
2.1 Multimedia Tasks and Concerns

Multimedia content is ubiquitous in software all around us, including in our phones, of course. We are interested in this subject from a computer science and engineering point of view, and we are also interested in making interactive applications (or “presentations”), using video editors such as Adobe Premiere or Cyberlink PowerDirector and still-image editors such as Adobe Photoshop in the first instance, but then combining the resulting resources into interactive programs by making use of “authoring” tools such as Flash and Director that can include sophisticated programming. Multimedia often generates problems and considerations that have a more general computer science flavor. For example, most cameras now are smart enough to find faces (with reasonable success)—but just recently such a task was firmly in the domain of Computer Vision, i.e., a branch of Artificial Intelligence dealing with trying to *understand* image content. So such more basic concerns do impact multimedia as it now appears in products, and will tend to increasingly influence the field. Continuing in the Computer Vision direction, a camera owner might be encouraged to think like a computer scientist and ask “What is going on in an image?” A less high-level question is “Where has this image been taken?” (scene recognition), or “Does the image contain a particular object?” (object classification). A still quite difficult question is “Where is an object of interest?” (object detection). And a lower level question might be “Which object does each pixel belong to?” (image segmentation). Thus it does not take long before we find ourselves fully engaged in a classic Computer Vision hierarchy of high-level to detailed description of an image, with scene recognition at the top and image segmentation at the bottom.

In this text, we take a moderate approach to difficulty level, and do not presume to answer such sophisticated questions as those posed above. Nonetheless, studying the fundamentals of the multimedia problem is indeed a fruitful concern and our aim in the book is to give readers the tools they would need to eventually tackle such difficult questions, for example in a work situation.

2.2 Multimedia Presentation

In this section, we briefly outline some effects to keep in mind for presenting multimedia content as well as some useful guidelines for content design [1,2].

Graphics Styles

Careful thought has gone into combinations of color schemes and how lettering is perceived in a presentation. Many presentations are meant for business projected displays, rather than appearing on a screen close to the eye. Human visual dynamics are considered in regard to how such presentations must be constructed. Most of the observations here are drawn from Vetter et al. [3], as is Fig. 2.1.

Color Principles and Guidelines

Some *color schemes* and *art styles* are best combined with a certain theme or style. Color schemes could be, for example, natural and floral for outdoor scenes and solid colors for indoor scenes. Examples of art styles are oil paints, watercolors, colored pencils, and pastels.

A general hint is to not use too many colors, as this can be distracting. It helps to be consistent with the use of color—then color can be used to signal changes in theme.

Fonts

For effective visual communication, large fonts (18 to 36 points) are best, with no more than six to eight lines per screen. As shown in Fig. 2.1, sans serif fonts work better than serif fonts (serif fonts are those with short lines stemming from and at an angle to the upper and lower ends of a letter's strokes). Figure 2.1 shows a comparison of two screen projections, (Figs. 2 and 3 from Vetter et al. [3]).

Figure 2.1 shows *good* use of color and fonts. It has a consistent color scheme, uses large and all sans serif (Arial) fonts. The bottom figure is *poor*, in that too many colors are used, and they are inconsistent. The red adjacent to the blue is hard to focus on, because the human retina cannot focus on these colors simultaneously. The serif (Times New Roman) font is said to be hard to read in a darkened, projection setting. Finally, the lower right panel does not have enough contrast—pretty pastel colors are often usable only if their background is sufficiently different.

A Color Contrast Program

Seeing the results of Vetter et al.'s research, we constructed a small Visual Basic program ¹ to investigate how readability of text colors depends on color and the color of the background.

¹ See <http://www.cs.sfu.ca/mmbook>. There, both the executable and the program source are given.

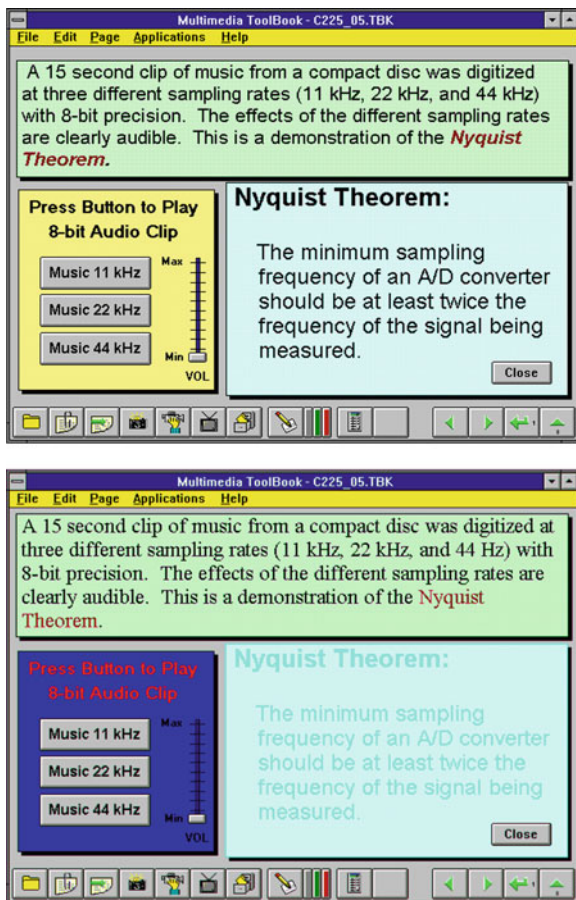


Fig. 2.1 Colors and fonts. *Courtesy of Ron Vetter*

The simplest approach to making readable colors on a screen is to use the principal complementary color as the background for text. For color values in the range 0–1 (or, effectively, 0–255), if the text color is some triple (Red, Green, Blue), or (R, G, B) for short, a legible color for the background is likely given by that color subtracted from the maximum:

$$(R, G, B) \Rightarrow (1 - R, 1 - G, 1 - B) \quad (2.1)$$

That is, not only is the color “opposite” in some sense (not the same sense as artists use), but if the text is bright, the background is dark, and vice versa.

In the Visual Basic program given, sliders can be used to change the background color. As the background changes, the text changes to equal the principal complementary color. Clicking on the background brings up a color-picker as an alternative to the sliders.

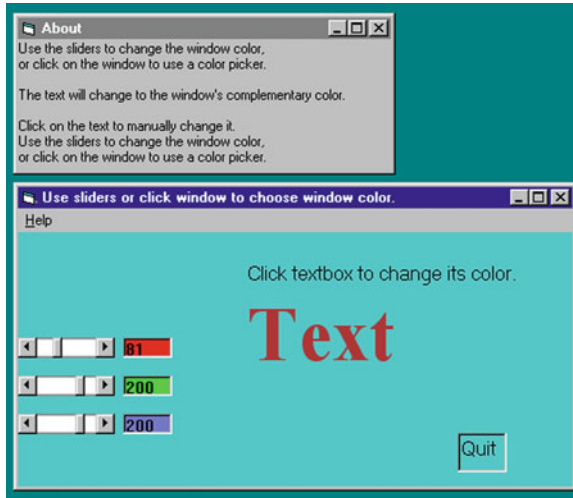


Fig. 2.2 Program to investigate colors and readability

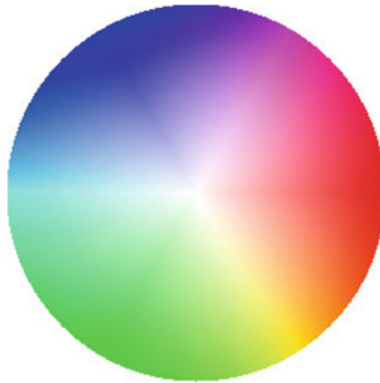


Fig. 2.3 Color wheel

If you feel you can choose a better color combination, click on the text. This brings up a color picker not tied to the background color, so you can experiment. (The text itself can also be edited.) A little experimentation shows that some color combinations are more pleasing than others—for example, a pink background and forest green foreground, or a green background and mauve foreground. Figure 2.2 shows this small program in operation.

Figure 2.3 shows a “color wheel,” with opposite colors equal to $(1 - R, 1 - G, 1 - B)$. An artist’s color wheel will not look the same, as it is based on feel rather than on an algorithm. In the traditional artist’s wheel, for example, yellow is opposite magenta, instead of opposite blue as in Fig. 2.3, and blue is instead opposite orange.

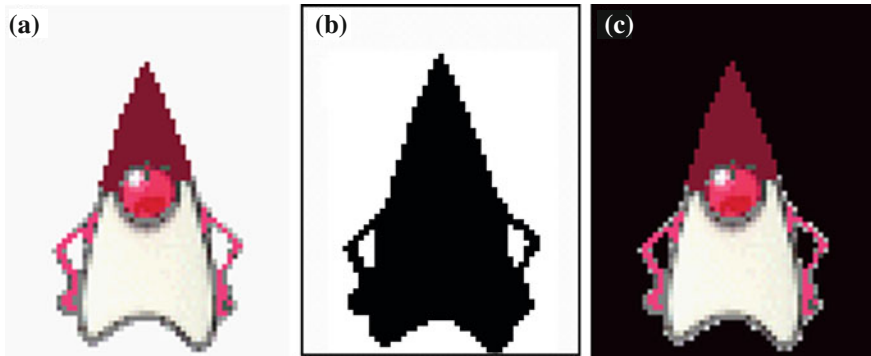


Fig. 2.4 Sprite creation: **a** original; **b** mask image M ; and **c** sprite S . “Duke” figure courtesy of Sun Microsystems

Sprite Animation

Sprites are often used in animation. For example, in Adobe Director (this used to be Macromedia Director), the notion of a sprite is expanded to an instantiation of any resource. However, the basic idea of sprite animation is simple. Suppose we have produced an animation figure, as in Fig. 2.4a. Then it is a simple matter to create a 1-bit mask M , as in Fig. 2.4b, black on white, and the accompanying sprite S , as in Fig. 2.4c.

