Real-Time Special Effects for Digitized Movies

by

Derrick Yim

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Science and Engineering
at the Massachusetts Institute of Technology
May 1992
Copyright © Derrick Yim 1992

The author hereby grants to M.I.T. permission to reproduce
and to distribute copies of this thesis document in whole or in part.

Author

Department of Electrical Engineering and Computer Science
May 16,1992

Certified by

Glorianna Davenport
Thesis Supervisor

Accepted by

Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

Abstract

Personal computers have become powerful enough to record and playback digital video "on the desktop". As PC's become faster, it is becoming possible to not only playback digitized movies, but to manipulate them in various ways as they are playing. Such effects, and how they can reshape our perception of cinematic storytelling, will be explored in this thesis.

Table of Contents

1. **Introduction** 1
2. **Special Effects** 3
    2.1 Real-Time Editing 3
    2.2 Compositing 3
3. **Run-Time Compositing in QuickTime** 5
    3.1 A Typical User Scenario 5
    3.2 How it all Works 9
        3.2.1 Keying 9
        3.2.2 Storing (and using) the Key Information 12
            Conversion of Filtered out Pixels into Key Value 12
        3.2.3 Storing Key Information: Alternative Techniques 13
            Lookup Table 13
            Mask Track 13
4. **The Future** 15

List of Figures

1. Typical Foreground and Background Movie 6
2. Foreground Movie composited onto the Background Movie 6
3. HSV Tolerances Dialog Box 7
4. Typical CompositeMaker Window.. 8
5. HSV "Cone" 9
6. HSV Key Region 10
7. Mask Track 13

1. Introduction

Digital video for the masses is becoming a reality. Due in large part to the availability of relatively inexpensive, powerful computers with cheap mass storage, contemporary PC's posses the ability to manipulate a fairly large amount of data quickly enough to produce reasonable quality video "on the desktop". With certain forms of mass storage (namely CD-ROM), packages consisting of hundreds of megabytes of digital video data can be put into circulation for only a few dollars per copy. In addition, it is possible to obtain hard disk drives which can store a gigabyte of information for several thousand dollars.

The biggest (no pun intended) problem with digital video on small computers has always been that of bandwidth. Most systems simply do not have the processing power nor the bus architecture to handle the amount of data required for even NTSC resolution images in real time. Consider an image sequence at a resolution of 512 by 320 pixels, 16 bits per pixel, at 30 frames per second. This would work out to a data rate of almost 10 megabytes per second. In addition to the demand placed on the computer itself, a 10 second sequence would require 100 megabytes of secondary storage (read disk space) for archival purposes.

The way around the bandwidth problem is compression. By reducing the amount of data flowing from storage to CPU, it becomes possible to play digital video in real time. How close one can get to full-resolution, full motion video obviously depends on how much computing power one has to work with, but with some systems available today, with no specific video-compression hardware, it is possible to get enough digital video for use in a multimedia presentation.

Besides being able to show movies of real-world occurrences, digital video can also be used to "trick" the user into believing that a computer has more power than it really does. For instance, using digital video, a rather complex computer animation sequence (say of a fluid modeling with ray tracing), pre-rendered and pre-animated, can be shown in real time. Even though there might have been no way to compute the sequence on-the-fly on a particular machine, the sequence could nonetheless be made into a digital movie and played back.

An important point to observe about the aforementioned "silicon-based handwaving" scenario is that it is basically a tradeoff of processing power vs. storage space. That is to say, for a fairly long sequence, a program written to generate the animation in real-time (were it possible) would require much less memory space than storing the entire sequence itself. However, given a large amount of memory, it would be possible to get a similar net effect with a lot less processing power. This tradeoff, like any, has its advantages and its disadvantages. On the one hand, having such powerful hardware may simply be unfeasible, while having lots of storage may not be a problem. On the other hand, since the sequences are pre-generated, a great deal of limitation is placed on being able to interact in real-time with the sequence. A small amount of interactivity may be possible

if, say a set of sequences were generated corresponding to different parameter settings, but for a case where there are a fairly large number of such settings, the amount of storage space required would grow very quickly.

