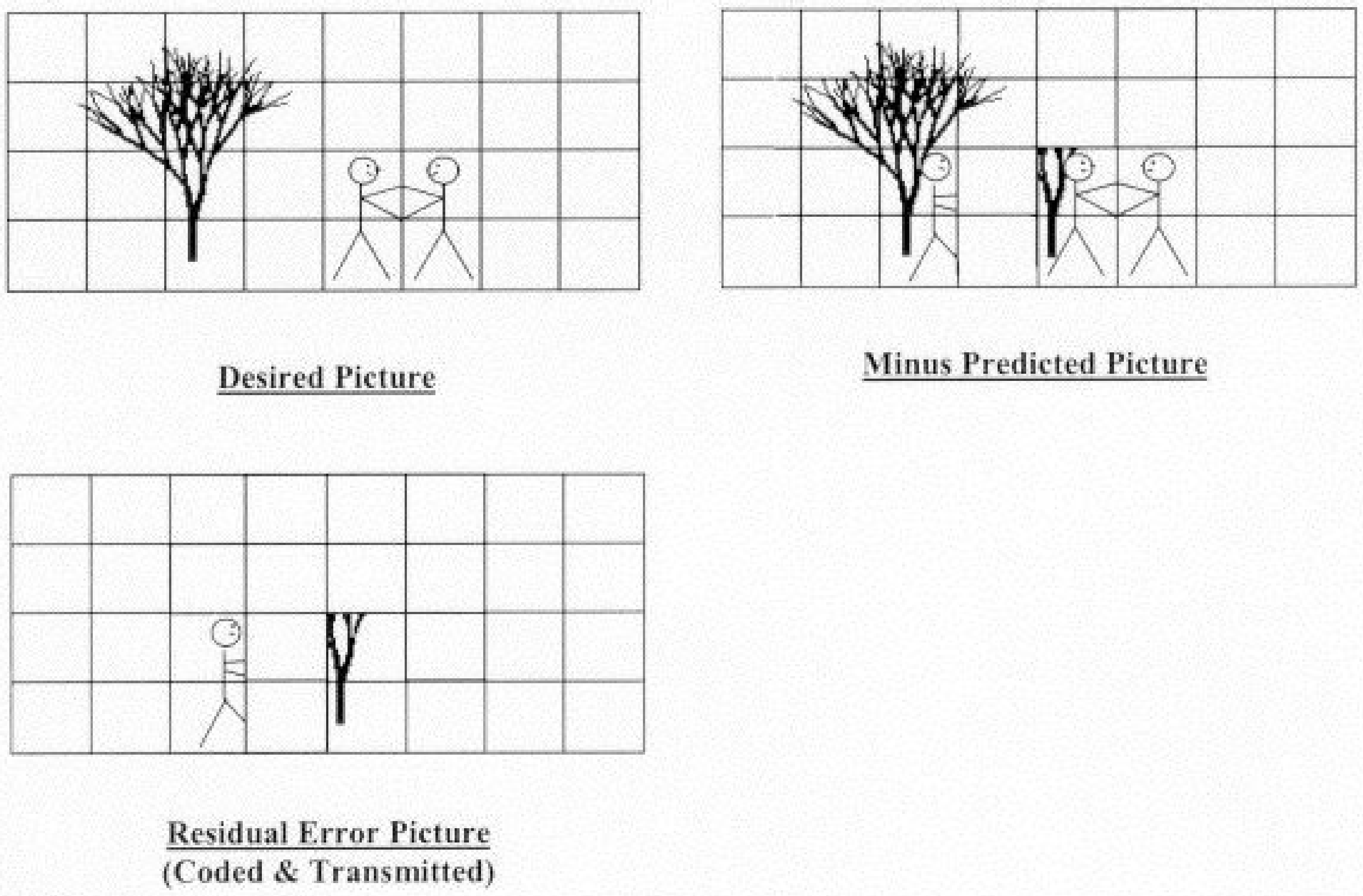


Figure 9.10: Final Motion Estimation Prediction

Of course not every macroblock search will result in an acceptable match. If the encoder decides that no acceptable match exists (again, the "acceptable" criterion is not MPEG defined, and is up to the system designer) then it has the option of coding that particular macroblock as an intra macroblock, even though it may be in a P or B frame. In this manner, high quality video is maintained at a slight cost to coding efficiency.

Estimating the Motion Vectors

Basic Ideas is to search for Macroblock (MB) Within a $\pm$ nxm pixel search window and work out Sum of Absolute Difference (SAD) (or Mean Absolute Error (MAE) for each window but this is computationally more expensive) is a minimum.

Where SAD is given by:

```
For i = -n to +n
  For j = -m to +m
```

$$SAD(i,j) = \Sigma_{k=0}^{l=N-1}\Sigma_{j=0}^{j=N-1} \mid C(x+k,y+l) - R(x+i+k,y+j+l) \mid$$

Here N = is size of Macroblock window typically (16 or 32 pixels), (x, y) the position of the original MB, C, and R is the region to compute the SAD.

It is sometimes applicable for an alpha mask to be applied to SAD calculation to mask out certain pixels.

$$SAD(i,j) = \Sigma_{k=0}^{l=N-1}\Sigma_{j=0}^{j=N-1} \mid C(x+k,y+l) - R(x+i+k,y+j+l) \mid *(!alpha_c(i,j) = 0)$$

So for a $\pm$ 2x2 Search Area is given by dashed lines and a 2x2 Macroblock window example, the SAD is given by bold dot dash line (near top right corner) in Figure 9.11.

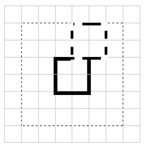


Figure 9.11: SAD Window search Example

Selecting Intra/Inter Frame coding

Based upon the motion estimation a decision is made on whether INTRA or INTER coding is made.

To determine INTRA/INTER MODE we do the following calculation:

$$MB_{mean} = \frac{\Sigma_{i=0,j=0}^{N-1} |C(i,j)|}{N}$$

$$A = \Sigma_{i=0,j=0}^{n,m} \mid C(i,j) - MB_{mean} \mid *(!alpha_c(i,j) = 0)$$

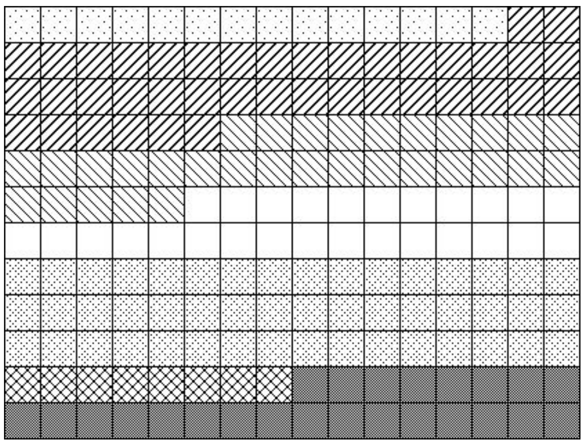
If $A < (SAD - 2N)$ INTRA Mode is chosen.

Coding of Predicted Frames:Coding Residual Errors

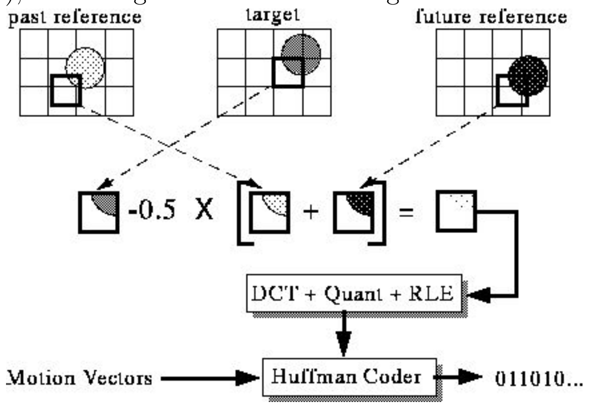
After a predicted frame is subtracted from its reference and the residual error frame is generated, this information is spatially coded as in I frames, by coding 8x8 blocks with the DCT, DCT coefficient quantization, run-length/amplitude coding, and bitstream buffering with rate control feedback. This process is basically the same with some minor differences, the main ones being in the DCT coefficient quantization. The default quantization matrix for non-intra frames is a flat matrix with a constant value of 16 for each of the 64 locations. This is very different from that of the default intra quantization matrix which is tailored for more quantization in direct proportion to higher spatial frequency content. As in the intra case, the encoder may choose to override this default, and utilize another matrix of choice during the encoding process, and download it via the encoded bitstream to the decoder on a picture basis. Also, the non-intra quantization step function contains a dead-zone around zero that is not present in the intra version. This helps eliminate any lone DCT coefficient quantization values that might reduce the run-length amplitude efficiency. Finally, the motion vectors for the residual block information are calculated as differential values and are coded with a variable length code according to their statistical likelihood of occurrence.

Differences from H.261

- Larger gaps between I and P frames, so expand motion vector search range.
- To get better encoding, allow motion vectors to be specified to fraction of a pixel (1/2 pixels).
- Bitstream syntax must allow random access, forward/backward play, etc.
- Added notion of *slice* for synchronization after loss/corrupt data. Example: picture with 7 slices:



- B frame macroblocks can specify *two* motion vectors (one to past and one to future), indicating result is to be averaged.



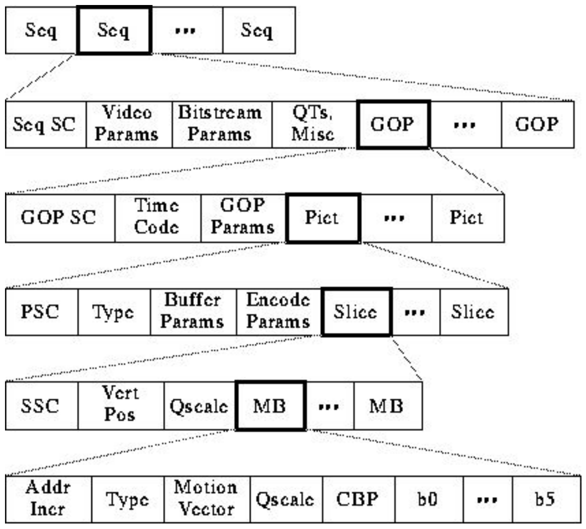
- Compression performance of MPEG 1

Type	Size	Compression
I	18 KB	7:1
P	6 KB	20:1
B	2.5 KB	50:1
Avg	4.8 KB	27:1

9.4.2 The MPEG Video Bitstream

The MPEG Video Bitstream is summarised as follows:

- Public domain tool **mpeg_stat** and **mpeg_bits** will analyze a bitstream.



- Sequence Information

  1. *Video Params* include width, height, aspect ratio of pixels, picture rate.
  2. *Bitstream Params* are bit rate, buffer size, and constrained parameters flag (means bitstream can be decoded by most hardware)
  3. Two types of QTs: one for intra-coded blocks (I-frames) and one for inter-coded blocks (P-frames).

- Group of Pictures (GOP) information

  1. *Time code*: bit field with SMPTE time code (hours, minutes, seconds, frame).
  2. *GOP Params* are bits describing structure of GOP. Is GOP *closed*? Does it have a dangling pointer *broken*?

- Picture Information

  1. *Type*: I, P, or B-frame?
  2. *Buffer Params* indicate how full decoder's buffer should be before starting decode.
  3. *Encode Params* indicate whether half pixel motion vectors are used.

- Slice information

  1. *Vert Pos*: what line does this slice start on?
  2. *QScale*: How is the quantization table scaled in this slice?

- Macroblock information

1. *Addr Incr*: number of MBs to skip.
2. *Type*: Does this MB use a motion vector? What type?
3. *QScale*: How is the quantization table scaled in this MB?
4. *Coded Block Pattern (CBP)*: bitmap indicating which blocks are coded.

9.4.3 Decoding MPEG Video in Software

- Software Decoder goals: portable, multiple display types
- Breakdown of time

Function	% Time
Parsing Bitstream	17.4%
IDCT	14.2%
Reconstruction	31.5%
Dithering	24.5%
Misc. Arith.	9.9%
Other	2.7%

Intra Frame Decoding

To decode a bitstream generated from the encoder of Figure 9.12, it is necessary to reverse the order of the encoder processing. In this manner, an I frame decoder consists of an input bitstream buffer, a Variable Length Decoder (VLD), an inverse quantizer, an Inverse Discrete Cosine Transform (IDCT), and an output interface to the required environment (computer hard drive, video frame buffer, etc.). This decoder is shown in Figure 9.13.

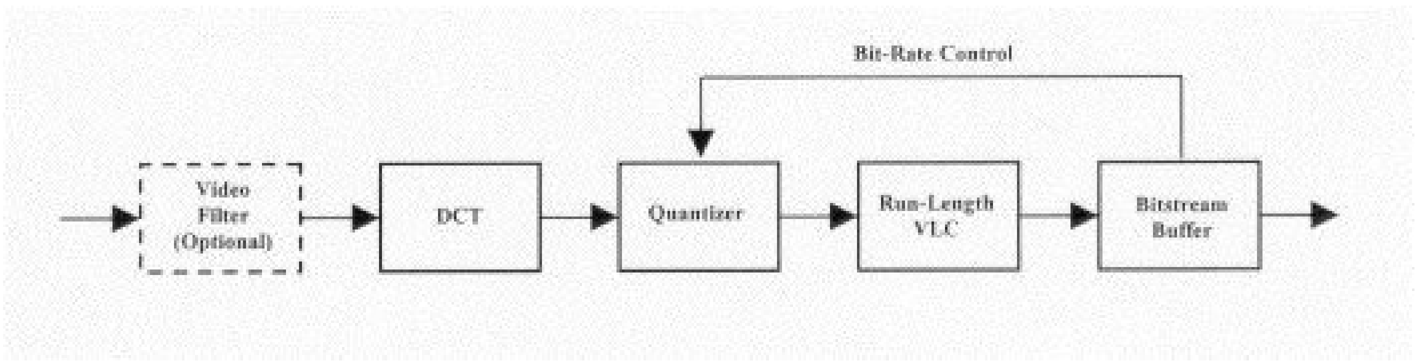


Figure 9.12: Intra Frame Encoding

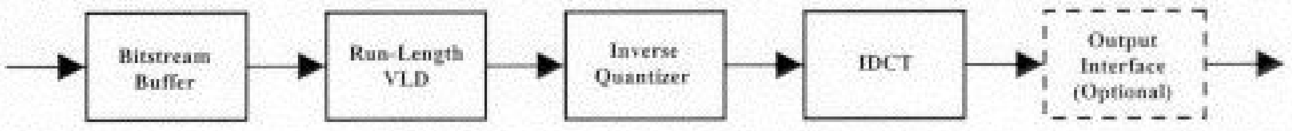


Figure 9.13: Intra Frame Decoding

The input bitstream buffer consists of memory that operates in the inverse fashion of the buffer in the encoder. For fixed bit-rate applications, the constant rate bitstream is buffered in the memory and read out at a variable rate depending on the coding efficiency of the macroblocks and frames to be decoded.

The VLD is probably the most computationally expensive portion of the decoder because it must operate on a bit-wise basis (VLD decoders need to look at every bit, because the boundaries between variable length codes are random and non-aligned) with table look-ups performed at speeds up to the input bit-rate. This is generally the only function in the receiver that is more complex to implement than its corresponding function within the encoder, because of the extensive high-speed bit-wise processing necessary.

The inverse quantizer block multiplies the decoded coefficients by the corresponding values of the quantization matrix and the quantization scale factor. Clipping of the resulting coefficients is performed to the region 2048 to +2047, then an IDCT mismatch control is applied to prevent long term error propagation within the sequence.

Non-Intra Frame Decoding

It was shown previously that the non-intra frame encoder built upon the basic building blocks of the intra frame encoder, with the addition of motion estimation and its associated support structures. This is also true of the non-intra frame decoder, as it contains the same core structure as the intra frame decoder with the addition of motion compensation support. Again, support for intra frame decoding is inherent in the structure, so I, P, and B frame decoding is possible. The decoder is shown in Figure 9.14.

MPEG-2, MPEG-3, and MPEG-4

- MPEG-2 target applications

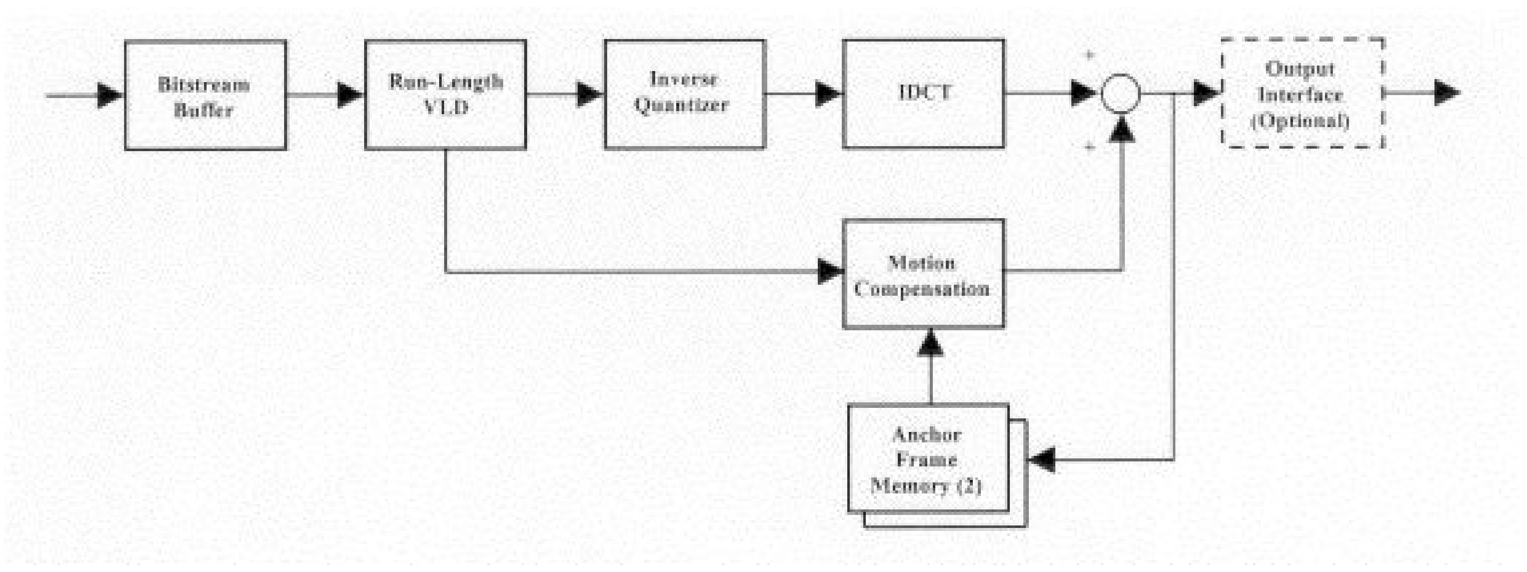


Figure 9.14: Non-Intra Frame Decoding

Level	size	Pixels/sec	bit-rate (Mbits)	Application
Low	352 x 240	3 M	4	consumer tape equiv.
Main	720 x 480	10 M	15	studio TV
High 1440	1440 x 1152	47 M	60	consumer HDTV
High	1920 x 1080	63 M	80	film production

- Differences from MPEG-1
  1. Search on fields, not just frames.
  2. 4:2:2 and 4:4:4 macroblocks
  3. Frame sizes as large as 16383 x 16383
  4. Scalable modes: Temporal, Progressive,...
  5. Non-linear macroblock quantization factor
  6. A bunch of minor fixes (see MPEG FAQ for more details)
- MPEG-3: Originally for HDTV (1920 x 1080), got folded into MPEG-2
- MPEG-4: Originally targeted at very low bit-rate communication (4.8 to 64 kb/sec). Now addressing video processing...

9.4.4 Further Reading/Information

- G.K. Wallace, *The JPEG Still Picture Compression Standard*
- CCITT, *Recommendation H.261*
- D. Le Gall, *MPEG: A Video Compression Standard for Multimedia Applications*
- K. Patel, et. al., *Performance of a Software MPEG Video Decoder*
- P. Cosman, et. al., *Using Vector Quantization for Image Processing*
- "Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s," ISO/IEC 11172-2: Video (November 1991).
- "Generic Coding of Moving Pictures and Associated Audio Information: Video," ISO/IEC 13818-2 : Draft International Standard (November 1994).
- Barry G. Haskell, Atul Puri, Arun N. Netravali, Digital Video: An Introduction to MPEG-2, Chapman and Hall, 1997.
- K.R. Rao, P. Yip, Discrete Cosine Transform Algorithms, Advantages, Applications, Academic Press, Inc., 1990.
- Majid Rabbani, Paul W. Jones, Digital Image Compression Techniques, SPIE Optical Engineering Press, 1991.
- Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, Didier J. LeGall, MPEG Video Compression Standard, Chapman and Hall, 1997.
- IEEE Micro Magazine Media Processing, IEEE Computer Society, Volume 16 Number 4, August 1996.
- *http://www.mpeg.org/MPEG/* — MPEG Resources on the Web.
- *http://drogo.cselt.stet.it/mpeg/* — The Official MPEG Committee

9.5 Audio Compression

As with video a number of compression techniques have been applied to audio.

9.5.1 Simple Audio Compression Methods

Traditional lossless compression methods (Huffman, LZW, etc.) usually don't work well on audio compression (the same reason as in image compression).

The following are some of the Lossy methods applied to audio compression:

- Silence Compression - detect the "silence", similar to run-length coding
- Adaptive Differential Pulse Code Modulation (ADPCM)

  e.g., in CCITT G.721 – 16 or 32 Kbits/sec.

  (a) encodes the difference between two consecutive signals,

  (b) adapts at quantization so fewer bits are used when the value is smaller.

  - It is necessary to predict where the waveform is headed –> difficult
  - Apple has proprietary scheme called ACE/MACE. Lossy scheme that tries to predict where wave will go in next sample. About 2:1 compression.
- Linear Predictive Coding (LPC) fits signal to speech model and then transmits parameters of model. Sounds like a computer talking, 2.4 kbits/sec.
- Code Excited Linear Predictor (CELP) does LPC, but also transmits error term – audio conferencing quality at 4.8 kbits/sec.

9.5.2 Psychoacoustics

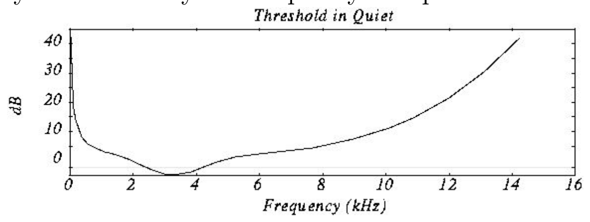
These methods are related to how humans actually hear sounds:

Human hearing and voice

- Range is about 20 Hz to 20 kHz, most sensitive at 2 to 4 KHz.
- Dynamic range (quietest to loudest) is about 96 dB
- Normal voice range is about 500 Hz to 2 kHz
  - Low frequencies are vowels and bass
  - High frequencies are consonants

Question: How sensitive is human hearing?

- Experiment: Put a person in a quiet room. Raise level of 1 kHz tone until just barely audible. Vary the frequency and plot



Frequency Masking

Question: Do receptors interfere with each other?

- Experiment: Play 1 kHz tone (*maskingtone*) at fixed level (60 dB). Play *test tone* at a different level (e.g., 1.1kHz), and raise level until just distinguishable.
- Vary the frequency of the test tone and plot the threshold when it becomes audible:
- Repeat for various frequencies of masking tones

Critical Bands

- Perceptually uniform measure of frequency, non-proportional to width of masking curve

  About 100 Hz for masking frequency $<$ 500 Hz, grow larger and larger above 500 Hz.
- The width is called the size of the *critical band*

Barks

- Introduce new unit for frequency called a *bark* (after Barkhausen)

  1 Bark = width of one critical band

  For frequency $<$ 500 Hz,

  For frequency $>$ 500 Hz,
- Masking Thresholds on critical band scale:

Temporal masking

- If we hear a loud sound, then it stops, it takes a little while until we can hear a soft tone nearby
- Question: how to quantify?

- Experiment: Play 1 kHz *masking tone* at 60 dB, plus a *test tone* at 1.1 kHz at 40 dB. Test tone can't be heard (it's masked).

  Stop masking tone, then stop test tone after a short delay.

  Adjust delay time to the shortest time that test tone can be heard (e.g., 5 ms).

  Repeat with different level of the test tone and plot:

- Try other frequencies for test tone (masking tone duration constant). Total effect of masking

Summary

- If we have a loud tone at, say, 1 kHz, then nearby quieter tones are masked.
- Best compared on critical band scale – range of masking is about 1 critical band
- Two factors for masking – frequency masking and temporal masking
- Question: How to use this for compression?

9.5.3 MPEG Audio Compression

Some facts

- MPEG-1: 1.5 Mbits/sec for audio and video

  About 1.2 Mbits/sec for video, 0.3 Mbits/sec for audio

  (Uncompressed CD audio is 44,100 samples/sec * 16 bits/sample * 2 channels $>$ 1.4 Mbits/sec)

- Compression factor ranging from 2.7 to 24.
- With Compression rate 6:1 (16 bits stereo sampled at 48 KHz is reduced to 256 kbits/sec) and optimal listening conditions, expert listeners could not distinguish between coded and original audio clips.
- MPEG audio supports sampling frequencies of 32, 44.1 and 48 KHz.
- Supports one or two audio channels in one of the four modes:
  1. Monophonic – single audio channel
  2. Dual-monophonic – two independent channels (similar to stereo)
  3. Stereo – for stereo channels that share bits, but not using joint-stereo coding
  4. Joint-stereo – takes advantage of the correlations between stereo channels

Steps in algorithm:

1. Use convolution filters to divide the audio signal (e.g., 48 kHz sound) into frequency subbands that approximate the 32 critical bands –> *sub-band filtering*.
2. Determine amount of masking for each band caused by nearby band using the results shown above (this is called the *psychoacoustic model*).
3. If the power in a band is below the masking threshold, don't encode it.
4. Otherwise, determine number of bits needed to represent the coefficient such that noise introduced by quantization is below the masking effect (Recall that 1 bit of quantization introduces about 6 dB of noise).
5. Format bitstream

Example:

- After analysis, the first levels of 16 of the 32 bands are these:

Band	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Level (db)	0	8	12	10	6	2	10	60	35	20	15	2	3	5	3	1

- If the level of the 8th band is 60dB,

  it gives a masking of 12 dB in the 7th band, 15dB in the 9th.

  Level in 7th band is 10 dB (< 12 dB), so ignore it.

  Level in 9th band is 35 dB (> 15 dB), so send it.

  –> Can encode with up to 2 bits (= 12 dB) of quantization error.

MPEG Layers

- MPEG defines 3 layers for audio. Basic model is same, but codec complexity increases with each layer.
- Divides data into frames, each of them contains 384 samples, 12 samples from each of the 32 filtered subbands as shown below.

  Figure: Grouping of Sub-band Samples for Layer 1, 2, and 3
- Layer 1: DCT type filter with one frame and equal frequency spread per band. Psychoacoustic model only uses frequency masking.
- Layer 2: Use three frames in filter (before, current, next, a total of 1152 samples). This models a little bit of the temporal masking.
- Layer 3: Better critical band filter is used (non-equal frequencies), psychoacoustic model includes temporal masking effects, takes into account stereo redundancy, and uses Huffman coder.

Effectiveness of MPEG audio

Layer	Target bitrate	Ratio	Quality @ 64 kbits	Quality @ 128 kbits	Theoretical Min. Delay
Layer 1	192 kbit	4:1	---	---	19 ms
Layer 2	128 kbit	6:1	2.1 to 2.6	4+	35 ms
Layer 3	64 kbit	12:1	3.6 to 3.8	4+	59 ms

- 5 = perfect, 4 = just noticeable, 3 = slightly annoying, 2 = annoying, 1 = very annoying
- Real delay is about 3 times theoretical delay

9.5.4 Streaming Audio (and video)

Popular new delivery medium for the Web and other Multimedia networks

Real Audio (*http://www.realaudio.com/*), Shockwave (*http://www.macromedia.com*) and .wav files are examples of streamed audio (and video)

- Buffered Data:
  - Trick get data to destination before it's needed
  - Temporarily store in memory (Buffer)
  - Server keeps feeding the buffer
  - Client Application reads buffer
- Needs Reliable Connection, moderately fast too.
- Specialised client, Steaming Audio Protocol (PNM for real audio).

9.5.5 Further Exploration

- *http://www.raum.com/mpeg/* — MPEG Audio Page
- *http://www.hitsquad.com/smm/news/9903_109/?nl9905* — MP3 Beginner's Guide

A good article on this subject is:

- "A Tutorial on MPEG/Audio Compression", Davis Pan, *IEEE Multimedia*, pp. 60-74, 1995.

See Video MPEG Further Reading Resources above also

9.6 Audio Compression

As with video a number of compression techniques have been applied to audio.

9.6.1 Simple Audio Compression Methods

Traditional lossless compression methods (Huffman, LZW, etc.) usually don't work well on audio compression (the same reason as in image and video compression).

The following are some of the simple methods applied to audio compression, we have seen the basic methods working (sometimes with audio examples) previously:

- Silence Compression - detect the "silence", similar to run-length coding (seen example before)
- Differential Pulse Code Modulation (DPCM)

  Relies on the fact that difference in amplitude in successive samples is small then we can used reduced bits to store the difference (seen examples before)
- Adaptive Differential Pulse Code Modulation (ADPCM)

  e.g., in CCITT G.721 – 16 or 32 Kbits/sec.

  (a) encodes the difference between two consecutive signals but a refinement on DPCM, (b) adapts at quantisation so fewer bits are used when the value is smaller.

  – It is necessary to predict where the waveform is headed –> difficult
  – Apple had a proprietary scheme called ACE/MACE. Lossy scheme that tries to predict where wave will go in next sample. About 2:1 compression.
- Adaptive Predictive Coding (APC) typically used on Speech.

  – Input signal is divided into fixed segments (*windows*)
  – For each segment, some sample *characteristics* are computed, *e.g. pitch, period, loudness.*
  – These characteristics are used to predict the signal
  – Computerised talking (Speech Synthesisers use such methods) but low bandwidth: acceptable quality at 8 kbits/sec
- Linear Predictive Coding (LPC) fits signal to speech model and then transmits parameters of model as in APC.

  Speech Model:

  - Speech Model: pitch, period, loudness, vocal tract parameters (voiced and unvoiced sounds).
  - synthesised speech
  - Still sounds like a computer talking,
  - Bandwidth as low as 2.4 kbits/sec.

- Code Excited Linear Predictor (CELP) does LPC, but also transmits error term.
  - Based on more sophisticated model of vocal tract than LPC
  - Better perceived speech quality
  - audio conferencing quality at 4.8 kbits/sec.

9.6.2 Psychoacoustics or Perceptual Coding

Just as we have seen with image (JPEG) and Video compression if we can employ some knowledge of how humans perceive sound, we can then exploit areas where the human ear is *less sensitive* to sound to achieve compression. MPEG audio works on this principal.

How do we hear sound?

Sound is produced by a vibrating source. The vibrations disturb air molecules and thus produce variations in air pressure: lower than average pressure, *rarefactions*, and higher than average, *compressions.* This produces sound waves.

