The Making of *Pencil Test*

GALYN SUSMAN

ABSTRACT

The Advanced Technology Group at Apple Computer, Inc. recently produced an animation entitled Pencil Test, created entirely with Macintosh II and Macintosh Plus equipment. This paper discusses the challenges and obstacles faced, and the set of solutions chosen in producing an animation on a platform that had not previously been utilized by the animation industry. Animation is both an entertaining and effective communication tool. The conclusions set forth in this paper present some of the issues that need to be addressed to facilitate easier creation of animation on personal computers.

Keywords: Macintosh, personal computers, 3D character animation

1. INTRODUCTION

In the fall of 1987 Apple's Advanced Technology Group decided that it was time to create a production quality animation for SIGGRAPH. The goal was to produce a piece of 3D character animation with high quality rendering. The challenge was to create this piece entirely on Apple computers, specifically on the Macintosh II. In just six months we formed a group that designed, produced, and scored our first animation, *Pencil Test*, which debuted at SIGGRAPH in July 1988.

The 3D animation problem can conveniently be divided into six steps: design, modeling, animation, rendering, sound and the final transfer to some medium.

Design is the creation of a story and script, storyboards (pictorial representations of changes in action), and animatics (video recordings of the storyboard that show the timing of the transitions).

Modeling is the creation of three-dimensional models for every object and character shown in the storyboards.

These models, along with the animatics, are used to create the *animation*, where all of the objects are placed, scaled, and rotated to their actual positions within a scene. A scene consists of all actions that take place from one camera position (or sequence of camera positions as in a pan, zoom, or fly-by). As soon as the camera alters its position, orientation, or path of motion, there is a change of scene. All camera and object movement within a scene is achieved by defining keyframes(set positions, scales and orientations of an object throughout a scene). A keyframe animation system interpolates between key-frames to produce all of the intermediate frames.

Rendering takes these frames along with the models and generates two-dimensional images from the three-dimensional mathematical descriptions. In computer animation this generally involves modeling a natural environment where there are lights and a camera, and objects have color, material properties and even textures. A software rendering package will then take this information and, depending on the algorithm, generate images as simple as cartoon frames or as rich as photographs.

These images are then *transferred* to some medium, usually film or video tape. In some cases this tape needs additional editing. For example, for special effects like fades or for overlayed credits, this tape must be taken to an editing studio where these effects can be achieved.

The *sound* track is usually designed while the graphics are being produced. Generally, a professional recording studio is used to record and lay the sound track to tape.