The plusses and minuses of such a tradeoff have to be weighed in terms of the situation. For instance, in the example given above, where a huge amount of processing power is required, the tradeoff would be valid. On the other hand, if there was a case where the amount of processing power required was small, whereas the amount of storage required was very large, it would make more sense to compute the sequence in real time; One would probably not want to pre-generate a series of long movies to show a bunch of relatively simple cellular automatons being executed, for example, with each movie corresponding to every possible set of initial conditions.

This thesis is concerns such a tradeoff issue, in the area of digitized video databases. For the most part, it doesn't make much sense to render a video sequence on a personal computer which would be much easier to just go out and shoot. Nowadays, a lot of special effects in motion pictures are computer rendered, but those are of events which would be either impossible or too costly to set up in real life. Also, they are usually rendered on either very powerful workstations or even on supercomputers, and definitely not in real time. On the other hand, it is possible to do some simple effects on pre-recorded digital video sequences, on a PC, in real time. These effects could not only be used to add to the content of a digital movie, but also to relate them to one another in such a way as to balance out the performance/storage tradeoff.

2. Special Effects

2.1 Real-Time Editing

One such example of real time computation on a digital movie database would be real-time editing. Suppose you had a large database consisting of digital movies. And suppose that a lot of those movie sequences had overlapping scenes. That is, same scenes that were in different movies, maybe to the point where the only difference between a lot of those movies was the order of the scenes, with different parts of the scene being used in a particular movie, and with only a few other scenes exclusive to a particular movie. In such a case, it would make a lot of sense to store movies not as entire movies, but as a database of scenes. Then one could use another program, (such as a script), to select and string together parts of scenes from the database to construct full movies on the fly. These ideas are currently being developed in several interactive movie projects here in the Interactive Cinema Group at the Media Lab.

Such a method would be particularly useful in a multimedia periodical, where parts of the database could be updated from time to time to keep up with current events in any given subject, while other scenes in the database, stock footage of geographical and other basic static concepts, remain unchanged. As the footage (as well as other aspects of the presentation) were updated, so too would the edit lists, and one could come up with new movies without having to unnecessarily alter the database or waste storage space with redundant data.

2.2 Compositing

Another example of constructing movies as they are viewed is digital compositing. Take a project such as the "Elastic Map of Boston", where guides are used to take the user around a map of Boston. What the guides have to say may apply to more than one area or landmark on the map. Placing the guides into specific scenes represented on the map can be done in several ways. One method would be to actually film the guide on location for all of the different places that she appears on the map presentation. The problem with this method is that to do any filming, the actor for the guide must be available at the time, and only those camera crews who have the actor with them (discounting any genetically identical siblings) could be involved in a shoot at one given moment. In addition, for a long project spanning several months, or even years, it would be difficult to maintain continuity, especially if scenes were shot out of order,

A different approach, which might solve the problems of on location shooting, would be to shoot the guide against a bluescreen, and then to composite the guide onto the location footage (via a post production switcher, for instance) to produce composited movies. This method gives a bit more flexibility in the actual making of the footage, but a movie must be made of every combination and permutation of guide and location, and

each of these movies must be stored in the database. Also, if one did not want the guide to be present in a particular scene, such a scene would have to be put in the database as well. Finally, if some permutation of guide and location was not covered which the user requested at some point in the future, the user would simply be out of luck.

The way to get around these problems is to composite the guide into a scene in *run-time.* That is, store the footage of the guides and of the scenes separately, and when the two are requested to appear in a movie, do all of the compositing as the new movie is shown. In such a case, every different combination of guide/location would not have to be thought out and constructed beforehand, and a much more versatile database could be put together given the same amount of storage space.