Now we can overlay the sprite on a colored background B , as in Fig. 2.5a, by first ANDing B and M , then ORing the result with S , with the final result as in Fig. 2.5e. Operations are available to carry out these simple compositing manipulations at frame rate and so produce a simple 2D animation that moves the sprite around the frame but does not change the way it looks.

Video Transitions

Video transitions can be an effective way to indicate a change to the next section. Video transitions are syntactic means to signal “scene changes” and often carry semantic meaning. Many different types of transitions exist; the main types are *cuts*, *wipes*, *dissolves*, *fade-ins*, and *fade-outs*.

A cut, as the name suggests, carries out an abrupt change of image contents in two consecutive video frames from their respective clips. It is the simplest and most frequently used video transition.

A wipe is a replacement of the pixels in a region of the viewport with those from another video. If the boundary line between the two videos moves slowly across the screen, the second video gradually replaces the first. Wipes can be left-to-right, right-to-left, vertical, horizontal, like an iris opening, swept out like the hands of a clock, and so on.

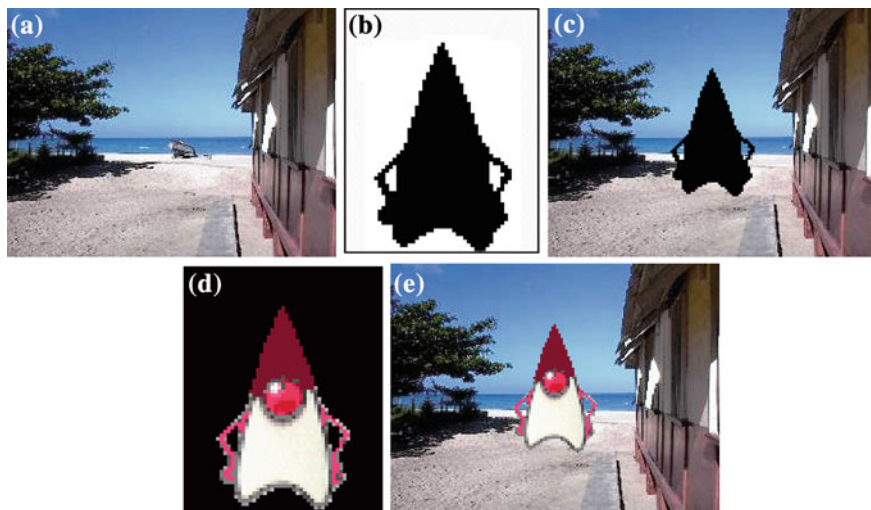


Fig. 2.5 Sprite animation: **a** Background B ; **b** Mask M ; **c** B and M ; **d** Sprite S ; **e** B and M or S

A dissolve replaces every pixel with a mixture over time of the two videos, gradually changing the first to the second. A fade-out is the replacement of a video by black (or white), and fade-in is its reverse. Most dissolves can be classified into two types, corresponding, for example, to *cross dissolve* and *dither dissolve* in Adobe Premiere video editing software.

In type I (cross dissolve), every pixel is affected gradually. It can be defined as

$$\mathbf{D} = (1 - \alpha(t)) \cdot \mathbf{A} + \alpha(t) \cdot \mathbf{B} \quad (2.2)$$

where \mathbf{A} and \mathbf{B} are the color 3-vectors for video A and video B. Here, $\alpha(t)$ is a transition function, which is often linear with time t :

$$\alpha(t) = kt, \quad \text{with } kt_{\max} \equiv 1 \quad (2.3)$$

Type II (dither dissolve) is entirely different. Determined by $\alpha(t)$, increasingly more and more pixels in video A will abruptly (instead of gradually, as in Type I) change to video B. The positions of the pixels subjected to the change can be random or sometimes follow a particular pattern.

Obviously, fade-in and fade-out are special types of a Type I dissolve, in which video A or B is black (or white). Wipes are special forms of a Type II dissolve, in which changing pixels follow a particular geometric pattern.

Despite the fact that many digital video editors include a preset number of video transitions, we may also be interested in building our own. For example, suppose we wish to build a special type of wipe that slides one video out while another video slides in to replace it. The usual type of wipe does not do this. Instead, each video stays in place, and the transition line moves across each “stationary” video, so that the left part of the viewport shows pixels from the left video, and the right part shows pixels from the right video (for a wipe moving horizontally from left to right).



Fig. 2.6 a Video_L ; b Video_R ; c Video_L into place and pushing out Video_R

Suppose we would like to have each video frame not held in place, but instead move progressively farther into (out of) the viewport: we wish to slide Video_L in from the left and push out Video_R . Figure 2.6 shows this process. Each of Video_L and Video_R has its own values of R, G, and B. Note that R is a function of position in the frame, (x, y) , as well as of time t . Since this is video and not a collection of images of various sizes, each of the two videos has the same maximum extent, x_{\max} . (Premiere actually makes all videos the same size—the one chosen in the preset selection—so there is no cause to worry about different sizes).

As time goes by, the horizontal location x_T for the transition boundary moves across the viewport from $x_T = 0$ at $t = 0$ to $x_T = x_{\max}$ at $t = t_{\max}$. Therefore, for a transition that is linear in time, $x_T = (t/t_{\max})x_{\max}$.

So for any time t , the situation is as shown in Fig. 2.7a. The viewport, in which we shall be writing pixels, has its own coordinate system, with the x -axis from 0 to x_{\max} . For each x (and y) we must determine Fig. 2.7a from which video we take RGB, i.e., (Red, Green, Blue) values, and Fig. 2.7b from what x position in the *unmoving* video we take pixel values—that is, from what position x from the left video, say, in its own coordinate system. It is a video, so of course the image in the left video frame is changing in time.

Let us assume that dependence on y is implicit. In any event, we use the same y as in the source video. Then for the red channel (and similarly for the green and blue), $R = R(x, t)$. Suppose we have determined that pixels should come from Video_L . Then the x -position x_L in the *unmoving* video should be $x_L = x + (x_{\max} - x_T)$, where x is the position we are trying to fill in the viewport, x_T is the position in the viewport that the transition boundary has reached, and x_{\max} is the maximum pixel position for any frame.

To see this, we note from Fig. 2.7b that we can calculate the position x_L in Video_L 's coordinate system as the sum of the distance x , in the viewport, and the difference $x_{\max} - x_T$.

Substituting the fact that the transition moves linearly with time, $x_T = x_{\max}(t/t_{\max})$, we can set up a pseudocode solution as in Fig. 2.8. In Fig. 2.8, the slight change in formula if pixels are actually coming from Video_R instead of from Video_L is easy to derive.

The Exercise section below contains suggestions for further such video transitions. As a computer scientist or engineer, you should be easily capable of constructing your own video transitions, rather than relying on simply choosing items from a menu.

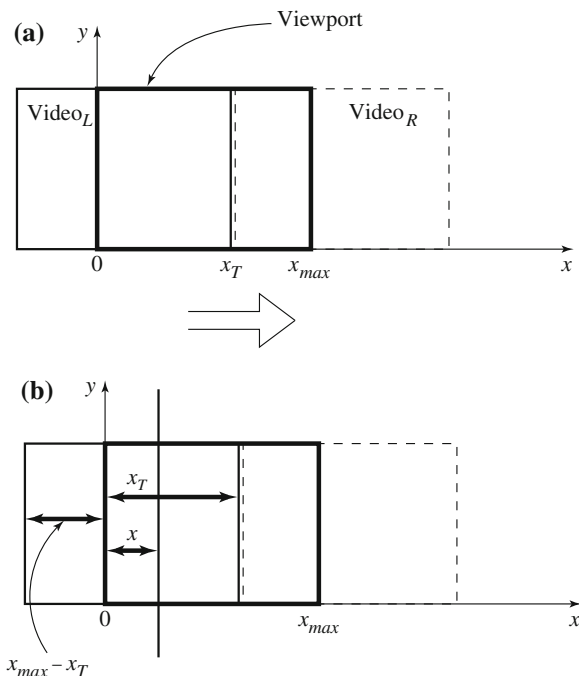


Fig. 2.7 **a** Geometry of Video_L pushing out Video_R ; **b** Calculating position in Video_L from where pixels are copied to the viewport

```

for  $t$  in  $0..t_{max}$ 
  for  $x$  in  $0..x_{max}$ 
    if  $(\frac{x}{x_{max}} < \frac{t}{t_{max}})$ 
       $R = R_L (x + x_{max} * [1 - \frac{t}{t_{max}}], t)$ 
    else
       $R = R_R (x - x_{max} * \frac{t}{t_{max}}, t)$ 

```

Fig. 2.8 Pseudocode for slide video transition

A career in multimedia involves addressing interesting and sometimes challenging tasks that no-one actually solved before!