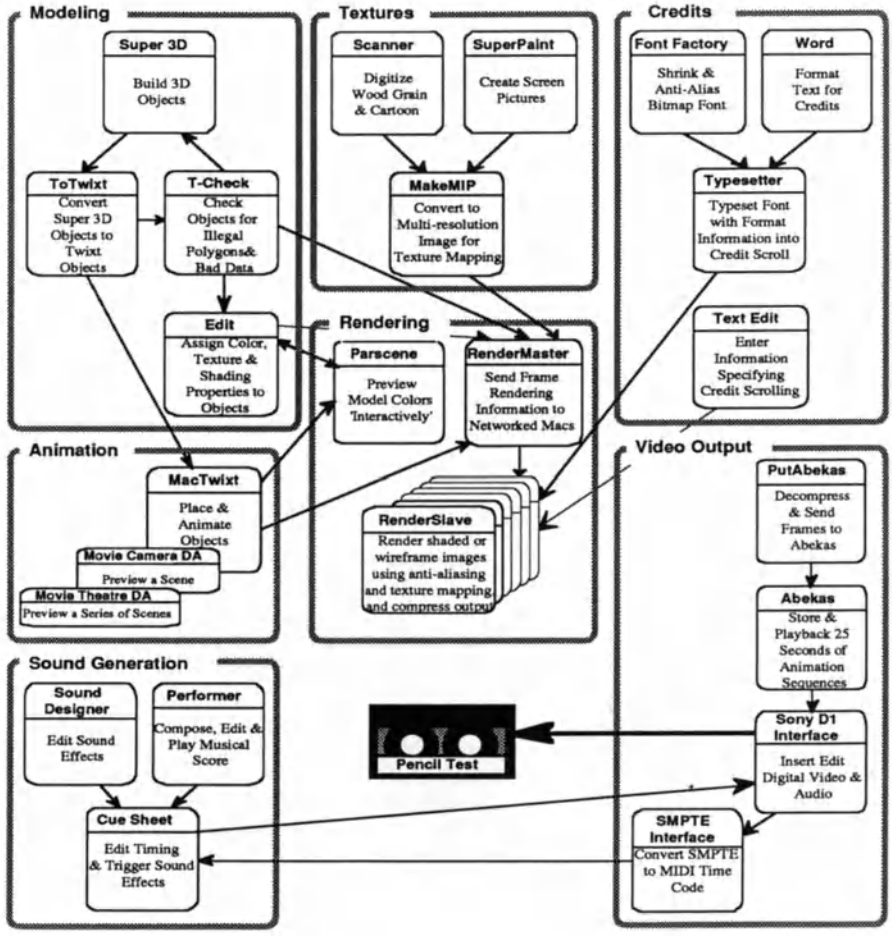

Fig. 1. Software flow diagram

2. OVERVIEW

The high-end systems that are typically used for animation projects have large, integrated software packages to create animations. There is no such software for the Macintosh. Instead we used existing programs to solve parts of the problem and integrated them with custom software (Fig . 1). We chose to do all of the design by hand. For modeling we chose *Super3D* from Silicon Beach Software. We convinced the author of *Twixt*, a public domain animation package from Ohio State University, to port his code to the Macintosh for us (*MacTwixt*). The majority of the rendering software was written in-house. We used a scanner and *SuperPaint* from Silicon Beach Software to create some of the textures. The credits were generated during the rendering process and we chose *Microsoft Word* to format them. A big breakthrough occurred when we realized that we could create the entire sound track on the Macintosh, something that high-end systems generally do not address. In producing the sound, we used one package for editing sound effects (*Sound Designer* by Digidesign), another for composing the score (*Professional Performer* by Mark of the Unicorn), and a third for cuing the sound to video tape (*Cue Sheet* by Digidesign). Finally, there was no software or hardware in place to help us transfer our piece to video tape; we had to provide these ourselves. All of the components shown in Fig. 1 that are not discussed in this paragraph were written by us specifically for this project.

3. DESIGN

Though the primary motivation for this work was technical, we were not without artistic goals. We wanted our piece to tell a story; to be funny and endearing. The design we developed had to accomplish this while allowing for the limitations of the software and the hardware. We were working with beta, alpha, and pre-alpha software, and many of the tools were relatively primitive by Apple standards. To achieve simplicity, we decided that our story would have only one primary character. The rest of the objects would, for the most part, remain stationary and have few or no movable joints.

With these criteria in mind we wrote a script that we thought would be reasonable to work with and still satisfied our artistic goals. Unfortunately, there was no existing software for the Macintosh to aid in the process of storyboarding and creating animatics[1], so we contracted an artist to do this work for us. When we received our first set of storyboards and started to work with the character, we realized that as unadorned as our initial design had been, it wasn't going to be simple enough. The main character had been visualized as an articulate, curvy pencil that bent and twisted in all directions (Fig. 2a). All we had to work with was a polygonal modeler and an animation package that did not allow us to animate control points on a flexible object. If we wanted to model the pencil with splines we were going to have to write an animation package that let us animate splines. We reconsidered the motions for the pencil and came up with the segmented polygonal design as it exists today (Fig. 2b).

Fig. 2. Character design: (a) original curvy design, (b) final segmented design.

1 *VideoWorks* does not provide the timing control needed to stage animatics from hand drawn storyboards, and it is much too complicated to use for such a basic process. Upon reflection, *HyperCard* probably would have sufficed if we had scanned the storyboards and then written a script that allowed timed playback and manipulation.