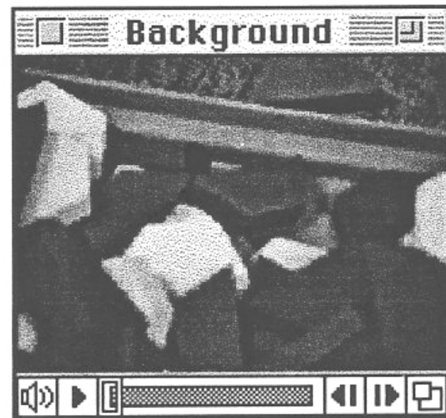
3. Run-Time Compositing in QuickTime

Using a Macintosh running with QuickTime (the new multimedia system software extension from Apple Computer), I was able to implement run-time compositing for that particular platform. The next several sections will discuss many of the issues involved in such a process.

In the process of writing my thesis, I developed two main programs, which I call *CompositeMaker*, and *Comp*. CompositeMaker is utilized in pre-processing one movie to allow it to be played over another movie in the program Comp during run-time. I will first describe these utilities from a user perspective, then describe the details of how they work.

3.1 A Typical User Scenario

Playing back composited QuickTime movies involves a little bit of pre-processing (i.e. processing not done before the movie is to be used). Namely, some processing has to be done to the *foreground* movie in order for it to be "comped" during run time over another *background* movie. No pre-processing need be done on the background movie. In the case of the guides example, the footage of the guide would be the foreground movie (see figure 1). From the user's point of view, this pre-processing involves choosing a parameter such that parts of the foreground movie becomes *transparent* while other parts of the movie remain *visible.* Those areas of the movie set to be transparent do not appear in the final movie when the foreground is composited over the background movie, allowing parts of the background movie to show through. From the computer's point of view, pre-processing identifies, at the time the movie is to be shown, what parts of the movie the user has selected to be transparent (and what parts remain visible), and also makes the task of compositing during run time a little easier for the CPU, allowing for bigger and faster movies.



Foreground Movie

Background Movie

Figure 1: Typical Foreground and Background movie

The pre-processing for a foreground movie is done in a program called *CompositeMaker*. Using *CompositeMaker*, the user opens a movie which she wants to preprocess. Since the window in which her movie has the standard movie player controls, she can play the movie, and make sure that it is the movie she wants to pre-process.

Next, using the mouse, she clicks on the region of the movie she wants to *key* out. Keying out an area of the movie means selecting a property of a certain region of the movie such that other areas of the movie sharing a particular characteristic of that region (in our case, Hue, Saturation and/or Value) will not appear in the final composited movie, and the background movie will show through. Figure 2 more clearly illustrates this concept.

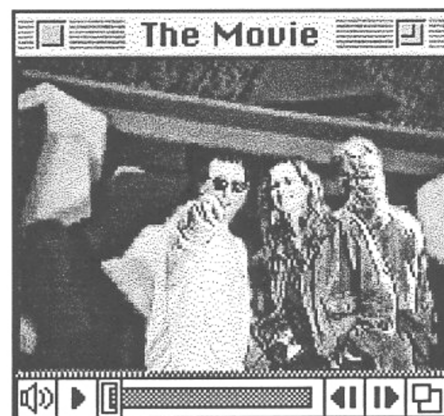


Figure 2: Foreground movie composited onto the background movie

On (what was) the empty region to the right of the movie in the *CompositeMaker* window, an image appears wherein the areas of the movie which will be transparent appear to be black. By examining this image, the user can determine whether or not her selection of the key value will work. In the case where the image appears unsatisfactory, the user can do one of a couple of things. First, she can try picking a different key value (i.e. clicking somewhere else on the movie with a different value). Or, she can try adjusting the tolerances *for Hue, Saturation*, and *Value*, using the provided dialog box (see Figure 3). The values in this dialog box define a range over which if any of the pixels in the movie fall in, it is considered to be keyed out. Adjusting the values in this dialog box in various ways also determines whether the movie would be keyed by hue, saturation, or value. For instance, to key by hue, the user would make the hue tolerance value fairly small (say under 10,000 over a total possible range of 65535 in the case of *CompositeMaker*), while keeping the saturation and value tolerances fairly large (around 20,000). An analogous range of values could be set to key for value or saturation.