2.3 Data Compression

One of the most evident and important challenges of using multimedia is the necessity to compress data. Table 2.1 shows some values for standard-definition and for high-definition broadcast video. Clearly, we need excellent and fast data compression in

Table 2.1 Uncompressed video sizes

<i>Standard definition video</i>		
640×480 full color	=	922 kB/frame
@ 30 frames/s	=	28 MB/s
	=	221 Mb/s
× 3,600 s/h	=	100 GB/h
<i>High definition video</i>		
1,920×1,080 full color	=	6.2 MB/frame
@ 30 frames/s	=	187 MB/s
	=	1.5 Gb/s
× 3,600 s/h	=	672 GB/h

order to avoid such high data rates that cause problems for storage and networks, if we tried to share such data, and also for disk I/O.

How much compression is required? In effect, this depends on the application, on the capability of the viewing computer and display, and on the bandwidth (in bits per second) available to perhaps stream and certainly to view the decompressed result.

In the ubiquitous JPEG image compression standard the amount of compression is controlled by a value *Q* in the range 0–100 (and see Sect. 9.1 for details). The “quality” of the resulting image is best for *Q* = 100 and worst for *Q* = 0.

Figure 2.9a shows an original, uncompressed image taken by a digital camera that allows full-accuracy images to be captured, with no data compression at all. For this image, there are 364 rows and 485 columns of pixel data (reduced from 2424 by 3232 to better see the effect of *Q*); so with 8-bit accuracy in each of Red, Green, and Blue pixel values, the total file size is $364 \times 485 \times 3 = 529,620$ bytes (not including file-header information, which stores such values as the row and column size).

In Table 2.2 we show results using different Quality Factors in JPEG compression. Indeed, we can greatly shrink the file size down, but for small values of *Q* the resulting image is poor.

We can see in Fig. 2.9 that while *Q* = 25 is not terrible, if we insist on going down to a Quality Factor of *Q* = 5 we do end up with an unusable image. However this exercise does shows us something interesting: the color part, as opposed to the black-and-white (i.e., the grayscale) may well be the less noticeable problem for high compression ratios (i.e., low amounts of data surviving compression). We will see how color and grayscale are in fact treated differently, in Chap. 9.

Compression indeed saves the day, but at a price too. JPEG compression can effect a compression ratio of 25:1 with little loss of quality. For video compression the MPEG video compression standard, set out in Chap. 11, can produce a compression ratio of 100:1 while retaining reasonable quality (Fig. 2.9).

However, let us look at how expensive image and video processing is in terms of processing in the CPU. Suppose we have an image whose pixels we wish to darken, by a factor of 2. The following code fragment is pseudocode for such an operation:



Fig. 2.9 JPEG compression: **a** original uncompressed image; **b** JPEG compression with Quality Factor $Q = 75$ (the typical default); **c**, **d** Factors $Q = 25$ and $Q = 5$

Table 2.2 JPEG file sizes (bytes) and percentage size of data for JPEG compression with Quality Factor $Q = 75, 25$, and 5

Quality factor	Compressed file size	Percentage of original (%)
–	529,620	100
75	37,667	7.11
25	16,560	3.13
5	5,960	1.13

```
for x = 1 to columns
  for y 1 to rows
    image[x,y].red /= 2;
    image[x,y].green /= 2;
    image[x,y].blue /= 2;
  }
}
```

On a RISC machine, the loop amounts to one increment, one check, and one branch instruction. There are also three loads, three shifts, and three stores. This makes a total of 12 instructions per pixel, i.e., per 3 bytes. So, we have 4 instructions per image-byte. For standard-definition video we have 28 MB/s, meaning $28 \times 4 = 112$

mega-instructions per second. For high-definition, at 187 MB/s, we need 748 mega-instructions per second.

This is certainly possible. However, JPEG compression takes some 300 instructions per pixel, or in other words 100 instructions per image byte. This yields numbers of 2.8 billion instructions per second for standard-definition and 19 billion instructions per second for high-definition, which begins to be a real constraint! Clearly, clever techniques are required, and we will view these in later chapters.

Other issues arise from trying to interact with multiple streams of data; e.g., what happens if we tried to show video of a news interview, plus some video of background information, plus data streams of additional information, etc. Is compositing (putting together) such information first, and then compressing the best way forward? Or is compositing at the receiver end? Multimedia tends to open up new questions on Computer Science itself. Multiple data streams place new burdens on operating systems, in terms of scheduling and resource management.

In addition, new capabilities can imply new demands: what happens if the rock band needs to rehearse music together, but they are not in the same place (a Distributed Music problem). The question becomes one of how much latency (time-lag) is acceptable when we are doing compression, for various applications. For music rehearsal, all the band members have to hit the lead note at very close to the same time!

2.4 Multimedia Production

A multimedia project can involve a host of people with specialized skills. In this book we emphasize technical aspects, but also multimedia production can easily involve an art director, graphic designer, production artist, producer, project manager, writer, user interface designer, sound designer, videographer, and 3D and 2D animators, as well as programmers.

The production timeline would likely only involve programming when the project is about 40 % complete, with a reasonable target for an alpha version (an early version that does not contain all planned features) being perhaps 65–70 % complete. Typically, the design phase consists of storyboarding, flowcharting, prototyping, and user testing, as well as a parallel production of media. Programming and debugging phases would be carried out in consultation with marketing, and the distribution phase would follow.

A storyboard depicts the initial idea content of a multimedia concept in a series of sketches. These are like “keyframes” in a video—the story hangs from these “stopping places.” A flowchart organizes the storyboards by inserting navigation information—the multimedia concept’s structure and user interaction. The most reliable approach for planning navigation is to pick a traditional data structure. A hierarchical system is perhaps one of the simplest organizational strategies.

Multimedia is not really like other presentations, in that careful thought must be given to organization of movement between the “rooms” in the production. For

example, suppose we are navigating an African safari, but we also need to bring specimens back to our museum for close examination—just how do we effect the transition from one locale to the other? A flowchart helps imagine the solution.

The flowchart phase is followed by development of a detailed functional specification. This consists of a walk-through of each scenario of the presentation, frame by frame, including all screen action and user interaction. For example, during a mouseover for a character, the character reacts, or a user clicking on a character results in an action.

The final part of the design phase is prototyping and testing. Some multimedia designers use a specialized multimedia authoring tool at this stage already, even if the intermediate prototype will not be used in the final product or continued in another tool. User testing is, of course, extremely important before the final development phase.

2.5 Multimedia Sharing and Distribution

Multimedia content, once produced, needs to be published and then shared among users. In recent years, traditional storage and distribution media, such as optical disks, have been largely replaced by USB flash drives or solid-state drives (SSD), or more conveniently, the Internet.

Consider YouTube, the most popular video sharing site over the Internet, as an example. A user can easily create a Google account and channel (as YouTube is now owned by Google), and then upload a video, which will be shared to everyone or to selected users. YouTube further enables titles and tags that are used to classify the videos and link similar videos together (shown as a list of related videos). Figure 2.10 shows the webpage for uploading a video from a local computer to YouTube. The video, captured by us for a 1905 Edison Fireside phonograph with a cygnet horn playing a cylinder record, can be searched from YouTube's homepage with "Edison Phonograph Multimedia Textbook," which was the title and tag supplied by us.

Figure 2.11 shows the YouTube page of this video. It also offers a list of related videos, recommended by YouTube using the title and the tags. Ideally, we expect that it is linked to other videos about Edison phonographs or multimedia textbooks. The results shown in Fig. 2.11 are not exactly what we would expect, but they do relate to a certain degree. Note here only the texts of titles and tags are used for related videos, not the video content itself. Indeed, multimedia content retrieval and recommendation remains quite difficult, and we will review some of the basic techniques in Chap. 20.

The link to this video can be fed into such other social networking sites such as Facebook or Twitter as well, potentially propagating to many users of interest in a short time, as we will examine in Chap. 18.