When a sound wave impinges on a surface (*e.g.* eardrum or microphone) it causes the surface to vibrate in sympathy:

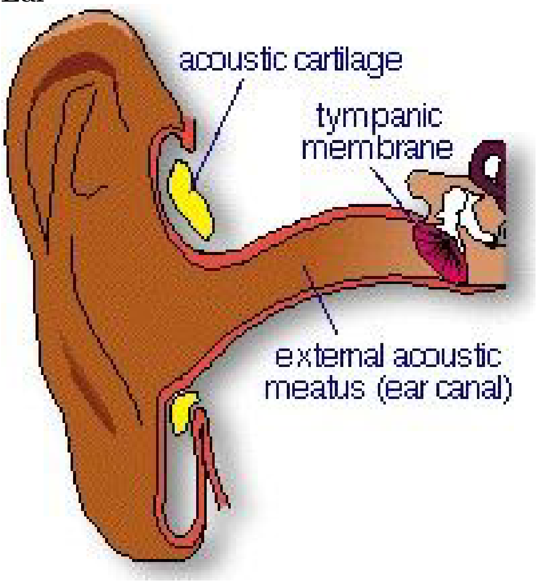
In this way *acoustic energy* is transferred from a source to a receptor.

So in human hearing, upon receiving the the waveform the eardrum vibrates in sympathy and through a variety of mechanisms the acoustic energy is transferred to nerve impulses that the brain interprets as sound.

The ear can be regarded as being made up of 3 parts: The outer, middle and inner ear.

Let us now consider the function of the main parts of the ear and in particular how the transmission of sound is processed.

The Outer Ear

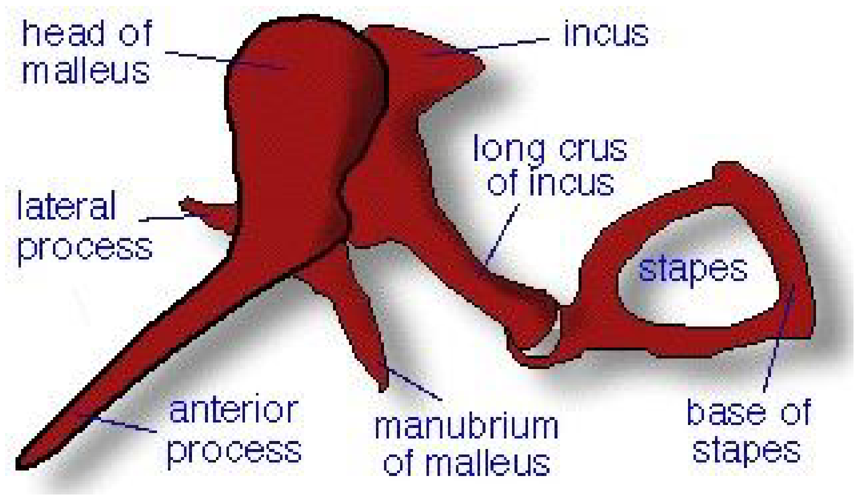


The outer ear has the following main components:

Ear Canal: The ear canal is open at the outer end which is surrounded by the pinna or auricle (below left). The pinna plays an important spacial focusing role in hearing. The canal then narrows slightly and widens toward its inner end, which is sealed off by the eardrum, or tympanic membrane. Thus the canal is a shaped tube enclosing a resonating column of air - with the combination of open and closed ends. This makes it rather like an organ pipe.

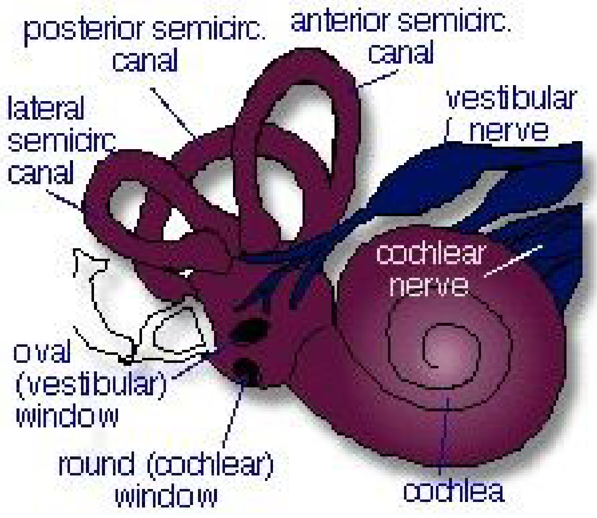
Eardrum (Tympanic Membrane): This acts as the interface between the external and middle ear. At this point, sound is converted into mechanical vibrations in the solid materials of the middle ear. Sounds (air pressure waves) first set up sympathetic vibrations on the membrane of the eardrum, just as they do in the diaphragm of some types of microphone. The eardrum passes these vibrations on to the middle ear structure.

The Middle Ear



The middle ear contains three small bones which are known collectively as the *ossicles* and individually as the *Malleus*, *Incus*, and *Stapes*. These bones form a system of levers which are linked together and driven by the eardrum; Malleus pushing Incus, Incus pushing Stapes. Working together as a lever system, the bones amplify the force of sound vibrations.

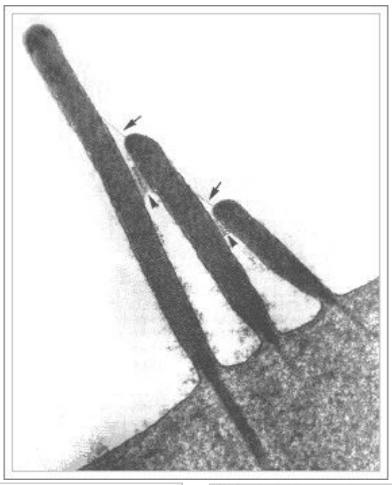
The Inner Ear

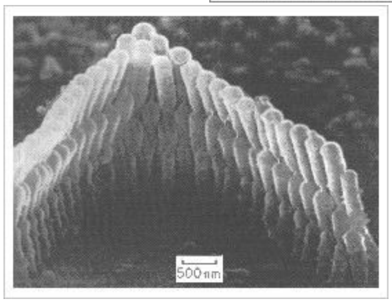


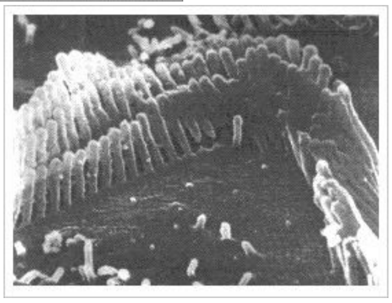
The Cochlea: The amplified mechanical force transmitted from the middle ear to the inner ear by the ossicles is immediately transformed in the cochlea into hydraulic pressure, as the cochlea is filled with fluid. This hydraulic pressure imparts movement to the cochlear duct and to the organ of Corti. It is staggering to note that this process is accomplished in the cochlea which is no bigger than the tip of a little finger.

How the Cochlea Works: The pressure waves in the cochlea exert energy along a route that begins at the oval window and ends abruptly at the membrane-covered round window, where the pressure is dissipated. In keeping with the principles of hydraulics, the pressure applied to the oval window at the stirrup (stape) is transmitted to all parts of the cochlea.

In addition to being filled with fluid, the inner surface of the cochlea (*, the basilar membrane*) is lined with over 20,000 hair-like nerve cells, called stereocilia, which perform one of the most critical roles in our ability to hear.







Hydraulic pressure waves in the cochlea induce a wave-like ripple in the basilar membrane which travels from the tight towards the loose end. High tones create their greatest crests where the membrane is tight, low tones where the wall is slack. This is because resonant frequency is correlated with tension as in a tight string. The position of this crest is important because it determines which nerve fibres will send signals to the brain. High frequency tones cause the crest to occur at the base of the cochlea and the lower frequencies toward the apex.

The stereocilia differ in length by minuscule amounts; they also have different degrees of resiliency to the fluid which passes over them. As a compressional wave moves from the interface between the hammer of the middle ear and the oval window of the inner ear through the cochlea, the stereocilia will be set in motion, vibrating in sympathy. Each hair cell has a natural sensitivity to a particular frequency of vibration. When the frequency of the compressional wave matches the natural frequency of the stereocilia, that stereocilia will resonate with a larger amplitude of vibration. This increased vibrational amplitude induces the stereocilia cell to release an electrical impulse which passes along the auditory nerve towards the brain. In a process which is not clearly understood, the brain is capable of interpreting the qualities of the sound upon reception of these electric nerve impulses.

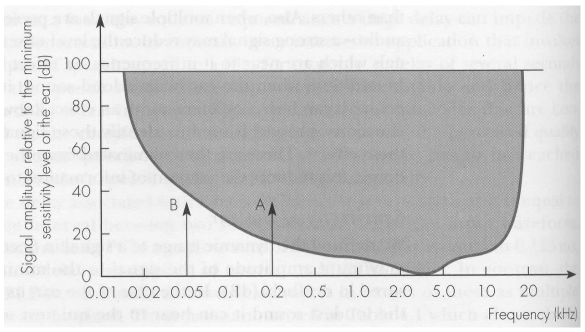
The other parts of the inner ear, *the semicircular canals* are the body's balance mechanism and it is thought that it plays no part in hearing. One of the two walls of the cochlear duct is the vibrating basilar membrane whose function is to separate sounds according to frequency. The membrane is narrow and tight at the end near the stirrup and wider and more pliant at the other.

Sensitivity of the Ear

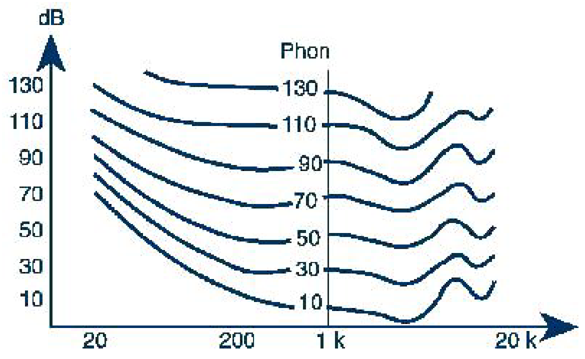
- Range is about 20 Hz to 20 kHz, most sensitive at 2 to 4 KHz.
- Dynamic range (quietest to loudest) is about 96 dB
- Approximate threshold of pain: 130 dB
- Hearing damage: > 90 dB (prolonged exposure)
- Normal conversation: 60-70 dB
- Typical classroom background noise: 20-30 dB
- Normal voice range is about 500 Hz to 2 kHz
  - Low frequencies are vowels and bass
  - High frequencies are consonants

Question: How sensitive is human hearing?

The sensitivity of the human ear with respect to frequency is given by the following graph.



But, it's even more complicated than this. The frequency dependence is also level dependent! To illustrate this rather complex phenomenon, we often use a set of graphs called Loudness Curves or Fletcher-Munson Curves (named after those who originally made these back in the 1930s).



These curves shown here indicate how the perceived loudness is a function of both the frequency and the level of a sinusoidal sound signal. The equal loudness curves are contours of equal loudness and express how much a sound level must be changed as the frequency varies, to maintain a certain perceived loudness

The unit used is called phon. At 1 kHz the phon and the dB SPL values are identical. As frequency varies, the phon follows the contour curve while the dB remains constant.

What do the curves mean?

They tell that for some frequency regions a sound must be increased in level to appear equally loud as a sound of 1 kHz, while for other ranges, the sound must be attenuated to maintain equal perceived loudness.

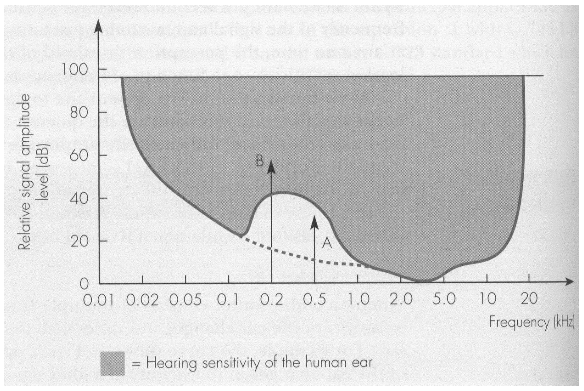
The frequency-range your hearing accentuates, happens to coincide with the frequency range in which very important lingual sounds have their major spectral contents. Typically, sounds like "p" and "t" have very important parts of their spectral energy within the *accentuated* range, making them more easy to discriminate between.

The ability to hear sounds of the *accentuated* range (around a few kHz) is thus vital for speech communication.

Frequency Masking

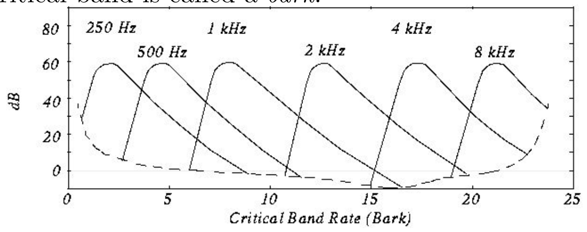
Frequency Masking: When an Audio signal consists of multiple frequencies the sensitivity of the ear changes with the relative amplitude of the signals. If the frequencies are close and the amplitude of one is less than the other close frequency then the second frequency may not be heard.

For example in the figure below



Note how the sensitivity of the ear changes in the vicinity of the loud signal B. Signal A is smaller in amplitude but close in frequency. The basic sensitivity is distorted in the region of B so as a result signal A will no longer be heard.

The range of closeness for frequency masking (*the Critical Bands*) depends on the frequencies and relative amplitudes. The critical bandwidth for average human hearing is a constant 100 Hz for frequencies less than 500 Hz and increases (approximately) linearly by 100 Hz for each additional 500 Hz. The width of the critical band is called a *bark*.



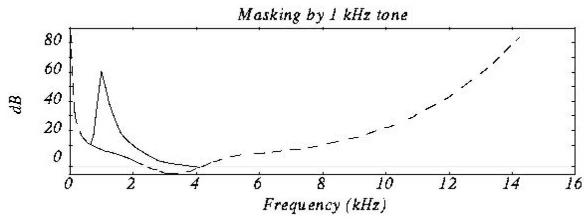
What is the cause of Frequency Masking?

As we have seen above, within the human (inner) ear there are the stereocilia. These are excited by air pressure variations, transmitted via outer and middle ear. We also noted that different stereocilia respond to different ranges of frequencies, the critical bands

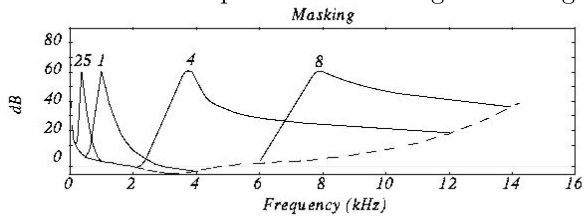
Frequency Masking occurs because after excitation by one frequency further excitation by a less strong similar frequency of the same group of cells is not possible.

- Example: Play 1 kHz tone (*maskingtone*) at fixed level (60 dB). Play *test tone* at a different level (e.g., 1.1 kHz), and raise level until just distinguishable.

- Vary the frequency of the test tone and plot the threshold when it becomes audible:



- If we repeat for various frequencies of masking tones we get:



Temporal masking

Temporal Masking: After the ear hears a loud sound it takes a further short while before it can hear a quieter sound.

Why is this so?

Stereocilia vibrate with corresponding force of input sound stimuli. If the stimuli is strong then the stereocilia will be in a high state of excitation and get fatigued. After extended listening to loud music or headphones this sometime manifests itself with ringing in the ears and even temporary deafness. Prolonged exposure to noise permanently damages the *Stereocilia.*

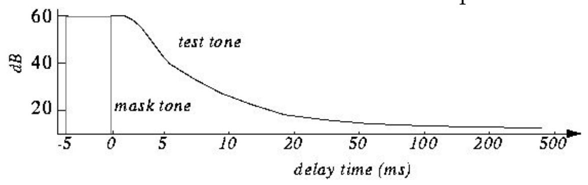
Temporal Masking occurs because the hairs take time to settle after excitation to respond again.

- Example: Play 1 kHz *masking tone* at 60 dB, plus a *test tone* at 1.1 kHz at 40 dB. Test tone can't be heard (it's masked).

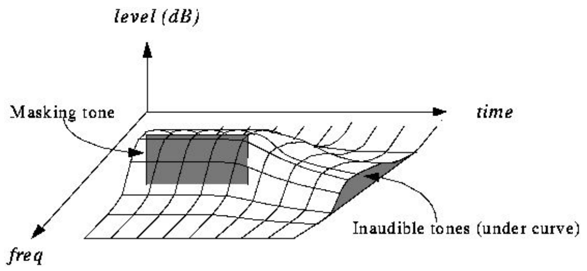
  Stop masking tone, then stop test tone after a short delay.

  Adjust delay time to the shortest time that test tone can be heard (e.g., 5 ms).

  Repeat with different level of the test tone and plot:



- Try other frequencies for test tone (masking tone duration constant). Total effect of masking:



Summary

- If we have a loud tone at, say, 1 kHz, then nearby quieter tones are masked.
- Best compared on critical band scale – range of masking is about 1 critical band
- Two factors for masking – frequency masking and temporal masking
- Question: How to use this for compression? Two examples:
  - MPEG Audio
  - Dolby

How to compute?

We have met basic tools:

- Fourier and Discrete Cosine Transforms
- Work in frequency space
- Band Pass Filtering — Visualise a graphic equaliser

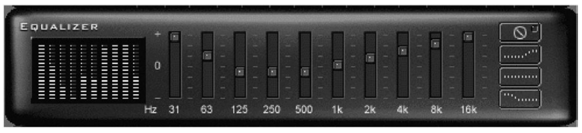


Figure 9.15: A Graphic Equaliser

9.6.3 MPEG Audio Compression

MPEG audio compression basically aims to exploit the psychoacoustic models described above. Frequency masking is always utilised and more complex forms of MPEG also employ temporal masking.

MPEG audio compression basically works by dividing the audio signal up into a set of frequency subbands (that approximate *critical bands*). Each band is then quantised according to the *audibility of quantisation noise* within that band. Quantisation is the key to MPEG audio compression and is the reason why it is lossy. Although MPEG *claims* to be perceptually lossless:

- Human tests (part of standard development), Expert listeners.
- 6-1 compression ratio, stereo 16 bit samples at 48 Khz compressed to 256 kbits/sec
- *Difficult*, real world examples.
- *Under Optimal listening conditions* no statistically distinguishable difference between original and MPEG.

Basic MPEG

MPEG defines a set of standards for for use of video **with** sound. The compression methods or *coders* associated with audio compression are called *MPEG audio coders.*

The MPEG standard allows for a variety of different coders to employed. The difference is their level of sophistication in applying perceptual compression ideas. There are different *layers*. One key aspect of MPEG is that whilst some complex psychoacoustic modelling may be applied in *coding phase* this modelling is **not** required in the *decoding* phase. This is a desirable feature as this allows for real time (Hardware or software) decompression which is essential for broadcast purposes. This also means decompression is independent of the psychoacoustic models used: different models can be used or if there is enough bandwidth no models at all.

There have also been evolving standards for MPEG audio compression. MPEG-1 is by the most prevalent but as we have already seen (Section 6.3.7) the standard now extends to MPEG-4 (structured audio). For now we concentrate on MPEG-1. We will discuss aspects of others later.

Some basic facts:

- MPEG-1: 1.5 Mbits/sec for audio and video

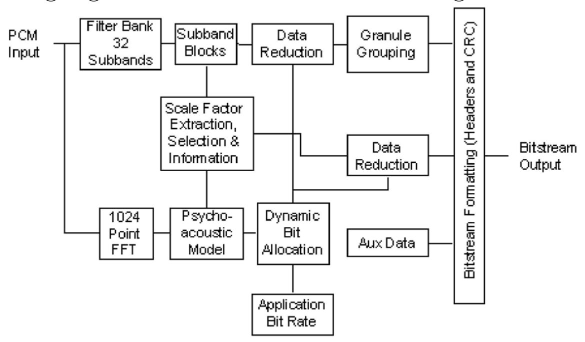
  About 1.2 Mbits/sec for video, 0.3 Mbits/sec for audio

  (Uncompressed CD audio is 44,100 samples/sec * 16 bits/sample * 2 channels $>$ 1.4 Mbits/sec)
- Compression factor ranging from 2.7 to 24.
- MPEG audio supports sampling frequencies of 32, 44.1 and 48 KHz.
- Supports one or two audio channels in one of the four modes:

  1. Monophonic – single audio channel
  2. Dual-monophonic – two independent channels (functionally identical to stereo)
  3. Stereo – for stereo channels that share bits, but not using joint-stereo coding
  4. Joint-stereo – takes advantage of the correlations between stereo channels

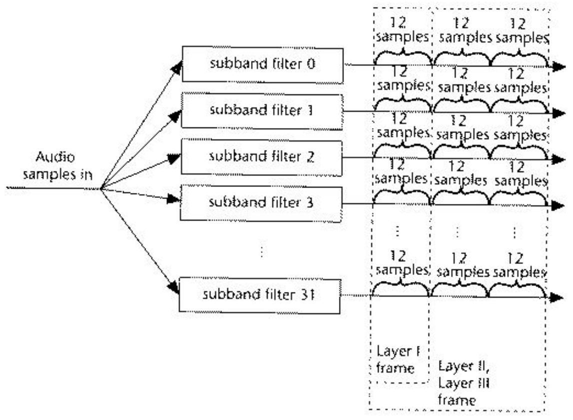
Basic MPEG-1 Compression algorithm

The basic encoding algorithm is summarised in the figure below:



The main stages of the algorithm are:

1. The audio signal is first samples and quantised use PCM
    - Application dependent: Sample rate and number of bits
2. The PCM samples are then divided up into a number of *frequency subband* and compute *subband scaling factors*:



- *Analysis filters* (also called *critical-band filters*, break signal up into equal width subbands
- Use discrete fourier transform (or the so called efficient implementation of DFT, the fast Fourier transform (FFT)) or discrete cosine transform (DCT) to filter signal.
- Filters divide audio signal into frequency subbands that approximate the 32 critical bands
- Each band is known as a *sub-band sample.*
- *Example*: 16 kHz signal frequency, Sampling rate 32 kbits/sec gives each subband a bandwidth of 500 Hz.

- Time duration of each sampled segment of input signal is time to accumulate 12 successive sets of 32 PCM (subband) samples, *i.e.* 32*12 = 384 samples.
- In addition to filtering the input, analysis banks determine
  - maximum amplitude of 12 subband samples in each subband.
  - each known as the *scaling factor* of the subband.
  - passed to *psychoacoustic model* and *quantiser blocks*

3. *Psychoacoustic modeller*
   - Frequency Masking and may employ temporal masking.
   - Performed concurrently with filtering and analysis operations.
   - Determine amount of masking for each band caused by nearby bands.
   - Input: set hearing thresholds and subband masking properties (model dependent) and scaling factors (above).
   - Output: a set of *signal-to-mask* ratios:
     - Indicate those frequencies components whose amplitude is below the audio threshold.
     - If the power in a band is below the masking threshold, don't encode it.
     - Otherwise, determine number of bits (from scaling factors) needed to represent the coefficient such that noise introduced by quantisation is below the masking effect (Recall that 1 bit of quantisation introduces about 6 dB of noise).

Example:

- Assume that after analysis, the first levels of 16 of the 32 bands are these:

Band	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Level (db)	0	8	12	10	6	2	10	60	35	20	15	2	3	5	3	1

- If the level of the 8th band is 60 dB,

  then assume (according to model adopted) it gives a masking of 12 dB in the 7th band, 15 dB in the 9th.

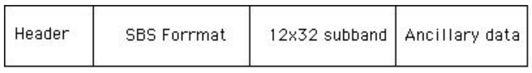
  Level in 7th band is 10 dB ($<$ 12 dB), so ignore it.

  Level in 9th band is 35 dB ($>$ 15 dB), so send it.

  –> Can encode with up to 2 bits (= 12 dB) of quantisation error.

Output bitstream

The basic output stream for a basic MPEG encoder is as follows:



- *Header*: contains information such as the sample frequency and quantisation,.
- *subband sample (SBS) format*: Quantised scaling factors and 12 frequency components in each subband.
  - Peak amplitude level in each subband quantised using 6 bits (64 levels)
  - 12 frequency values quantised to 4 bits
- *Ancillary data*: Optional. Used, for example, to carry additional coded samples associated with special broadcast format (*e.g surround sound*)

Decoding the bitstream

- Dequantise the subband samples after demultiplexing the coded bitstream into subbands.
- *Synthesis bank* decodes the dequantised subband samples to produce PCM stream.
  - This essentially involves applying the inverse fourier transform (IFFT) on each substream and multiplexing the channels to give the PCM bit stream.

MPEG Layers

MPEG defines 3 levels of processing layers for audio. Each with increasing levels of sophistication and thus greater compression ratios.

Level 1 is the basic mode, Levels 2 and 3 more advance (use temporal masking). Level 3 is the most common form for audio files on the Web — our beloved MP3 files that record companies claim are bankrupting their industry. Strictly speaking these files should be called *MPEG-1 level 3* files.

Level 1

- Best suited for bit rate bigger than 128 kbits/sec per channel.
- Example: Phillips Digital Compact Cassette uses Layer 1 192 kbits/sec compression

- Divides data into frames, each of them contains 384 samples, 12 samples from each of the 32 filtered subbands as shown above.
- Psychoacoustic model only uses frequency masking.
- Optional Cyclic Redundancy Code (CRC) error checking.

Layer 2

- Targeted at bit rates of around 128 kbits/sec per channel.
- Examples: Coding of Digital Audio Broadcasting (DAB) on CD-ROM, CD-I and Video CD.
- Enhancement of level 1.
- Codes audio data in larger groups:
  - Use three frames in filter (before, current, next, a total of 1152 samples).
  - This models a little bit of the temporal masking.
- Imposes some restrictions on bit allocation in middle and high subbands.
- More compact coding of scale factors and quantised samples.
- Better audio quality due to saving bits here so more bits cab be used in quantised subband values

Layer 3

- Targeted at bit rates of 64 kbits/sec per channel.
- Example: audio transmission of ISDN or suitable bandwidth network.
- Much more complex approach.
- Psychoacoustic model includes temporal masking effects,
- Takes into account stereo redundancy.
- Better critical band filter is used (non-equal frequencies)
- Uses a modified DCT (MDCT) for lossless subband transformation.
- Two different block lengths: 18 (long) or 6 (short)

- 50% overlap between successive transform windows gives window sizes of 36 or 12 — *accounts for temporal masking*
- Greater frequency resolution accounts for poorer time resolution
- Uses Huffman coding on quantised samples for better compression.

Comparison of MPEG Levels

Layer	Target bitrate	Ratio	Quality @ 64 kbits	Quality @ 128 kbits	Theoretical Min. Delay
Layer 1	192 kbit	4:1	---	---	19 ms
Layer 2	128 kbit	6:1	2.1 to 2.6	4+	35 ms
Layer 3	64 kbit	12:1	3.6 to 3.8	4+	59 ms

- 5 = perfect, 4 = just noticeable, 3 = slightly annoying, 2 = annoying, 1 = very annoying
- Real delay is about 3 times theoretical delay

Bit Allocation

The bit allocation process determines the number of code bits to be allocated to each subband based on information from the psychoacoustic model. For Layer I and 2, this process starts by computing the mask-to-noise ratio as given by the following equation:

$MNR_{dB} = SNR_{dB} - SMR_{dB}$

where

MNR_{dB} is the mask-to-noise ratio, SNR_{dB} is the signal-to-noise ratio, and SMR_{dB} is the signal-to-mask ratio from the psychoacoustic model. All values are in decibels.

The MPEG audio standard provides tables that give estimates for the signal-to-noise ratio resulting from quantising to a given number of quantiser levels. Designers are free to try other methods of getting the signal-to-noise ratios.

Once the bit allocation unit has mask-to-noise ratios for all the subbands, it searches for the subband with the lowest mask-to noise ratio and allocates code bits to that subband. When a subband gets allocated more code bits, the bit allocation unit looks up the new estimate for the signal-to-noise ratio and recomputes that subband's mask-to-noise ratio. The process repeats until no more code bits can be allocated.

The Layer 3 encoder uses *noise allocation.* The encoder iteratively varies the quantisers in an orderly way, quantises the spectral values, counts the number of Huffman code bits required to code the audio data and actually calculates the resulting noise. If, after quantisation, there are still scale factor bands with more than the allowed distortion, the encoder amplifies the values in those scale factor bands and effectively decreases the quantiser step size for those bands. After this the process repeats. The process stops if any of these three conditions is true:

- None of the scale factor bands have more than the allowed distortion.
- The next iteration would cause the amplification for any of the bands to exceed the maximum allowed value.
- The next iteration would require all the scale factor bands to be amplified.

Real-time encoders also can include a time-limit exit condition for this process.

Stereo Redundancy Coding

The MPEG audio compression algorithm supports two types of stereo redundancy coding: intensity stereo coding and Middle/Side (MS) stereo coding. All layers support intensity stereo coding. Layer III also supports MS stereo coding. Both forms of redundancy coding exploit another perceptual property of the human auditory system.

Simply stated at low frequencies, the human auditory system cant detect where the sound is coming from. So save bits and encode it mono. Psychoacoustic results show that above about 2 kHz and within each critical band, the human auditory system bases its perception of stereo imaging more on the temporal envelope of the audio signal than its temporal fine structure.

In intensity stereo mode the encoder codes some upper-frequency subband outputs with a single summed signal instead of sending independent left and right channel codes for each of the 32 subband outputs. The intensity stereo decoder reconstructs the left and right channels based only on a single summed signal and independent left and right channel scale factors. With intensity stereo coding, the spectral shape of the left and right channels is the same within each intensity-coded subband but the magnitude is different. The MS stereo mode encodes the left and right channel signals in certain frequency ranges as middle (sum of left and right) and side (difference of left and right) channels. In this mode, the encoder uses specially tuned threshold values to compress the side channel signal further.

Further MPEG Audio Standards

Since MPEG-1 there have been three other standards. The second phase of the MPEG audio compression standard, MPEG-2 audio, was completed in November of 1994 when it became an ISO standard.

This standard extends the MPEG-1 standard in the following ways:

- *Multichannel audio support*: The enhanced standard supports up to 5 high fidelity audio channels, plus a low frequency enhancement channel, thus it will be applicable for the compression of audio for High Definition Television or digital movies.

- *Multilingual audio support*: The standard supports up to 7 additional commentary channels.
- *Lower compressed audio bit rates*: The standard supports additional lower, compressed bit rates down to 8 kbits/sec.
- *Lower audio sampling rates*: Besides 32, 44.1, and 48 kHz, the new standard accommodates 16, 22.05, and 24 kHz sampling rates as well. The commentary channels can have a sampling rate that is half the high fidelity channel sampling rate.

In many ways this new standard is compatible with the first MPEG audio standard (MPEG-1). MPEG-2 audio decoders can decode MPEG-1 audio bitstreams. In addition, MPEG-1 audio decoders can decode two main channels of MPEG-2 audio bitstreams. This backward compatibility is achieved by combining suitably weighted versions of each of the up to 5 channels into a *down-mixed* left and right channel. These two channels fit into the audio data framework of a MPEG-1 audio bitstream. Information needed to recover the original left, right, and remaining channels fit into the ancillary data portion of a MPEG-1 audio bitstream, or in a separate auxiliary bitstream.

MPEG-3 audio:

- does not exist anymore — merged with MPEG-2

MPEG-4 audio:

- Already studied in Section 6.3.7
- uses structures audio concept
- delegates audio production to client synthesis where appropriate
- otherwise compress audio stream as above.

9.6.4 Dolby Audio Compression

Application areas:

- FM radio Satellite transmission and broadcast TV audio (DOLBY AC-1)
- Common compression format in PC sound cards (DOLBY AC-2)
- High Definition TV standard *advanced television* (ATV) (DOLBY AC-3). MPEG competitor in this area.