In all we devoted about two months to design, including writing the script, creating storyboards, shooting the animatic, and designing the character and objects. At this point we were ready to begin modeling.

4. MODELING

Modeling was the one phase of this project where we solicited involvement from as many people as possible (with the hopes of encouraging further participation). Any intersted party could model one of the many objects that appeared on the desk (Fig. 3). Because the majority of these people had no previous modeling experience, we needed a modeler with a simple user interface and straight-forward tools like revolution and extrusion. In addition, the modeler had to have a published data format or at least export the model data in ASCII so that we could import our models into the animation and rendering software. Based on these criteria we chose *Super3D* by Silicon Beach Software.

Though the version of *Super3D* that we worked with was a beta version, the software was relatively reliable and easy to work with. It maintained the click and drag interface of the Macintosh in the 3D environment by having a third scroll bar that controlled motion in the z dimension. The scroll bars controlled rotation, while clicking on pan and zoom icons controlled the translation and scale. There were, however, a few setbacks working with this package. One of the greatest difficulties was the lack of visual feedback. Because the only solid visual feedback was a flat shading mechanism, approximated for an eight bit frame buffer, it was very difficult to tell if an object was closed or inside out. Smooth shading and data consistency checking should be included in this kind of modeling package to guarantee a model's data integrity. Data anomalies did appear and greatly hindered us later in the project.

Super3D exported its data in a very intelligible ASCII format that we could easily parse and convert. Our first (but certainly not our last) data conversion program, *ToTwixt*, was written to take this output and convert it into a format that the animation software could use. The objects could then be placed within the 3D environment.

5. ANIMATION

From the start it was never really clear what the quality of the rendering was going to be for the final piece. However, we did know that limitations in rendering time would not allow for photo-realistic rendering. This increased the importance of concentrating on the quality of the animation. If the story wasn't successfully told by the expressiveness of the main character and the information carried in the sound it would not be understood. We needed an animation package that would allow us to squash and stretch our objects and define transformational relationships between objects. It also had to create good curved interpolations for the paths of motion.

It was fortunate that *MacTwixt* satisfied these criteria, for it was the only 3D animation package available to us (by coercion) on the Macintosh. Working with *MacTwixt* was a new experience for us point-and-click Macintosh users. It had a command line interface and lacked the ability to move objects relative to their current position (i.e. move object x one unit to the right). In addition, the software was not able to do real-time playback of animation. Drawing one 1/4 screen wireframe image on the screen took anywhere from 2 to 15 seconds. This made it impossible to view our motion.

As a result, we wrote a tool (*MovieCamera*) that we could run with *MacTwixt* to capture all of the drawing commands and play them back in something approximating real-time (timed to the refresh rate of the monitor). In this way we could see clips of the basic wireframe animation and study our motion. However, that was not quite good enough. We then wanted to see a series of scenes strung together. *MovieCamera* was modified to save the captured bitmaps in a file and another tool was written (*MovieTheater*) that could take a collection of these files and string them together for playback. By playing back the scenes in sequence we were able to get an idea of the overall timing.

After scenes had been animated, *MacTwixt* would write out scene files, which were a collection of transformation matrices for every object at every frame. Depending on the scene, this process could take anywhere from one to twenty-four hours, an unexpected delay.[2] We minimized this process by editing these scene files by hand for simple changes, and by writing out very small subsections of scenes and cutting them into the larger files for more complex changes. These scene files were the frame descriptions used by the renderer.

6. RENDERING

Our plan was always to get the best quality rendering possible in the allotted development time (approximately three months). At the very least we had to have smooth shading (Fig. 4), and we aimed for Phong shading which would produce specular highlights (Fig. 5). We also needed texture mapping because much of the information relating to the story was to be told by texture maps appearing on the Macintosh screen (Fig. 6). Anti-aliasing was also necessary or the quality would not suffice for the SIGGRAPH Film and Video Show. Though it was not clear how time-consuming the rendering process was going to be, we knew that we would need a distributed system to allow for time to render several versions of the film. With this in mind we initiated the renderer project and the distributed systems project.