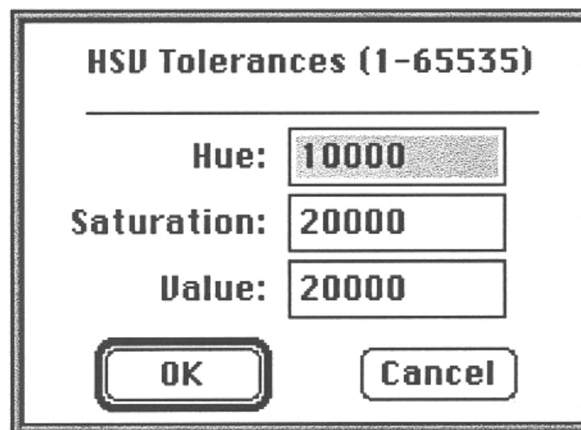


Figure 3: HSV Tolerances Dialog Box

Given that all goes well, the *CompositeMaker* window should look something like figure 4. After a key value has been selected, the user can view parts of the movie using the standard movie controller. By clicking on the right hand image of the window, the user can see what each frame would look like, with the key color filtered out. This could be done to ensure that what appears fine for some parts of the movie don't degrade as the movie progresses. If it turns out that other parts of the movie are incompatible with previous settings, the entire key/tolerance selection process can be repeated. Once the image is to the user's satisfaction, she can then make the final foreground movie, by selecting "Save Movie" from the "File" menu. The progress can be monitored by watching the movie controller. The final images of the movie being made can also be watched on the right part of the window. The user can abort the movie making procedure during this time by holding down the mouse button. Notice that all of the regions falling within the key value tolerances have been turned into the specific key value.

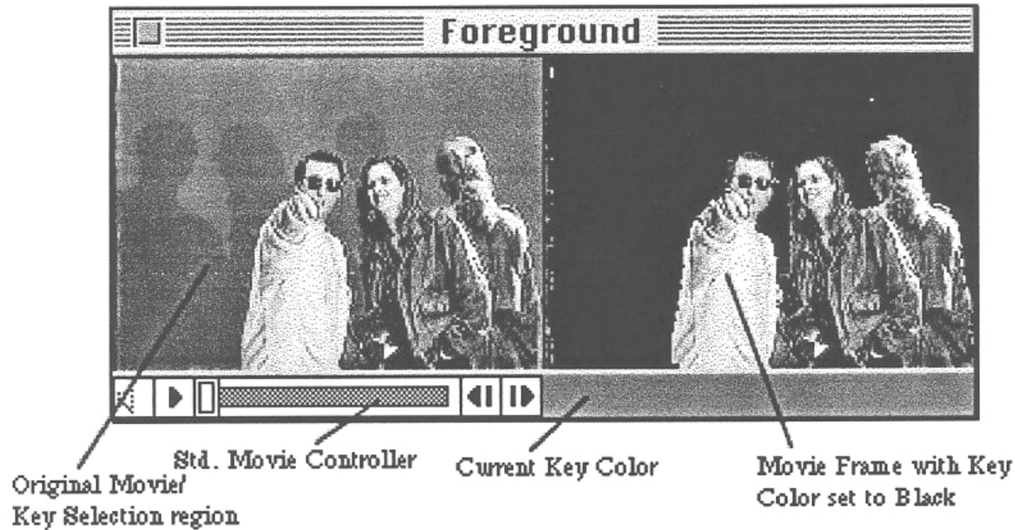


Figure 4: Typical CompositeMaker window

Now, once the foreground movie has been made, it can be played back and viewed with a background movie in two different ways. One is to use the "Comp" program, which is a simple stand alone application which asks for two separate movies and plays them back, the first movie being the foreground movie, the second being the background movie. Note that the first movie has to be of a movie generated by the *CompositeMaker* program.