Differences with MPEG:

- MPEG perceptual coders control quantisation accuracy of each subband by computing bit numbers for each sample.
- MPEG needs to store each quantise value with each sample.
- MPEG Decoder uses this information to dequantise: *forward adaptive bit allocation*
- *Advantage of MPEG?*: no need for psychoacoustic modelling in the decoder due to store of every quantise value.
- DOLBY: Use *fixed bit rate allocation* for each subband. No need to send with each frame.
- DOLBY encoders and decoder need this information.

Fixed Bit Rate Allocation:

- Bit allocations are determined by known sensitivity characteristics of the ear.

Different Dolby standards:

DOLBY AC-1 : Low complexity psychoacoustic model

- 40 subbands at sampling rate of 32 kbits/sec or
- (proportionally more) subbands at 44.1 or 48 kbits/sec
- typical compressed bit rate of 512 kbits per second for stereo.
- Example: FM radio Satellite transmission and broadcast TV audio

DOLBY AC-2 : Variation to allow subband bit allocations to vary

- **NOW** Decoder needs copy of psychoacoustic model.
- Minimised encoder bit stream overheads at expense of transmitting encoded frequency coefficients of sampled waveform segment — known as the *encoded spectral envelope*.
- Mode of operation known as *backward adaptive bit allocation mode*
- HIgh (hi-fi) quality audio at 256 kbits/sec.
- Not suited for broadcast applications:
  - encoder cannot change model without changing (remote/distributed) decoders
- Example: Common compression format in PC sound cards.

DOLBY AC-3 : Development of AC-2 to overcome broadcast challenge

- Use *hybrid backward/forward adaptive bit allocation mode*
- Any model modification information is encoded in a frame.
- Sample rates of 32, 44.1, 48 kbits/sec supported depending on bandwidth of source signal.
- Each encoded block contains 512 subband samples, with 50% (256) overlap between successive samples.
- For a 32 kbits/sec sample rate each block of samples is of 8 ms duration, the duration of each encoder is 16 ms.
- Audio bandwidth (at 32 kbits/sec) is 15 KHz so each subband has 62.5 Hz bandwidth.
- Typical stereo bit rate is 192 kbits/sec.
- Example: High Definition TV standard *advanced television* (ATV). MPEG competitor in this area.

9.6.5 Streaming Audio (and video)

Popular new delivery medium for the Web and other Multimedia networks

Real Audio (*http://www.realaudio.com/*), Shockwave (*http://www.macromedia.com*) and .wav files are examples of streamed audio (and video)

- Need to compress and uncompress data in realtime
- Buffered Data:
  - Trick get data to destination before it's needed
  - Temporarily store in memory (Buffer)
  - Server keeps feeding the buffer
  - Client Application reads buffer
- Needs Reliable Connection, moderately fast too.
- Specialised client, Steaming Audio Protocol (PNM for real audio).

9.6.6 Further Exploration

- *http://www.raum.com/mpeg/* — MPEG Audio Page
- *http://www.hitsquad.com/smm/news/9903_109/?nl9905* — MP3 Beginner's Guide

A good article on this subject is:

- "A Tutorial on MPEG/Audio Compression", Davis Pan, *IEEE Multimedia*, pp. 60-74, 1995.

See Video MPEG Further Reading Resources above also

Chapter 10

Multimedia Integration, Interaction and Interchange

10.1 Integrating Multimedia

So far we have been primarily concerned with each media type or format individually. We have noted that certain media (individually) are based on spatial and/or temporal representations, other may be static.

Once we start to integrate media spacial and temporal implications become even more critical. For example static text may need to index or label a portion of video at a given instant or segment of time and there the integration becomes temporal and spatial if the label is placed at a given location (or locations moving over time).

Clearly, it is important to know the tolerance and limits for each medium as integration will require knowledge of these for synchronisation and indeed create further limits (*e.g.* bandwidth of two media types increase, if audio is encoded at a 48 Khz sampling rate and it needs to accompany video being streamed out at 60 frames per second then inter-stream synchronisation is not necessarily straightforward.

It is common (obvious) that media types are bundled together for ease of delivery, storage *etc.*. Therefore, it is not surprising that formats have been developed to support, store and deliver media in an integrated form.

The need for interchange between different multimedia applications probably running on different platforms have lead to the evolution of common interchange file formats. Many of these formats build on underlying individual media formats (MPEG, JPEG *etc.*) however further relationships are necessary when the media is truly integrated to become *multimedia*. Spatial, temporal structural and procedural constraints will exist between the media. This especially true now that interaction is a common feature of multimedia.

10.2 Interactive Multimedia

Modern multimedia presentation and applications are becoming increasingly interactive.

Simple interactions that simply start movie clips, audio segments animations etc are very common. Recently complex interactions between media is available. Following hyperlinks is instinctively non-linear and the advent of digital TV has lead to the need of a wide choice and the need for interactivity.

Interactivity now needs to be incorporated as part of the media representation/format. The MHEG format (see below) has been developed expressly for such purposes.

We have now briefly addressed the need for integrated and interactive media formats. One last topic before we discuss specific formats is the need for common interchange formats.

10.3 Multimedia Interchange

A Variety of multimedia applications running on different platforms will need to communicate with each other particularly if they are running on a distributed network.

Until recently (and it may even still pose some problems) the lack of a common interchange file format was a serious impediment to development of a market of multimedia applications.

A common interchange format needs to be widely adopted (be supported by many applications) and be sufficiently expressive to represent a wide variety of media content. These may be conflicting requirements since only when a wide variety of media is supported will it be widely adopted. Propriety application on support a small variety of media they require and may not readily adapt to other formats. Fortunately some widely accept standard that support a wide variety of media (with open standards even) are now developed.

The need for interchange formats are significant in several applications:

- As a final storage model for the creation and editing of multimedia documents.
- As a format for delivery of final form digital media. *E.g.* Compact Discs to end-use players.
- As a format for real-time delivery over a distributed network
- for interapplication exchange of data.

We will look at two broad area where the interchange and integration of multimedia:

- Desktop media — Quicktime (also main file format in MPEG-4)
- Digital TV — MHEG

10.4 Quicktime

10.4.1 Introduction

QuickTime is the most widely used cross-platform multimedia technology available today. QuickTime now has powerful streaming capabilities, so you can enjoy watching live events as they happen. QuickTime 6 is the latest version (2002) and it includes wide support of main media types and formats, streaming capabilities as well as the tools needed to create, edit, and save QuickTime movies. These tools include the QuickTime Player, PictureViewer, and the QuickTime Plug-in.

QuickTime developed out of a multimedia extension for Apple's Macintosh(proprietary) System 7 operating system. It is now an international standard for multimedia interchange and is available for many platforms and as Web browser plug ins.

The following main features are summarised below:

Versatile support for web-based media

- Access to *live* and stored *streaming media* content with the QuickTime Player
- High-Quality Low-Bandwidth delivery of multimedia
- Easy view of QuickTime movies (with enhanced control) in Web Browsers and applications.
- Multi platform support.
- Built in support for most popular Internet media formats (Over 35 formats).
- Easy import/export of movies in the QuickTime Player

Sophisticated playback capabilities

- Play back full-screen video
- Play slide shows and movies continuously
- Work with video, still-image, and sound files in all leading formats

Easy content authoring and editing

- Create new QuickTime streaming movies by copying and pasting content from any supported format
- Enhance movies and still pictures with filters for sharpening, color tinting, embossing, and more
- Save files in multiple formats, including the new DV format for high-quality video

- Create slide shows from pictures
- Add sound to a slide show

QuickTime is an *open standard* — it embraces other standards and incorporates them into its environment. It supports every major file format for pictures, including BMP, GIF, JPEG, PICT, and PNG. QuickTime also supports every important professional file format for video, including AVI, AVR, DV, M-JPEG, MPEG-1, and OpenDML. Key standards for web streaming, including HTTP, RTP, and RTSP as set forth by the Internet Engineering Task Force, are supported as well. QuickTime supports Timecode tracks, including the critical standard for video Timecode set forth by SMPTE. And for musicians, QuickTime supports MIDI standards such as the Roland Sound Canvas and the GS format extensions. QuickTime is not a proprietary environment. Not only can QuickTime movies be played back on both Windows- and Mac OSbased systems (including Windows 95, Windows 98, and Windows NT), it can also be used on web servers in UNIX, Windows, or Mac OS environments. QuickTime movies can also be played back in any standard web browser, including Microsoft Internet Explorer, Netscape Navigator, Netscape Communicator, and America Online. Unlike more limited or proprietary formats, QuickTime makes it easy to combine media types and authoring tools from multiple platforms. Content creators can work on the platform of their choice and then deliver the output to a wide range of playback devices and computer platforms. Robust multiplatform support dramatically reduces production time, because different creators can simultaneously work on the same content using different platforms. QuickTime 4 extends this capability to any RTP/RTSP standards-based server running a QuickTime Streaming Server.

10.4.2 Quicktime Support of Media Formats

QuickTime can work with more types of media than any other technology. Whether youre creating streaming video web sites, CD-ROMs, DVDs, or professional video, QuickTime gives you the best options for quality and bandwidth efficiency.

Video — QuickTime supports AVI, AVR, DV, OpenDML, and other professional digital video formats. AVI and other files can contain only audio and video, QuickTime can easily enhance these files with text, additional music tracks, and any other supported media types. QuickTime 6 features a variety of video compressors and decompressors that can handle needs ranging from CD-ROMs and DVDs to dial-up Internet access. QuickTime can stream video over the Internet even with 28.8-Kbps modems. QuickTime 6 includes support for Cinepak, IMA, Intel Indeo Video, the industry-standard H.263 compressor, Sorenson Video 3, and many others.

MPEG format — The MPEG standard for Macintosh is used extensively for consumer products, combining high-quality audio with low data rates.

An Apple extension for QuickTime 6 for Macintosh provides direct access to MPEG-1 audio and video, including the popular MPEG-1, Layer 3 (MP3). You can also play back MP3 files with QuickTime for Windows. The QuickTime file format for MPEG-4 has been adopted as an ISO standard since Quicktime 4. In February 1998, the International Standards Organization (ISO) formally adopted the QuickTime file format as the starting point for the MPEG-4 (Fig 10.1) file format. Quicktime 6 is the first major iteration to offer support of all MPEG-4 video and audio specifications, though not all features are supported.

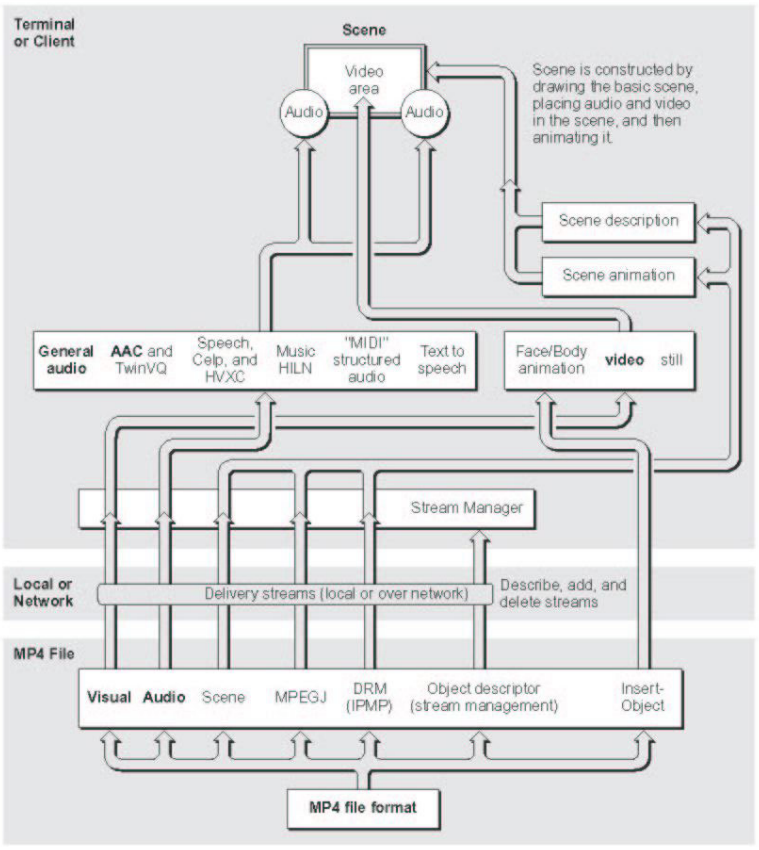


Figure 10.1: MPEG-4 File Structure

Audio and Speech — QuickTime supports high-quality digital audio which can be used in many types of applications. Audio is used by itself (*e.g.* MP3 player) or integrated with other media types (film audio track). Quicktime also offers special support for speech — Qualcomms PureVoice technology allows for some of the highest quality voice compression available (14.4 Kbps modem transmission rate). Support for MPEG-4 structured audio is also available in Quicktime 6 — `.mp4` files.

MIDI — MIDI music is an integral part of the QuickTime architecture. It can be used alone or with video, animation, still images, or other visual elements. In addition to playing music through internal or external speakers, QuickTime can route musical information to external MIDI devices, effects processors, and drum machines. QuickTime 6 provides CD-quality, low-bandwidth music by supporting over 200 instruments with the Roland Sound Canvas sound set. It also supports GS format extensions, which allow additional expressions for General or standard MIDI sequences. On Windows systems, QuickTime supports the use of the MIDI Mapper for use with external MIDI hardware. Midi is also a component of the MPEG-4 structured audio format.

Images and Graphics – The extensive collection of still-image importers in QuickTime 4 allows media authors to leverage photography and illustrations created in a wide range of formats. QuickTime 4 now has the ability to open FlashPix images with PictureViewer and export to PNG, TIFF, TARGA, and MacPaint images, while supporting 16-bit-per-channel files. It also supports multiple images in TIFF, FlashPix, and Adobe Photoshop formats. To enable you to work easily within workgroups, QuickTime supports all the main image formats *e.g.*BMP, GIF, JPEG, PICT, PNG, and SGI formats. Quicktime 6 also supports JPEG 2000 standard which os based on wavelets rather than the DCT.

Text — QuickTime supports searchable text tracks, with a *Find* command in the QuickTime Player or through Quicktime C/Java API. A single QuickTime movie can have multiple text tracks, simplifying the creation of multilingual movies. Text annotation can be used with any QuickTime-enabled media types: AVI for video files, and WAV, AIFF, or MPEG-1 for audio files. It can also be deployed as an HREF track, allowing you to embed URLs in your movies.

Animations — Macromedia Flash animation can now be played with any QuickTime application, including the QuickTime Plug-in. Quicktime currently offers support for Flash 5. For high quality animation with compact files, QuickTime 6 integrates a curve-based vector animation compressor and flexible sprite capabilities. Recall that for some types of animations, these vector-based tools can produce a dramatically smaller file than traditional compressors and prerendered video tracks can produce. QuickTime 6 also supports alpha channel compositing and special effects up to 16 bits per pixel.

10.4.3 QuickTime Concepts

The following concepts QuickTime are used by Quicktime:

Movies and Media Data Structures —

A traditional movie, whether stored on film, laser disk, or tape, is a continuous stream of data. A QuickTime movie can be similarly constructed, but it need not be: a QuickTime movie can consist of data in sequences from different forms, such as analog video and CD-ROM. The movie is not the medium; it is the organizing principle.

A QuickTime movie may contain several tracks. Each track refers to a media that contains references to the movie data, which may be stored as images or sound on hard disks, floppy disks, compact discs, or other devices. The data references constitute the track's media. Each track has a single media data structure.

Components —

QuickTime provides components so that every application doesn't need to know about all possible types of audio, visual, and storage devices. A component is a code resource that is registered by the Component Manager. The component's code can be available as a system wide resource or in a resource that is local to a particular application. Each QuickTime component supports a defined set of features and presents a specified functional interface to its client applications. Applications are thereby isolated from the details of implementing and managing a given technology. For example, you could create a component that supports a certain data encryption algorithm. Applications could then use your algorithm by connecting to your component through the Component Manager, rather than by implementing the algorithm over again.

Image Compression —

Image data requires a large amount of storage space. Storing a single 640-by-480 pixel image in 32-bit color can require as much as 1.2 MB. Similarly, sequences of images, like those that might be contained in a QuickTime movie, demand substantially more storage than single images. This is true even for sequences that consist of fairly small images, because the movie consists of a large number of those images. Consequently, minimizing the storage requirements for image data is an important consideration for any application that works with images or sequences of images.

The Image Compression Manager provides your application with an interface for compressing and decompressing images and sequences of images that is independent of devices and algorithms.

Time —

Image compression is difficult but worthwhile–images, not to mention long sequences of images, take a lot of memory. Time management in QuickTime is equally essential. You must understand time management to understand the QuickTime functions and data structures.

Seemingly simple issues prove interesting–for example, determining the proper length (duration) of a movie. For many movies, the proper duration is the time required to play them in ”real” time–that is, a rate in which human actions appear natural, and objects fall to earth accelerating at 32 feet per second per second. But what is the length of a movie that shows spreadsheet data charted over time, or a map of the earth that recapitulates continental drift? Add to this the differing clock speeds of different platforms, and the need to decompress in real time, and time proves, as ever, complex.

To manage these situations, QuickTime defines time coordinate systems, which anchor movies and their media data structures to a common temporal reality, the second. A time coordinate system contains a time scale that provides the translation between real time and the time in a movie. Time scales are marked in time units. The number of units that pass per second quantifies the scale–that is, a time scale of 26 means that 26 units pass per second and each time unit is 1/26 of a second. A time coordinate system also contains a duration, which is the length of a movie or a media in the number of time units it contains. Particular points in a movie can be identified by a time value, the number of time units elapsed to that point.

Each media has its own time coordinate system, which starts at time 0. The Movie Toolbox maps each type of media data from the movie's time coordinate system to the media's time coordinate system.

10.4.4 The QuickTime Architecture

QuickTime comprises two managers: the Movie Toolbox and the Image Compression Manager. QuickTime also relies on the Component Manager, as well as a set of predefined components. Figure 1-1 shows the relationships of these managers and an application that is playing a movie.

The Movie Toolbox —

Your application gains access to the capabilities of QuickTime by calling functions in the Movie Toolbox. The Movie Toolbox allows you to store, retrieve, and manipulate time-based data that is stored in QuickTime movies. A single movie may contain several types of data. For example, a movie that contains video information might include both video data and the sound data that accompanies the video.

The Movie Toolbox also provides functions for editing movies. For example, there are editing functions for shortening a movie by removing portions of the video and sound tracks, and there are functions for extending it with the addition of new data from other QuickTime movies.

The Movie Toolbox is described in the chapter ”Movie Toolbox” later in this book. That chapter includes code samples that show how to play movies.

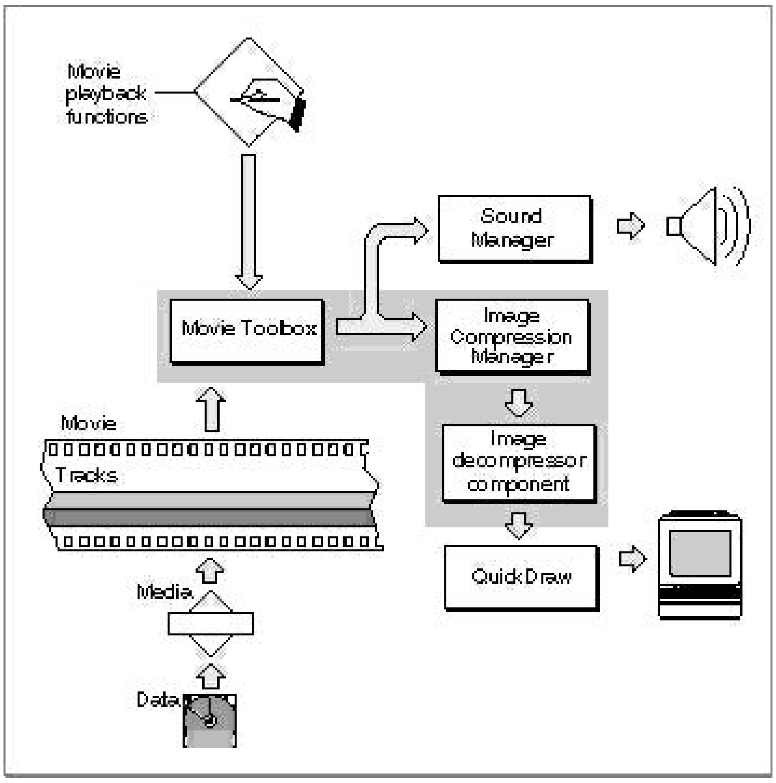


Figure 10.2: Quicktime Architecture

The Image Compression Manager —

The Image Compression Manager comprises a set of functions that compress and decompress images or sequences of graphic images.

The Image Compression Manager provides a device-independent and driver-independent means of compressing and decompressing images and sequences of images. It also contains a simple interface for implementing software and hardware image-compression algorithms. It provides system integration functions for storing compressed images as part of PICT files, and it offers the ability to automatically decompress compressed PICT files on any QuickTime-capable Macintosh computer.

In most cases, applications use the Image Compression Manager indirectly, by calling Movie Toolbox functions or by displaying a compressed picture. However, if your application compresses images or makes movies with compressed images, you will call Image Compression Manager functions.

The Image Compression Manager is described in the chapter "Image Compression Manager" later in this book. This chapter also includes code samples that show how to compress images or make movies with compressed images.

The Component Manager —

Applications gain access to components by calling the Component Manager. The Component Manager allows you to define and register types of components and communicate with components using a standard interface. A component is a code resource that is registered by the Component Manager. The component's code can be stored in a system wide resource or in a resource that is local to a particular application.

Once an application has connected to a component, it calls that component directly. If you create your own component class, you define the function-level interface for the component type that you have defined, and all components of that type must support the interface and adhere to those definitions. In this manner, an application can freely choose among components of a given type with absolute confidence that each will work.

10.4.5 QuickTime Components

QuickTime includes several components that are provided by Apple. These components provide essential services to your application and to the managers that make up the QuickTime architecture. The following Apple-defined components are among those used by QuickTime:

- movie controller components, which allow applications to play movies using a standard user interface standard image compression dialog components, which allow the user to specify the parameters for a compression operation by supplying a dialog box or a similar mechanism

- image compressor components, which compress and decompress image data sequence grabber components, which allow applications to preview and record video and sound data as QuickTime movies video digitizer components, which allow applications to control video digitization by an external device
- media data-exchange components, which allow applications to move various types of data in and out of a QuickTime movie derived media handler components, which allow QuickTime to support new types of data in QuickTime movies
- clock components, which provide timing services defined for QuickTime applications preview components, which are used by the Movie Toolbox's standard file preview functions to display and create visual previews for files sequence grabber components, which allow applications to obtain digitized data from sources that are external to a Macintosh computer
- sequence grabber channel components, which manipulate captured data for a sequence grabber component
- sequence grabber panel components, which allow sequence grabber components to obtain configuration information from the user for a particular sequence grabber channel component

We will study programming aspects of the above with Java in the next Chapter.

10.4.6 Quicktime File Format

The Quicktime Movie File Format is a published (*http://developer.apple.com/techpubs/quicktime/qtdevdocs/PDF/QTFileFormat.pdf*) file format for storing multimedia content for Quicktime presentation. Several players are available on many platforms.

The Quicktime file format uses a track model for organising the temporally data of a movie. A movie can contain one or more tracks — a track is a time-ordered sequence of a media type. The media are addressed using an *edit list* which is a list of end-points of digital media clips or segments.

See handouts, Ch 14 in *Multimedia Systems* by J. Buford (Addison Wesley) and the online Apple documentation for further details.

10.4.7 Further Information

Quicktime Software, information, news etc . may b e obtained from Apple's Quicktime Web site: *http://www.apple.com/quicktime*

Specific documents of interest to this course are:

- *http://developer.apple.com/techpubs/quicktime/qtdevdocs/RM/qthead4.htm* — quicktime documentation online

- *http://www.apple.com/quicktime/pdf/QuickTime4_FS-a.pdf* — Quicktime Information
- *http://www.apple.com/quicktime/pdf/QuickTime4Pro_DS-a.pdf* — Quicktime Pro Features
- *http://developer.apple.com/techpubs/quicktime/qtdevdocs/PDF/QTFileFormat.pdf* — Quicktime file format

10.5 Open Media Framework Interchange (OMFI) Format

The OMFI is a common interchange framework developed in repsonse to an industry led standardisation effort (including Avid — a major digital video hardware/applications vendor)

Like Quicktime the primary concern of the OMFI format is concerned with temporal representation of media (such as video and audio) and a track model is used.

The primary emphasis is video production and an number of additional features reflect this:

- Source (analogue) material object represent videotape and film so that the origin of the data is readily identified. Final footage may resort to this original form so as to ensure highest possible quality.
- Special track types store (SMPTE) time codes for segments of data.
- Transitions and effects for overlapping and sequences of segments are predefined.
- *Motion Control* — the ability to play one track at a speed which is a ratio of the speed of another track is supported.

The OMFI file format incorporates:

- a header — include indices for objects contained in file
- Object dictionary — to enhance the OMFI class hierarchy in an application
- Object data
- Track data

10.6 Multimedia and Hypermedia Information Encoding Expert Group (MHEG)

The development of MHEG arose directly out of the increasing convergence of broadcast and interactive technologies for Digital TV. It specifies an encoding format for multimedia applications independently of service paradigms and network protocols. Like Quicktime and OMFI it is concerned with time-based media objects, whose encodings are determined by other standards.However, the scope of MHEG is larger in that it directly supports interactive media and real-time delivery over networks.

There have been a progressions of MHEG standards (much like MPEG) The current widespread standard is MHEG-5 but drafts standards exist up to MHEG-7.

Every design generally represents a compromise between conflicting goals. MHEG's design is no exception, especially if you consider that MHEG-5 (and later) targets a continuously growing and fiercely competitive market where broadcast and interactive technologies converge almost daily.

Converging technologies have often stimulated adopting standard solutions. Multimedia applications standards provide more than just the obvious objectives of portability and interoperability. A good multimedia standard can become a reference solution for system developers and application programmers. It also promotes the use of modular architectures that rely on common components that accomplish a specific functionality, such as interpreting and presenting MHEG applications to users. This task is performed by a compliant runtime engine (RTE), a resident software component that schedules delivery of an application to the user. It's aimed at a wide installation base within complete solutions, like a Video on Demand or an Interactive TV system. RTEs help improve a product's return on investment, abate a product's per unit costs, and provide high quality, robust products due to extensive product testing.

10.6.1 Practical MHEG: Digital Terrestrial TV

MHEG has been used as the Media interchange format in Digital TV set top boxes (Fig 10.3).

In the UK, ITV digital used this format and the newer Freeview digital terrestrial (Fig: 10.4 services use this. MHEG is also widely used in European Digital TV.

UK digital TV interests are managed by the Digital TV Group UK — *http://www.dtg.org.uk/*.

Note that the other (satellite) digital TV interest in the UK, SKY, uses a proprietary API format, called OPEN (!!). The advantage is that MHEG **is a truly open** format (ISO standard). MHEG is the only open standard in this area.

Some excellent and interesting document relating to the implementation and delivery of MHEG digital TV is available at

Figure 10.3: Digital TV set top box

Figure 10.4: UK Digital TV Consortium

http://www.dtg.org.uk/reference/mheg/_mheg_index.html

Digital TV services

What sort of multimedia services does digital TV provide. The figure below (Fig. 10.5) summarises and illustrates typical digital TV services.

MHEG has been designed to provide such functionality.

10.6.2 The family of MHEG standards

MHEG encompasses the family of standards issued by the ISO/IEC JTC1 joint technical committee's working group WG12information technology subcommittee SC29, coding of audio, picture multimedia, and hypermedia information. See Table 10.1 for the complete list of MHEG standards.

Version	**Complete Name**	**Status**
MHEG-1	MHEG object representation-base notation (ASN.1)	International standard
MHEG-2	MHEG object representation-alternate notation (SGML)	Withdrawn
MHEG-3	MHEG script interchange representation	International standard
MHEG-4	MHEG registration procedure	International standard
MHEG-5	Support for base-level interactive applications	International standard
MHEG-6	Support for enhanced interactive applications	International standard (April 1998)
MHEG-7	Interoperability and conformance testing for ISO/IEC 13522-5	Draft international standard (Jan 1999)

Table 10.1: MHEG Standards

Since it was introduced first, MHEG-1 received the most attention. It's the generic standard for encoding multimedia objects without specific assumptions on the application area or on the target platform used for delivering and rendering these objects to the user. MHEG-3 provides a script extension to MHEG-1. MHEG-4 specifies a registration procedure for identifiers used by the objects to identify, for example, a specific format for content data. MHEG-5 can conceptually be considered a simplifying profile of MHEG-1. It addresses terminals with limited resources, like the set-top unit. Actually, an MHEG-1 decoder can't decode MHEG-5 applications due to some slightly different provisions added to optimize performance in VoD/ITV environments. MHEG-6 extends the *declarative* MHEG-5 approach with procedural code capabilities typical of a scripting language. It defines the interface (MHEG-5 API) and a script engine's runtime environment on top of an MHEG-5 engine using the Java virtual machine to provide a complete solution for application representation. MHEG-7, a new standard, addresses the conformance and interoperability of MHEG-5 engines and applications.

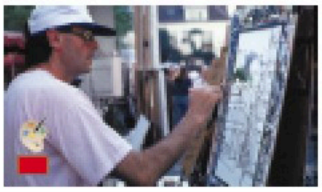
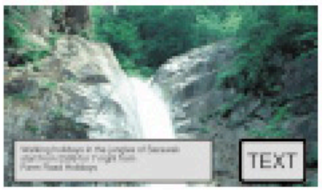
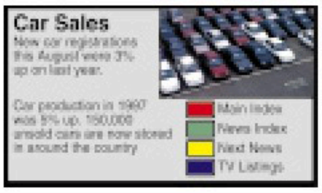
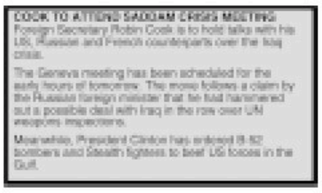
Visual Appearance	Description
	1. Conventional TV
	2. TV with visual prompt of available information
 Walking holidays in the jungles of Sarawak start from £698 for 7 night from Penn Road Holidays	3. TV with information overlaid
 **Car Sales** New car registrations this August were 3% up on last year. Car production in 1997 was 5% up. 150,000 unsold cars are now stored in around the country 🟥 Main Index 🟩 News Index 🟨 Next News 🟦 TV Listings	4. Information with video or picture inset
 COOK TO ATTEND SADDAM CRISIS MEETING Foreign Secretary Robin Cook is to hold talks with his US, Russian and French counterparts over the Iraq crisis. The Geneva meeting has been scheduled for the early hours of tomorrow. The move follows a claim by the Russian foreign minister that he had hammered out a possible deal with Iraq in the row over UN weapons inspections. Meanwhile, President Clinton has ordered B-52 bombers and Stealth fighters to beef US forces in the Gulf.	5. Just information

Figure 10.5: UK Digital TV Consortium

10.6.3 MHEG-5 overview

SInce MHEG-5 is the widest MHEG standard in operation and is widely known we will highlight this MHEG standard.