6.1. Preparation

Before we could render any of our objects we needed to modify them to include various rendering attributes. We wrote a program called *Edit* that allowed us to assign colors, smooth edges, material properties and surface textures to the objects.

Some early rendering tests revealed topological inconsistencies in our model data. The anomalies fell into three categories: zero area polygons and multiple neighbor polygons causing cracks in the shaded models; open solids (where one or more polygons of the model were missing) resulting in holes in the shaded models; and reversed polygon normals, appearing as holes or causing the entire object to appear inside-out. To fix some of these problems we wrote *TCheck*, a program that would read in the objects, remove zero area polygons, and flag all of the other bad data. All models that failed to pass *TCheck* were rebuilt in *Super3D*. This cycle was extraordinarily time consuming. Each object was broken into its subparts and all the subparts were run through *TCheck*. Any part found with bad data was rebuilt and rechecked until all of the parts were renderable. Then the object could be reassembled, and read into *Edit* to be recolored and resmoothed.

An early addition to our rendering environment was a program to preview rendered objects. *Parscene* began as a module that parsed scene description files from *MacTwixt* and object description files from *Edit*. It then converted them into a form usable by the renderer. At this point it was

---

2 This was due to a bug in the ported version of *Twixt* and not an inherent limitation in the *MacTwixt* application.

Figure 3. Sample modeled objects

Figure 4. An example of smooth shading

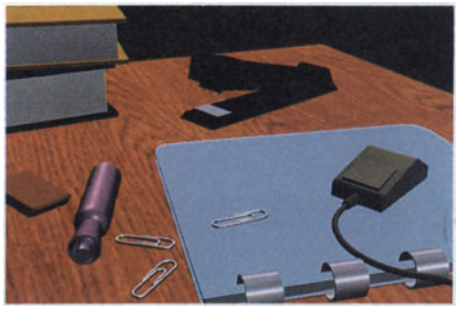

Figure 5. Objects with specular highlights

Figure 6. A texture map showing an active Macintosh screen

expanded to be an interactive front-end that allowed us to read in a scene and display the frames on a 24-bit monitor, using an experimental 24-bit version of *Color QuickDraw* (the Macintosh drawing package) and a prototype 24-bit video card. Once this was accomplished we added an interface that allowed us to interactively move the camera, add and manipulate lights, and experiment with different rendering methods. We could also remove one degree of freedom and use the mouse to move the camera and lights.

6.2. Renderer

All of the rendering code was written in-house. The renderer we implemented provided flat shading, smooth shading, and Phong shading. Multiple (four) colored light sources were implemented; up to three sources of white light were used in *Pencil Test*. A 24-bit Z-buffer was used to eliminate hidden surfaces (16-bit was not able to resolve front-most polygons in the ranges with which we were working).

Texture Mapping allowed planar mapping of textures onto polygons.[3] Pyramidal parametrics, (Williams 1983) often called MipMapping, was the fundamental technology used here. MipMapping generates a set of prefiltered source images of various resolutions and then maps the image closest in size to the polygon, thus minimizing aliasing effects and assuring better continuity of the textures between successive frames. A utility program called *MakeMip* was written to create

3 We did not implement mapping onto arbitrary curves because we did not have any objects constructed with patches.

multiple resolution MipMaps from texture files (RLE or PICT format.) These input files could be either scanned data (the woodgrain for the desk) or painted images created with *SuperPaint* (screen shots for the Macintosh).
Anti-Aliasing was achieved by rendering each frame at higher than target resolution (4,9,or 16 times) and then decimating the image with a digital filter (Lanczos Windowed Sinc Function) to the target resolution.[4] As a result, even machines with eight megabytes of memory could not compute a whole frame with a large number of objects and textures at nine times the resolution. We therefore modified our renderer to render smaller bands of the image. This became very useful when setting up network rendering because it allowed us to render on more memory limited machines. We also implemented anti-aliased wireframe output for the production of flicker-free wire-frame test.[5]

6.3. Optimizations

Because rendering is so time-consuming we needed to take any shortcuts available to speed up rendering time. The first of these shortcuts was to convert the computations of the renderer from floating point to fixed point. However, peculiar results like tearing textures indicated that we were exceeding the numeric range of our fixed point numbers. As a result, the radix location for any given variable depended on the number range for that variable, meaning that we had to keep track of the radix location for every variable. However, we found that we achieved an order of magnitude performance improvement using fixed point over the hardware floating point on the Macintosh.