However, this program was used as an early development tool, and is not very useful in general, as the movies cannot be viewed in any form of context. A more useful way to show movies being composited in real time is to use the Comp XCMD, which can be called within a HyperCard (2.0 or later only) stack. The user has to merely install the XCMD like she would any other, and then call it, using the following syntax:

Comp "<foreground movie>, <background movie>, <point>"

where <point> specifies the movie window location. As in the case of the standalone application, the first argument must contain the name of a pre-processed movie. Full file paths are required, unless the movies are in one of the folders that either HyperCard or the stack knows about (e.g. in the same folder as the stack itself). This is the same convention as used by most other XCMD's where a filename must be specified.

3.2. How it all Works

3.2.1 Keying

A fundamental design issue involved in making real time compositing happen was that of being able to specify how to key out parts of a foreground movie. For most applications, chroma keying (keying by color) is an adequate method. In the analog world, such a process is relatively trivial, as composite video signals are generally made up of a chroma component and a luminance component (in NTSC the chroma is constructed as a combination of two color difference signals, while luminance is carried on its own signal). To chroma-key, one simply needs to construct a band-pass filter which takes out that part of the signal that falls in the key region.

In the digital domain, such a simple process cannot be as easily implemented, because most digital images are represented as a combination of red, green and blue components. If the color to be keyed out happens to be one of these three colors, it is easy to recognize (simply find the difference between the particular color component and the other components). But in the general case, the only "correct" way to do a true chroma key would be to map the color coordinate system in a way such that the color to be keyed represented a primary value, and other two new components lay in a complementary position relative to this color in the colorspace.

Because I wanted to make my system as general as possible, and because the RGB color system is more a result of making the underlying display hardware easier to interface with than a result of keeping an intuitive system of color representation, I chose to use the Hue Saturation Value (HSV) model for color. In the HSV model, hue is more or less an angle on a circle representing various chromatic values, saturation is how much of that color is present in the pixel, and value is the pixel's "distance" away from black (see figure 5).

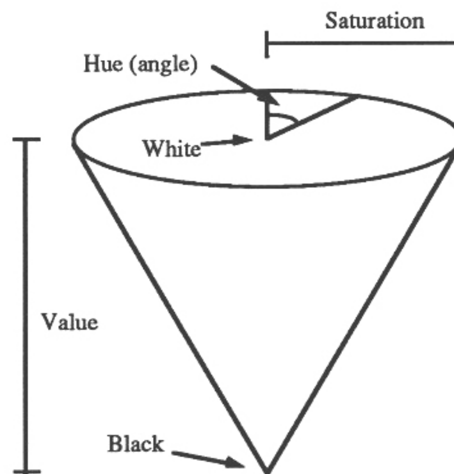


Figure 5: HSV "Cone"

To do a chroma key of HSV space, one merely has to pick an angle corresponding to a range of color to be keyed, and pass all pixels that fall outside that value. In addition, the saturation and value components can be restricted as well. Once the tolerances are specified, the key region looks something like figure 6.