A multimedia application can be conceived as a set of self-contained objects based on synchronization and spatial-temporal relationships of multiple media formats, structural composition, event-action associations, navigation, and user interaction capabilities. Controlling the playback of time-dependent contents, like streams of multiplexed audiovisual data requires specific support. These streams demand VCR control functions (play, pause, fast forward, and so on), as well as the capability to manage events generated during their presentation. For example, rendering text subtitles can be synchronized with timecode events generated during the playback of a stream. MHEG-5 represents an application, as a set of scenes, which contain objects common to all scenes. A scene supports the spatially and temporally coordinated presentation of audiovisual content consisting of graphics, bitmaps, text, and streams (based on the multiplex of audio and video components). Interaction can be performed via graphic elements like buttons, sliders, text entry boxes, and hypertext selections. Every scene, as well as an entire application, is a self-contained entity that can represent its localized behavior by links that are event-action associations. Events can be generated by users, expiration of timers, playback of streams, and other conditions within the RTE.

The global scope of MHEG-5 is to define the syntax and semantics of a set of object classes that can be used for interoperability of multimedia applications across minimal-resources platforms. The developed applications will reside on a server, and as portions of the application are needed, they will be downloaded to the client. In a broadcast environment, this download mechanism could rely, for instance, on cyclic rebroadcasting of all portions of the application. It is the responsibility of the client to have a runtime that interprets the application parts, presents the application to the user, and handles the local interaction with the user.

The major goals of MHEG-5 are:

- To provide a good standard framework for the development of client/server multimedia applications intended to run on a memory-constrained Client.
- To define a final-form coded representation for interchange of applications across platforms of different versions and brands.
- To provide the basis for concrete conformance levelling, guaranteeing that a conformant application will run on all conformant terminals.
- To allow the runtime engine on the Client to be *small* and *easy* to implement.
- To be free of strong constraints on the architecture of the Client.

- To allow the building of a wide range of applications. This means also providing access to external libraries. An application using external libraries will only be partly portable.
- To allow for application code that is guaranteed to be "safe" in the sense that it cannot harm other code in the Client, nor put the Client in an abnormal state.
- To allow automatic static analysis of (final-form) application code in order to help insure bug-free applications and minimize the debugging investment needed to get a robust application. Note that this analysis should be possible to implement independently of the authoring environment.
- To promote rapid application development by providing high-level primitives and provide a declarative paradigm for the application development.

The MHEG-5 model is object-oriented. The actions are methods targeted to objects from different classes to perform a specific behavior and include:

- preparation,
- activation,
- controlling the presentation,
- user interaction,
- getting the value of attributes,
- and so on.

To allow interoperability across heterogeneous systems, MHEG-5 specifies the precise encoding syntax. Two notations are possible: the ASN.1 notation, which MHEG-1 also adopts, and a textual notation as illustrated below:.

In a client-server architecture (Fig 10.6), MHEG-5 applications are stored on the server and downloaded to the terminal for the RTE to interpret. This model is not limited to storage and retrieval services. In the broadcast environment, for example, the set of channels transmitted on a broadcast network can be considered a virtual server, where the download mechanism relies on cyclic rebroadcast of all portions of an application.

10.6.4 MHEG Programming Principles

MHEG-5 provides suitable abstractions for managing active, autonomous, and reusable entities (since it adopts an object-oriented approach).

A class is specified by three kinds of properties:

- attributes that make up an object's structure,
- events that originate from an object, and

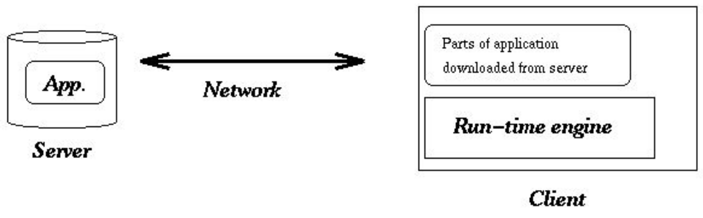


Figure 10.6: MHEG Client-Server Interaction

- actions that target an object to accomplish a specific behavior or to set or get an attribute's value.

The most significant classes of MHEG-5 are now briefly described:

Root — A common Root superclass provides a uniform object identification mechanism and specifies the general semantics for preparation/destruction and activation/deactivation of objects, including notification of changes of an object's availability and running status. These general provisions are further specialized moving downwards through the inheritance tree, which first branches into the Group and Ingredient classes.

Group — This abstract class handles the grouping of objects in the Ingredient class as a unique entity of interchange. In fact, Group objects can be addressed and independently downloaded from the server. A Group can be specialized into Application and Scene classes.

Application — An MHEG-5 application is structurally organized into one Application and one or more Scene objects. The Application object represents the entry point that performs a transition to the presentation's first Scene. Generally, this transition occurs at startup (see below code examples) because a presentation can't happen without a Scene running.

The Launch action activates an Application after quitting the active Application. The Quit action ends the active Application, which also terminates the active Scene's presentation. The Ingredients of an Application are available to the different Scenes that become active, thereby allowing an uninterrupted presentation of contents (for example, a bitmap can serve as the common background for all Scenes in an Application).

Scene — This class allows spatially and temporally coordinated presentations of Ingredients. At most, one Scene can be active at one time. Navigating within an Application is performed via the `TransitionToaction` that closes the current Scene, including its Ingredients, and activates the new one. The `SceneCoordinateSystem` attribute specifies the presentation space's 2D size for the Scene. If a user interaction occurs in this space, a UserInput event is generated. A Scene also supports timers. A Timer event is generated when a timer expires.

Ingredient — This abstract class provides the common behavior for all objects that can be included in an Application or a Scene.

The `OriginalContent` attribute maps object and content data. It contains either included content or a reference to an external data source (such as a URL or a DSMCC file name). The `ContentHook` attribute specifies the encoding format for the content. However, MHEG-5 does not list the supported encoding formats. See coding examples below for the use of content references and hooks.

The action `Preload` gives hints to the RTE for making the content available for presentation. Especially for streams, this action does not completely download the content, it just sets up the proper network connection to the site where the content is stored. The action `Unload` frees allocated resources for the content.

The Presentable, Stream, and Link classes are subclasses of the Ingredient class.

Presentable — This abstract class specifies the common aspects for information that can be seen or heard by the user. The `Run` and `Stop` actions activate and terminate the presentation, while generating the `IsRunning` and `IsStopped` events.

Visible — The Visible abstract class specializes the `Presentable` class with provisions for displaying objects in the active Scene's presentation space.

The `OriginalBoxSize` and `OriginalPosition` attributes respectively specify the size and position of the object's bounding box relative to the Scene's presentation space. The actions `SetSize` and `SetPosition` change the current values of these attributes.

The specialized objects in the Visible class include:

- *Bitmap* — This object displays a 2D array of pixels. The Tiling attribute specifies whether the content will be replicated throughout the `BoxSize` area. The action `ScaleBitmap` scales the content to a new size.
  Example, to create a simple bitmap object:
  ```
  (bitmap: BgndInfo
      content-hook: #bitmapHook
      content-data: referenced-content: "Info.bitmap"
      box-size: ( 320 240 )
      original-position: ( 0 0 )
  )
  ```
- *LineArt, DynamicLineArt*— A LineArt is a vectorial representation of graphical entities, like polylines and ellipses. `DynamicLineArt` draws lines and curves on the fly in the `BoxSize` area.

- *Text* — This object represents a text string with a set of rendition attributes. Essentially, these attributes specify fonts and formatting information like justification and wrapping.

Stream — This class (a subclass of Ingredient) controls the synchronized presentation of multiplexed audio-visual data (such as an MPEG-2 file). A Stream object consists of a list of components from the `Video`, `Audio`, and `RTGraphics` (animated graphics) classes. The `OriginalContent` attribute of the Stream object refers to the whole multiplex of data streams.

When a Stream object is running, its streams can be switched on and off independently. This lets users switch between different audio trails (different languages) or choose which video stream(s) to present among a range of available ones. For example, the Turin code example below contains an MPEG-1 Stream composed of one audio and one video component. These components automatically activate when a run action targets the whole Stream because their `InitiallyActive` attribute is set to *true*.

Specific events are associated with playback: `StreamPlaying/StreamStopped` notifies the actual initiation/termination and `CounterTrigger` notifies the system when a previously booked time-code event occurs. The Turin code example below shows how the `CounterTrigger` event can be used to synchronize text subtitling and also illustrates the `SetCounterPosition` and `SetCounterEndPosition` actions to specify a temporal segment for presentation.

Link — The Link class implements event-action behavior by a condition and an effect. The `LinkCondition` contains an `EventSource` - a reference to the object on which the event occurs - an `EventType` that specifies the kind of event and a possible `EventData` that is a data value associated with the event.

MHEG-5 Action objects consist of a sequence of elementary actions. Elementary actions are comparable to methods in an object-oriented paradigm. The execution of an Action object means that each of its elementary actions are invoked sequentially.

As an example, consider the following Link, which transitions to another Scene when the character `A` is entered in the EntryField `EF1`.

Example, to create a simple link:

```
(link: Link1
    event-source: EF1
    event-type: #NewChar
    event-data: 'A'
    link-effect:
        (action: transition-to: Scene2)
)
```

In the Turin code example below, `Link 49` triggers only if a `CounterTrigger` event on the video clip occurs with `EventData = 3`.

Specifically, this lets you associate a different effect with every booked value of the `CounterPosition`. The `LinkEffect` comprises a set of actions that are executed in sequence when an event that matches with the `LinkCondition` occurs. Every action specifies the target object and, possibly, other parameters depending on the type of action. MHEG-5 specifies more than 100 kinds of actions.

Interactible — This abstract class provides a way for users to interact with objects within the following sub-classes:

Hotspot, PushButton, and SwitchButton — These subclasses implement button selection capability and generate the `IsSelected` event.

Example, to create a simple `SwitchButton`:

```
(switchbutton: Switch1
    style: #radiobutton
    position: ( 50 70 )
    label: "On"
)
```

Hypertext — This class extends the Text class with *anchors*. When selected, these anchors link text content to associated information.

Slider and EntryField — Respectively, these objects let users adjust a numeric value (such as the volume of an audio stream) and edit text.

Example, to create a simple slider:

```
(slider: Slider1
    box-size: ( 40 5 )
    original-position: ( 100  100 )
    max-value: 20
    orientation: #right
)
```

UK Digital Terrestrial MHEG Support: *EuroMHEG*

Above we described the main classes of MHEG. There a few other classes that we will not address in this course. It is enough that we basically gain a broad understanding of how MHEG works and the basic classes that support this.

Not all MHEG engines support all MHEG classes. In fact the MHEG standard use by UK digital TV needed to be initially retricted to meet production timescales for the launch. An *EuroMHEG* standard was thus defined. The standard was defined to be extensible so as to be able to include updates and additions in due course.

The MHEG classes supported by EuroMHEG are:

Root	Group	Application
Scene	Ingredient	Link
Program	ResidentProgram	RemoteProgram
Palette	Font	CursorShape
Variable	BooleanVariable	IntegerVariable
OctetStringVariable	ObjectRefVariable	ContentRefVariable
Presentable	TokenManager	TokenGroup
ListGroup	Visible	Bitmap
LineArt	Rectangle	DynamicLineArt
Text	Stream	Audio
Video	RTGraphics	Interactible
Slider	EntryField	HyperText
Button	HotSpot	PushButton
SwitchButton	Action	

10.6.5 Interaction within a Scene

The MHEG application is event-driven, in the sense that all actions are called as the result of an event firing a link. Events can be divided into two main groups: synchronous events and asynchronous events. Asynchronous events are events that occur asynchronously to the processing of Links in the MHEG engine. These include timer events and user input events. An application area of MHEG-5 (such as DAVIC) must specify the permissible UserInput events within that area. Synchronous events are events that can only occur as the result of an MHEG-5 action being targeted to some objects. A typical example of a synchronous event is `IsSelected`, which can only occur as the result of the MHEG-5 action Select being invoked. Synchronous events are always dealt with immediately; asynchronous events are queued.

The mechanism at the heart of the MHEG engine, therefore, is the following:

1. After a period of of idleness, an asynchronous event occurs. The event can be a user input event, a timer event, a stream event, or some other type of event.

2. Possibly, a link that reacts on the event is found. This link is then fired. If no such link is found, the process starts again at 1.

3. The result of a link being fired is the execution of an action object, which is a sequence of elementary actions. These can change the state of other objects, create or destroy other objects, or cause events to occur.

4. As a result of the actions being performed, synchronous events may occur. These are dealt with immediately, i.e., before processing any other asynchronous events queued.

When all events have been processed, the process starts again at 1.

10.6.6 Availability; Running Status

Before doing anything to an object, the MHEG-5 engine must prepare it. Preparing an object typically entails retrieving it from the server, decoding the interchange format and creating the corresponding internal data structures, and making the object available for further processing. The preparation of an object is asynchronous; its completion is signalled by an IsAvailable event.

All objects that are part of an application or a scene have a RunningStatus, which is either true or false. Objects whose RunningStatus is true are said to be running, which means that they perform the behaviour they are programmed for. More concretely:

- only running Visibles are actually visible on the screen,
- only running Audio objects are played out through the loudspeaker,
- only running Links will execute the action part if the associated event occurs, etc.

10.6.7 Interactibles

The MHEG-5 mix-in class Interactible groups some functionality associated with user interface-related objects (Slider, HyperText, EntryField, Buttons).

These objects can all be highlighted (by setting their HighlightStatus to True).

They also have the attribute InteractionStatus, which, when set to true, allows the object to interact directly with the user, thus bypassing the normal processing of UserInput events by the MHEG-5 engine.

Exactly how an Interactible reacts when its InteractionStatus is true is implementation-specific.

As an example, the way that a user enters characters in an EntryField can be implemented in different ways in different MHEG-5 engines.

At most one Interactible at a time can have its InteractionStatus set to True. Interactibles

10.6.8 Visual Representation

For objects that are visible on the screen, the following rules apply :

- Objects are drawn downwards and to the right of their position on the screen. This point can be changed during the life cycle of an object, thus making it possible to move objects.
- Objects are drawn without scaling. Objects that do not fit within their bounding box are clipped.
- Objects are drawn with "natural" priority, *i.e.*, on top of already existing objects. However, it is possible to move objects to the top or the bottom of the screen, as well as putting them before or after another object.

- The screen can be frozen, allowing the application to perform many (possibly slow) changes and not update the screen until it's unfrozen.

10.6.9 Object Sharing Between Scenes

It is possible within MHEG-5 to share objects between some or all scenes of an application. As an example, this can be used to have variables retain their value over scene changes, or to have an audio stream play on across a scene change. Shared objects are alway contained in an Application object. Since there is always exactly one Application object running whenever a scene is running, the objects contained in an application object are visible to each of its scenes.

10.6.10 Object Encoding

The MHEG-5 specification does not prescribe any specific formats for the encoding of content. For example, it is conceivable that a Video object is encoded as MPEG or as motion-JPEG. This means that the group using MHEG-5 must define which content encoding schemes to apply for the different objects in order to achieve interoperability.

However, MHEG-5 does specify a final-form encoding of the MHEG-5 objects themselves. This encoding is an instance of ASN.1, using the Basic Encoding Rules (BER).

10.6.11 Conformance

The issue of conformance, though of crucial importance, has not yet been extensively addressed by the MHEG committee. It is expected that a conformance definition for the standard will have to be drafted, probably lagging the standard itself by some period of time.

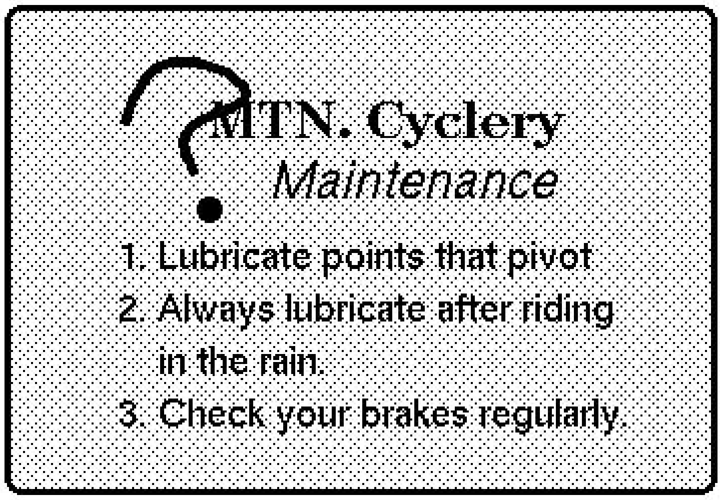
10.6.12 MHEG Coding Examples

A Simple MHEG Example

Below (Fig 10.7) is a very simple scene that displays a bitmap and text. The user can press the 'Left' button on the input device and a transition is made from the current scene, `InfoScene1`, to a new scene, `InfoScene2`.

The pseudo-code from the above scene may look like the following:

```
(scene:InfoScene1
    <other scene attributes here>
    group-items:
      (bitmap: BgndInfo
          content-hook: #bitmapHook
          original-box-size: (320 240)
          original-position: (0  0)
          content-data: referenced-content: "InfoBngd"
```


MTN. Cyclery
Maintenance

1. Lubricate points that pivot
2. Always lubricate after riding
   in the rain.
3. Check your brakes regularly.

Figure 10.7: Simple MHEG Example

```
        )
        (text:
            content-hook: #textHook
            original-box-size: (280 20)
            original-position: (40 50)
            content-data: included-content: "1. Lubricate..."
        )
        links:
            (link: Link1
                event-source: InfoScene1
                event-type: #UserInput
                event-data: #Left
                link-effect: action: transition-to: InfoScene2
             )
)
```

A More Complex Example (Turin)
Another example application lets users retrieve tourist information about the city of Turin, Italy. URL: *http://drogo.cselt.stet.it/ufv/mediatouch/mh5webapp/to_audio_e_0.mh5*

Figure 10.8 shows a screen shot of the *main_scene* object. This scene consists of

- a text title,
- a bitmap background,

- a video clip, which is playing a segment of Porta Nuova, the main railway station, enriched with a text subtitle object (left side),
- some interactive thumbnails (right side),
- a set of buttons that provide VCR controls,
- functions to switch between viewing the video in *normal* and *zoom* modes, and item a *return to previous screen* capability.

The playback of the video clip synchronizes with the subtitle's content. Selecting an interactive thumbnail constrains the playback to the associated temporal segment.

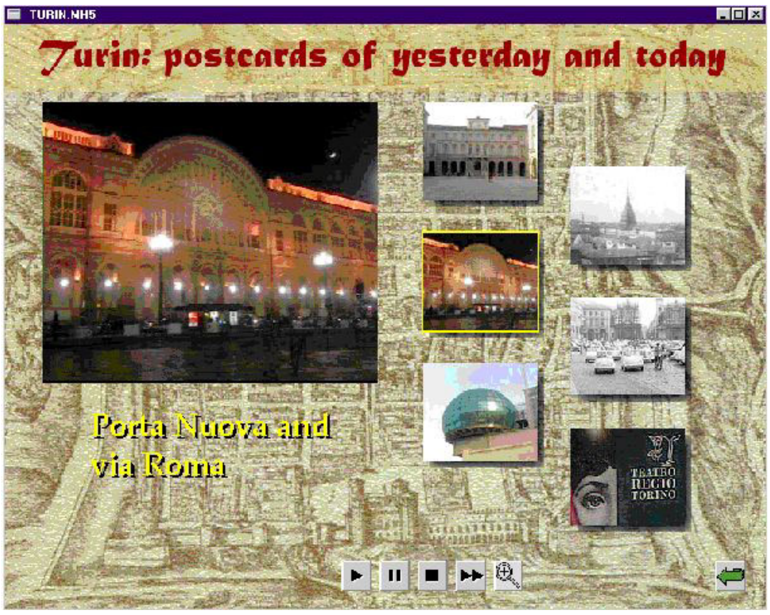


Figure 10.8: MHEG Example Presentation (Turin Guide)

The code listing below is an excerpt from the Turin application's MHEG textual notation:

```
{:Application ("turin.mh5" 0)
:OnStartUp ( // sequence of initialization
actions
:TransitionTo (("main_scene.mh5" 0)) //
activation of the first scene
)
}
{:Scene ("main_scene.mh5" 0)
```

```
:OnStartUp ( // sequence of initialization
actions
preload (2) // the connection to the
source of the video clip is set up
...
setCounterTrigger (2 3 190000) // book a
time code event at 190000 msec
...
)
:Items ( // both presentable ingredients and
links
{:Bitmap 1 // background bitmap
:InitiallyActive true
:CHook 3 // JPEG
:OrigContent
:ContentRef ("background.jpg")
:OrigBoxSize 800 600
:OrigPosition 0 0
}
{:Stream 2 // video clip
:InitiallyActive false
:CHook 101 // MPEG-1
:OrigContent
:ContentRef ("turin.mpg")
:Multiplex (
{:Audio 3 // audio component of the
video clip
:ComponentTag 1 // refers to audio
elementary stream
:InitiallyActive true
}
{:Video 4 // video component of the
video clip
:ComponentTag 2 // refers to
video elementary stream
:InitiallyActive true
:OrigBoxSize 352 288
:OrigPosition 40 80
}
)
}
{:HotSpot 20 // "Porta Nuova" hotspot
:InitiallyActive true
:OrigBoxSize 120 100
:OrigPosition 440 214
}
... // 25 more presentable ingredients
{:Link 30 // selecting an "interactive"
thumbnail
:EventSource (20) // "Porta Nuova"
hotspot
:EventType IsSelected
:LinkEffect (
:SetSpeed (2 1 1) // video clip:
speed is set to 1/1 (normal)
:SetCounterPosition (2 190000) //
initial point of the segment
:SetCounterEndPosition (2 246500) //
end point of the segment
:Run (2) // activate playback with
the above settings
)
}
{:Link 49 // the video clip crosses a pre
defined time code position
:EventSource (2) // video clip
:EventType CounterTrigger
:EventData 3 // booked at startup by
setCounterTrigger (2 3 190000)
:LinkEffect (
:SetData (5 // text subtitle is set
to a new string, that is
:NewRefContent ("st9.txt")) //
"Porta Nuova and via Roma"
:SetHighlightStatus (20 true) //
hotspot 20 is highlighted
)
}
... // 58 more links
)
:SceneCS 800 600 // size of the scene's
presentation space
}
```

10.6.13 An MHEG Player Java Applet — Further MHEG Examples

The Technical University of Berlin have produced a MHEG Java Engine (*http://enterprise.prz.tu-berlin.de/imw/"¿http://enterprise.prz.tu-berlin.de/imw/*

Java Class libraries (with JavaDoc documentation) and details on installation/compilation *etc* are also available. Several examples of MHEG coding, including an MHEG Introduction written in MHEG.

Running the MHEG Engine

The MHEG engine exists as a Java applet (although you can of course is the class libraries in your own java code (applications and applets)).

The MHEG engine is available in the *MHEG:MHEG Java Applet* folder on the Macintosh *Applications HD* in the Multimedia Lab.

You can run the applet through any Java enabled browser or appletviewer.

Here as an example of how to run the main applet provided for the *demo* MHEG example:

```
<applet name="MHEG 5 Engine"
 code="mheg5/POM/Mheg5Applet.class"
 codebase="applications/"
 archive="mhegwww.zip"
 width="510"
 height="346"
 align="center"
 <param name="objectBasePath" value="file:.">
 <param  name="groupIdentifier" value="demo/startup">
 <param name="mon" value="false">
</applet>
```

If you use the applet yourself you may need to change:

- the `code` and `codebase` paths — these specify where the applications and applet classes reside.
- the `groupIdentifier` *value* — for most of the application demos a `startup` MHEG file is reference first in a folder for each application. (see other examples below.

MHEG Example — The Simple MHEG Presentation

The Simple example produces the following output shown in Fig 10.9

The presentation create two buttons, labelled “Hello” and “World” respectively, and some rectangle graphics.

When pressed the button is brought to the foreground of the display.

The applet is called via:

```
<html>
  <head>
    <title>MHEG-5 Engine written in Java</title>
  </head>
  <body bgcolor="#ffffff">
    <center>
    <h1>MHEG-5 Engine</h1>
      <applet
```

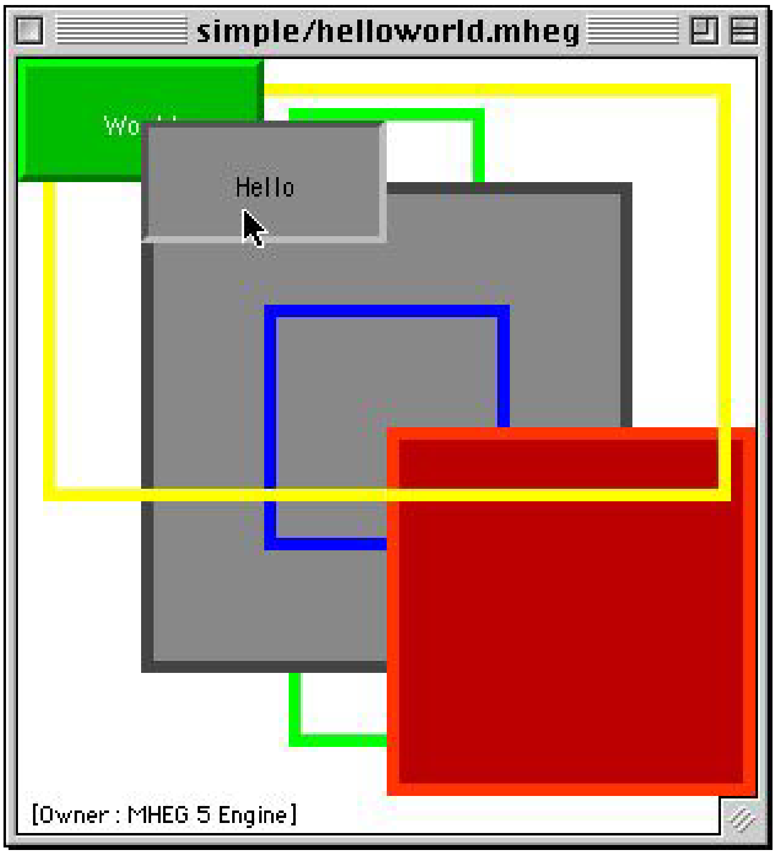


Figure 10.9: MHEG Simple application Example

```
 name="MHEG 5 Engine"
 code="mheg5/POM/Mheg5Applet.class"
 codebase="applications/"
 archive="mhegwww.zip"
 width="510"
 height="346"
 align="center"
 alt="If you had a java-enabled browser, you would see an applet here.">
 <hr>If your browser recognized the applet tag,
   you would see an applet here.<hr>
   <param name="objectBasePath" value="file:.">
   <param  name="groupIdentifier" value="simple/startup">
   <param name="mon" value="false">
      </applet>
    </center>
  </body>
</html>
```

The MHEG modules for this presentation are:

startup — calls `helloworld.mheg`

helloworld.mheg — sets up main presentation

scene1.mheg — called in `helloworld.mheg`

The MHEG code for each module is a follows:
startup:

```
{:Application  ("simple/startup" 0)
  //:OnStartUp    (:TransitionTo(("simple/helloworld.mheg" 0)))
  :Items(
  {:Link 1
   :EventSource 1
   :EventType IsRunning
   :LinkEffect (:TransitionTo(("simple/helloworld.mheg" 0)))
          }
)
}
\end{verbatim


{\bf helloworld.mheg}

\footnotesize
\begin{verbatim}
```

```
{:Scene ( "simple/helloworld.mheg" 0 )

 :Items
 (

   {:Rectangle 4000
    :OrigBoxSize 80 260 :OrigPosition 110 20
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Green :OrigRefFillColour transparent
   }
   {:Rectangle 4001
    :OrigBoxSize 200 200 :OrigPosition 50 50 :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour DarkGray :OrigRefFillColour Gray
   }
   {:Rectangle 4002
    :OrigBoxSize 100 100 :OrigPosition 100 100
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Blue :OrigRefFillColour transparent
   }
   {:Rectangle 4003
    :OrigBoxSize 150 150 :OrigPosition 150 150
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour "#FF2211" :OrigRefFillColour DarkRed
   }
   {:Rectangle 4004
    :OrigBoxSize 280 170 :OrigPosition 10 10
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Yellow :OrigRefFillColour transparent
   }

   {:PushButton 4005 :InitiallyActive true
    :OrigBoxSize 100 50
    :ButtonRefColour DarkGreen :OrigLabel "World"
   }
   {:PushButton 4006 :InitiallyActive true
    :OrigBoxSize 100 50 :OrigPosition 50 25
    :ButtonRefColour Gray :OrigLabel "Hello"
   }


   {:Link 1015
    :EventSource 4006 :EventType IsSelected
    :LinkEffect ( :BringToFront ( 4005 )
                  :SetHighlightStatus ( 4005 true ) )
   }
   {:Link 1016
    :EventSource 4005 :EventType IsSelected
    :LinkEffect ( :BringToFront ( 4006 ) )
   }
   {:Link 1017
    :EventSource 4005 :EventType CursorEnter
    :LinkEffect ( :BringToFront ( 4005 ) )
   }
   {:Link 1018
    :EventSource 4006 :EventType CursorEnter
    :LinkEffect ( :BringToFront ( 4006 ) )
   }

   {:ObjectRefVar 100 :OrigValue :ObjectRef 4001 }
   {:ObjectRefVar 101 :OrigValue :ObjectRef 4002 }
   {:Link 1000
    :EventSource 0 :EventType UserInput :EventData 1
    :LinkEffect ( :SetPosition( 4006 50 25 ) )
   }
   {:Link 1001
    :EventSource 0 :EventType UserInput :EventData 2
    :LinkEffect ( :SetPosition ( 4006 50 200 ) )
   }
   {:Link 1002
    :EventSource 0 :EventType UserInput :EventData 3
    :LinkEffect ( :Stop ( :IndirectRef 101 ) )
   }
   {:Link 1003
    :EventSource 0 :EventType UserInput :EventData 4
    :LinkEffect ( :Run ( :IndirectRef 101 ) )
   }

   {:Link 1004
    :EventSource 0 :EventType UserInput :EventData 15
    :LinkEffect ( :TransitionTo( ( "simple/scene1.mheg" 0) ) )
   }

   {:Link 1005
    :EventSource 0 :EventType UserInput :EventData 5
    :LinkEffect ( :BringToFront ( 4000 ) )
   }
   {:Link 1006
    :EventSource 0 :EventType UserInput :EventData 6
    :LinkEffect ( :BringToFront ( 4001 ) )
   }
   {:Link 1007
    :EventSource 0 :EventType UserInput :EventData 7
```

```
    :LinkEffect ( :BringToFront ( 4002 ) )
   }
   {:Link 1008
    :EventSource 0 :EventType UserInput :EventData 8
    :LinkEffect ( :BringToFront ( 4003 ) )
   }
   {:Link 1009
    :EventSource 0 :EventType UserInput :EventData 9
    :LinkEffect ( :BringToFront ( 4004 ) )
   }

   {:Link 1010
    :EventSource 0 :EventType UserInput :EventData 10
    :LinkEffect ( :SendToBack ( 4000 ) )
   }
   {:Link 1011
    :EventSource 0 :EventType UserInput :EventData 11
    :LinkEffect ( :SendToBack ( 4001 ) )
   }
   {:Link 1012
    :EventSource 0 :EventType UserInput :EventData 12
    :LinkEffect ( :SendToBack ( 4002 ) )
   }
   {:Link 1013
    :EventSource 0 :EventType UserInput :EventData 13
    :LinkEffect ( :SendToBack ( 4003 ) )
   }
   {:Link 1014
    :EventSource 0 :EventType UserInput :EventData 14
    :LinkEffect ( :SendToBack ( 4004 ) )
   }
 )

 :InputEventReg 1
 :SceneCS 300 300
 :MovingCursor true
}
```