An additional performance improvement was added to the Phong shading algorithm: Gouraud shading was automatically used instead of Phong shading if a quick test indicated negligible specular reflection in a given polygon.

6.4. Extras

The credits were done by formatting a high resolution bitmap and then decimating it to the target size (as was done with our anti-aliased frames.) The text was laid into a long scroll file whose scan-lines were indexed by frame number. Once the section of credits was determined it was composited onto the background frame. This compositing (blending, not overwriting) meant that we could lay the credits onto any colored background.

A few special effects were achieved. To avoid having to do post-production, we implemented our own fade to black. The only trick here was realizing that the Y value in YUV is not a pure luminance value. Decreasing this Y value does not bring all color to black. As a result, all of the components had to be scaled.

We also implemented a cheap form of motion blur for one scene involving a fast camera pan that produced rough motion. The difficulty here was that we were not set up to render *fields* (our animation files were keyed to the *frame* rate and our renderer shaded full *frame* images.) Our solution was to first reduce the vertical resolution to that appropriate for fields, and then estimate the motion difference between frames and approximate the blur of the horizontal pan with a horizontal smear.

---

4 This is clearly not the fastest method of doing anti-aliasing. However, it was the most quickly implemented, given the limited amount of time that we had remaining to develop the piece. Certainly, one of the first changes to our system will be to implement a faster method of anti-aliasing.

5 There are several additions we would have made to the renderer if we had had more time. The obvious is an enhanced anti-aliasing algorithm. Another would have been a cheap implementation of shadows. Without shadows our objects sometimes appeared as though they were floating in a space slightly above the desk.

6.5. Distributed System

Fortunately, we anticipated that we were going to require a distributed rendering system to be able to compute the frames in a realistic amount of time. This was later confirmed when timings showed that it took approximately thirty minutes to render a frame (96 days to render the entire film on one Macintosh II). However, when the project began there was no renderer to work with, so the distributed computational environment had to be implemented as an independent module from the renderer. The environment set up was a master/slave system where the master handled the data and file management, and the slave controlled the actual rendering. The master was a modified version of *Parscene* (see section 6.1.) that parsed the scene file and broke it into individual frames. The slave was a generic module that attached to any program to handle I/O, data transfer and program initiation. Because this slave was generic we were able to continually modify the renderer and simply attach new versions to the slave. This process was so robust that we could even substitute renderers while the slave programs were running.[6] The slave program communicated with the master, telling the master its available resources, such as memory. The master then selected an appropriate job (e.g. a frame renderable within the slave's memory constraints) and passed back the name of the current rendering program, the frame file to render as well as the location of this file and the output file. The slave would then fetch objects and textures from a file server and initiate the attached renderer. The output frame was passed back to the server where it was stored until recording.

In the end we were able to have 25 to 30 Macintosh II's running at any given time. There were approximately 5000 frames to be rendered, each frame taking anywhere from 20 to 40 minutes to compute (depending on the amount of texture mapping and the number of reflective surfaces.) One complete turn of the animation could be completed in just over three days.

7. SOUND

The sound track to *Pencil Test* needed to be much more than pretty background music. It was very important that the sound effects be dramatic because they were conveying parts of the story that were not represented graphically (e.g. you are aware of a human presence even though you never see a human figure). The accompanying music needed to be finely tuned to these sound effects to help emphasize but not overpower them. At the same time the music needed to help set the mood and pace of the piece. This careful timing required sequencers and an electronic cue sheet.