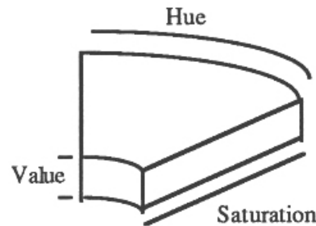


Figure 6: HSV Key Region

To convert from RGB to HSV space, I used an algorithm from Foley & Van Damme, which I converted to the following C code (all values are manipulated in fixed point for performance reasons):

```
Void RGBToHSV(RGBColor *RGB, HSVColor *HSV)
{
        unsigned long max, min;
        unsigned long rc, gc, bc;

        unsigned long r, g, b;
        unsigned long h;

        /* Setup fixed point implementation for r, g, and b */

        r  =  (unsigned long) RGB->red<-16;
        g  =  (unsigned long) RGB->green<-16;
        b  =  (unsigned long) RGB->blue<-16;

        /* Get Max & Min of RGB */

        if (r > g)
        {
                max = r;
                min = g;
        }
        else
        {
                max = g;
```

```
			min = r;
		}
		if (b > max)
			max = b;
		else if (b < min)
			min = b;

		/* value is max */

		HSV-> value = max >> 16;

		/* do saturation */

		if (max)
			HSV-> saturation = ((max - min) * (UNIT / max)) >> 16;
		else
			HSV-> saturation = 0;

		/* now do hue (which is kind of hard) */

		if (HSV-> saturation)
		{
			rc = (max - r) * (UNIT / (max - min);  /* distance of color from red */
			rg = (max - g) * (UNIT / (max - min);  /* distance of color from green
*/
			rb = (max - b) * (UNIT / (max - min);  /* distance of color from blue */

			if (max == r)
				h = (bc - gc) / ONE_SIXTH;
			else if (max == g)
				h = UNIT_3 + (rc - bc) / ONE_SIXTH;
			else
				h = UNIT_6 + (gc - rc) / ONE_SIXTH;

			/* convert hue back to int */

			HSV-> hue = h >> 16;
		}
		else
			HSV-> hue = 0;
			/* actually, it's undefined, but 0 seems to be the best value */
}
```

Once an HSV value is obtained from an RGB value, and tolerances are specified, it is a relatively simple matter to determine whether or not a pixel's value falls in the key region, and when the new movie is made, whether that particular pixel can be included or excluded.

3.2.2 Storing (and using) the Key Information

Now that a key region has been defined and a method is available for determining what is to be keyed out or not, that information must be stored in the new movie, and a way to play that movie composited over a background movie is required. As mentioned before, the point of pre-processing the foreground movie is to:

a) Store the key information.
b) Make the movie easier to play back for the computer.

The main difficulty for the computer with real-time compositing is determining which pixels should be passed. To complicate matters, since the algorithm for converting from RGB to HSV is somewhat involved, there is not enough time to do such a conversion for every pixel while the movies are playing.

Conversion of Filtered out Pixels into Key Value

A simple way to satisfy both the storage and simplification of processing requirements in the preprocessed movie (and the method I chose for full implementation) is to change every pixel which falls in the key region into a single RGB value during pre-processing. This way, that particular RGB value can be stored in a header to the movie (in my case it was stored in the User Data region provided by QuickTime, which is also used to identify the movie as being a pre-processed movie), and it is relatively easy for the computer to target the particular RGB value as the null value. At playback, both the background and the foreground movies are played onto off-screen buffers. As they are copied onscreen, the foreground movie is copied over the background movie, less pixels having the null RGB value, and a composited movie sequence is created during playback. Choosing the particular null RGB value was a relatively simple matter. Since it is guaranteed, by definition, that no other areas in the movie will have pixels with the original key value, I chose that value as the null value. Converting all of the pixels falling into the defined key region into the original key value has a further advantage: because fairly large areas of the images are a single RGB value, the movies compress very well under any compression schemes that use any kind of run-length encoding.

3.2.3 Storing Key Information: Alternative Techniques

In addition to converting filtered key values into a single RGB value, there are a couple of alternative techniques which were studied.

Lookup Table

Another method for pre-processing movies could be the use of lookup tables. Since most of the movies I was using were represented by 16 bits per pixel, a lookup table showing whether or not a pixel should be passed or keyed out, for every possible color value, would have 65,536 entries, meaning that the whole table could be as little as 8,192 kilobytes. While a movie was playing the computer could utilize the lookup table to determine whether or not a pixel from the foreground movie should be copied over the background movie. The only real advantage to this method over the previous method would be that it would not be as vulnerable to having essential key data (i.e. the converted pixels) corrupted by the quantization associated with some compression methods. However, one would not gain the extra compression associated with the advantages of run-length encoding, unless a combination of the two methods was utilized. One would, however, have to be careful that the information stored in the lookup table corresponds to the image after compression, in a lossy compression scheme.