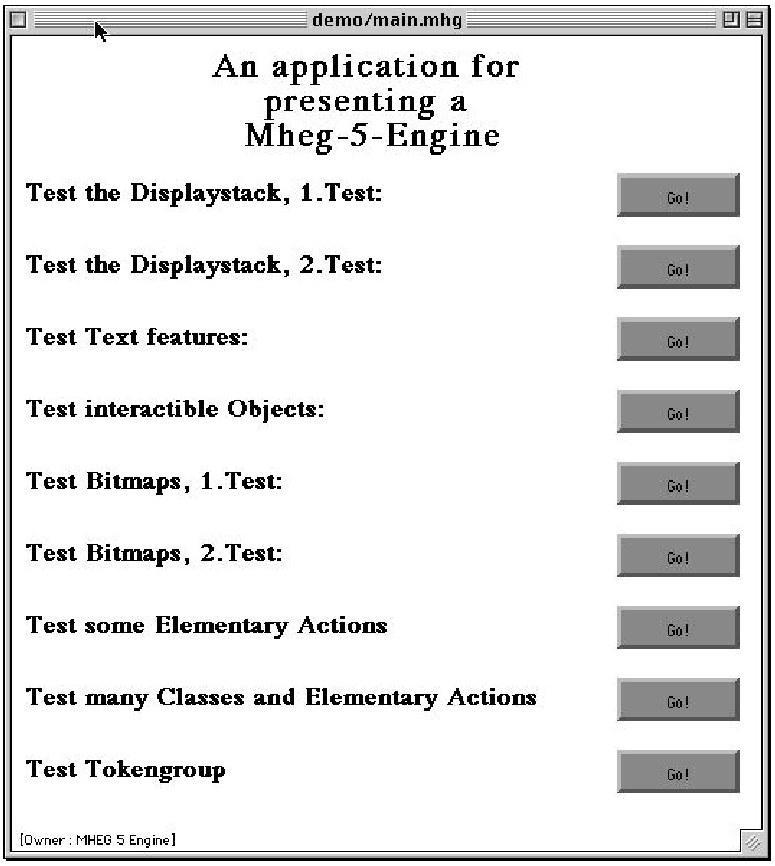
scene1.mheg

```
{:Scene ( "simple/scene1.mheg" 0 )

 :Items
 (
   {:Rectangle 4000
      :OrigBoxSize 300 300 //  200 200
    :OrigPosition 20 80
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour "Black" :OrigRefFillColour DarkRed
  }
  {:Rectangle 4001
    :OrigBoxSize 200 200 :OrigPosition 100 180
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Blue :OrigRefFillColour DarkBlue
  }

  {:Link 1000
    :EventSource 0 :EventType UserInput :EventData 6
    :LinkEffect ( :PutBehind ( 4003 4000 ) )
  }
  {:Link 1001
    :EventSource 0 :EventType UserInput :EventData 7
    :LinkEffect ( :PutBehind ( 4002 4000 ) )
  }
  {:Link 1002
    :EventSource 0 :EventType UserInput :EventData 8
    :LinkEffect ( :PutBehind ( 4001 4000 ) )
  }

  {:Link 1003
    :EventSource 0 :EventType UserInput :EventData 9
    :LinkEffect ( :PutBefore ( 4003 4000 ) )
  }
  {:Link 1004
    :EventSource 0 :EventType UserInput :EventData 10
    :LinkEffect ( :PutBefore ( 4002 4000 ) )
  }
  {:Link 1005
    :EventSource 0 :EventType UserInput :EventData 11
    :LinkEffect ( :PutBefore ( 4001 4000 ) )
  }

  {:Link 1006
    :EventSource 0 :EventType UserInput :EventData 1
    :LinkEffect ( :SetPosition ( 4000 20 80 ) )
  }
  {:Link 1007
    :EventSource 0 :EventType UserInput :EventData 2
    :LinkEffect ( :SetPosition ( 4000 20 120 ) )
  }

  {:Link 1008
    :EventSource 0 :EventType UserInput :EventData 3
      :LinkEffect ( :SetBoxSize ( 4000 200 300 ) )   // 200 200
  }
  {:Link 1009
    :EventSource 0 :EventType UserInput :EventData 4
    :LinkEffect ( :SetBoxSize ( 4000 300 300 ) )
  }

  {:Link 1100 :InitiallyActive true
    :EventSource 0 :EventType UserInput :EventData 16
    :LinkEffect ( :TransitionTo( ( "simple/helloworld.mheg" 0) ) )
  }
)
```

```
 :InputEventReg 1
 :SceneCS 400 400
}
```

MHEG Example — The Demo MHEG Presentation

The Demo example produces the output shown in Fig 10.10.


demo/main.mhg

An application for presenting a Mheg-5-Engine

Test the Displaystack, 1.Test: Go!

Test the Displaystack, 2.Test: Go!

Test Text features: Go!

Test interactible Objects: Go!

Test Bitmaps, 1.Test: Go!

Test Bitmaps, 2.Test: Go!

Test some Elementary Actions Go!

Test many Classes and Elementary Actions Go!

Test Tokengroup Go!

[Owner : MHEG 5 Engine]

Figure 10.10: MHEG Demo application Example

As can be seen many of the key features of MHEG are illustrated in further sub-windows (click on button to move to respective window). Try these out for yourself.

The following MHEG modules are used:

startup — initial module

main.mhg — Called by startup

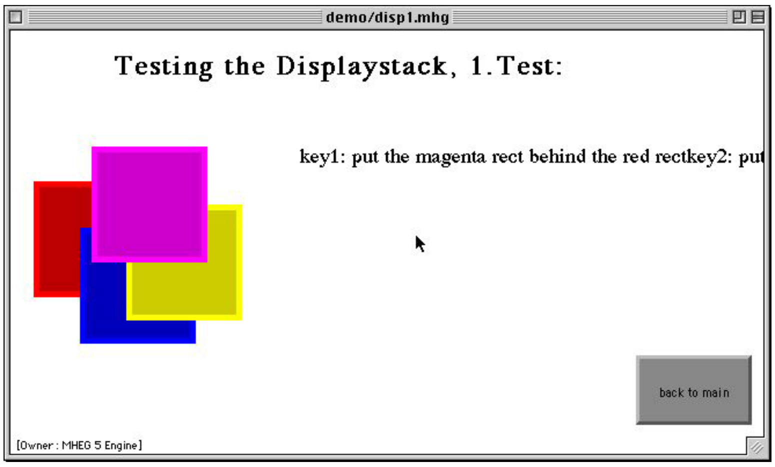

demo/disp1.mhg
Testing the Displaystack, 1.Test:
key1: put the magenta rect behind the red rectkey2: put
back to main
[Owner: MHEG 5 Engine]

Figure 10.11: MHEG Demo application Display1 Example

disp1.mhg — input from numeric keys 1 and 2 to tile rectangles (Fig 10.11)

disp2.mhg — input from numeric keys 1 and 2 to tile rectangles (different display) (Fig 10.12)

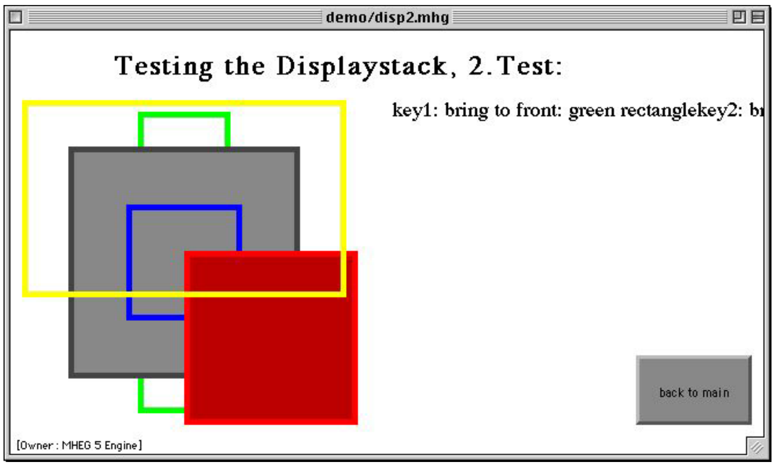

demo/disp2.mhg
Testing the Displaystack, 2.Test:
key1: bring to front: green rectanglekey2: b
back to main
[Owner: MHEG 5 Engine]

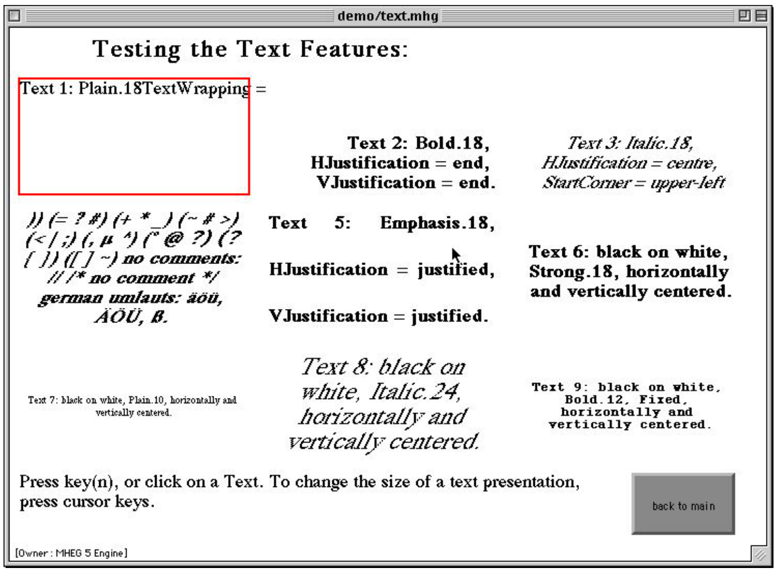
Figure 10.12: MHEG Demo application Display2 Example

text.mhg — illustrates MHEG control of text display (Fig 10.13)

intact.mhg — illustrates MHEG interactive objects (Fig 10.14)

bitmap1.mhg — illustrates MHEG display of bitmaps

bitmap2.mhg — illustrates MHEG display of bitmaps


demo/text.mhg

Testing the Text Features:

Text 1: Plain.18TextWrapping =

Text 2: Bold.18, HJustification = end, VJustification = end.

Text 3: Italic.18, HJustification = centre, StartCorner = upper-left

)) (= ? #) (+ * _) (~ # >) (< | ;) (, µ ^) (° @ ?) (? [ ]) ([ ] ~) no comments: // /* no comment */ german umlauts: äöü, ÄÖÜ, ß.

Text 5: Emphasis.18, HJustification = justified, VJustification = justified.

Text 6: black on white, Strong.18, horizontally and vertically centered.

Text 7: black on white, Plain.10, horizontally and vertically centered.

Text 8: black on white, Italic.24, horizontally and vertically centered.

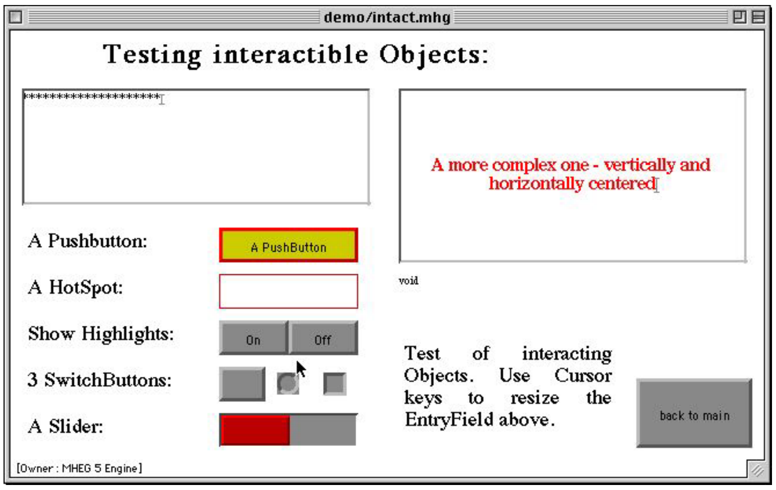
Text 9: black on white, Bold.12, Fixed, horizontally and vertically centered.

Press key(n), or click on a Text. To change the size of a text presentation, press cursor keys.

back to main

[Owner : MHEG 5 Engine]

Figure 10.13: MHEG Demo application Text Example


demo/intact.mhg

Testing interactible Objects:

****

A more complex one - vertically and horizontally centered

A Pushbutton: A PushButton

A HotSpot:

Show Highlights: On Off

3 SwitchButtons:

A Slider:

void

Test of interacting Objects. Use Cursor keys to resize the EntryField above.

back to main

[Owner : MHEG 5 Engine]

Figure 10.14: MHEG Demo application Interactive Objects Example

ea.mhg — illustrates MHEG elementary actions (Fig 10.15)

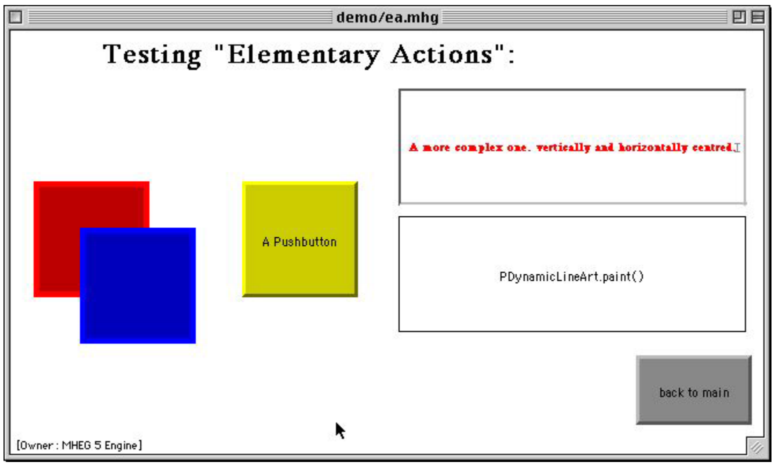

demo/ea.mhg
Testing "Elementary Actions":
A more complex one. vertically and horizontally centred.
A Pushbutton
PDynamicLineArt.paint()
back to main
[Owner: MHEG 5 Engine]

Figure 10.15: MHEG Demo application Elementary Actions Example

allcl.mhg — MHEG concrete classes and elementary actions (Fig 10.16)

token.mhg — MHEG token groups example (Fig 10.17)

The MHEG code for the modules is as follows:
Startup:

```
{:Application  ("demo/startup" 0)
  // :OnStartUp    (:TransitionTo(("demo/main.mhg" 0)))
  :Items(
  {:Link 1
   :EventSource 1
   :EventType IsRunning
   :LinkEffect (:TransitionTo(("demo/main.mhg" 0)))
          }
)
}
```

main.mhg:

```
{:Scene ( "demo/main.mhg" 0 )

 :Items
 (
   {:Text 100
    :OrigContent  'An application for presenting a Mheg-5-Engine'
    :OrigBoxSize 320 80
    :OrigPosition 90 10
    :FontAttributes Bold.24 :FontName Proportional
    :HJustification centre
    :TextWrapping true
   }
```

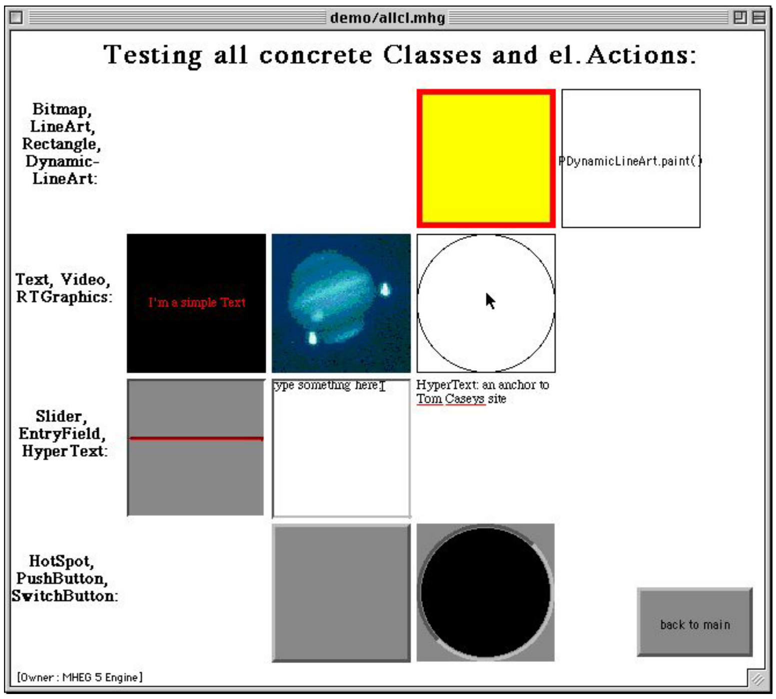

demo/allcl.mhg
Testing all concrete Classes and el. Actions:
Bitmap, LineArt, Rectangle, Dynamic-LineArt:
PDynamicLineArt.paint()
Text, Video, RTGraphics:
I'm a simple Text
Slider, EntryField, HyperText:
ype something here
HyperText: an anchor to Tom Caseys site
HotSpot, PushButton, SwitchButton:
back to main
[Owner : MHEG 5 Engine]

Figure 10.16: MHEG Demo application Concrete Classes Example

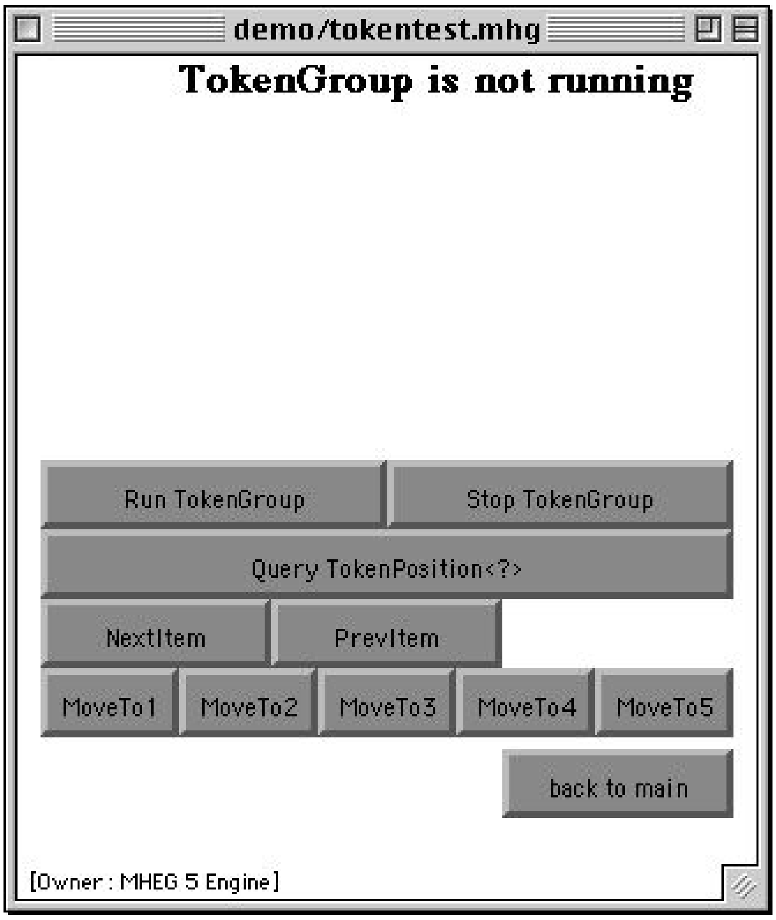


Figure 10.17: MHEG Demo application Token Groups Example

```
{:Bitmap 101
 :OrigContent :ContentRef ( "demo/tu_klein.gif" )
  :OrigBoxSize 51 39     // 0 0
  :OrigPosition 10 15
  :Tiling false
  }

{:Bitmap 102
 :OrigContent :ContentRef ( "demo/fsp_pv.gif")
          :OrigBoxSize 48 43
  :OrigPosition 420 15
  :Tiling false
  }

{:Bitmap 103
 :OrigContent :ContentRef ( "demo/prz.gif" )
          :OrigBoxSize 48 43
  :OrigPosition 470 15
  :Tiling false
  }

{:Link  110
 :EventSource 0
 :EventType UserInput
 :EventData 16
 :LinkEffect (
              :Quit (( "demo/startup" 0 ))
            )
}


// ----------------------------------------------------------------

{:Text 1000
 :OrigContent  'Test the Displaystack, 1.Test:'
 :OrigBoxSize 300 30
 :OrigPosition 10 100
 :FontAttributes Bold.18 :FontName Proportional
}

{:PushButton 1001
 :OrigBoxSize   85  30
 :OrigPosition 420 95
 :ButtonRefColour gray
 :OrigLabel "Go!"
}

{:Link 1002
 :EventSource 1001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/disp1.mhg" 0) )
            )
}

{:Link 1003
 :EventSource 1001 :EventType CursorEnter
 :LinkEffect ( :Activate(1005) )
 }
 {:Link 1004
  :EventSource 1001 :EventType CursorLeave
  :LinkEffect ( :DeActivate(1005) )
 }
 {:Link 1005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
    :LinkEffect ( :TransitionTo( ( "demo/disp1.mhg" 0) )
)
 }


 // ---------------------------------------------------------------

 {:Text 2000
  :OrigContent  'Test the Displaystack, 2.Test:'
  :OrigBoxSize 300 30
  :OrigPosition 10 150
  :FontAttributes Bold.18 :FontName Proportional
 }

 {:PushButton 2001
  :OrigBoxSize   85  30
  :OrigPosition 420 145
  :ButtonRefColour gray
  :OrigLabel "Go!"
 }

 {:Link 2002
  :EventSource 2001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/disp2.mhg" 0) )
              )
 }


 {:Link 2003
  :EventSource 2001 :EventType CursorEnter
  :LinkEffect ( :Activate( 2005 ) )
 }
 {:Link 2004
  :EventSource 2001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 2005 ) )
 }
 {:Link 2005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
    :LinkEffect ( :TransitionTo( ( "demo/disp2.mhg" 0) )
)
 }


 // ---------------------------------------------------------------
 {:Text 3000
  :OrigContent  'Test Text features:'
  :OrigBoxSize 300 30
  :OrigPosition 10 200
  :FontAttributes Bold.18 :FontName Proportional
 }

 {:PushButton 3001
  :OrigBoxSize   85  30
  :OrigPosition 420 195
  :ButtonRefColour gray
  :OrigLabel "Go!"
 }

 {:Link 3002
  :EventSource 3001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/text.mhg" 0) )
              )
 }
 {:Link 3003
  :EventSource 3001 :EventType CursorEnter
  :LinkEffect ( :Activate( 3005 ) )
 }
 {:Link 3004
  :EventSource 3001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 3005 ) )
 }
 {:Link 3005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
    :LinkEffect ( :TransitionTo( ( "demo/text.mhg" 0) )
)
 }

 // ----------------------------------------------------------------

 {:Text 4000
  :OrigContent  'Test interactible Objects:'
  :OrigBoxSize 300 30
  :OrigPosition 10 250
  :FontAttributes Bold.18 :FontName Proportional
 }

 {:PushButton 4001
  :OrigBoxSize   85  30
  :OrigPosition 420 245
  :ButtonRefColour gray
  :OrigLabel "Go!"
 }

 {:Link 4002
  :EventSource 4001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/intact.mhg" 0) )
              )
 }
 {:Link 4003
  :EventSource 4001 :EventType CursorEnter
  :LinkEffect ( :Activate( 4005 ) )
 }
 {:Link 4004
  :EventSource 4001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 4005 ) )
 }
 {:Link 4005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
    :LinkEffect ( :TransitionTo( ( "demo/intact.mhg" 0) )
)
 }

// ----------------------------------------------------------------

 {:Text 5000
  :OrigContent  'Test Bitmaps, 1.Test:'
  :OrigBoxSize 300 30
  :OrigPosition 10 300
  :FontAttributes Bold.18 :FontName Proportional
 }

 {:PushButton 5001
  :OrigBoxSize   85  30
  :OrigPosition 420 295
  :ButtonRefColour gray
  :OrigLabel "Go!"
 }

 {:Link 5002
  :EventSource 5001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/bitmap1.mhg" 0) )
             )
 }

 {:Link 5003
  :EventSource 5001 :EventType CursorEnter
  :LinkEffect ( :Activate( 5005 ) )
 }
 {:Link 5004
  :EventSource 5001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 5005 ) )
 }
 {:Link 5005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
    :LinkEffect ( :TransitionTo( ( "demo/bitmap1.mhg" 0) )
)
 }
 // ---------------------------------------------------------------

 {:Text 6000
  :OrigContent  'Test Bitmaps, 2.Test:'
  :OrigBoxSize 300 30
  :OrigPosition 10 350
  :FontAttributes Bold.18 :FontName Proportional
 }

 {:PushButton 6001
  :OrigBoxSize   85  30
  :OrigPosition 420 345
  :ButtonRefColour gray
  :OrigLabel "Go!"
 }

 {:Link 6002
  :EventSource 6001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/bitmap2.mhg" 0) )
              )
 }

 {:Link 6003
  :EventSource 6001 :EventType CursorEnter
  :LinkEffect ( :Activate( 6005 ) )
 }
 {:Link 6004
  :EventSource 6001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 6005 ) )
 }
 {:Link 6005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
    :LinkEffect ( :TransitionTo( ( "demo/bitmap2.mhg" 0) )
)
 }


// ---------------------------------------------------------------

 {:Text 7000
  :OrigContent  'Test some Elementary Actions'
  :OrigBoxSize 300 30
  :OrigPosition 10 400
  :FontAttributes Bold.18 :FontName Proportional
 }

 {:PushButton 7001
  :OrigBoxSize   85  30
  :OrigPosition 420 395
  :ButtonRefColour gray
  :OrigLabel "Go!"
 }
 {:Link 7002
  :EventSource 7001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/ea.mhg" 0) )
              )
 }
 {:Link 7003
  :EventSource 7001 :EventType CursorEnter
  :LinkEffect ( :Activate( 7005 ) )
 }
 {:Link 7004
  :EventSource 7001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 7005 ) )
 }
 {:Link 7005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
    :LinkEffect ( :TransitionTo( ( "demo/ea.mhg" 0) )
)
 }

 // ------------------------------------------------------------------
 {:Text 8000
  :OrigContent  'Test many Classes and Elementary Actions'
  :OrigBoxSize 400 30
  :OrigPosition 10 450
  :FontAttributes Bold.18 :FontName Proportional
 }

 {:PushButton 8001
  :OrigBoxSize   85  30
  :OrigPosition 420 445
  :ButtonRefColour gray
  :OrigLabel "Go!"
 }

 {:Link 8002
  :EventSource 8001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/allcl.mhg" 0) )
              )
 }

 {:Link 8003
  :EventSource 8001 :EventType CursorEnter
  :LinkEffect ( :Activate( 8005 ) )
 }
 {:Link 8004
  :EventSource 8001 :EventType CursorLeave
  :LinkEffect ( :DeActivate( 8005 ) )
 }
 {:Link 8005
    :InitiallyActive false
    :EventSource 0
    :EventType UserInput
    :EventData 15
```

```
      :LinkEffect ( :TransitionTo( ( "demo/allcl.mhg" 0) )
  )
   }

   // ---------------------------------------------------------------

   {:Text 9000
    :OrigContent  'Test Tokengroup'
    :OrigBoxSize 400 30
    :OrigPosition 10 500
    :FontAttributes Bold.18 :FontName Proportional
   }

   {:PushButton 9001
    :OrigBoxSize   85  30
    :OrigPosition 420 495
    :ButtonRefColour gray
    :OrigLabel "Go!"
   }

   {:Link 9002
    :EventSource 9001 :EventType IsSelected
    :LinkEffect (  :TransitionTo( ( "demo/token.mhg" 0) )
                )
   }
   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/token.mhg" 0) )
  )
   }


 )
 :InputEventReg 1
 :SceneCS 520 550
 :MovingCursor true
}
```

disp1.mhg:

```
{:Scene ( "demo/disp1.mhg" 0 )

 :Items
 (
  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                  :Quit (( "demo/startup" 0 ))
                )
   }


   {:Text 1000
    :OrigContent  'Testing the Displaystack, 1.Test:'
    :OrigBoxSize 500 30
    :OrigPosition 90 20
    :FontAttributes Bold.24 :FontName Proportional
   }

   {:Text 1001
    :OrigContent  'key1: put the magenta rect behind the red rect
key2: put the yellow rect behind the red rect
key3: put the blue rect behind the red rect
key4: put the magenta rect before the red rect
key5: put the yellow rect before the red rect
key6: put the blue rect before the red rect.'
    :OrigBoxSize 400 200
    :OrigPosition 250 100
    :FontAttributes Plain.18 :FontName Proportional
    :TextWrapping false
   }

   {:Bitmap 1010
    :OrigContent :ContentRef ( "demo/tu_klein.gif" )
     :OrigBoxSize 51 39     // 0 0
     :OrigPosition 10 15
     :Tiling false
     }

   {:Bitmap 1011
    :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
     :OrigPosition 540 15
     :Tiling false
     }

   {:Bitmap 1012
    :OrigContent :ContentRef ( "demo/prz.gif" )
             :OrigBoxSize 48 43
     :OrigPosition 590 15
     :Tiling false
     }



   // ------------------------------------------------------------------

   {:Rectangle 4000
    :OrigBoxSize 100 100 :OrigPosition 20 130
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Red :OrigRefFillColour DarkRed
   }
```

```
{:Rectangle 4001
 :OrigBoxSize 100 100 :OrigPosition 60 170
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Blue :OrigRefFillColour DarkBlue
}
{:Rectangle 4002
 :OrigBoxSize 100 100 :OrigPosition 100 150
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Yellow :OrigRefFillColour DarkYellow
}
{:Rectangle 4003
 :OrigBoxSize 100 100 :OrigPosition 70 100
 :OrigLineWidth 5 :OrigLineStyle 1
 :OrigRefLineColour Magenta :OrigRefFillColour DarkMagenta
}

// ---------------------------------------------------------

{:Link 2000
 :EventSource 0 :EventType UserInput :EventData 6
 :LinkEffect ( :PutBehind ( 4003 4000 ) )
}
{:Link 2001
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect ( :PutBehind ( 4002 4000 ) )
}
{:Link 2002
 :EventSource 0 :EventType UserInput :EventData 8
 :LinkEffect ( :PutBehind ( 4001 4000 ) )
}

{:Link 2003
 :EventSource 0 :EventType UserInput :EventData 9
 :LinkEffect ( :PutBefore ( 4003 4000 ) )
}
{:Link 2004
 :EventSource 0 :EventType UserInput :EventData 10
 :LinkEffect ( :PutBefore ( 4002 4000 ) )
}
{:Link 2005
 :EventSource 0 :EventType UserInput :EventData 11
 :LinkEffect ( :PutBefore ( 4001 4000 ) )
}

// ---------------------------------------------------------

{:PushButton 9001
 :OrigBoxSize   100  60
 :OrigPosition  540 280
 :ButtonRefColour gray
 :OrigLabel "back to main"
}

{:Link 9002
 :EventSource 9001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
}
```

```
   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
  )
   }

 )

 :InputEventReg 1
 :SceneCS 650 350
 :MovingCursor true
}
```