Sound effects were gathered from existing prerecorded sources and from hand-recorded sounds. We discovered immediately that a normal sound taken out of context (like a footstep) is unrecognizable without visual cues and the natural acoustic environment that normally surrounds it. We therefore needed to record greatly exaggerated sounds. For example, the footsteps of the person leaving the room were recorded by having a large person walking loudly across a cafeteria table. The sounds were then edited using *Sound Designer* by Digidesign and external effects boxes (reverb, etc.) all controlled with a Macintosh Plus.

The music was performed and edited using *Professional Performer*, a Macintosh sequencer. This program enabled easy experimentation with ideas of sequences and orchestration. Once the music was created, it was carefully cued to the timing of the animation. Individual bars of the music were fit to actions within the film. This stretching and scaling of time on a bar by bar basis required a powerful sequencer.

---

6 It would be simple to take a slave and attach it to any distributed process.

Though the sound effects were synchronized to particular frames of action within the film, much additional tweaking was needed to compensate for the psycho-acoustical properties of the effect, i.e. the moment you expect to hear the sound based on what you see. This often required bumping sounds backwards and forwards by as much as five frames from the actual event, and demanded at least half frame accuracy. All of this was done with *Cue Sheet* and the Opcode Time Code machine (which performed with 100% reliability). This kind of editing requires VITC equivalent timecode, which is a vertical interval time code that is frame accurate (a longitudinal time code can not maintain accuracy when single-stepping through tape). In general, a thorough understanding of SMPTE, differences between drop frame and non-drop frame striping and other additional timing complexities were needed to lay the sound to tape. This volume of expertise should be hidden from the user in future systems.

We have received many favorable comments on how well the music and the sound effects worked with each other. It was important to make sure that the music did not cover sound effects that gave pertinent information to the story line. Yet, too much sound and not enough score tended to slow down the motion of the piece. The synergy was due to the sound effects people and the music people working together constantly, not because of a particular software package. However, the software was well enough integrated to allow the various people to work together and share information.

8. OUTPUT

8.1. Images to Tape

When it came time to put the animation to tape we faced some very large problems. First, the Macintosh has no direct video output so we could not transfer the frames in the analog domain. Second, to save the entire film in RGB format without some kind of compression would have required seven gigabytes of storage (about 10,000 floppies). Finally, to transfer these digital frames to a digital frame store over Ethernet would take minutes a frame, translating into days for the entire film.

To circumvent the lack of video output we decided to use the Abekas A60 to transfer our files to tape. The Abekas is a digital sequence store that can be written to one frame at a time, and can play back 25 seconds of digital video in real-time. The Abekas' frame store saves frames in YUV format. A procedure was written to convert our RGB frames to YUV. These frames were then compressed using the standard Macintosh PackBits routine, packing each component separately (a scan line of Y, a scan line of U and then a scan line of V). We were able to compress the entire video down to approximately 1.7 gigabytes from the original 7 gigabyte figure. This fit easily on our 2.4 gigabyte file server (four 600 megabyte Racet drives.)

Though this compression solved the storage problem, these frames still needed to be transferred to the Abekas. To accomplish this we wrote our own Ethernet protocol to communicate with the Abekas. This worked reasonably well when communication was happening between two nodes on the same subnet, but it failed miserably when trying to cross bridges. This limited the number of machines we could actually use for rendering.[7] Shipping compressed frames to the Abekas over Ethernet still turned out to be a very time-consuming process. For every frame it took about 10 seconds to *decompress* the frame and another 10 seconds to *transfer* it to the Abekas. We finally reduced this time by abandoning Ethernet altogether and transferring the images via SCSI. This reduced our *transfer* time to about one second.

Once the images were on the Abekas they could be recorded, in real-time, onto video tape. For recording we used a Sony D1 digital tape drive. By using a digital medium we were able to prevent one generation loss in the recording process.

7 This number was further limited because of bugs in the file server. For some unknown reason, the file server could handle no more than 30 clients.