The Internet is reshaping traditional TV broadcasting, as well. In the UK, the BBC's iPlayer has been successfully broadcasting high-quality TV programs to both TV subscribers with set-top boxes and public Internet users with Adobe

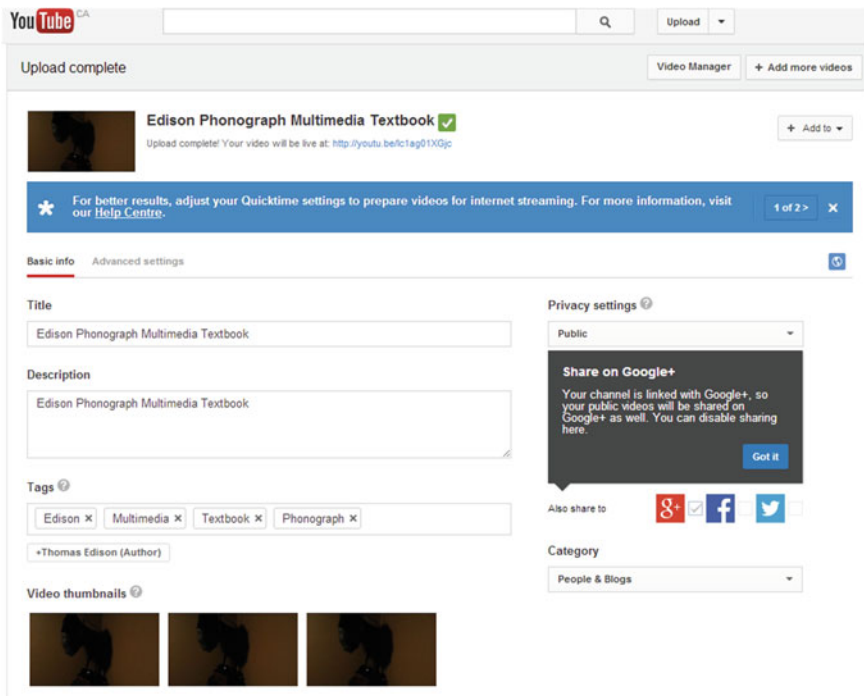


Fig.2.10 The webpage for uploading a YouTube video. The video, titled “Edison Phonograph Multimedia Textbook” is open to all users (Privacy settings: Public) and can be searched in the YouTube homepage using the title or tags. Note that the video thumbnails are automatically generated by YouTube

Flashplayer since 2007; in the US, CNBC, Bloomberg Television, and Showtime use live-streaming services from the BitGravity’s Content Distribution Network to stream live television to paid subscribers. China, the largest *Internet Protocol TV* (IPTV) market by subscribers (12.6 million) to date, is probably the most vigorous market, seeing a wide range of technologies competing with each other and with dedicated IPTV networks.

Users’ viewing habits are also changing. Compelling content is the core foundation of any IPTV proposition, which remains true today; yet IPTV services are becoming highly personalized, integrated, portable, and on-demand. Most service providers are moving beyond basic video offerings toward richer user experiences, particularly with the support for multi-screen viewing across TVs, PCs, tablets, and smartphones. Meanwhile, 3D, multi-view, and multi-streaming are being developed, in which multiple video streams from the same event are delivered to a user, who will be able to switch between camera views. This is a real recognition by service providers of what is happening in homes across the planet—that families are voraciously and simultaneously consuming streamed high-definition video on devices other than the traditional set-top box/TV pairs.

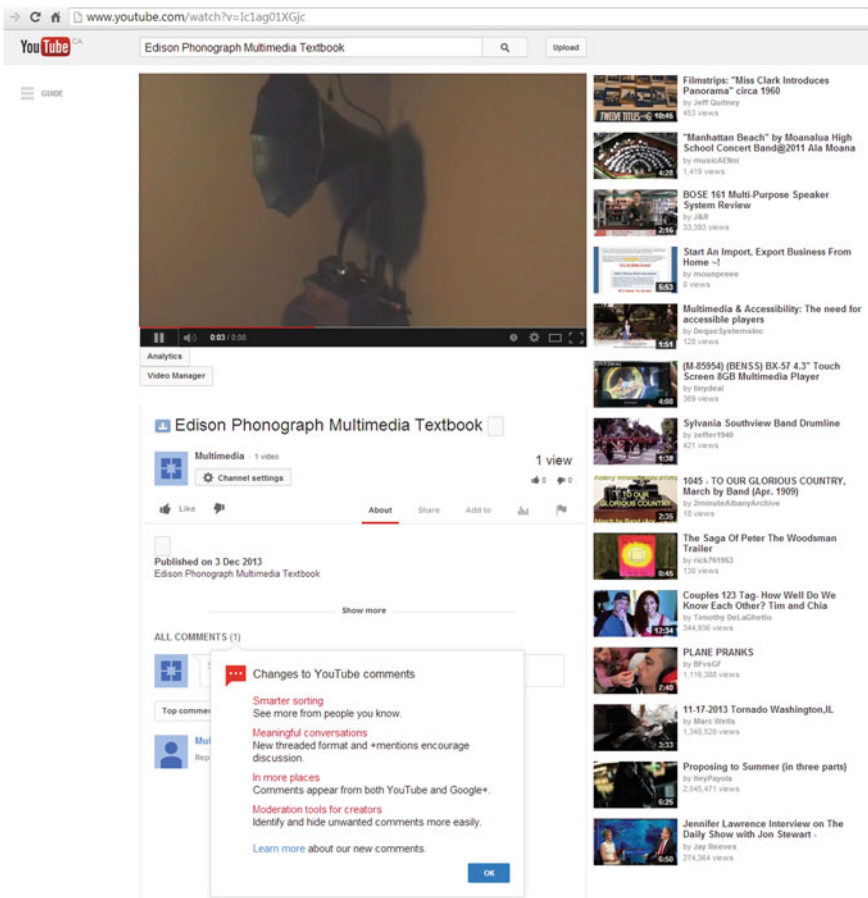


Fig. 2.11 The YouTube page for the video uploaded. The list of related videos are shown on the *right side*, and users can post their comments, too

The scaling challenge for multimedia content distribution however is enormous [4]. To reach 100 million viewers, delivery of TV quality video encoded in MPEG-4 (1.5Mbps) will require an aggregate capacity of 1.5 Tbps. To put things into perspective, consider two large-scale Internet video broadcasts: the CBS broadcast of the NCAA tournament in March 2006, which at the peak had 268,000 simultaneous viewers, and the opening ceremony of the London Summer Olympics in July 2012, which drew a peak broadcast audience of 27.1 million, of which 9.2 million were via BBC's mobile site and 2.3 million on tablets. Even with low bandwidth Internet video of 400Kbps, the CBS/NCAA broadcast needs more than 100Gbps server and network bandwidth; on the busiest day of the London Olympics, BBC's website delivered 2.8 petabytes, with the peak traffic at 700 Gbps. These can hardly be handled

by any single server. Later, in Chaps. 16 and 19, we will see effective solutions using peer-to-peer, content distribution networks, or the cloud to deal with such challenges.

2.6 Some Useful Editing and Authoring Tools

This text is primarily concerned with principles of multimedia—the fundamentals to be grasped for a real understanding of this subject. Nonetheless, we need real vehicles for showing this understanding, and straight programming in C++ or Java is not always the best way of showing your knowledge. Most introductory multimedia courses ask you to at least start off with delivering some multimedia product (e.g., see Exercise 10).

Therefore, we will consider some popular authoring tools. Since the first step in creating a multimedia application is probably creation of interesting video clips, we start off with looking at a video editing tool. This is not really an authoring tool, but video creation is so important that we include a small introduction to one such program.

The tools we look at are the following (which all happen to be Adobe products):

- Premiere
- Director
- Flash.

While this is of course by no means an exhaustive list, these tools are often used in creating multimedia content.

2.6.1 Adobe Premiere

Premiere Basics

Adobe Premiere is a very simple yet powerful video editing program that allows you to quickly create a simple digital video by assembling and merging multimedia components. It effectively uses a “score” authoring metaphor, in that components are placed in “tracks” horizontally, in a Timeline window that in a sense resembles a musical score.

The `File > New Project` command opens a window that displays a series of “presets”—assemblies of values for frame resolution, compression method, and frame rate. There are many preset options, many of which might conform to a DV-NTSC or DV-PAL video standard, HDV, MPEG2, etc. depending on your installation.

Start by importing resources, such as AVI (Audio Video Interleave) video files and WAV sound files and dragging them from the Project window onto tracks 1 or 2. (In fact, you can use up to 99 video and 99 audio tracks!).

Usually you see only three tracks: Video 1, Video 2, and Video 3. Video transitions, meaning how we change from one video segment to another, are in the Effects window

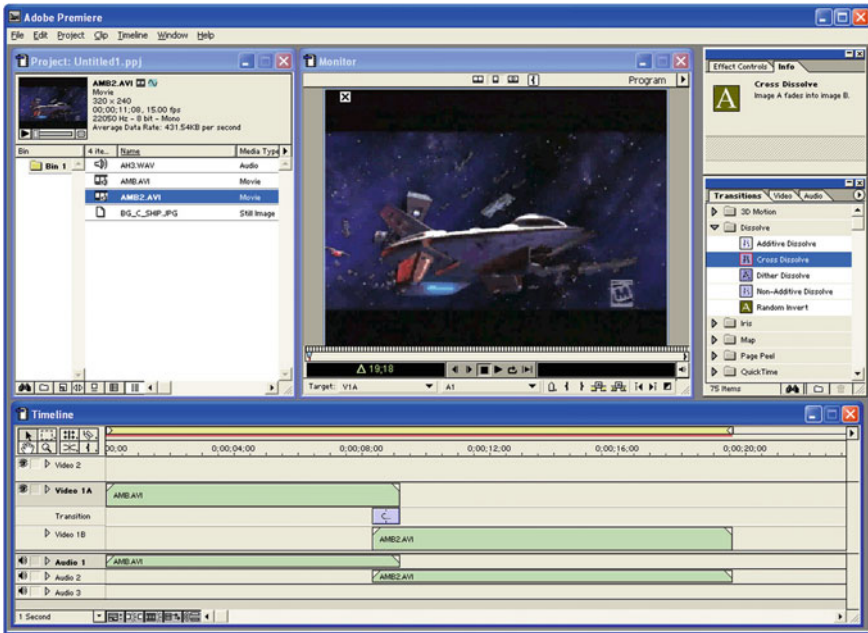


Fig. 2.12 Adobe Premiere screen

Transitions are dragged into the Transitions track from the Transition window, such as a gradual replacement of Video 1 by Video 2 (a dissolve), sudden replacement of random pixels in a checkerboard (a dither dissolve), or a wipe, with one video sliding over another. There are many other transitions to choose from.