disp2.mhg:

```
{:Scene ( "demo/disp2.mhg" 0 )

 :Items
 (
  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                  :Quit (( "demo/startup" 0 ))
                )
   }

   {:Text 1000
    :OrigContent  'Testing the Displaystack, 2.Test:'
    :OrigBoxSize 500 30
    :OrigPosition 90 20
    :FontAttributes Bold.24 :FontName Proportional
   }

   {:Text 1001
    :OrigContent  'key1: bring to front: green rectangle
key2: bring to front: gray rectangle
key3: bring to front: blue rectangle
key4: bring to front: red rectangle
key5: bring to front: yellow rectangle
key6: send to back: green rectangle
key7: send to back: gray rectangle
key8: send to back: blue rectangle
key9: send to back: red rectangle
key0: send to back: yellow rectangle'
    :OrigBoxSize 300 220
    :OrigPosition 330 60
    :FontAttributes Plain.18 :FontName Proportional
    :TextWrapping false
   }

   {:Bitmap 101
    :OrigContent :ContentRef ( "demo/tu_klein.gif" )
     :OrigBoxSize 51 39     // 0 0
     :OrigPosition 10 15
     :Tiling false
     }

   {:Bitmap 102
    :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
     :OrigPosition 540 15
     :Tiling false
     }

   {:Bitmap 103
    :OrigContent :ContentRef ( "demo/prz.gif" )
             :OrigBoxSize 48 43
     :OrigPosition 590 15
     :Tiling false
     }


   // ----------------------------------------------------------------

   {:Rectangle 4000
    :OrigBoxSize 80 260 :OrigPosition 110 70
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Green :OrigRefFillColour transparent
   }
   {:Rectangle 4001
    :OrigBoxSize 200 200 :OrigPosition 50 100
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour DarkGray :OrigRefFillColour Gray
   }
   {:Rectangle 4002
    :OrigBoxSize 100 100 :OrigPosition 100 150
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Blue :OrigRefFillColour transparent
   }
   {:Rectangle 4003
    :OrigBoxSize 150 150 :OrigPosition 150 190
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Red :OrigRefFillColour DarkRed
   }
   {:Rectangle 4004
    :OrigBoxSize 280 170 :OrigPosition 10 60
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Yellow :OrigRefFillColour transparent
   }
```

```
// -----------------------------------------------------------------


{:Link 1005
 :EventSource 0 :EventType UserInput :EventData 6
 :LinkEffect ( :BringToFront ( 4000 ) )
}
{:Link 1006
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect ( :BringToFront ( 4001 ) )
}
{:Link 1007
 :EventSource 0 :EventType UserInput :EventData 8
 :LinkEffect ( :BringToFront ( 4002 ) )
}
{:Link 1008
 :EventSource 0 :EventType UserInput :EventData 9
 :LinkEffect ( :BringToFront ( 4003 ) )
}
{:Link 1009
 :EventSource 0 :EventType UserInput :EventData 10
 :LinkEffect ( :BringToFront ( 4004 ) )
}

{:Link 1010
 :EventSource 0 :EventType UserInput :EventData 11
 :LinkEffect ( :SendToBack ( 4000 ) )
}
{:Link 1011
 :EventSource 0 :EventType UserInput :EventData 12
 :LinkEffect ( :SendToBack ( 4001 ) )
}
{:Link 1012
 :EventSource 0 :EventType UserInput :EventData 13
 :LinkEffect ( :SendToBack ( 4002 ) )
}
{:Link 1013
 :EventSource 0 :EventType UserInput :EventData 14
 :LinkEffect ( :SendToBack ( 4003 ) )
}
{:Link 1014
 :EventSource 0 :EventType UserInput :EventData 5
 :LinkEffect ( :SendToBack ( 4004 ) )
}

// ----------------------------------------------------------

{:PushButton 9001
 :OrigBoxSize   100  60
 :OrigPosition  540 280
 :ButtonRefColour gray
 :OrigLabel "back to main"
}

{:Link 9002
 :EventSource 9001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
```

```
   }

   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
  )
   }

 )

 :InputEventReg 1
 :SceneCS 650 350
 :MovingCursor true
}
```

text.mhg:

```
{:Scene ( "demo/text.mhg" 0 )

 :Items
 (
   {:Text 1000
    :OrigContent  'Testing the Text Features:'
    :OrigBoxSize 500 30
    :OrigPosition 80 10
    :FontAttributes Bold.24 :FontName Proportional
   }


   {:Text 3000
    :OrigContent  'Text 1: Plain.18
TextWrapping = false'
    :OrigBoxSize 220 110
    :OrigPosition 10 50
    :OrigFont 'OctetString in OriginalFont'
    :FontAttributes Plain.18
    :FontName Proportional
    :TextColour black
    :BackgroundColour white
    :CharacterSet 1848
    :HJustification start
    :VJustification start
    :LineOrientation horizontal
    :StartCorner upper-left
    :TextWrapping false
   }
```

```
{:Bitmap 101
  :OrigContent :ContentRef ( "demo/tu_klein.gif" )
   :OrigBoxSize 51 39     // 0 0
   :OrigPosition 10 5
   :Tiling false
   }

 {:Bitmap 102
  :OrigContent :ContentRef ( "demo/fsp_pv.gif")
          :OrigBoxSize 48 43
   :OrigPosition 625 5
   :Tiling false
   }

 {:Bitmap 103
  :OrigContent :ContentRef ( "demo/prz.gif" )
          :OrigBoxSize 48 43
   :OrigPosition 675 5
   :Tiling false
   }


{:Link  110
  :EventSource 0
  :EventType UserInput
  :EventData 16
  :LinkEffect (
                :Quit (( "demo/startup" 0 ))
              )
 }

 {:HotSpot 3100
  :OrigBoxSize 220 110
  :OrigPosition 10 50
 }

 {:Text 3001
  :OrigContent  'Text 2: Bold.18,
                 HJustification = end,
                 VJustification = end.'
  :OrigBoxSize 220 110
  :OrigPosition 250 50
  :OrigFont 'OctetString in OriginalFont'
  :FontAttributes Bold.18
  :TextColour black
  :BackgroundColour white
  :CharacterSet 1848
  :HJustification end
  :VJustification end
  :LineOrientation horizontal
  :StartCorner upper-left
  :TextWrapping true
 }
 {:HotSpot 3101
  :OrigBoxSize 220 110
  :OrigPosition 250 50
```

```
}

{:Text 3002
 :OrigContent  'Text 3: Italic.18,
                HJustification = centre,
                StartCorner = upper-left'
 :OrigBoxSize 220 110
 :OrigPosition 490 50
 :OrigFont 'OctetString in OriginalFont'
 :FontAttributes Italic.18
 :TextColour black
 :BackgroundColour white
 :CharacterSet 1848
 :HJustification centre
 :VJustification end
 :LineOrientation horizontal
 :StartCorner upper-left
 :TextWrapping true
}
{:HotSpot 3102
 :OrigBoxSize 220 110
 :OrigPosition 490 50
}

{:Text 3003
 :OrigContent  :ContentRef ( "demo/to_load.txt" )
 :OrigBoxSize 220 110
 :OrigPosition 10 180
 :OrigFont 'OctetString in OriginalFont'
 :FontAttributes Bold-Italic.18
 :TextColour black
 :BackgroundColour white
 :CharacterSet 1848
 :HJustification centre
 :VJustification end
 :LineOrientation horizontal
 :StartCorner upper-left
 :TextWrapping true
}
{:HotSpot 3103
 :OrigBoxSize 220 110
 :OrigPosition 10 180
}

{:Text 3004
 :OrigContent  'Text 5: Emphasis.18,
                HJustification = justified,
                VJustification = justified.'
 :OrigBoxSize 220 110
 :OrigPosition 250 180
 :OrigFont 'OctetString in OriginalFont'
 :FontAttributes Emphasis.18
 :TextColour black
 :BackgroundColour white
 :CharacterSet 1848
 :HJustification justified
 :VJustification justified
 :LineOrientation horizontal
 :StartCorner upper-left
 :TextWrapping true
}
{:HotSpot 3104
 :OrigBoxSize 220 110
 :OrigPosition 250 180
 }

{:Text 3005
 :OrigContent  'Text 6: black on white, Strong.18,
               horizontally and vertically centered.'
 :OrigBoxSize 220 110
 :OrigPosition 490 180
 :FontAttributes Strong.18
 :TextColour black
 :BackgroundColour white
 :HJustification centre
 :VJustification centre
 :TextWrapping true
}
{:HotSpot 3105
 :OrigBoxSize 220 110
 :OrigPosition 490 180
}

{:Text 3006
 :OrigContent  'Text 7: black on white, Plain.10,
               horizontally and vertically centered.'
 :OrigBoxSize 220 110
 :OrigPosition 10 310
 :FontAttributes Plain.10
 :TextColour black
 :BackgroundColour white
 :HJustification centre
 :VJustification centre
 :TextWrapping true
}
{:HotSpot 3106
 :OrigBoxSize 220 110
 :OrigPosition 10 310
}

{:Text 3007
 :OrigContent  'Text 8: black on white, Italic.24,
               horizontally and vertically centered.'
 :OrigBoxSize 220 110
 :OrigPosition 250 310
 :FontAttributes Italic.24
 :TextColour black
 :BackgroundColour white
 :HJustification centre
 :VJustification centre
 :TextWrapping true
}
{:HotSpot 3107
 :OrigBoxSize 220 110
 :OrigPosition 250 310
}

{:Text 3008
 :OrigContent  'Text 9: black on white, Bold.12, Fixed,
                horizontally and vertically centered.'
 :OrigBoxSize 220 110
 :OrigPosition 490 310
 :FontAttributes Bold.12
 :FontName Fixed
 :TextColour black
 :BackgroundColour white
 :HJustification centre
 :VJustification centre
 :TextWrapping true
}
{:HotSpot 3108
 :OrigBoxSize 220 110
 :OrigPosition 490 310
}

// ---------------------------------------------------------

{:Rectangle 2000
 :OrigBoxSize 224 114 :OrigPosition 8 48
 :OrigLineWidth 2 :OrigLineStyle 1
 :OrigRefLineColour Red
 :OrigRefFillColour transparent
}

{:IntegerVar 32000 :OrigValue 350 }
{:IntegerVar 32001 :OrigValue 120 }
{:ObjectRefVar 32002 :OrigValue :ObjectRef 3000 }   // the text
{:ObjectRefVar 32003 :OrigValue :ObjectRef 3100 }   // the hotspot

{:IntegerVar 2001  :OrigValue 8 }
{:IntegerVar 2002  :OrigValue 48 }
{:IntegerVar 2003  :OrigValue 224 }
{:IntegerVar 2004  :OrigValue 114 }


{:Link 1004
 :EventSource 0 :EventType UserInput :EventData 1
 :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
               :GetBoxSize (:IndirectRef 32002 2003 2004 )
               :Subtract ( 32001 1 )
               :Add       ( 2003 4 )
               :Add       ( 2004 3 )
               :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef 32001 )
               :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef 32001 )
               :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
               :BringToFront (2000)
               :BringToFront (:IndirectRef 32003)
             )
}
{:Link 1005
 :EventSource 0 :EventType UserInput :EventData 2
 :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
               :GetBoxSize (:IndirectRef 32002 2003 2004 )
               :Add ( 32001 1 )
               :Add      ( 2003 4 )
               :Add      ( 2004 5 )
               :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef 32001 )
               :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef 32001 )
               :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
               :BringToFront (2000)
               :BringToFront (:IndirectRef 32003)
             )
}
{:Link 1006
 :EventSource 0 :EventType UserInput :EventData 3
 :LinkEffect ( :GetBoxSize (:IndirectRef 32002  32000 32001 )
               :GetBoxSize (:IndirectRef 32002 2003 2004 )
               :Subtract ( 32000 1 )
               :Add      ( 2003 3 )
               :Add      ( 2004 4 )
               :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef 32001 )
               :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef 32001 )
               :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
               :BringToFront (2000)
               :BringToFront (:IndirectRef 32003)
             )
}
{:Link 1007
 :EventSource 0 :EventType UserInput :EventData 4
 :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
               :GetBoxSize (:IndirectRef 32002 2003 2004 )
               :Add ( 32000 1 )
               :Add      ( 2003 5 )
               :Add      ( 2004 4 )
               :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef 32001 )
               :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef 32001 )
               :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
               :BringToFront (2000)
               :BringToFront (:IndirectRef 32003)

             )
}


// ----------------------------------------------------------

{:Link 1008
 :EventSource 0 :EventType UserInput :EventData 6
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3000 )
                :SetVariable ( 32003  :GObjectRef 3100 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )

             )
}
{:Link 1108
 :EventSource 3100 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3000 )
                :SetVariable ( 32003  :GObjectRef 3100 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1009
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3001 )
                :SetVariable ( 32003  :GObjectRef 3101 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1109
 :EventSource 3101 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3001 )
                :SetVariable ( 32003  :GObjectRef 3101 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1010
 :EventSource 0 :EventType UserInput :EventData 8
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3002 )
                :SetVariable ( 32003  :GObjectRef 3102 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1110
 :EventSource 3102 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3002 )
                :SetVariable ( 32003  :GObjectRef 3102 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1011
 :EventSource 0 :EventType UserInput :EventData 9
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3003 )
                :SetVariable ( 32003  :GObjectRef 3103 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1111
 :EventSource 3103 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3003 )
                :SetVariable ( 32003  :GObjectRef 3103 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1012
 :EventSource 0 :EventType UserInput :EventData 10
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3004 )
                :SetVariable ( 32003  :GObjectRef 3104 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1112
 :EventSource 3104 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3004 )
                :SetVariable ( 32003  :GObjectRef 3104 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}
```

```
// ---

{:Link 1013
 :EventSource 0 :EventType UserInput :EventData 11
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3005 )
                :SetVariable ( 32003  :GObjectRef 3105 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1113
 :EventSource 3105 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3005 )
                :SetVariable ( 32003  :GObjectRef 3105 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1014
 :EventSource 0 :EventType UserInput :EventData 12
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3006 )
                :SetVariable ( 32003  :GObjectRef 3106 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )                 )
}
{:Link 1114
```

```
 :EventSource 3106 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3006 )
                :SetVariable ( 32003  :GObjectRef 3106 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1015
 :EventSource 0 :EventType UserInput :EventData 13
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3007 )
                :SetVariable ( 32003  :GObjectRef 3107 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1115
 :EventSource 3107 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3007 )
                :SetVariable ( 32003  :GObjectRef 3107 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}

// ---

{:Link 1016
```

```
 :EventSource 0 :EventType UserInput :EventData 14
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3008 )
                :SetVariable ( 32003  :GObjectRef 3108 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1116
 :EventSource 3108 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3008 )
                :SetVariable ( 32003  :GObjectRef 3108 )
    :GetPosition ( :IndirectRef 32002 2001 2002 )
    :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
    :Subtract ( 2001 2 )
    :Subtract ( 2002 2 )
    :Add      ( 2003 4 )
    :Add      ( 2004 4 )
    :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
    :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
    :BringToFront ( 2000 )
    :BringToFront ( :IndirectRef 32002 )
    :BringToFront ( :IndirectRef 32003 )
             )
}


// ----------------------------------------------------------


{:Text 9000
 :OrigContent  'Press key(n), or click on a Text.
      To change the size of a text presentation,
      press cursor keys.'
 :OrigBoxSize 550 110
 :OrigPosition 10 430
 :FontAttributes Plain.18 :FontName Proportional
 :TextWrapping true
}

{:PushButton 9001
 :OrigBoxSize   100  60
 :OrigPosition  600 430
 :ButtonRefColour gray
 :OrigLabel "back to main"
}

{:Link 9002
 :EventSource 9001 :EventType IsSelected
```

```
    :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
   }

   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
  )
   }

 )

 :InputEventReg 1
 :SceneCS 730 500
 :MovingCursor true
}
```

intact.mhg:

```
{:Scene ( "demo/intact.mhg" 0 )

 :Items
 (
  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                  :Quit (( "demo/startup" 0 ))
                )
   }
   {:Text 1000
    :OrigContent  'Testing interactible Objects:'
    :OrigBoxSize 500 30
    :OrigPosition 80 10
    :FontAttributes Bold.24 :FontName Proportional
   }

  {:Bitmap 101
    :OrigContent :ContentRef ( "demo/tu_klein.gif" )
     :OrigBoxSize 51 39     // 0 0
     :OrigPosition 10 5
     :Tiling false
     }

   {:Bitmap 102
    :OrigContent :ContentRef ( "demo/fsp_pv.gif")
```

```
          :OrigBoxSize 48 43
  :OrigPosition 545 5
  :Tiling false
  }

{:Bitmap 103
 :OrigContent :ContentRef ( "demo/prz.gif" )
          :OrigBoxSize 48 43
  :OrigPosition 595 5
  :Tiling false
  }



{:Text 1001
 :OrigContent  'Test of interacting Objects. Use Cursor keys to resize
                the EntryField above.'
 :OrigBoxSize 180 200
 :OrigPosition 340 270
 :FontAttributes Plain.18 :FontName Proportional
 :HJustification justified
 :TextWrapping true
}


{:EntryField 3001
 :OrigContent 'A simple Entryfield.'
 :OrigBoxSize 300 100
 :OrigPosition 10 50
 :ObscuredInput true
}

//  'A more complex one - vertically and horizontally centered'
{:EntryField 3002
 :OrigContent 'A more complex one - vertically and horizontally centered'
 :OrigBoxSize 300 150
 :OrigPosition 335 50
 :FontAttributes Bold.16
 :FontName Proportional
 :TextColour red
 :BackgroundColour white
 :HJustification centre
 :VJustification centre
}

{:OStringVar 3003
 :OrigValue  "void"
}

{:Link 3004
 :EventSource 3002 :EventType InteractionCompleted
 :LinkEffect ( :GetTextData ( 3002 3003 )
               :SetData (3005 :indirectref 3003) )
}

{:Text 3005
 :OrigContent  'void'
 :OrigBoxSize 300 50
 :OrigPosition 335 210
 :FontAttributes Plain.10 :FontName Proportional
}


// ---------------------------------------------------------


{:Text 4000
 :OrigContent  'A Pushbutton:'
 :OrigBoxSize 120 30
 :OrigPosition 15 173
 :FontAttributes Plain.18 :FontName Proportional
}

{:PushButton 4001
 :OrigBoxSize  120 30
 :OrigPosition  180 170
 :ButtonRefColour darkYellow
 :OrigLabel "A PushButton"
}

{:Rectangle 4002
 :InitiallyActive false
 :OrigBoxSize 30 30
 :OrigPosition 130 170
 :OrigLineWidth 5
 :OrigLineStyle 1
 :OrigRefLineColour DarkRed
 :OrigRefFillColour Red
}

// ----

{:Text 4010
 :OrigContent  'A HotSpot:'
 :OrigBoxSize 120 30
 :OrigPosition 15 213
 :FontAttributes Plain.18 :FontName Proportional
}

{:HotSpot 4011
 :OrigBoxSize  120 30
 :OrigPosition   180 210
}

{:Rectangle 4012
 :InitiallyActive false
 :OrigBoxSize 30 30
 :OrigPosition 130 210
 :OrigLineWidth 5
 :OrigLineStyle 1
 :OrigRefLineColour DarkRed
 :OrigRefFillColour Red
}
```

```
// ----

{:Text 4020
 :OrigContent  '3 SwitchButtons:'
 :OrigBoxSize 135 30
 :OrigPosition 15 293
 :FontAttributes Plain.18 :FontName Proportional
}

{:SwitchButton 4021
 :OrigBoxSize  40 30
 :OrigPosition   180 290
 :OrigLabel "A SwitchButton"
 :ButtonStyle pushbutton
}
{:SwitchButton 4022
 :OrigBoxSize  20 20
 :OrigPosition   230 295
 :OrigLabel "A SwitchButton"
 :ButtonStyle radiobutton
}
{:SwitchButton 4023
 :OrigBoxSize  20 20
 :OrigPosition   270 295
 :OrigLabel "A SwitchButton"
 :ButtonStyle checkbox
}

{:Text 4025
 :OrigContent  'A Slider:'
 :OrigBoxSize 135 30
 :OrigPosition 15 333
 :FontAttributes Plain.18 :FontName Proportional
}

{:Slider 4026
 :OrigBoxSize  120 30
 :OrigPosition   180 330
 :Orientation right
 :MaxValue 100
 :InitialPortion 50
 :SliderStyle proportional
}

//{:Rectangle 4029
// :InitiallyActive false
// :OrigBoxSize 30 30
// :OrigPosition 130 250
// :OrigLineWidth 5
// :OrigLineStyle 1
// :OrigRefLineColour DarkRed
// :OrigRefFillColour Red
//}

// ---------------------------------------------------------

{:Text 4040
```

```
 :OrigContent  'Show Highlights:'
 :OrigBoxSize 150 30
 :OrigPosition 15 253
 :FontAttributes Plain.18 :FontName Proportional
}

{:PushButton 4041
 :OrigBoxSize   60  30
 :OrigPosition 180 250
 :ButtonRefColour gray
 :OrigLabel "On"
}

{:PushButton 4051
 :OrigBoxSize   60  30
 :OrigPosition 240 250
 :ButtonRefColour gray
 :OrigLabel "Off"
}

// ---------------------------------------------------------


{:Link 5000
 :EventSource 4041 :EventType IsSelected
 :LinkEffect ( :SetHighlightStatus ( 4001 true )
               :SetHighlightStatus ( 4011 true )
             )
}
{:Link 5001
 :EventSource 4051 :EventType IsSelected
 :LinkEffect ( :SetHighlightStatus ( 4001 false )
               :SetHighlightStatus ( 4011 false )
             )
}

// ---------------------------------------------------------

{:Link 5010
 :EventSource 4001 :EventType IsSelected
 :LinkEffect ( :Run ( 4002 ) )
}
{:Link 5011
 :EventSource 4001 :EventType IsDeselected
 :LinkEffect ( :Stop ( 4002 ) )
}

{:Link 5020
 :EventSource 4011 :EventType IsSelected
 :LinkEffect ( :Run ( 4012 ) )
}
{:Link 5021
 :EventSource 4011 :EventType IsDeselected
 :LinkEffect ( :Stop ( 4012 ) )
}

// ---------------------------------------------------------
```

```
{:IntegerVar 32000 :OrigValue -1 }
{:IntegerVar 32001 :OrigValue -1 }

{:Link 1020
 :EventSource 0 :EventType UserInput :EventData 4
 :LinkEffect ( :GetBoxSize ( 3002 32000 32001 )
               :Add        ( 32000 4 )
               :SetBoxSize ( 3002 :IndirectRef 32000 :IndirectRef 32001 ))}
{:Link 1021
 :EventSource 0 :EventType UserInput :EventData 3
 :LinkEffect ( :GetBoxSize ( 3002 32000 32001 )
               :Subtract   ( 32000 4 )
               :SetBoxSize ( 3002 :IndirectRef 32000 :IndirectRef 32001 ))}
{:Link 1022
 :EventSource 0 :EventType UserInput :EventData 2
 :LinkEffect ( :GetBoxSize ( 3002 32000 32001 )
               :Add        ( 32001 4 )
               :SetBoxSize ( 3002 :IndirectRef 32000 :IndirectRef 32001 ))}
{:Link 1023
 :EventSource 0 :EventType UserInput :EventData 1
 :LinkEffect ( :GetBoxSize ( 3002 32000 32001 )
               :Subtract   ( 32001 4 )
               :SetBoxSize ( 3002 :IndirectRef 32000 :IndirectRef 32001 ))}

// -----------------------------------------------------------

{:PushButton 9001
 :OrigBoxSize   100  60
 :OrigPosition  540 300
 :ButtonRefColour gray
 :OrigLabel "back to main"
}

{:Link 9002
 :EventSource 9001 :EventType IsSelected
 :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
}

{:Link 9003
 :EventSource 9001 :EventType CursorEnter
 :LinkEffect ( :Activate( 9005 ) )
}
{:Link 9004
 :EventSource 9001 :EventType CursorLeave
 :LinkEffect ( :DeActivate( 9005 ) )
}
{:Link 9005
   :InitiallyActive false
   :EventSource 0
   :EventType UserInput
   :EventData 15
   :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
)
}
```

```
 )

 :InputEventReg 1
 :SceneCS       650 370
 :MovingCursor  true
}
```

bitmap1.mhg:

```
{:Scene ( "demo/bitmap.mhg" 0 )

  :Items (
       {:Text 100
        :OrigContent  'Testing the drawing of Bitmaps, 1.Test:'
        :OrigBoxSize 500 30
        :OrigPosition 80 10
        :FontAttributes Bold.24
        :FontName Proportional
       }

       {:Bitmap 101
             :OrigContent :ContentRef ( "demo/tu_klein.gif" )
     :OrigBoxSize 51 39     // 0 0
     :OrigPosition 10 5
     :Tiling false
     }

       {:Bitmap 102
        :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
     :OrigPosition 595 5
     :Tiling false
     }

       {:Bitmap 103
        :OrigContent :ContentRef ( "demo/prz.gif" )
             :OrigBoxSize 48 43
     :OrigPosition 645 5
     :Tiling false
     }

  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                 :Quit (( "demo/startup" 0 ))
               )
   }
  // ---------------------------

  {:Bitmap 3000
     :OrigContent
     :ContentRef ( "demo/pirogue.jpg" )
     :OrigBoxSize 158 158
     :OrigPosition 10 100
     :Tiling false
     }
          {:HotSpot 3001
             :OrigBoxSize 158 158
             :OrigPosition 10 100
          }
  {:Text 3002
             :OrigContent  'A JPEG-encoded picture, drawn in another size:'
             :OrigBoxSize 330 40
             :OrigPosition 10 48
             :FontAttributes Plain.18
             :FontName Proportional
             :TextWrapping true
          }



  // ---------------------------

  {:Bitmap 3100
     :OrigContent
     :ContentRef ( "demo/pirogue.gif" )
     :OrigBoxSize 158 158
     :OrigPosition 350 100
     :Tiling false
     }
          {:HotSpot 3101
             :OrigBoxSize 158 158
             :OrigPosition 350 100
          }
  {:Text 3102
             :OrigContent  'The GIF-encoded picture rescaled:'
             :OrigBoxSize 300 20
             :OrigPosition 350 48
             :FontAttributes Plain.18 :FontName Proportional
             :TextWrapping true
          }


  // ---------------------------

  {:Bitmap 3200
     :OrigContent
     :ContentRef ( "demo/pirogue.jpg" )
     :OrigBoxSize 0 0
     :OrigPosition 10 310
     :Tiling false
     }
  {:Text 3202
             :OrigContent  'The JPEG-encoded picture in original size:'
             :OrigBoxSize 320 20
             :OrigPosition 10 270
             :FontAttributes Plain.18 :FontName Proportional
          }

  // ---------------------------
```

```
{:Bitmap 3300
   :OrigContent
   :ContentRef ( "demo/pirogue.gif" )
   :OrigBoxSize 0 0
   :OrigPosition 350 310
   :Tiling false
   }
{:Text 3302
           :OrigContent  'The GIF-encoded picture in original size:'
           :OrigBoxSize 320 20
           :OrigPosition 350 270
           :FontAttributes Plain.18 :FontName Proportional
        }

        // ----------------------------------------------------------

        {:Rectangle 2000
         :OrigBoxSize 162 162 :OrigPosition 8 98
         :OrigLineWidth 2 :OrigLineStyle 1
         :OrigRefLineColour Red
         :OrigRefFillColour transparent
        }

        {:IntegerVar 32000 :OrigValue 158 }
        {:IntegerVar 32001 :OrigValue 158 }
        {:ObjectRefVar 32002 :OrigValue :ObjectRef 3000 }        // the bitmap
        {:ObjectRefVar 32003 :OrigValue :ObjectRef 3001 }        // the hotspot

        {:IntegerVar 2001  :OrigValue 8 }
        {:IntegerVar 2002  :OrigValue 98 }
        {:IntegerVar 2003  :OrigValue 162 }
        {:IntegerVar 2004  :OrigValue 162 }


        {:Link 1004
         :EventSource 0 :EventType UserInput :EventData 1
         :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
                       :GetBoxSize (:IndirectRef 32002 2003 2004 )
                       :Subtract ( 32001 1 )
                       :Add       ( 2003 4 )
                       :Add       ( 2004 3 )
                       :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef 32001 )
                       :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef 32001 )
                       :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                       :BringToFront (2000)
                       :BringToFront (:IndirectRef 32003)
              )
        }
        {:Link 1005
         :EventSource 0 :EventType UserInput :EventData 2
         :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
                       :GetBoxSize (:IndirectRef 32002 2003 2004 )
                       :Add ( 32001 1 )
                       :Add       ( 2003 4 )
                       :Add       ( 2004 5 )
                       :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef 32001 )
                       :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef 32001 )
                       :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                       :BringToFront (2000)
                       :BringToFront (:IndirectRef 32003)
              )
        }
        {:Link 1006
         :EventSource 0 :EventType UserInput :EventData 3
         :LinkEffect ( :GetBoxSize (:IndirectRef 32002  32000 32001 )
                       :GetBoxSize (:IndirectRef 32002 2003 2004 )
                       :Subtract ( 32000 1 )
                       :Add      ( 2003 3 )
                       :Add      ( 2004 4 )
                       :SetBoxSize (:IndirectRef 32002 :IndirectRef 32000 :IndirectRef 32001 )
                       :SetBoxSize (:IndirectRef 32003 :IndirectRef 32000 :IndirectRef 32001 )
                       :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                       :BringToFront (2000)
                       :BringToFront (:IndirectRef 32003)
              )
        }
        {:Link 1007
         :EventSource 0 :EventType UserInput :EventData 4
         :LinkEffect ( :GetBoxSize (:IndirectRef 32002 32000 32001 )
                       :GetBoxSize (:IndirectRef 32002 2003 2004 )
                       :Add ( 32000 1 )
                       :Add      ( 2003 5 )
                       :Add      ( 2004 4 )
                       :SetBoxSize (:IndirectRef 32002
                          :IndirectRef 32000 :IndirectRef 32001 )
                       :SetBoxSize (:IndirectRef 32002
                          :IndirectRef 32000 :IndirectRef 32001 )
                       :SetBoxSize (2000  :IndirectRef 2003 :IndirectRef 2004)
                       :BringToFront (2000)
                       :BringToFront (:IndirectRef 32003)
              )
        }


        // ------------------------------------------------------------

        {:Link 1008
         :EventSource 0 :EventType UserInput :EventData 6
         :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3000 )
                        :SetVariable ( 32003  :GObjectRef 3001 )
        :GetPosition ( :IndirectRef 32002 2001 2002 )
        :GetBoxSize     ( :IndirectRef 32002 2003 2004 )
        :Subtract ( 2001 2 )
        :Subtract ( 2002 2 )
        :Add      ( 2003 4 )
        :Add      ( 2004 4 )
        :SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
        :SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
        :BringToFront ( 2000 )
        :BringToFront ( :IndirectRef 32002 )
        :BringToFront ( :IndirectRef 32003 )
                    )
       }
       {:Link 1108
```

```
 :EventSource 3001 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3000 )
                :SetVariable ( 32003  :GObjectRef 3001 )
:GetPosition ( :IndirectRef 32002 2001 2002 )
:GetBoxSize     ( :IndirectRef 32002 2003 2004 )
:Subtract ( 2001 2 )
:Subtract ( 2002 2 )
:Add      ( 2003 4 )
:Add      ( 2004 4 )
:SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
:SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
:BringToFront ( 2000 )
:BringToFront ( :IndirectRef 32002 )
:BringToFront ( :IndirectRef 32003 )
            )
}

// ---

{:Link 1009
 :EventSource 0 :EventType UserInput :EventData 7
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3100 )
                :SetVariable ( 32003  :GObjectRef 3101 )
:GetPosition ( :IndirectRef 32002 2001 2002 )
:GetBoxSize     ( :IndirectRef 32002 2003 2004 )
:Subtract ( 2001 2 )
:Subtract ( 2002 2 )
:Add      ( 2003 4 )
:Add      ( 2004 4 )
:SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
:SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
:BringToFront ( 2000 )
:BringToFront ( :IndirectRef 32002 )
:BringToFront ( :IndirectRef 32003 )
             )
}
{:Link 1109
 :EventSource 3101 :EventType IsSelected
 :LinkEffect (  :SetVariable ( 32002  :GObjectRef 3100 )
                :SetVariable ( 32003  :GObjectRef 3101 )
:GetPosition ( :IndirectRef 32002 2001 2002 )
:GetBoxSize     ( :IndirectRef 32002 2003 2004 )
:Subtract ( 2001 2 )
:Subtract ( 2002 2 )
:Add      ( 2003 4 )
:Add      ( 2004 4 )
:SetPosition (2000 :IndirectRef 2001 :IndirectRef 2002 )
:SetBoxSize (2000 :IndirectRef 2003 :IndirectRef 2004 )
:BringToFront ( 2000 )
:BringToFront ( :IndirectRef 32002 )
:BringToFront ( :IndirectRef 32003 )
            )
}

// -------------------------------------------------------

 {:Text 9000
```

```
          :OrigContent  'Press key1 or key2,
                         or click on one of the upper bitmaps.
                         To change the size of the
                         bitmap presentation, press cursor keys.'
          :OrigBoxSize 550 110
          :OrigPosition 10 490
          :FontAttributes Plain.18 :FontName Proportional
          :TextWrapping true
         }

        {:PushButton 9001
          :OrigBoxSize   100  60
          :OrigPosition  590 480
          :ButtonRefColour gray
          :OrigLabel "back to main"
         }

         {:Link 9002
          :EventSource 9001 :EventType IsSelected
          :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
         }

         {:Link 9003
          :EventSource 9001 :EventType CursorEnter
          :LinkEffect ( :Activate( 9005 ) )
         }
         {:Link 9004
          :EventSource 9001 :EventType CursorLeave
          :LinkEffect ( :DeActivate( 9005 ) )
         }
         {:Link 9005
          :InitiallyActive false
          :EventSource 0
          :EventType UserInput
          :EventData 15
          :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
  )
         }

   )

   :InputEventReg 1
   :SceneCS 700 560
   :MovingCursor true
}
```