Mask Track

Perhaps the most interesting method of pre-processing a movie is the use of a separate mask track in the movie, which would dynamically change as the movies were played, and provide a mask for copying only the 'valid" parts of the foreground movie over the background movie (see figure 7). In addition to providing the basic compositing data, a mask track could also be used as a form of alpha channel, allowing, in the simplest case, anti-aliasing to smooth out the edges between (say) the guide and the background. But other more interesting effects could be applied, such as a guide being translucent, or even fading in and out of a scene.

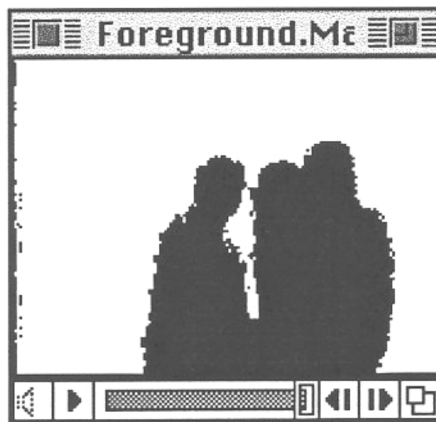


Figure 7: Mask Track

The mask track would be (assuming it was compressed in lossless form, as it should be) impervious to the effects of lossy compression, and could be counted on to accurately hold compositing data regardless of the loss in the original image. The mask track need not be very large, either. The simplest case of a one bit per pixel mask track would obviously be very small compared to a 16 bits per pixel video track, and even with a mask track that was only 4 bits deep, one would have an alpha channel capable of representing 16 levels of translucency.

The drawback to the mask track is that of all of the methods discussed, it would be the most computationally taxing, as yet a third movie must be played along with the background and foreground movie. The appendix includes source code for a special version of *CompositeMaker* which creates a movie with both a video and mask track.

4. The Future

What can be done in real time is limited by processing power. With a higher end Macintosh running QuickTime, many simple special effects can be done on the fly. While only simple effects are possible for the moment, more complicated ADO effects, such as effects where movies are mapped onto metamorphasizing spheres and the like, may be possible in the very near future if efficient enough algorithms and powerful enough hardware becomes available. I believe that many such interesting effects could be calculated while a movie (or movies) is playing to provide variety into multimedia presentations, and without taking up unnecessary storage space. In addition, the quality of special effects can also be improved. In the case of compositing, for instance, anti-aliasing and a general improvement of image quality in the composite could be accomplished through alpha tracks and more intricate filtering algorithms.

Such technologies, especially in the case of run-time compositing, will alter the ways we think about cinematic storytelling, as it will be possible to separate the character in a movie from the location, and decisions about which location to put the character could be made as the movie is being viewed, providing a new degree of interactivity.

As an example, I have included the source code for a simple player which will play two movies in succession with some interesting fade/blend effects at the endpoints. This is a fairly primitive example, but again, many things are possible, and I hope that what I have discussed in this paper will lead to lots of interesting digital video database based multimedia applications.

List of References

S. Applin, D. Blackketter, E. Chen, J. Hanan, E. Hoffert, G. Miller, E. Patterson, D. Yim, "The Virtual Museum -- Interactive 3D Navigation of a Multimedia Database", Proc. of Imagina conf. 1992, Monte Carlo, Fr. Dec. 1991.

J.F. Blinn, "Raster Graphics", Reprinted from *Tutorial: Computer Graphics* by Kellogg S. Booth, Feb. 1979. © 1979 by the Institute of Electrical and Electronics Engineers, Inc.

H.P. Brondmo, G. Davenport, "Creating and Viewing the Elastic Charles -- A Hypermedia Journal", MIT Media Laboratory, July 1989.

J.D. Foley, A. Van Dam, Fundamentals of Interactive Computer Graphics, © 1982, Addison-Wesley Publishing Company, Inc.

A.V. Lord, "NTSC Colour System", Reprinted from *The Focal Encyclopedia of Film and Television Techniques*, © 1969 Focal Press Ltd. (by permission of Hastings House Publishers, Inc., New York, 10016, USA distributors).

J. Watlington. "Synthetic Movies", S.M. Thesis, Massachusetts Institute of Technology, September, 1989.