You can import WAV sound files by dragging them to Audio 1 or Audio 2 of the Timeline window or to any additional sound tracks. You can edit the properties of any sound track by right-clicking on it.

Figure 2.12 shows what a typical Premiere screen might look like. The yellow ruler at the top of the Timeline window delineates the working timeline—drag it to the right amount of time. The 1 s dropdown box at the bottom represents showing one video keyframe per 1 s.

To “compile” the video, go to `Sequence > Render Work Area` to have a look at the product you are making, and save the project as a .prproj file. To save the movie, select `File > Export > Movie`. Now it gets interesting, because you must make some choices here, involving how and in what format the movie is to be saved. Figure 2.13 shows the project options. The dialogs that tweak each codec are provided by the codec manufacturer; bring these up by clicking on the parts of the project being controlled in a panel or via a `Configure` button. Compression codecs (compression–decompression protocols) are often in hardware on the video capture card. If you choose a codec that requires hardware assistance, someone else’s system may not be able to play your brilliant digital video, and all is in vain!

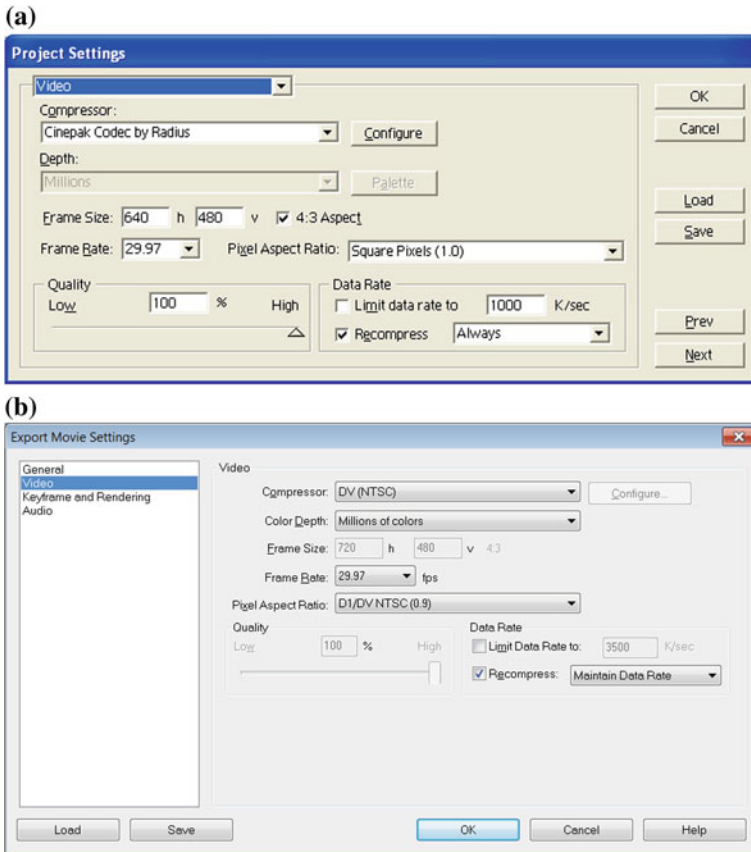


Fig. 2.13 **a** Adobe Premiere output options in project settings; **b** Adobe Premiere submenu for compression options

Images can also be inserted into tracks. We can use transitions to make the images gradually appear or disappear in the final video window. To do so, set up a “mask” image, as in Fig. 2.14. Here, we have imported an Adobe Photoshop layered image, with accompanying alpha channel made in Photoshop.

Then in Premiere, we click on the image, which has been placed in its own video track, and in the Effect Controls window, click the triangle next to the Opacity property to enter a new opacity value, making the image partially transparent. Premiere controls making the face shown in the image have a transparent background using the alpha channel. It is also simple to use Motion effect to have the image fly in and out of the frame.

In Photoshop, we set up an alpha channel as follows:

1. Use an image you like—a .JPG, say.
2. Make the background some solid color—white, say.
3. Make sure you have chosen Image > Mode > RGB Color.

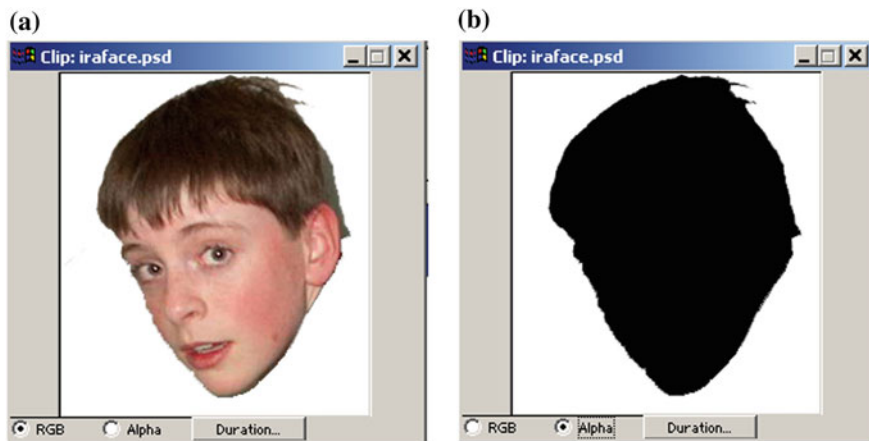


Fig. 2.14 Adobe Premiere preview clip viewer, for an Adobe Photoshop image with an alpha-channel layer. **a:** RGB channels: color on white background. **b:** Alpha channel: *black* on *white* background

4. Select that background area (you want it to remain opaque in Premiere)—use the magic wand tool.
5. Go to `Select > Save Selection...`
6. Ensure that `Channel = New`. Press OK.
7. `GotoWindow > Show Channels`, double-click the new channel, and rename it Alpha; make its color (0, 0, 0).
8. Save the file as a PSD.

If the alpha channel you created in Photoshop (the “key” we’re using) has a white background, you will need to reverse the key in Premiere.

Premiere has its own simple method of creating titles (to give credit where credit is due) for your digital video.

Another nice feature of Premiere is that it is simple to use in capturing video from old analog sources. To form a digital video from a videotape or camcorder input, go to `File > Capture`. (The menu for video/audio capture options appears by right-clicking the capture window.)

2.6.2 Adobe Director

Director Windows

Director is a complete environment (see Fig. 2.15) for creating interactive “movies.” The movie metaphor is used throughout Director, and the windows used in the program reflect this. The main window, on which the action takes place, is the Stage.

The other two main windows are Cast and Score. A Cast consists of resources a movie may use, such as bitmaps, sounds, vector-graphics shapes, Flash movies,

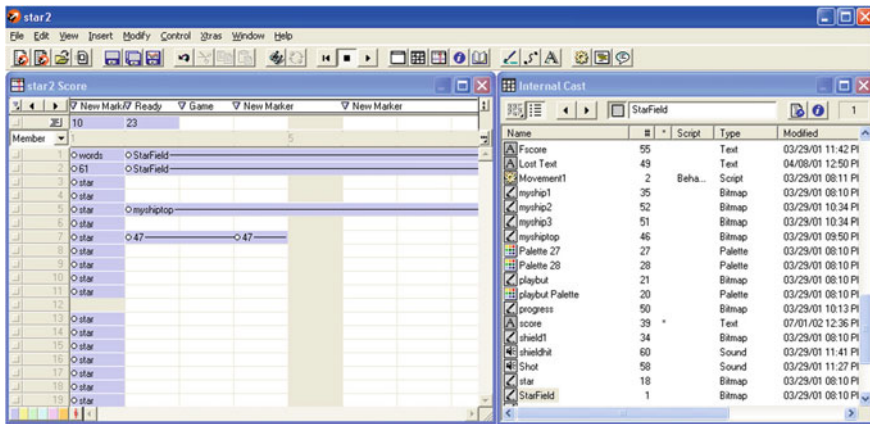


Fig. 2.15 Director: main windows

digital videos, and scripts. Cast members can be created directly or simply imported. Typically you create several casts, to better organize the parts of a movie. Cast members are placed on the Stage by dragging them there from the Cast window. Because several instances may be used for a single cast member, each instance is called a sprite. Typically, cast members are raw media, whereas sprites are objects that control where, when, and how cast members appear on the stage and in the movie.

Sprites can become interactive by attaching “behaviors” to them (for example, make the sprite follow the mouse) either prewritten or specially created. Behaviors are in the internal script language of Director, called Lingo. Director is a standard event-driven program that allows easy positioning of objects and attachment of event procedures to objects. A very useful part of Lingo is called Imaging Lingo, which can directly manipulate images from within Director. This means that image manipulation can be carried out in code, making for code-based visual effects.