8.2. Gotchas

While these steps solved all of the known problems, we still encountered a few unknowns that were almost devastating to the project. The most pronounced of these was the 'disappearing file syndrome'. With only a few days left until the submission deadline we found that some of our files began to disappear. They still occupied space on the disk, but the directories no longer had any record of them. Fortunately, we were using the D1 for recording, and this became our backup device. As soon as we rendered something, we shipped it to the Abekas and then transfered it to the D1. If there were problems with individual frames in a scene, we could copy the scene from the D1 to the Abekas where we could replace those frames and then transfer the entire scene back to tape. However, by the time we grasped what was happening we had lost about one third of the film and many of the frames needed to be rerendered.

8.3. Sound to Tape

Once all of the graphics were laid to tape we were ready to lay down the sound track. Our original intention was to use a Macintosh-controlled 24-track studio for recording. However, after laying the first pass of the sound track to tape in such a studio, we realized that with two Macintoshs and three samplers we could record the entire sound track *live* with a mix down instead of taking the time to lay all of the sound effects onto different tracks of tape. In addition, synchronizing to the 24-track tape deck was no simple feat, whereas controlling the samplers with the Opcode machine was much more reliable. However, to record live, all of the sound effects had to be staged and divided between the three samplers to avoid generating overlapping sounds. This was done by hand with a giant multi-colored scheduling chart. Though this was manageable for a three minute piece it would be a nightmare for anything longer. Using the information from this scheduling chart, the cue sheet was filled with the location and destination of the music and sound effects, and the sound was loaded into the sequencers. The score was then ready to be laid to tape. This was an entirely Macintosh-driven process. In fact, there were no people in the sound lab when the music was laid to tape because everyone went into the graphics lab to watch the video, a dramatic climax to end our production!

9. CONCLUSIONS

One of the most prominent problems throughout the creation of the piece was the lack of data interchangeability between software packages. Different stages of the project produced 19 distinct file formats (Fig. 7), with the development of all conversion software up to us. This is completely unmanageable for an animator and even for most engineers. However, in the personal computer environment it is unreasonable to expect a single developer to provide an all-in-one package doing modeling, animation, rendering, I/O, etc. The environment is such that developers produce high-quality solutions for particular portions of the animation process, giving the user the flexibility to pick the packages that best meet her needs. Therefore, someone, preferably Apple, should define a common data format for modeling and animation and encourage the development of animation frameworks that provide presentation and data integration.[8]

Sound technology seems to be much further ahead than the graphics technology, and Apple has taken a lead here in promoting interchangeability. There are already timing and music data standards that are well accepted, and passing music and sound effects from one program to another is virtually standardized. Synchronizing to video is possible because there is already a

---

8 This problem also holds true outside of Apple. Developers for high-end workstations like the Silicon Graphics *Iris* machine tend to supply one-package solutions, and there are several such packages. However, there are no common data formats that would allow an animator to select a modeler, an animation module, and a renderer, each from a different vendor.

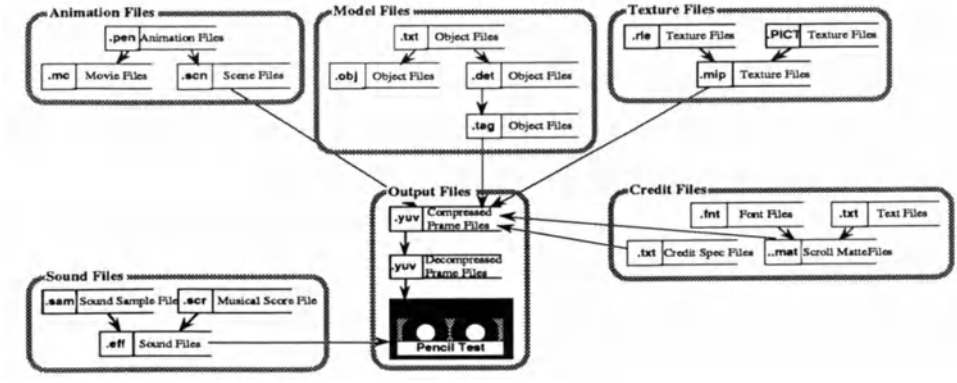

Figure 7. File Flow Diagram

mechanism for synchronizing to prerecorded material, a process that requires precision beyond that of video frame rates. However, there is still room for progress. A clearly defined interface to sound does not yet exist. The concepts of cut and paste, duplicate, etc. are very well understood methods of manipulating graphics, but the same ease of use and standard interaction protocol for sound is not in place. Apple should provide basic techniques for interaction with time ordered operations like sound and animated graphics.