bitmap2.mhg:

```
{:Scene ( "demo/bitmap2.mhg" 0 )

  :Items (
         {:Text 1000
          :OrigContent  'Testing the drawing of Bitmaps, 2.Test:'
          :OrigBoxSize 500 30
          :OrigPosition 80 10
          :FontAttributes Bold.24
          :FontName Proportional
```

```
        }

 {:Bitmap 101
           :OrigContent :ContentRef ( "demo/tu_klein.gif" )
    :OrigBoxSize 51 39     // 0 0
    :OrigPosition 10 5
    :Tiling false
    }

 {:Bitmap 102
           :OrigContent :ContentRef ( "demo/fsp_pv.gif")
           :OrigBoxSize 48 43
    :OrigPosition 595 5
    :Tiling false
    }

        {:Bitmap 103
           :OrigContent :ContentRef ( "demo/prz.gif" )
           :OrigBoxSize 48 43
    :OrigPosition 645 5
    :Tiling false
    }

 {:Link  110
   :EventSource 0
   :EventType UserInput
   :EventData 16
   :LinkEffect (
                :Quit (( "demo/startup" 0 ))
              )
  }

 {:Bitmap 3000
    :OrigContent
    :ContentRef ( "demo/colors.gif" )
    :OrigBoxSize 680 400
    :OrigPosition 10 50
    :Tiling false
    }


        {:Bitmap 3100
    :OrigContent
    :ContentRef ( "demo/capbla.gif" )
    :OrigBoxSize 180 208
    :OrigPosition 50 100
    :Tiling false
    }
        {:Bitmap 3200
    :OrigContent
    :ContentRef ( "demo/capblb.gif" )
    :OrigBoxSize  180 208
    :OrigPosition 150 100
    :Tiling false
    }
        {:Bitmap 3300
    :OrigContent
    :ContentRef ( "demo/capblc.gif" )
    :OrigBoxSize 180 208
    :OrigPosition 50 200
    :Tiling false
    }
         {:Bitmap 3400
    :OrigContent
    :ContentRef ( "demo/capbld.gif" )
    :OrigBoxSize 180 208
    :OrigPosition 150 200
    :Tiling false
    }

         // {:Bitmap 3500
 //   :OrigContent
 //   :ContentRef ( "demo/globe.gif" )
 //   :OrigBoxSize 0 0
 //   :OrigPosition 450 200
 //   :Tiling false
 //   }

 // ---------------------------

         {:Bitmap 5000
            :InitiallyActive false
    :OrigContent
    :ContentRef ( "demo/red_ball.gif" )
    :OrigBoxSize 50 50
    :OrigPosition 10 10
    :Tiling true
    }

 // ---------------------------

         {:IntegerVar 32000 :OrigValue 320 }
         {:IntegerVar 32001 :OrigValue 158 }

         {:Link 1020
            :EventSource 0 :EventType UserInput :EventData 4
            :LinkEffect ( :GetBoxSize ( 5000 32000 32001 )
                          :Add        ( 32000 5 )
                          :SetBoxSize ( 5000
                            :IndirectRef 32000 :IndirectRef 32001 ))}

         {:Link 1021
            :EventSource 0 :EventType UserInput :EventData 3
            :LinkEffect ( :GetBoxSize ( 5000 32000 32001 )
                          :Subtract   ( 32000 5 )
                          :SetBoxSize ( 5000
                           :IndirectRef 32000 :IndirectRef 32001 ))}

         {:Link 1022
            :EventSource 0 :EventType UserInput :EventData 2
            :LinkEffect ( :GetBoxSize ( 5000 32000 32001 )
                          :Add        ( 32001 5 )
                          :SetBoxSize ( 5000
                             :IndirectRef 32000 :IndirectRef 32001 ))}
```

```
{:Link 1023
   :EventSource 0 :EventType UserInput :EventData 1
   :LinkEffect ( :GetBoxSize ( 5000 32000 32001 )
                 :Subtract   ( 32001 5 )
                 :SetBoxSize ( 5000
                    :IndirectRef 32000 :IndirectRef 32001 ))}

// -------


{:Link 1030
   :EventSource 0 :EventType UserInput :EventData 14
   :LinkEffect ( :Run ( 5000 )
                 :BringToFront ( 9001 )
               )
}

{:Link 1031
   :EventSource 0 :EventType UserInput :EventData 5
   :LinkEffect ( :Stop ( 5000 )
               )
}

{:Link 2000
   :EventSource 0 :EventType UserInput :EventData 6
   :LinkEffect ( :BringToFront ( 3100 ) )
}
{:Link 2001
   :EventSource 0 :EventType UserInput :EventData 7
   :LinkEffect ( :BringToFront ( 3200 ) )
}
{:Link 2002
   :EventSource 0 :EventType UserInput :EventData 8
   :LinkEffect ( :BringToFront ( 3300 ) )
}
{:Link 2003
   :EventSource 0 :EventType UserInput :EventData 9
   :LinkEffect ( :BringToFront ( 3400 ) )
}

{:Link 2004
   :EventSource 0 :EventType UserInput :EventData 10
   :LinkEffect ( :PutBefore ( 3100 3000 ) )
}
{:Link 2005
   :EventSource 0 :EventType UserInput :EventData 11
   :LinkEffect ( :PutBefore ( 3200 3000 ) )
}
{:Link 2006
   :EventSource 0 :EventType UserInput :EventData 12
   :LinkEffect ( :PutBefore ( 3300 3000 ) )
}
{:Link 2007
   :EventSource 0 :EventType UserInput :EventData 13
   :LinkEffect ( :PutBefore ( 3400 3000 ) )
}
```

```
        // ----------------------------------------------------------

        {:Text 9000
         :OrigContent  'Press Key9 to activate a tiled bitmap,
                        Key0 to deactivate it,
                        and Cursor Keys to resize that bitmap.'
         :OrigBoxSize 550 110
         :OrigPosition 10 480
         :FontAttributes Plain.18 :FontName Proportional
         :TextWrapping true
        }

       {:PushButton 9001
         :OrigBoxSize   100  60
         :OrigPosition  590 460
         :ButtonRefColour gray
         :OrigLabel "back to main"
        }

        {:Link 9002
         :EventSource 9001 :EventType IsSelected
         :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
        }

        {:Link 9003
         :EventSource 9001 :EventType CursorEnter
         :LinkEffect ( :Activate( 9005 ) )
        }
        {:Link 9004
         :EventSource 9001 :EventType CursorLeave
         :LinkEffect ( :DeActivate( 9005 ) )
        }
        {:Link 9005
         :InitiallyActive false
         :EventSource 0
         :EventType UserInput
         :EventData 15
         :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
      )
        }


   )

   :InputEventReg 1
   :SceneCS 700 530
   :MovingCursor true
}
```

ea.mhg:

```
{:Scene ( "demo/ea.mhg" 0 )

 :Items
 (
   {:Text 1000
     :OrigContent  'Testing "Elementary Actions":'
     :OrigBoxSize 500 30
     :OrigPosition 80 10
     :FontAttributes Bold.24 :FontName Proportional
   }

     {:Bitmap 101
             :OrigContent :ContentRef ( "demo/tu_klein.gif" )
     :OrigBoxSize 51 39     // 0 0
     :OrigPosition 10 5
     :Tiling false
     }

       {:Bitmap 102
        :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
     :OrigPosition 545 5
     :Tiling false
     }

       {:Bitmap 103
        :OrigContent :ContentRef ( "demo/prz.gif" )
             :OrigBoxSize 48 43
     :OrigPosition 595 5
     :Tiling false
     }

  {:Link  110
    :EventSource 0
    :EventType UserInput
    :EventData 16
    :LinkEffect (
                  :Quit (( "demo/startup" 0 ))
                )
   }

   // ----------------------------------------------------

   // ElementaryActions for Rectangle

   {:Rectangle 4000
    :OrigBoxSize 100 100 :OrigPosition 20 130
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Red :OrigRefFillColour DarkRed
   }
   {:Rectangle 4001
    :OrigBoxSize 100 100 :OrigPosition 60 170
    :OrigLineWidth 5 :OrigLineStyle 1
    :OrigRefLineColour Blue :OrigRefFillColour DarkBlue
   }

   // apply all ElementaryActions possible for Rectangle
   {:Link 4102
    :EventSource 0 :EventType UserInput :EventData 6
    :LinkEffect (
                  // Root
```

```
                  :GetAvailabilityStatus( 4000 2103)
                  :GetRunningStatus( 4000 2103 )
               // Ingredient
                  :Clone( 4000 4200 )
                  :Preload( 4000 )
                  :SetData( 4000 "new data_1")
                  :Unload( 4000 )
               // Presentable
                  :Run( 4000 )
                  :SetData( 4000 "new data_2" )
                  :Stop( 4000 )
                  :Run( 4000 )
               // Visible
                  :BringToFront( 4000 )
                  :GetBoxSize( 4000 2101 2102 )
                  :GetPosition ( 4000 2101 2102 )
                  :PutBefore(4000 4001)
                  :PutBehind(4000 4001)
                  :SendToBack(4000)
                  :SetBoxSize(4000 150 100 )
                  :SetPaletteRef(4000 100)
                  :SetPosition(4000 20 130)
               // LineArt
          :SetFillColour (4000 :NewAbsoluteColour blue)
          :SetLineColour (4000 :NewAbsoluteColour red)
          :SetLineStyle (4000 3)
          :SetLineWidth (4000 6)
                  :PutBehind(4000 4001)
             )
 }


// ---------------------------------------------------------


// ElementaryActions for PushButton
{:PushButton 5000
 :OrigBoxSize  100 100
 :OrigPosition  200 130
 :ButtonRefColour darkYellow
 :OrigLabel "A Pushbutton"
}

{:Link 5100
 :EventSource 0 :EventType UserInput :EventData 7
 //:EventSource 5000 :EventType IsSelected
 :LinkEffect (
                //Root:
          :GetAvailabilityStatus(5000 2101)
          :GetRunningStatus(5000 2103)
                //Ingredient:
          :Clone(5000 5200)
          :Preload(5000)
          :SetData(5000 "setdata, ingredient")
          :Unload(5000)
                //Presentable:
          :Run(5000)
          :SetData(5000 "setdata, presentable")
          :Stop(5000)
                  :Run(5000)
                //Visible:
          :BringToFront(5000)
          :GetBoxSize(5000 2101 2102)
          :GetPosition(5000 2101 2102)
          :PutBefore(5000 4000)
          :PutBehind(5000 4000)
          :SendToBack(5000)
          :SetBoxSize(5000 120 90)
          :SetPaletteRef(5000 1)
          :SetPosition(5000 190 120)
                //Button:
          :Deselect(5000)
          :GetInteractionStatus(5000 2101)
          :Select(5000)
          :SetInteractionStatus(5000  true)
                  :Deselect(5000)
                //PushButton:
          :GetLabel(5000 5300)
          :Select(5000)
          :SetLabel(5000 "Label set by EA")
             )
}


 // -----------------------------------------------------

 // ElementaryActions for EntryField

 {:EntryField 2000
  :OrigContent 'A more complex one. vertically and horizontally centred.'
  :OrigBoxSize 300 100
  :OrigPosition 335 50
  :FontAttributes Bold.10 :FontName Proportional
  :TextColour red
  :BackgroundColour white
  :HJustification centre
  :VJustification centre
 }

 {:IntegerVar 2101 :OrigValue 1 }
 {:IntegerVar 2102 :OrigValue 1 }
 {:BooleanVar 2103 :OrigValue true}

// apply all ElementaryActions possible for Entryfield
 {:Link 2100
  :EventSource 0 :EventType UserInput :EventData 9
  :LinkEffect (
              // Root
                    :GetAvailabilityStatus( 2000 2101 )
                    :GetRunningStatus( 2000 2103 )
              // Ingredient
                    :Clone( 2000 4200 )
                    :Preload( 2000 )
                    :SetData( 2000 "new data_1")
```

```
                        :Unload( 2000 )
                  // Presentable
                        :Run( 2000 )
                        :SetData( 2000 "new data_2" )
                        :Stop( 2000 )
                        :Run( 2000 )
                  // Visible
                        :BringToFront( 2000 )
                        :GetBoxSize( 2000 2101 2102 )
                        :GetPosition ( 2000 2101 2102 )
                        :PutBefore(2000 4000)
                        :PutBehind(2000 4000)
                        :SendToBack(2000)
                        :SetBoxSize(2000 300 100 )
                        :SetPaletteRef(2000 100)
                        :SetPosition(2000 335 50)
                  // Text
                        :GetTextContent(2000 2101)
                        :GetTextData(2000 2101)
                        :SetData( 2000 "new data_3" )
                        //:SetFontRef(2000 "Bold:20")
                  // Interactible
                        :GetHighlightStatus(2000 2101)
                        :GetInteractionStatus(2000 2101)
                        :SetHighlightStatus(2000 true)
                        :SetInteractionStatus(2000 true)
                  // EntryField
                       :GetEntryPoint(2000 2101)
                       :GetOverwriteMode(2000 2101)
                       :SetEntryPoint(2000 5)
                       :SetOverwriteMode(2000 true)
                )
  }

// ----------------------------------------------------------

  // ElementaryActions for DynamicLineArt

  {:DynamicLineArt 3000
   :OrigBoxSize 300 100
   :OrigPosition 335 160
  }

  // apply all ElementaryActions possible for DynamicLineArt
  {:Link 3100
   :EventSource 0 :EventType UserInput :EventData 8
   :LinkEffect (
                  // Root
                        :GetAvailabilityStatus( 3000 2101 )
                        :GetRunningStatus( 3000 2103 )
                  // Ingredient
                        :Clone( 3000 4200 )
                        :Preload( 3000 )
                        :SetData( 3000 "new data_1")
                        :Unload( 3000 )
                  // Presentable
                        :Run( 3000 )
```

```
          :SetData( 3000 "new data_2" )
          :Stop( 3000 )
          :Run( 3000 )
    // Visible
          :BringToFront( 3000 )
          :GetBoxSize( 3000 2101 2102 )
          :GetPosition ( 3000 2101 2102 )
          :PutBefore(3000 4000)
          :PutBehind(3000 4000)
          :SendToBack(3000)
          :SetBoxSize(3000 300 100 )
          :SetPaletteRef(3000 100)
          :SetPosition(3000 335 160)

    // LineArt
  :SetFillColour(3000 :NewAbsoluteColour red)
  :SetLineColour(3000 :NewAbsoluteColour blue)
  :SetLineStyle(3000 3)
  :SetLineWidth(3000 2)

    //DynamicLineArt:
  :BringToFront(3000)
  :Clear(3000)


   :SetPosition(3000 335 160)
  :PutBefore(3000 2000)
  :PutBehind(3000 2000)
  :SendToBack(3000)
  :SetBoxSize(3000 300 100)

  :SetFillColour(3000 :NewAbsoluteColour red)
  :SetLineColour(3000 :NewAbsoluteColour green)
  :SetLineStyle(3000 2)
  :SetLineWidth(3000 2)
  :DrawArc(3000 5 5 100 50 30 180)

  :SetFillColour(3000 :NewAbsoluteColour  red)
  :SetLineColour(3000 :NewAbsoluteColour blue)
  :SetLineStyle(3000 2)
  :SetLineWidth(3000 10)
  :DrawLine(3000 5 5 300 100)

  :SetFillColour(3000 :NewAbsoluteColour green)
  :SetLineColour(3000 :NewAbsoluteColour yellow)
  :SetLineStyle(3000 1)
  :SetLineWidth(3000 2)
  :DrawOval(3000 5 50 100 45)

  :SetFillColour(3000 :NewAbsoluteColour red)
  :SetLineColour(3000 :NewAbsoluteColour yellow)
  :SetLineStyle(3000 2)
  :SetLineWidth(3000 6)
  :DrawPolygon(3000
      ((100 5) (100 45) ( 150 30) (199 45) (199 5)))

  :SetFillColour(3000 :NewAbsoluteColour red)
```

```
               :SetLineColour(3000 :NewAbsoluteColour black)
               :SetLineStyle(3000 2)
               :SetLineWidth(3000 3)
               :DrawPolyLine(3000
                   ((100 60) (100 95) (150 80 ) (199 95) (199 60)))

               :SetFillColour(3000 :NewAbsoluteColour gray)
               :SetLineColour(3000 :NewAbsoluteColour red)
               :SetLineStyle(3000 2)
               :SetLineWidth(3000 3)
               :DrawRectangle (3000 205 5  90 40)

               :SetFillColour(3000 :NewAbsoluteColour red)
               :SetLineColour(3000 :NewAbsoluteColour green)
               :SetLineStyle(3000 3)
               :SetLineWidth(3000 2)
               :DrawSector(3000 205 50 90 45 -30 180 )



 )
  }

  {:IntegerVar 3101 :OrigValue 30 }      // start_angle for arc
  {:IntegerVar 3102 :OrigValue 10 }      // linewidth for line
  {:IntegerVar 3103 :OrigValue 100 }     // x-size for oval
  {:IntegerVar 3104 :OrigValue 150 }     // xPos of a point of polygon
  {:IntegerVar 3105 :OrigValue 80 }      // yPos of a point for polyline
  {:IntegerVar 3106 :OrigValue 90 }      // xSize for rectangle
  {:IntegerVar 3107 :OrigValue 180 }      // arc_angle for sector


  {:Link 3200
   :EventSource 0 :EventType UserInput :EventData 1
   :LinkEffect (
                   :Add       ( 3101 5 )
                   :Add       ( 3102 1 )
                   :Add       ( 3103 10 )
                   :Add       ( 3104 5 )
                   :Add       ( 3105 5 )
                   :Add       ( 3106 10 )
                   :Add       ( 3107 10 )

               :Clear(3000)
                       :GetBoxSize( 3000 2101 2102 )

               :SetFillColour(3000 :NewAbsoluteColour red)
               :SetLineColour(3000 :NewAbsoluteColour green)
               :SetLineStyle(3000 2)
               :SetLineWidth(3000 2)
               :DrawArc(3000 5 5 100 50  :IndirectRef 3101 180)

               :SetFillColour(3000 :NewAbsoluteColour red)
               :SetLineColour(3000 :NewAbsoluteColour blue)
               :SetLineStyle(3000 3)
               :SetLineWidth(3000  :IndirectRef 3102)
               :DrawLine(3000 5 5 300 100)
```

```
            :SetFillColour(3000 :NewAbsoluteColour green)
            :SetLineColour(3000 :NewAbsoluteColour yellow)
            :SetLineStyle(3000 1)
            :SetLineWidth(3000 4)
            :DrawOval(3000 5 50  :IndirectRef 3103 45)

            :SetFillColour(3000 :NewAbsoluteColour red)
            :SetLineColour(3000 :NewAbsoluteColour yellow)
            :SetLineStyle(3000 2)
            :SetLineWidth(3000 6)
            :DrawPolygon(3000 ((100 5) (100 45)
                  (  :IndirectRef 3104 30) (199 45) (199 5)))

            :SetFillColour(3000 :NewAbsoluteColour red)
            :SetLineColour(3000 :NewAbsoluteColour black)
            :SetLineStyle(3000 2)
            :SetLineWidth(3000 3)
            :DrawPolyLine(3000 ((100 60) (100 95)
                  (150  :IndirectRef 3105 ) (199 95) (199 60)))

            :SetFillColour(3000 :NewAbsoluteColour gray)
            :SetLineColour(3000 :NewAbsoluteColour red)
            :SetLineStyle(3000 2)
            :SetLineWidth(3000 3)
            :DrawRectangle (3000 205 5   :IndirectRef 3106 40)

            :SetFillColour(3000 :NewAbsoluteColour red)
            :SetLineColour(3000 :NewAbsoluteColour green)
            :SetLineStyle(3000 3)
            :SetLineWidth(3000 2)
            :DrawSector(3000 205 50 90 45 -30  :IndirectRef 3107 )
         )
}
{:Link 3201
 :EventSource 0 :EventType UserInput :EventData 2
 :LinkEffect (
                    :Subtract       ( 3101 5 )
                    :Subtract       ( 3102 1 )
                    :Subtract       ( 3103 10 )
                    :Subtract       ( 3104 5 )
                    :Subtract       ( 3105 5 )
                    :Subtract       ( 3106 10 )
                    :Subtract       ( 3107 10 )

                    :Clear(3000)

            :SetFillColour(3000 :NewAbsoluteColour red)
            :SetLineColour(3000 :NewAbsoluteColour green)
            :SetLineStyle(3000 2)
            :SetLineWidth(3000 2)
            :DrawArc(3000 5 5 100 50  :IndirectRef 3101 180)

            :SetFillColour(3000 :NewAbsoluteColour red)
            :SetLineColour(3000 :NewAbsoluteColour blue)
            :SetLineStyle(3000 2)
            :SetLineWidth(3000  :IndirectRef 3102)
```

```
              :DrawLine(3000 5 5 300 100)

              :SetFillColour(3000 :NewAbsoluteColour green)
              :SetLineColour(3000 :NewAbsoluteColour yellow)
              :SetLineStyle(3000 1)
              :SetLineWidth(3000 2)
              :DrawOval(3000 5 50  :IndirectRef 3103 45)

              :SetFillColour(3000 :NewAbsoluteColour red)
              :SetLineColour(3000 :NewAbsoluteColour yellow)
              :SetLineStyle(3000 2)
              :SetLineWidth(3000 6)
              :DrawPolygon(3000 ((100 5) (100 45)
                    (  :IndirectRef 3104 30) (199 45) (199 5)))

              :SetFillColour(3000 :NewAbsoluteColour red)
              :SetLineColour(3000 :NewAbsoluteColour black)
              :SetLineStyle(3000 2)
              :SetLineWidth(3000 3)
              :DrawPolyLine(3000 ((100 60) (100 95)
                    (150  :IndirectRef 3105 ) (199 95) (199 60)))

              :SetFillColour(3000 :NewAbsoluteColour gray)
              :SetLineColour(3000 :NewAbsoluteColour red)
              :SetLineStyle(3000 2)
              :SetLineWidth(3000 3)
              :DrawRectangle (3000 205 5   :IndirectRef 3106 40)

              :SetFillColour(3000 :NewAbsoluteColour red)
              :SetLineColour(3000 :NewAbsoluteColour green)
              :SetLineStyle(3000 3)
              :SetLineWidth(3000 2)
              :DrawSector(3000 205 50 90 45 -30  :IndirectRef 3107 )
           )
  }


// ----------------------------------------------------------


 {:PushButton 9001
  :OrigBoxSize   100  60
  :OrigPosition  540 280
  :ButtonRefColour gray
  :OrigLabel "back to main"
 }

 {:Link 9002
  :EventSource 9001 :EventType IsSelected
  :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
 }

 {:Link 9003
  :EventSource 9001 :EventType CursorEnter
  :LinkEffect ( :Activate( 9005 ) )
```

```
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
  )
   }

 )

 :InputEventReg 1
 :SceneCS 650 350
 :MovingCursor true
}
```