The set of predefined events is rich and includes mouse events as well as network events (an example of the latter would be testing whether cast members are downloaded yet). The type of control achievable might be to loop part of a presentation until a video is downloaded, then continue or jump to another frame. Bitmaps are used for buttons, and the most typical use would be to jump to a frame in the movie after a button-click event.

The Score window is organized in horizontal lines, each for one of the sprites, and vertical frames. Thus the Score looks somewhat like a musical score, in that time is from left to right, but it more resembles the list of events in a MIDI file (see Chap.6.)

Both types of behaviors, prewritten and user-defined, are in Lingo. The Library palette provides access to all prewritten behavior scripts. You can drop a behavior onto a sprite or attach behaviors to a whole frame.

If a behavior includes parameters, a dialog box appears. For example, navigation behaviors must have a specified frame to jump to. You can attach the same behavior to

many sprites or frames and use different parameters for each instance. Most behaviors respond to simple events, such as a click on a sprite or the event triggered when the “playback head” enters a frame. Most basic functions, such as playing a sound, come prepackaged. Writing your own user-defined Lingo scripts provides more flexibility. Behaviors are modified using *Inspector* windows: the Behavior Inspector, or Property Inspector.

Animation

Traditional animation (cel animation) is created by showing slightly different images over time. In Director, this approach amounts to using different cast members in different frames. To control this process more easily, Director permits combining many cast members into a single sprite.

A less sophisticated-looking but simple animation is available with the *tweening* feature of Director. Here, you specify a particular image and move it around the stage without altering the original image. “Tweening” refers to the job of minor animators, who used to have to fill in between the keyframes produced by more experienced animators—a role Director fulfills automatically.

To prepare such an animation, specify the path on the stage for the tweened frames to take. You can also specify several keyframes and the kind of curve for the animation to follow between keyframes. You also specify how the image should accelerate and decelerate at the beginning and end of the movement (“ease-in” and “ease-out”). Figure 2.16 shows a tweened sprite.

A simple kind of animation called *palette animation* is also widely used. If images are 8-bit, cycling through the color lookup table or systematically replacing lookup table entries produces interesting (or strange) effects. Palette animation is explained in detail in Sect. 3.1.7

The Score window’s important features are channels, frames, and the playback head. The latter shows where we are in the score; clicking anywhere in the score repositions the playback head. Channels are the rows in the Score and can contain sprite instances of visible media. Therefore, these numbered channels are called Sprite channels.

At the top of the Score window are Special Effects channels for controlling the palettes, tempo, transitions, and sounds. Figure 2.17 shows these channels in the Score window. Frames are numbered horizontally in the Sprite and Special Effects channels. A frame is a single step in the movie, as in a traditional film. The movie’s playback speed can be modified by resetting the number of frames per second.

Control

You can place named markers at any frame. Then the simplest type of control event would be to jump to a marker. In Director parlance, each marker begins a Scene. Events triggered for frame navigation are Go To Frame, Go To Marker, or Hold on

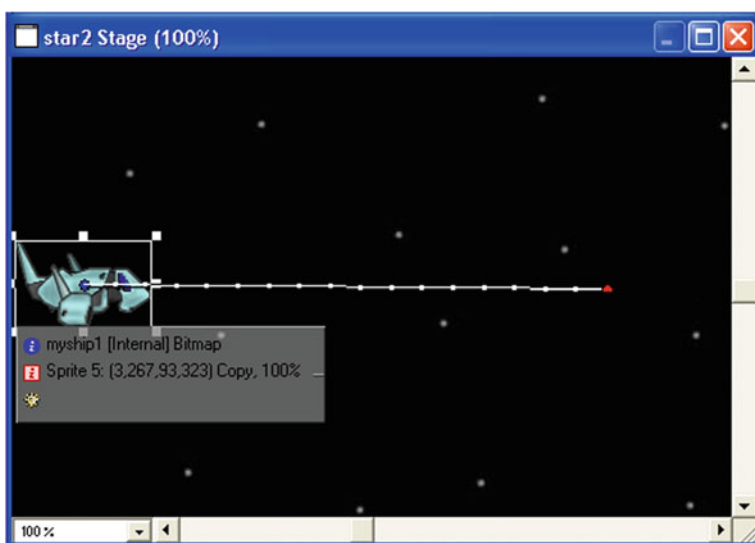


Fig.2.16 A tweened sprite

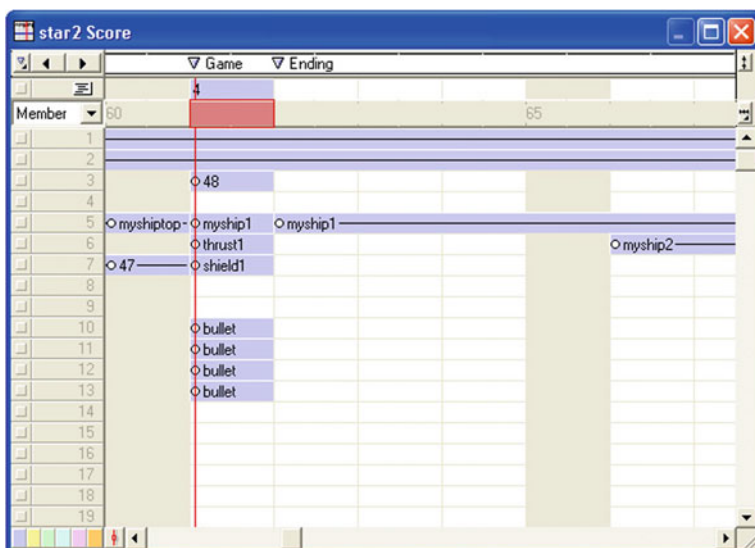


Fig.2.17 Score window

Current Frame, which stops the movie at that frame. Behaviors for frames appear in a Script Channel in the score window.

Buttons are simply bitmaps with behaviors attached. You usually make use of two bitmaps, one depicting the depressed state of the button and one for the undepressed state. Then the built-in event `on mouseUp` effects the jump.

Lingo Scripts

Director uses four types of scripts: behaviors, scripts attached to cast members, movie scripts, and parent scripts. Behaviors, movie scripts, and parent scripts all appear as cast members in the Cast window.

A “behavior” is a Lingo script attached to a sprite or a frame. You might use a script to determine whether a sprite moves, based on whether the user has clicked a button. A useful feature is that a script can control when a multimedia resource is played, depending on how much of the resource has already streamed from the Web. To attach a behavior, drag it from a cast to a sprite or frame in the Score or on the Stage.

Also used are Movie scripts, which are available to the entire movie. Movie scripts can control event responses when a movie starts, stops, or pauses and can also respond to events, such as key presses and mouse clicks. Parent scripts can be used to create multiple instances of an object without adding cast members to the score.

User-written Lingo scripts can be used to create animation or to respond to typical events, such as user actions with the keyboard and mouse. Scripts can also be used to stream videos from the Internet, perform navigation, format text, and so on.

Lingo scripts also extend behaviors beyond what the Score alone can do. The basic data type is a list, which is of course the fundamental data structure. Using lists, you can manipulate arrays as well. Math operations and string handling are also available. Lists are of two types: *linear* and *property*.

A linear list is simply a list as in LISP, such as `[12, 32, 43]`. A property list is an association list, again as in LISP: each element contains two values separated by a colon. Each property is preceded by a number sign. For example, statements to create two different property lists to specify the Stage coordinates of two sprites are as follows:

```
sprite1Location = [#left:100, #top:150, #right:300, #bottom:350]
sprite2Location = [#left:400, #top:550, #right:500, #bottom:750]
```

Lingo has many functions that operate on lists, such as `append` to add an element to the end of a list and `deleteOne` to delete a value from a list.

Lingo Specifics

- The function `the frame` refers to the current frame.
- Special markers `next` or `previous` refer to adjacent *markers* (not adjacent frames).

- `Function marker (-1)` returns the identifier for the previous marker. If the frame is marked and has a marker name, `marker (0)` returns the name of the current frame; otherwise, it returns the name of the previous marker.
- `movie "Jaws"` refers to the start frame of the global movie named "Jaws". This would typically be the name of another Director movie. The reference frame 100 of movie "Jaws" points into that movie. These details are well outlined in the Lingo Help portion of the online help.

Lingo is a standard, event-driven programming language. Event handlers are attached to specific events, such as a `mouseDown` message. Scripts contain event handlers. You attach a set of event handlers to an object by attaching the script to the object.

3D Sprites

A sophisticated feature in Director is the ability to create, import, and manipulate 3D objects on the stage. For e.g., a simple 3D object that can be added in Director is 3D text.

Properties and Parameters

Lingo behaviors can be created with more flexibility by specifying behavior parameters. Parameters can change a behavior by supplying input to the behavior when it is created. If no parameters are specified, a default value will be used.

Director Objects

Director has two main types of objects: those created in Lingo and those on the Score. Parent scripts are used to create a new object in Lingo. A behavior can be transformed into a parent script by changing the script type in the Property Inspector. Parent scripts are different from other behaviors, in that parameters are passed into the object when it is created in Lingo script.