Regardless of the speed of your computer, it is always possible to find a rendering algorithm that is complex enough to prevent rendering from happening in real-time; if you are using a personal computer this is almost a given. A standard method for using a distributed system to perform rendering (or any other computationally intensive activity) is an attractive solution, allowing differing qualities of animation to be produced in a reasonable amount of time. This applies not only to distribution across a network, but also to coprocessors within the host machine. We made a start at solving this problem by designing a generic slave that could be attached to any distributed task. Further efforts are needed to make distributed computing a standard on low-end machines.[9]

Finally, to this date, the phrase 'tools for computer animation' has been synonymous with 'packages for the creation of high-end 3D animation'. In focusing on solutions to complicated 3D animation problems, we have been overlooking some of the more basic problems. Software does not exist to manage the hoards of information that is generated for an animation. For example, there is no software to generate animatics from storyboard drawings or to manage scheduling data for music and sound effects. In addition, people have not been using computers to produce exciting 2-1/2D animation environments, something that is feasible for personal computers to achieve in real-time. Currently, the 2-1/2D world consists of basic motion control animation (moving bitmaps). Consider all the exciting work that could be done with animating splines and filled regions! Furthermore, interesting work can be done by combining the 3D and 2D environments which will reduce rendering time to something more manageable.

We need to be creative in finding new solutions for real-time interactive animation. In some ways, this may be the most challenging problem.

---

9 In the same vein, if frames are not going to be produced in real-time it is essential that personal computers provide some sort of frame-by-frame control along with the video output to allow standard recording equipment to be used. This is the area where enormous amounts of money are spent. To get work to video tape, one needs to buy a frame-by-frame controller or high-end video editing equipment. Both are expensive and neither is a reasonable solution for low-end computing.

10. ACKNOWLEDGMENTS

The following people made this project and paper possible: Jim Batson, Ernie Beernink, Mark Cutter, Sam Dicker, Toby Farrand, Jay Fenton, Jullian Gomez, Shaun Ho, Sampo Kaasila, Lisa Kleissner, Al Kossow, Mark Krueger, John Lasseter, Bruce Leak, Mark Lentczner, Tony Masterson, Steve Milne, Eric Mueller, Terri Munson, Daian Onaka, Wil Oxford, Jack Palevich, Steve Perlman, John Peterson, Mike Potel, Steve Roskowski, Andrew Stanton, Scott Stein, Carl Stone, Nancy Tague, Larry Tesler, Victor Tso, Ken Turkowski, Dave Wilson, John Worthington, Larry Yaeger.

REFERENCES

Foley JD, van Dam A, (1984) Fundamentals of Interactive Computer Graphics. Addison-Wesley, Reading, MA , chapters 9, 15 and 16.

Magnenat-Thalmann N, Thalmann D, (1985) Computer Animation. Springer-Verlag, Tokyo .

Williams L, (1983) Pyramidal Parametrics. SIGGRAPH '83 Proceedings 17(3):.

Wyszecki G, (1982) Color Science: Concepts and Methods, Quantitative Data and Formulae. John Wiley & Sons, Inc., New York.

Galyn Susman is currently a member of Apple's Advanced Technology Graphics Hardware group. She joined Apple in 1986 where she participated in the development of the color model for the Macintosh II, including writing the first color applications. Since then she has been involved in the architectural design of future Macs, and was project leader for the creation of the computer animated short, Pencil Test. She is currently managing a graphics software team focusing on animation, real-time graphics, and multimedia.

Prior to Apple, Galyn worked at Cadre Technologies developing display models in Postscript.

Galyn attended Brown University where she studied computer graphics.