allcl.mhg:

```
{:Scene ( "demo/allcl.mhg" 0 )

 :Items
 (

   {:Text 1000
     :OrigContent  'Testing all concrete Classes and el.Actions:'
     :OrigBoxSize 500 30
     :OrigPosition 80 10
     :FontAttributes Bold.24 :FontName Proportional
   }

     {:Bitmap 101
             :OrigContent :ContentRef ( "demo/tu_klein.gif" )
     :OrigBoxSize 51 39     // 0 0
     :OrigPosition 10 5
     :Tiling false
     }

       {:Bitmap 102
        :OrigContent :ContentRef ( "demo/fsp_pv.gif")
             :OrigBoxSize 48 43
     :OrigPosition 545 5
     :Tiling false
     }

       {:Bitmap 103
        :OrigContent :ContentRef ( "demo/prz.gif" )
             :OrigBoxSize 48 43
     :OrigPosition 595 5
     :Tiling false
     }

  {:Link  110
```

```
   :EventSource 0
   :EventType UserInput
   :EventData 16
   :LinkEffect (
                :Quit (( "demo/startup" 0 ))
              )
 }
// ---------------------------------------------------------

// Root 1008
// Group 1009
// Application 1010
// Scene 1011
// Ingredient 1012
// Link 1013

// ---- Programs (1014) ----

{:ResidentPrg 1015
 :Name "Hello World, I'm a ResidentProgram."
}

{:RemotePrg 1016
 :Name "Hello World, I'm a RemoteProgram."
}

{:InterchgPrg 1017
 :Name "Hello World, I'm a InterchangedProgram."
}

{:Palette 1018
}

{:Font 1019
}

{:CursorShape 1020
}

// ---- Variables 1021 ----

{:BooleanVar 1022
 :OrigValue true
}

{:IntegerVar 1023
 :OrigValue 1
}

{:OStringVar 1024
 :OrigValue "hello world"
}

{:ObjectRefVar 1025
 :OrigValue :ObjectRef 1024
}
```

```
{:ContentRefVar 1026
 :OrigValue :ContentRef "demo/to_load.txt"
}

// ---- Presentables 1027 ----
// TokenManager 1028

{:TokenGroup 1029
 :TokenGroupItems ( ( 1024 ) )
}

{:ListGroup 1030
 :TokenGroupItems ( ( 1024 ) )
 :Positions ( ( 10 10 ) )
}

{:Stream 1037
 :Multiplex ( {:Audio 1038 :ComponentTag 100 }  )
}

{:Audio 1038
 :ComponentTag 100
}

// ---- Visibles ----

{:Text 2000
  :OrigContent  'Bitmap, LineArt, Rectangle, Dynamic- LineArt:'
  :OrigBoxSize 95 95
  :OrigPosition 0 50
  :FontAttributes Bold.14
  :TextColour black
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }

{:Bitmap 1032
 :OrigContent :ContentRef ( "demo/pirogue.jpg" )
 :OrigBoxSize 120 120
 :OrigPosition 100 50
}

{:LineArt 1033
 :OrigBoxSize 120 120
 :OrigPosition 225 50
}

{:Rectangle 1034
 :OrigBoxSize 120 120
 :OrigPosition 350 50
 :OrigLineWidth 5
 :OrigLineStyle 1
 :OrigRefLineColour Red
 :OrigRefFillColour Yellow
}
```

```
{:DynamicLineArt 1035
 :OrigBoxSize 120 120
 :OrigPosition 475 50
}

// --------------------------

{:Text 2001
  :OrigContent  'Text, Video, RTGraphics:'
  :OrigBoxSize 95 95
  :OrigPosition 0 175
  :FontAttributes Bold.14
  :TextColour black
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }

{:Text 1036
 :OrigContent  "I'm a simple Text"
 :OrigBoxSize 120 120
 :OrigPosition 100 175
 :TextColour red
 :BackgroundColour black
 :HJustification centre
 :VJustification centre
 :TextWrapping true
}

{:Video 1039
 :InitiallyActive true
 :OrigContent
 :ContentRef ( "demo/saao.m1v" )
 :OrigBoxSize 120 120
 :OrigPosition 225 175
 :ComponentTag 100
 :Termination loop
}

{:RTGraphics 1040
 :OrigBoxSize 120 120
 :OrigPosition 350 175
 :ComponentTag 100
}

// ---- Interactibles 1041 ----

{:Text 2002
  :OrigContent  'Slider, EntryField, HyperText:'
  :OrigBoxSize 95 95
  :OrigPosition 0 300
  :FontAttributes Bold.14
  :TextColour black
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }
```

```
{:Slider 1042
 :OrigBoxSize 120 120
 :OrigPosition 100 300
 :Orientation up
 :MaxValue 100
 :SliderStyle proportional
}

{:EntryField 1043
 :OrigBoxSize 120 120
 :OrigPosition 225 300
}

{:HyperText 1044
 :OrigContent 'HyperText: an anchor to
     <a href="http://www.mheg.org">Tom Caseys</a> site'
 :OrigBoxSize 120 120
 :OrigPosition 350 300

}

// ---- Buttons 1045 ----

{:Text 2003
  :OrigContent  'HotSpot, PushButton, SwitchButton:'
  :OrigBoxSize 95 95
  :OrigPosition 0 425
  :FontAttributes Bold.14
  :TextColour black
  :HJustification centre
  :VJustification centre
  :TextWrapping true
 }

{:Hotspot 1046
 :OrigBoxSize 120 120
 :OrigPosition 100 425
}

{:PushButton 1047
 :OrigBoxSize 120 120
 :OrigPosition 225 425
}

{:SwitchButton 1048
 :OrigBoxSize 120 120
 :OrigPosition 350 425
 :ButtonStyle radiobutton
}

// TODO:  ___TODO___:
//{:Action
// (:Activate 1000)
//}
```

```
// ----------------------------------------------------------


{:IntegerVar   3100 :OrigValue 1 }
{:IntegerVar   3101 :OrigValue 1 }
{:BooleanVar   3102 :OrigValue true }
{:ObjectRefVAr 3103 :OrigValue :ObjectRef 3000}

// apply all ElementaryActions possible for all classes
{:Link 3000
 :EventSource 0 :EventType UserInput :EventData 6
 :LinkEffect (
               // Root (using the bitmap: 1032)
                :GetAvailabilityStatus( 1032 3102 )
                :GetRunningStatus( 1032 3102 )

        // Scene
 //:SendEvent(0 1)               // todo.
         :GetCursorPosition(0 3100 3101)
         :SetCursorPosition(0 100 100)
         :SetCursorShape(1000 1000)
         :SetTimer(0 1 2 :AbsoluteTime true)
         //:TransitionTo()              // not so useful here.

              //Application:
         //:CloseConnection()
         //:GetEngineSupport()
         //:Launch()
         //:LockScreen()
         //:OpenConnection()
         //:Quit()
         //:ReadPersistent()
         //:Spawn()
         //:StorePersistent()
         //:UnlockScreen()

               // Ingredient (using the bitmap: 1032)
                :Clone( 1032 1050 )
                :Preload( 1032 )
                :SetData( 1032 "new data_1")
                :Unload( 1032 )

               //Link:
                //:Activate()
         //:Deactivate()

               //Program (using resident program: 1015)
         //:Call(1015 )
         //:Fork(1015)
         //:SetData(1015)
         //:Stop(1015)

               //Presentable (using the bitmap: 1032)
         //:Run(1032)
         //:SetData(1032)
         //:Stop(1032)
```

```
       //Stream:
//:Clone()
:SetCounterEndPosition(1037 5)
:SetCounterPosition(1037 1)
:SetCounterTrigger(1037 2)
//:SetData(1037)
:SetSpeed(1037 25)

       //Visible (using the bitmap: 1032)
:BringToFront(1032)
:GetBoxSize(1032 3100 3101)
:GetPosition(1032 3100 3101)
:PutBefore(1032 1035)
:PutBehind(1032 1035)
:SendToBack(1032)
:SetBoxSize(1032 60 60)
//:SetPaletteRef(1032)
:SetPosition(1032 100 50)

       //RTGraphics:
//:Clone(1040)
//:SetData(1040)

       //LineArt:
//:SetFillColour(1033)
//:SetLineColour(1033)
:SetLineStyle(1033 3)
:SetLineWidth(1033 2)

       //DynamicLineArt:
:BringToFront(1035)
:Clear(1035)
:DrawArc(1035 10 10 30 30 0 180)
:DrawLine(1035 10 10 50 50)
:DrawOval(1035 20 20 50 30)
//:DrawPolygon(1035)
//:DrawPolyline(1035)
:DrawRectangle(1035 10 10 30 50)
:DrawSector(1035 30 30 40 40 90 180)
//:GetFillColour(1035)
//:GetLineColour(1035)
:GetLineStyle(1035 3100)
:GetLineWidth(1035 3100)
:PutBefore(1035 1032)
:PutBehind(1035 1032)
:SendToBack(1035)
:SetBoxSize(1035 90 90)
//:SetFillColour(1035)
//:SetLineColour(1035)
:SetLineStyle(1035 5)
:SetLineWidth(1035 6)
:SetPosition(1035 400 50)

       //Bitmap:
:ScaleBitmap(1032 50 50)
:SetTransparency(1032 25)
```

```
       //Text:
//:GetTextContent(1036)
//:GetTextData(1036)
//:SetData(1036)
//:SetFontRef(1036)

       //Interactible (using EntryField):
:GetHighlightStatus(1043 3102)
:GetInteractionStatus(1043 3102)
:SetHighlightStatus(1043 true)
:SetInteractionStatus(1043 true)

       //EntryField:
:GetEntryPoint(1043 3101)
:GetOverwriteMode(1043 3102)
:SetEntryPoint(1043 4)
:SetOverwriteMode(1043 false)

       //HyperText:
//:GetLastAnchorField(1044)

       //Video:
        //:Clone(1039)
:ScaleVideo(1039 30 30)
//:SetData(1039)

       //Slider:
:GetPortion(1042 3101)
:GetSliderValue(1042 3101)
:SetPortion(1042 50)
:SetSliderValue(1042 25)
:Step(1042 1)

       //Button (using PushButton)
:Deselect(1047)
:GetInteractionStatus(1047 3102)
:Select(1047)
:SetInteractionStatus(1047 true)

       //HotSpot:
:Select(1046)

       //PushButton:
//:GetLabel(1047)
:Select(1047)
:SetLabel(1047 "new Label")

       //SwitchButton:
:Deselect(1048)
:GetSelectionStatus(1048 3102)
:Select(1048)
:SetLabel(1048 "new Label")
//:Toggle(1048)

       //Audio:
        //:Clone(1038)
:GetVolume(1038 3100)
```

```
        //:SetData(1038)
        //:SetVolume(1038 100)

               //TokenGroup:
        //:CallActionSlot(1029)

               //ListGroup:
        :AddItem(1030 1 1032)
        :DelItem(1030 1032)
        :DeselectItem(1030 2)
        :GetCellItem(1030 2 1032)
        :GetFirstItem(1030 1032)
        //:GetItemStatus(1030)
        //:GetListItem(1030)
        //:GetListSize(1030)
        //:ScrollItems(1030)
        //:SelectItem(1030)
        //:SetFirstItem(1030)
        //:ToggleItem(1030)

               //TokenManager (using TokenGroup):
        //:GetTokenPosition(1029)
        //:Move(1029)
        //:MoveTo(1029)

               //Variable (using IntegerVariable):
        //:SetVariable(1023)
        //:TestVariable(1023)

               //IntegerVariable:
        //:Add(1023)
        //:Divide(1023)
        //:Modulo(1023)
        //:Multiply(1023)
                //:SetVariable(1023)
        //:Subtract(1023)

               //BooleanVariable:
                //:SetVariable(1022 )

               //OctetStringVariable:
                //:SetVariable(1024)
                //:Append(1024)

               //ObjectRefVariable:
                //:SetVariable(1025)

               //ContentRefVariable:
                //:SetVariable(1026)

 )
}

// ---------------------------------------------------------


 {:PushButton 9001
```

```
    :OrigBoxSize   100  60
    :OrigPosition  540 480
    :ButtonRefColour gray
    :OrigLabel "back to main"
   }

   {:Link 9002
    :EventSource 9001 :EventType IsSelected
    :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
   }

   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
  )
   }


 )

 :InputEventReg 1
 :SceneCS 650 550
 :MovingCursor true
}
```

token.mhg:

```
{:Scene ( "demo/tokentest.mhg" 0 )

 :Items
 (
   {:Text 500 :InitiallyActive false
    :OrigContent  'TokenGroup is running'
    :OrigBoxSize 500 30
    :OrigPosition 70 2
    :FontAttributes Bold.18 :FontName Proportional
   }
   {:Text 501 :InitiallyActive true
    :OrigContent  'TokenGroup is not running'
    :OrigBoxSize 500 30
    :OrigPosition 70 2
    :FontAttributes Bold.18 :FontName Proportional
   }

   {:PushButton 50 :InitiallyActive true
    :OrigBoxSize 150 30 :OrigPosition 10 175
    :ButtonrefColour Gray :OrigLabel "Run TokenGroup" }
   {:Link 60 :EventSource 50 :EventType IsSelected
    :LinkEffect ( :MoveTo( 1000 0 )
                  :CallActionSlot( 1000 1 )
                  :Run( 1000 ) )}

   {:PushButton 51 :InitiallyActive true
    :OrigBoxSize 150 30 :OrigPosition 160 175
    :ButtonRefColour Gray :OrigLabel "Stop TokenGroup" }
   {:Link 62 :EventSource 51 :EventType IsSelected
    :LinkEffect ( :MoveTo( 1000 0 )
                  :CallActionSlot( 1000 2 )
                  :Stop( 1000 ))}

   {:Link 100 :EventSource 101 :EventType IsSelected
    :LinkEffect ( :GetTokenPosition( 1000 104 )
                  :SetVariable( 103 :GInteger :IndirectRef 104 )
                  :SetVariable( 102 :GOctetString "Query TokenPosition<" )
                  :Append     ( 102 :IndirectRef 103 )
                  :Append     ( 102 ">" )
                  :SetLabel   ( 101 :IndirectRef 102 ) )
   }
   {:PushButton 101 :InitiallyActive true
    :OrigBoxSize 300 30 :OrigPosition 10 205
    :ButtonRefColour Gray :OrigLabel "Query TokenPosition<?>" }
   {:OStringVar 102 :OrigValue "" }
   {:OStringVar 103 :OrigValue "" }
   {:IntegerVar 104 :OrigValue -1 }

   {:PushButton 106 :InitiallyActive true
    :OrigBoxSize 100 30 :OrigPosition 10 235
    :ButtonRefColour Gray :OrigLabel "NextItem" }
   {:PushButton 107 :InitiallyActive true
    :OrigBoxSize 100 30 :OrigPosition 110 235
    :ButtonRefColour Gray :OrigLabel "PrevItem" }

   {:Link 111 :EventSource 106 :EventType IsSelected
    :LinkEffect( :Move( 1000 1 ) ) }
   {:Link 112 :EventSource 107 :EventType IsSelected
    :LinkEffect( :Move( 1000 2 ) ) }

   {:PushButton 120 :InitiallyActive true
    :OrigBoxSize 60 30 :OrigPosition 10 265
    :ButtonRefColour Gray :OrigLabel "MoveTo1" }
   {:PushButton 121 :InitiallyActive true
    :OrigBoxSize 60 30 :OrigPosition 70 265
    :ButtonRefColour Gray :OrigLabel "MoveTo2" }
   {:PushButton 122 :InitiallyActive true
    :OrigBoxSize 60 30 :OrigPosition 130 265
    :ButtonRefColour Gray :OrigLabel "MoveTo3" }
   {:PushButton 123 :InitiallyActive true
    :OrigBoxSize 60 30 :OrigPosition 190 265
    :ButtonRefColour Gray :OrigLabel "MoveTo4" }
   {:PushButton 124 :InitiallyActive true
    :OrigBoxSize 60 30 :OrigPosition 250 265
    :ButtonRefColour Gray :OrigLabel "MoveTo5" }
```

```
{:Link 125 :EventSource 120 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 1 ) ) }
{:Link 126 :EventSource 121 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 2 ) ) }
{:Link 127 :EventSource 122 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 3 ) ) }
{:Link 128 :EventSource 123 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 4 ) ) }
{:Link 129 :EventSource 124 :EventType IsSelected
 :LinkEffect( :MoveTo( 1000 5 ) ) }

{:Rectangle 4711 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 40 25 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkRed :OrigRefFillColour Red
}
{:Rectangle 4712 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 78 45 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkGray :OrigRefFillColour Gray
}
{:Rectangle 4713 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 116 65 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkBlue :OrigRefFillColour Blue
}
{:Rectangle 4714 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 154 85 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkGreen :OrigRefFillColour Green
}
{:Rectangle 4715 :InitiallyActive false
 :OrigBoxSize 64 64 :OrigPosition 192 105 :OrigLineWidth 4 :OrigLineStyle 1
 :OrigRefLineColour DarkMagenta :OrigRefFillColour Magenta
}

{:Link 1001 :EventSource 1000 :EventType TokenMovedFrom :EventData 1
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1002 :EventSource 1000 :EventType TokenMovedTo :EventData 1
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}
{:Link 1003 :EventSource 1000 :EventType TokenMovedFrom :EventData 2
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1004 :EventSource 1000 :EventType TokenMovedTo :EventData 2
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}
{:Link 1005 :EventSource 1000 :EventType TokenMovedFrom :EventData 3
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1006 :EventSource 1000 :EventType TokenMovedTo :EventData 3
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}
{:Link 1007 :EventSource 1000 :EventType TokenMovedFrom :EventData 4
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1008 :EventSource 1000 :EventType TokenMovedTo :EventData 4
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}
{:Link 1009 :EventSource 1000 :EventType TokenMovedFrom :EventData 5
 :LinkEffect ( :CallActionSlot( 1000 1 ) )}
{:Link 1010 :EventSource 1000 :EventType TokenMovedTo :EventData 5
 :LinkEffect ( :CallActionSlot( 1000 2 ) )}


{:TokenGroup 1000
 :InitiallyActive false
 :Shared false
```

```
:MovementTable
  (
    (2 3 4 5 1 )
    (5 1 2 3 4 )
  )
:TokenGroupItems
  (
    (4711
      :ActionSlots
      (
        (:SetFillColour(4711 :NewAbsoluteColour red))
        (:SetFillColour(4711 :NewAbsoluteColour transparent))
      )
    )
    ( 4712
      :ActionSlots
      (
        (:SetFillColour(4712 :NewAbsoluteColour gray))
        (:SetFillColour(4712 :NewAbsoluteColour transparent))
      )
    )
    ( 4713
      :ActionSlots
      (
        (:SetFillColour(4713 :NewAbsoluteColour blue))
        (:SetFillColour(4713 :NewAbsoluteColour transparent))
      )
    )
    ( 4714
      :ActionSlots
      (
        (:SetFillColour(4714 :NewAbsoluteColour green))
        (:SetFillColour(4714 :NewAbsoluteColour transparent))
      )
    )
    ( 4715
      :ActionSlots
      (
        (:SetFillColour(4715 :NewAbsoluteColour magenta))
        (:SetFillColour(4715 :NewAbsoluteColour transparent))
      )
    )
  )
:NoTokenActionSlots
  (
    (:Run( 500 ) :Stop( 501 ) )
    (:Run( 501 ) :Stop( 500 ) )
  )
}



{:PushButton 9001
 :OrigBoxSize   100  30
 :OrigPosition  210 300
 :ButtonRefColour gray
 :OrigLabel "back to main"
```

```
    }

    {:Link 9002
     :EventSource 9001 :EventType IsSelected
     :LinkEffect (  :TransitionTo( ( "demo/main.mhg" 0) ) )
    }


   {:Link 9003
    :EventSource 9001 :EventType CursorEnter
    :LinkEffect ( :Activate( 9005 ) )
   }
   {:Link 9004
    :EventSource 9001 :EventType CursorLeave
    :LinkEffect ( :DeActivate( 9005 ) )
   }
   {:Link 9005
      :InitiallyActive false
      :EventSource 0
      :EventType UserInput
      :EventData 15
      :LinkEffect ( :TransitionTo( ( "demo/main.mhg" 0) )
  )
   }



 )

 :InputEventReg 1
 :SceneCS 320 350
 :MovingCursor true
}
```

More Examples

Further examples are available in the MHEG applications folder:

bitmap — further examples of bitmaps in MHEG

interacting — further examples of interaction in MHEG

intvar — integer variables

jmf — video and audio

quiz2 — MHEG quiz in MHEG

text — further text in MHEG.

10.6.14 Relationships to Major Standards

Important relationships exist between MHEG-5 and other standards and specifications.

Davic (Digital Audio Visual Council) — aims to maximize interoperability across applications and services for the broadcast and interactive domains. It selects existing standards and specifies their interfaces in the realm of a complete reference model. This comprises the content provider system, the service provider system, and the service consumer system. This forum has prepared a series of specifications: Davic 1.0 selected MHEG-5 for encoding base level applications and Davic 1.1 relies on MHEG-6 to extend these applications in terms of the Java virtual machine that uses services from the MHEG-5 RTE.

DVB (Digital Video Broadcasting)s — provides a complete solution for digital television and data broadcasting across a range of delivery media where audio and video signals are encoded in MPEG-2. The specification includes an open service information system, known as DVB-SI, which provides the elements necessary for developing a basic electronic program guide (EPG) to support navigation amongst the new digital television services.

MPEG — the well-known(Chapter 7) family of standards used for coding audiovisual information (such as movies, video, and music) in a digital compressed format. MPEG-1 and MPEG-2 streams are likely to be used by MHEG-5 applications, which can easily control their playback through the facilities provided by the Stream class.

DSMCC (Digital Storage Media Command and Control) — specifies a set of protocols for controlling and managing MPEG streams in a client-server environment. The user-to-user protocol (both the client and server are users) consists of VCR commands for playback of streams stored on the server, as well as commands for downloading other data (bitmaps, text, and so on). The playback of MPEG streams from within MHEG-5 applications has a counterpart at the DSMCC layer.

10.6.15 MHEG Implementation

Several components may be requires in implementing and MHEG systems:

Runtime Engine (RTE) — MHEG-5 runtime engines generally run across a client-server architecture (See The Armida system (Figure 10.18) referenced below for an example application).

A preceding *Start-up* Module may be used to perform general initialization *etc.*:

- The client can be launched either as an autonomous Windows application or
- as a plug-in by an HTML browser, allowing seamless navigation between the World Wide Web and the webs of MHEG-5 applications. (See Armida system for more details).

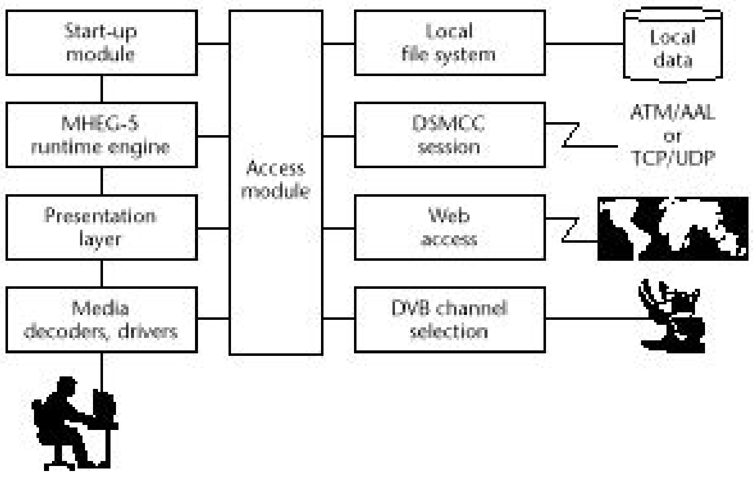


Figure 10.18: Armedia Client Architecture
Armedia is a client-server based interactive
multimedia application retrieval system.

The MHEG-5 RTE is the kernel of the client's architecture. It performs the pure interpretation of MHEG-5 objects and, as a platform-independent module, issues I/O and data access requests to other components that are optimized for the specific runtime platform.

The RTE performs two main tasks. First, it prepares the presentation and handles accessing, decoding, and managing MHEG-5 objects in their internal format. The second task is the actual presentation, which is based on an event loop where events trigger actions. These actions then become requests to the Presentation layer along with other actions that internally affect the engine.

Presentation layer — The presentation layer (PL) manages windowing resources, deals with low-level events, and performs decoding and rendering of contents from different media to the user. This functionality is available to the RTE via an object-oriented API that encapsulates all I/O platform specific aspects. The basic MHEG-5 Presentable classes have counterparts at this API level, which makes provisions for initialization/termination, data access and decoding, setting specific attributes (such as text font, color, and so on), and performing spatial and temporal controls. In addition, an informative flow exists from the PL back to the RTE, which notifies user interaction and stream events.

Access module — This module provides a consistent API for accessing information from different sources. It's used by the RTE to get objects and the PL to access content data (either downloaded or streamed). Note that the selection of a particular delivery strategy is out of MHEG-5's scope, and hence remains an implementation issue. Typical applications should support:

- bulk download for bitmaps, text, and MHEG-5 objects; and
- progressive download for audio and audiovisual streams.

The implementation of these mechanisms occurs via the DSMCC interface. The user has full interactive control of the data presentation, including playback of higher quality MPEG-2 streams delivered through an ATM network. Object and content access requests can also be issued to the Web via HTTP. However, this wide-spread protocol still suffers from limitations when high-quality video clips are progressively down-loaded, since the underlying networks may not yet provide adequate quality-of-service (QoS). When accessing the broadcast service, the Access module requires the DVB channel selection component to select the program referred to by a Stream object. To achieve this, you set the ContentHook to a value that means *Switched DVB* and the OriginalContent to the identifier of the channel to select. The MPEG-2 stream transported by the selected channel is displayed in the active Scene's presentation space.

MHEG Authoring Tool: MediaTouch —

The availability of an adequate authoring tool is mandatory to create the MHEG applications. The MediaTouch (Figure 10.19) application is one example developed for the Armida System (*http://drogo.cselt.stet.it/ufv/ArmidaIS/home_en.htm*). It is a visual-based Hierarchical Iconic authoring tool, similar to Authorware in many approaches.

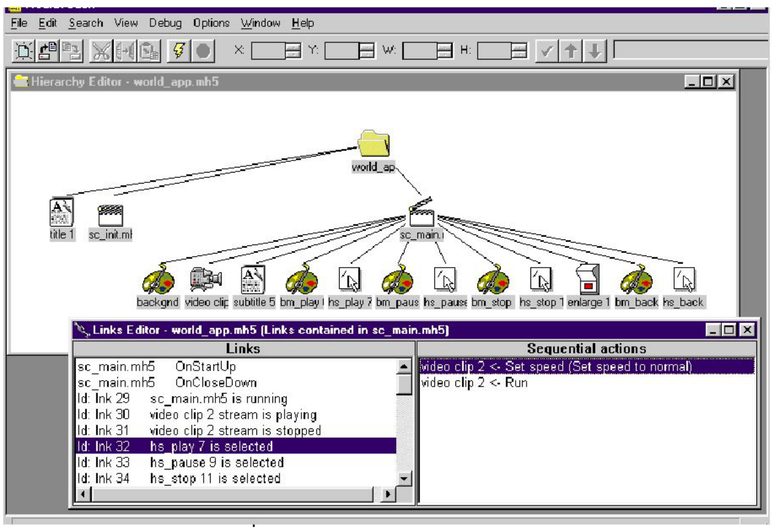
MediaTouch is based on the native approach, which lets the author operate at the level of MHEG-5 objects.

MediaTouch provides authors with the following functions:

Hierarchy Editor — Supports the interactive creation of the structure of Applications and Scenes objects. The author operates by adding, moving, and deleting Ingredients on the tree that represents the hierarchy's current status. Several Scenes from different Applications can be edited at one time and you can copy and move Ingredients between Scenes and Applications.

Properties Editor — The author sets the properties of Presentable Ingredients via an input form that is specific to each object class. The author does not need to type a value directly, since the system adopts a menu-based input interface that aims to minimize errors and inconsistencies.

Layout Editor — A Scene's layout can be set by interactively adjusting the position and size of the bounding box for every Visible Ingredient.



Links	
sc_main.mh5	OnStartUp
sc_main.mh5	OnCloseDown
Id: lnk 29	sc_main.mh5 is running
Id: lnk 30	video clip 2 stream is playing
Id: lnk 31	video clip 2 stream is stopped
Id: lnk 32	hs_play 7 is selected
Id: lnk 33	hs_pause 9 is selected
Id: lnk 34	hs_stop 11 is selected

Sequential actions
video clip 2 <- Set speed (Set speed to normal)
video clip 2 <- Run

Figure 10.19: MediaTouch MHEG Authoring Tool
(Hierarchy and Links Editor windows)

An Ingredient's content data, when available, is also displayed within its bounding box to make the layout match the actual presentation.

Links Editor — This function lets authors add, modify, and delete links and actions to Application and Scene objects. The author sets the link conditions, actions, and referenced objects via a menu. Links and actions can also be copied between different scenes and applications.

Figure 10.19) shows a screen shot from MediaTouch where an Application composed of two Scenes and one shared Text is open within the Hierarchy Editor window. One of these Scenes shows its Presentable Ingredients, and the author is editing its Links. The author can then launch the RTE from within MediaTouch to check the outcome of his work.

MHEG Authoring Tool: MHEGDitor — MHEGDitor is an MHEG-5 authoring tool based on Macromedia Director. MHEGDitor is composed of an Authoring extra to edit applications and of a Converter Extra to convert resulting movies into MHEG-5 applications. The two MHEGDitor extras work separately. With MHEGDitor Authoring Extra, create and edit your application within Macromedia Director 6. It opens a window to set preferences and links a specific external script castLib to your movie for you to create specific MHEG behaviours rapidly. You can test your application on the spot within Director, as if it were played by MHEGPlayer, the MHEG interpreter companion of MHEGDitor. With MHEGDitor Converter Extra, convert Macromedia Director 6 movies (edited with MHEGDitor Authoring Extra) into a folder containing all necessary items for an MHEG-5 application. Associated with Macromedia Director 6, MHEGDitor is the easiest way to author multimedia applications for Interactive TV for both Macintosh and Windows. This tool makes it possible for all system operators or content providers to deliver developed interactive programs to the widest range of potential users.

The Current Release is 1.4 which performs the following:

- Director 6.5 to MHEG-5
- Available on both Macintosh and Windows 95/NT platforms
- Advanced dynamic lists support, new Converter Extra settings (text data exported as external data...), converted applications optimisation, and more.

MHEG writing tool:MHEG Write — An editor to create and manipulate MHEG-5 applications by hand This editor is based on the free software "VIM", which is available via Internet from various sites for virtually all operating systems, including DOS, Windows, UNIX, etc. The MHEG-Write extension supports only the MHEG-5 textual notation. MHEG-Write provides macros for object templates, syntax highlighting and syntax error detection. Using MHEGWrite enables you to access every MHEG-5 features to edit your applications, since this editor is able to read native MHEG-5 code. It also gives good support for detailed examination, profile adaptation, and error correction of MHEG-5 applications produced with arbitrary other tools. By purchasing MHEGWrite, the MHEG Centre will also provide you with a set of MHEG-5 application samples, illustrating several MHEG-5 features, such as ListGroup together with cloning, using variables, sharing objects, etc.

Playing MHEG files — There are a few ways to *play* MHEG files:

- **MHEGPlayer** is an MHEG-5 interpreter which is able to execute MHEG-5 applications developed with MHEGDitor or any other authoring tool. It implements the MHEG-5 International Standard according to the additional requirements of the DAVIC application domain. The available content types are aligned with the underlying operating system. MHEGPlayer consists of a core interpreter with presentation system for Windows 95/NT. It is pre-configured to utilise the local file system to access MHEG objects and content data.

  Supported Formats include:

  - Bitmap: PNG, BMP, TIFF, JPEG, PSD
  - Video: MPEG-1, AVI, Quicktime (via MCI)
  - Audio: Wave, AIFF_C (via MCI)
  - MHEG-5: ASN.1 (DER) or Textual Notation

- MHEG Java Engine — Java Source code exists to compile a platform-independent MHEG player (*http://enterprise.prz.tu-berlin.de/imw/*)
- MHEG plug-ins for Netscape Browsers and Internet Explorer have been developed,

10.6.16 MHEG Future

Several companies and research institutes are currently developing MHEG tools and applications and conducting interoperability experiments for international projects and consortia. The MHEG Support Center is a European project that hopes to implement and operate an MHEG support and conformance testing environment for developers of multimedia systems and applications. Its partners include CCETT, Cril Ingenierie (France), De teBerkom, and GMD Fokus (Germany). IPA, the Information-technology Promotion Agency of Japan, is developing an MHEG-5 system with an authoring tool and viewer. The project members consist of ASCII, KDD, NEC, NTT Soft-ware, the Oki Electric Industry, Pioneer, Sony, and Toppan Printing. The purpose of this project is to do a profiling study and conduct interoperability testing of MHEG-5 systems.

Also, the following European achievements are worth mentioning:

- a Java implementation of an MHEG-5 engine, available from Phillips; (See below)
- MhegDitor by CCETT, an authoring tool based on Macromedia Director; and
- the results of the Jupiter project that addresses usability, performance, and interoperability of Davic-compliant services.

At CSELT, a partner of Jupiter, we're making more complete versions of MediaTouch and Armi**-da. We've demonstrated these systems at a number of events on behalf of Telecom Italia and showed such applications as movies-on-demand, news-on-demand, tourist information, and interactive TV.

A list containing the above applications and other initiatives is maintained by the MHEG Users Group (MUG), an unofficial forum begun by people from the standardization body to disseminate information and enable discussion on MHEG (see http://www.fokus.gmd.de/ovma/mug).

It's evident that the initial interest in MHEG-1 and MHEG-3 has recently decreased due to the incoming ITV/VoD profile, which is based on MHEG-5 and MHEG-6. The specification of this *complete solution*, urged and endorsed by Davic, was finalized by April 1998 when MHEG-6 is scheduled to become an International Standard. Further efforts will be devoted to MHEG-7, which is expected to provide the final specification of conformance and interoperability aspects for MHEG-5 by January 1999.

10.6.17 Further Reading/Information

The above notes on MHEG are base largely on an article *MHEG-5 — Aim, Concepts, and Implementation Issues*, by M. Echiffre, C. Marchisio, P. Marchisio, P. Panicciari and S. Del Rossi, IEEE Multimedia, Winter 1998, pp 84–91. Also available at URL: *http://www.fokus.gmd.de/ovma/mug/archives/doc/u1084.pdf*

Other good reference on MHEG and related multimedia standards are:

- T. Meyer-Boudnik and W. Effelsberg, *MHEG Explained*, IEEE Multimedia, Spring 1995, Vol. 2, No. 1, pp. 26-38.
- ISO 13522-5, Information TechnologyCoding of Multimedia and Hypermedia Information, Part 5: Support for Base-Level Interactive Applications, Int'l Org. for Standardization/Int'l Electronics Comm., Geneva, 1996.
- ISO Draft International Standard 13522-6, Information TechnologyCoding of Multimedia and Hypermedia Information, Part 6: Support for Enhanced Interactive Applications, Int'l Org. for Standardization/ Int'l Electronics Comm., Geneva, 1997.
- Davic 1.0 Specifications, Revision 5.0, Imprimeries de Versoix, Versoix, Switzerland, Dec. 1995.
- ETSI 300 468 2d ed.: Digital Broadcasting Systems for Television, Sound, and Data Services, Specification for Service Information (SI) in Digital Video Broadcasting (DVB) Systems, ETSI publications dept., Sophia Antipolis, France, 1997, http://www.etsi.fr.
- ISO 13818-6, Information TechnologyGeneric Coding of Moving Pictures and Associated Audio Information, Part 6: Digital Storage Media Command and Control, Int'l Org. for Standardization/Int'l Electronics Comm., Geneva, 1996.
- S. Dal Lago et al., *Armida: Multimedia Applications Across ATM-Based Networks Accessed via Internet Navigation*, Multimedia Tools and Applications, Vol. 5, No. 2, Sept. 1997.

Resources on the Web include:

The MHEG Centre — *http://www.mhegcentre.com/*: Info on MHEGDitor, MHEGWrite and more.

MHEG-5 User Group — *www.fokus.gmd.de/ovma/mug*

MHEG and the WWW — *http://www.prz.tu-berlin.de/ joe/mheg/mheg_intro.html*

Java MHEG Engine — *http://thunder.informatik.uni-kl.de:8080/MHEG5Engine/*, *http://enterprise.prz.tu-berlin.de/imw/* (Java Source).

MediaTouch — *http://drogo.cselt.stet.it/ufv/mediatouch/index.htm*

MediaTouch RTE (Free) — *http://drogo.cselt.stet.it/ufv/mediatouch/rte.htm*

GL-NET — *http://www.mheg5.de/*: MHEG Editor, Engine for Windows NT.

Moving Pictures Expert Group — *www.cselt.it/mpeg, www.mpeg.org*

Digital Audio Visual Council — *www.davic.org*

Digital Video Broadcasting — *www.dvb.org*

CSELT, Multimedia and Video Services — *www.cselt.it/ufv*

Digital TV Group UK — *http://www.dtg.org.uk/*

MHEG 5 UK — http://www.dtg.org.uk/reference/mheg/_mheg_index.html

Example MHEG Applications — *http://drogo.cselt.stet.it/ufv/mediatouch/applications.htm*