Parent scripts can be created and changed only in Lingo, while objects in the Score can only be manipulated. The most common objects used are the sprites in the Score. Sprites can be used only in the same time period as the Lingo script referencing them. Reference the sprite at the channel using the `Sprite` keyword followed by the sprite channel number.

2.6.3 Adobe Flash

Flash is a simple authoring tool that facilitates the creation of interactive movies. Flash follows the score metaphor in the way the movie is created and the windows

are organized. Here we give a brief introduction to Flash and provide some examples of its use.

Windows

A movie is composed of one or more scenes, each a distinct part of the movie. The command `Insert > Scene` creates a new scene for the current movie.

In Flash, components such as images and sound that make up a movie are called *symbols*, which can be included in the movie by placing them on the Stage. The stage is always visible as a large, white rectangle in the center window of the screen. Three other important windows in Flash are the Timeline, Library, and Tools.

Library Window

The Library window shows all the current symbols in the scene and can be toggled by the `Window > Library` command. A symbol can be edited by double-clicking its name in the library, which causes it to appear on the stage. Symbols can also be added to a scene by simply dragging the symbol from the Library onto the stage.

Timeline Window

The Timeline window manages the layers and timelines of the scene. The left portion of the Timeline window consists of one or more layers of the Stage, which enables you to easily organize the Stage's contents. Symbols from the Library can be dragged onto the Stage, into a particular layer. For example, a simple movie could have two layers, the background and foreground. The background graphic from the library can be dragged onto the stage when the background layer is selected.

Another useful function for layering is the ability to lock or hide a layer. Pressing the circular buttons next to the layer name can toggle their hidden/locked state. Hiding a layer can be useful while positioning or editing a symbol on a different layer. Locking a layer can prevent accidental changes to its symbols once the layer has been completed.

The right side of the Timeline window consists of a horizontal bar for each layer in the scene, similar to a musical score. This represents the passage of time in the movie. The Timeline is composed of a number of keyframes in different layers. An event such as the start of an animation or the appearance of a new symbol must be in a keyframe. Clicking on the timeline changes the current time in the movie being edited.

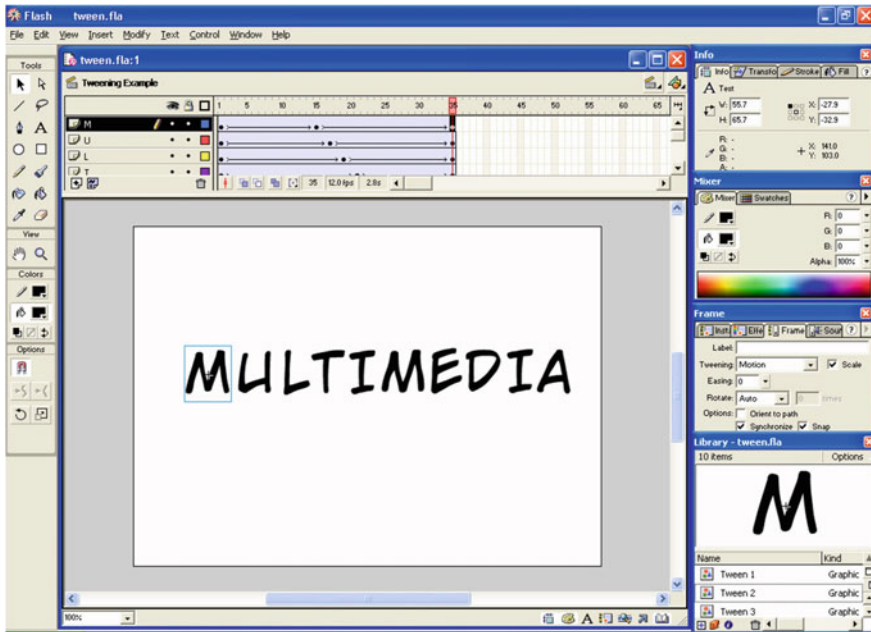


Fig. 2.18 Adobe Flash

Tools Window

The Tools window, which allows the creation and manipulation of images, is composed of four main sections: *Tools*, *View*, *Colors*, and *Options*. Tools consists of selection tools that can be used to demarcate existing images, along with several simple drawing tools, such as the pencil and paint bucket. View consists of a zoom tool and a hand tool, which allow navigation on the Stage. Colors allows foreground and background colors to be chosen, and symbol colors to be manipulated. Options allows additional options when a tool is selected. Figure 2.18 shows the basic Flash screen.

Symbols

Symbols can be either composed from other symbols, drawn, or imported into Flash. Flash is able to import several audio, image, and video formats into the symbol library. Symbols can take on one of three behaviors: a *button*, a *graphic*, or a *movie*. Symbols, such as a button, can be drawn using the Tools window.

Buttons

To create a simple button, create a new symbol with the button behavior. The Timeline window should have four keyframes: `up`, `down`, `over`, and `hit`. These keyframes show different images of the button when the specified action is taken. Only the `up` keyframe is required and is the default; all others are optional. A button can be drawn by selecting the rectangular tool in the Tools window and then dragging a rectangle onto the Stage.

To add images, so that the button's appearance will change when an event is triggered, click on the appropriate keyframe and create the button image. After at least one keyframe is defined, the basic button is complete, although no action is yet attached to it. Actions are discussed further in the ActionScripts section below.

Creating a symbol from other symbols is similar to creating a scene: drag the desired symbols from the Library onto the Stage. This allows the creation of complex symbols by combining simpler symbols.

Animation in Flash

Animation can be accomplished by creating subtle differences in each keyframe of a symbol. In the first keyframe, the symbol to be animated can be dragged onto the stage from the Library. Then another keyframe can be inserted, and the symbol changed. This can be repeated as often as needed. Although this process is time-consuming, it offers more flexibility than any other technique for animation. Flash also allows specific animations to be more easily created in several other ways. *Tweening* can produce simple animations, with changes automatically created between keyframes.

Tweening

There are two types of tweening: *shape* and *movement* tweening. Shape tweening allows you to create a shape that continuously changes to a different shape over time. Movement tweening allows you to place a symbol in different places on the Stage in different keyframes. Flash automatically fills in the keyframes along a path between the start and finish. More advanced tweening allows control of the path as well as of acceleration. Movement and shape tweenings can be combined for additional effect.

Mask animation involves the manipulation of a layer mask—a layer that selectively hides portions of another layer. For example, to create an explosion effect, you could use a mask to cover all but the center of the explosion. Shape tweening could then expand the mask, so that eventually the whole explosion is seen to take place. Figure 2.19 shows a scene before and after a tweening effect is added.



Fig. 2.19 Before and after tweening letters

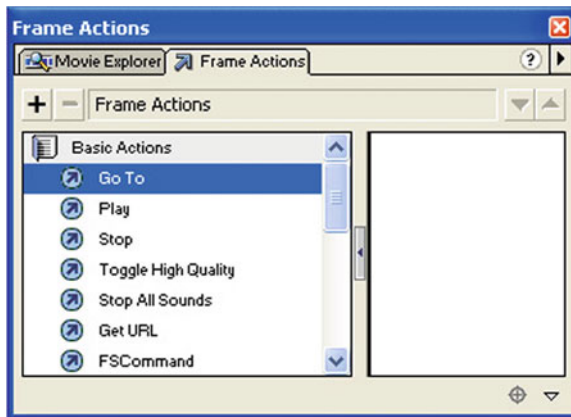


Fig. 2.20 ActionScripts window

ActionScripts

ActionScripts allow you to trigger events such as moving to a different keyframe or stopping the movie. ActionScripts can be attached to a keyframe or symbols in a keyframe. Right-clicking on the symbol and pressing **Actions** in the list can modify the actions of a symbol. Similarly, by right-clicking on the keyframe and pressing **Actions** in the pop-up, you can apply actions to a keyframe. A Frame Actions window will come up, with a list of available actions as well as the current actions being applied. ActionScripts are broken into six categories: *Basic Actions*, *Actions*, *Operators*, *Functions*, *Properties*, and *Objects*. Figure 2.20 shows the Frame Actions window.

Basic Actions allow you to attach many simple actions to the movie. Some common actions are

- **Goto**. Moves the movie to the keyframe specified and can optionally stop. The stop action is commonly used to stop interactive movies when the user is given an option.
- **Play**. Resumes the movie if the movie is stopped.
- **Stop**. Stops the movie if it is playing.

- **Tell Target.** Sends messages to different symbols and keyframes in Flash. It is commonly used to start or stop an action on a different symbol or keyframe.

The *Actions* category contains many programming constructs, such as `Loops` and `Goto` statements. Other actions are also included, similar to those in typical high-level, event-driven programming languages, such as Visual Basic. The *Operators* category includes many comparison and assignment operators for variables. This allows you to perform operations on variables in the `ActionScript`.

The *Functions* category contains built-in functions included in Flash that are not specific to a Flash object. The *Properties* section includes all the global variables predefined in Flash. For example, to refer to the current frame, the variable `_currentframe` is defined. The *Objects* section lists all objects, such as movie clips or strings and their associated functions.

Buttons need `ActionScripts`—event procedures—so that pressing the button will cause an effect. It is straightforward to attach a simple action, such as replaying the Flash movie, to a button.

2.7 Exercises

1. What extra information is multimedia good at conveying?
 - (a) What can spoken text convey that written text cannot?
 - (b) When might written text be better than spoken text?
2. Find and learn Autodesk 3ds Max (formerly 3D Studio Max) in your local lab software. Read the online tutorials to see this software's approach to a 3D modeling technique. Learn texture mapping and animation using this product. Make a 3D model after carrying out these steps.
3. Design an interactive webpage using Adobe Dreamweaver. HTML 4 provides *layer* functionality, as in Adobe Photoshop. Each layer represents an HTML object, such as text, an image, or a simple HTML page (and the Adobe HTML5 Pack is an extension to Adobe Dreamweaver). In Dreamweaver, each layer has a marker associated with it. Therefore, highlighting the layer marker selects the entire layer, to which you can apply any desired effect. As in Flash, you can add buttons and behaviors for navigation and control. You can create animations using the Timeline behavior.
4. Suppose we wish to create a simple animation, as in Fig. 2.21. Note that this image is exactly what the animation looks like at some time, not a figurative representation of the *process* of moving the fish; the fish is repeated as it moves. State what we need to carry out this objective, and give a simple pseudocode solution for the problem. Assume we already have a list of (x, y) coordinates for the fish path, that we have available a procedure for centering images on path positions, and that the movement takes place on top of a video.
5. For the slide transition in Fig. 2.8, explain how we arrive at the formula for x in the unmoving right video R_R .



Fig. 2.21 Sprite, progressively taking up more space

6. Suppose we wish to create a video transition such that the second video appears under the first video through an opening circle (like a camera iris opening), as in Fig. 2.22. Write a formula to use the correct pixels from the two videos to achieve this special effect. Just write your answer for the red channel.
7. Now suppose we wish to create a video transition such that the second video appears under the first video through a moving radius (like a clock hand), as in Fig. 2.23. Write a formula to use the correct pixels from the two videos to achieve this special effect for the red channel.
8. Suppose you wish to create a wavy effect, as in Fig. 2.24. This effect comes from replacing the image x value by an x value offset by a small amount. Suppose the image size is 160 rows \times 120 columns of pixels.
 - (a) Using float arithmetic, add a sine component to the x value of the pixel such that the pixel takes on an RGB value equal to that of a different pixel in the original image. Make the maximum shift in x equal to 16 pixels.
 - (b) In Premiere and other packages, only integer arithmetic is provided. Functions such as `sin` are redefined so as to take an `int` argument and return an `int`. The argument to the `sin` function must be in $0 \dots 1,024$, and the value of `sin` is in $-512 \dots 512$: `sin(0)` returns 0, `sin(256)` returns 512, `sin(512)` returns 0, `sin(768)` returns -512 and `sin(1,024)` returns 0. Rewrite your expression in part (a) using integer arithmetic.
 - (c) How could you change your answer to make the waving time-dependent?
9. How would you create the color-wheel image in Fig. 2.3? Write a small program to make such an image. *Hint:* Place R, G, and B at the corners of an equilateral triangle inside the circle. It is best to go over *all columns and rows* in the output image rather than simply going around the disk and trying to map results back to (x, y) pixel positions.

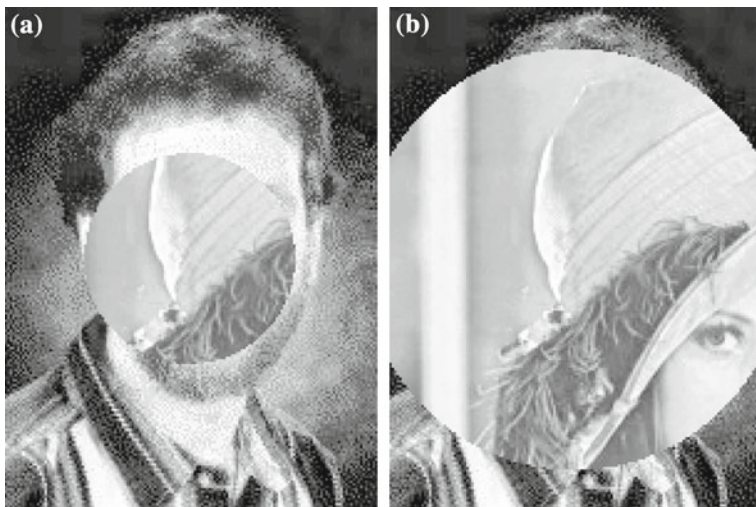


Fig. 2.22 Iris wipe: **a** iris is opening; **b** at a later moment

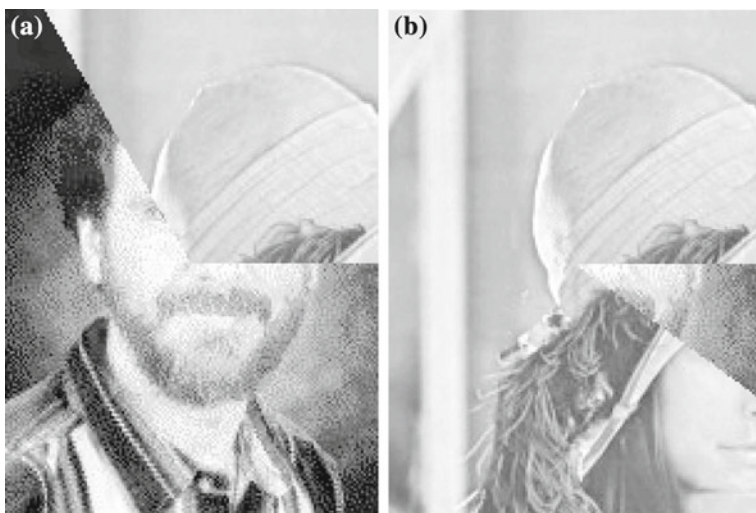


Fig. 2.23 Clock wipe: **a** clock hand is sweeping out; **b** at a later moment

10. As a longer exercise for learning existing software for manipulating images, video, and music, make a 1-minute digital video. By the end of this exercise, you should be familiar with PC- or Apple-based equipment and know how to use a video editor (e.g., Adobe Premiere), an image editor (especially Photoshop), some music notation program for producing MIDI, and perhaps digital-audio manipulation software such as Adobe Audition, as well as other multimedia software.



Fig. 2.24 Filter applied to video

- (a) Acquire (or find) at least three digital video files. You can either use a camcorder or download some from the net, or use the video setting on still-image camera, phone, etc. (or, for interesting legacy video, use video-capture through Premiere or an equivalent product to make your own, from an old analog Camcorder or VCR—this is challenging, and fun).
- (b) Try to upload one of the videos to YouTube. Check the time that is taken to upload the video, and discuss its relation with your video's quality and size. Is this time longer or shorter than the total playback time of the video?
- (c) Compose (or edit) a small MIDI file with music-manipulation software.
- (d) Create (or find) at least one WAV file (ok—could be MP3). You may either digitize your own or find some on the net, etc. You might like to edit this digital-audio file using software such as Audition, Audacity, etc.
- (e) Use Photoshop to create a title and an ending. This is not trivial; however, you cannot say you know about multimedia without having worked with Photoshop.

A useful feature to know in Photoshop is how to create an alpha channel:

- Use an image you like: a .JPG, say.
- Make the background some solid color, white, say.
- Make sure that you have chosen Image > Mode > RGB Color.
- Select that background area (you want it to remain opaque in Premiere): MagicWandTool
- Select > Save Selection > Channel=New; OK

- Window > ShowChannels; Double click the new channel and rename it Alpha; make its color (0,0,0)
- Save the file as a .PSD

If the alpha channel you created in Photoshop has a white background, you will need to choose ReverseKey in Premiere when you choose Transparency > Alpha.

- (f) Combine all of the above to produce a movie about 60 s long, including a title, some credits, some soundtracks, and at least three transitions. The plotline of your video should be interesting, to you!
- (g) Experiment with different compression methods; you are encouraged to use MPEG for your final product. We are very interested in seeing how the concepts in the textbook go over into the production of actual video. Adobe Premiere can use the DivX codec to generate movies, with the output movie actually playable on (that) machine; but wouldn't it be interesting to try various codecs?
- (h) The above constitutes a minimum statement of the exercise. You may be tempted to get very creative, and that is fine, but don't go overboard and take too much time away from the rest of your life!

References

1. A.C. Luther, *Authoring Interactive Multimedia* (The IBM Tools Series). AP Professional (1994)
2. D.E. Wolfram, in *Creating Multimedia Presentations* (QUE, Indianapolis, 1994)
3. R. Vetter, C. Ward, S. Shapiro, Using color and text in multimedia projections. *IEEE Multimed.* **2**(4), 46–54 (1995)
4. J. Liu, S.G. Rao, B. Li, H. Zhang, Opportunities and challenges of peer-to-peer internet video broadcast. *Proc. IEEE* **96**(1), 11–24 (2008)

Fundamentals of Multimedia

Li, Z.-N.; Drew, M.; Liu, J.

2014, XXIV, 727 p. 350 illus., 97 illus. in color.,

Hardcover

ISBN: 978-3-319-05289-2