1995

A Question of balance: Constructing a mobile software simulator

Michael Wulf Axelrod

A Question of Balance: Constructing a Mobile Software Simulator

by

Michael Wulf Axelrod

Submitted in Partial Fulfillment of the Requirements for the Degree
MASTER OF FINE ARTS

MFA COMPUTER ANIMATION PROGRAM
SCHOOL OF PHOTOGRAPHIC ARTS AND SCIENCES
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
1995

Marla Schweppe
Associate Professor
Film/Video Department
School of Photographic Arts and Sciences

Steve Kurtz
Associate Professor
Department of Information Technology

Francis C. Biddy
Producer/Astronomer
RMSC Strasenburgh Planetarium

Introduction

Initial inspiration for this project comes from the world of moving sculptures called mobiles. Artists such as Alexander Calder have pioneered the way by showing the world that mobiles can do more than entertain infants. This project brings the world of moving scultures to the computer and extends the artistic role mobiles can play for the average person.

The primary objective of this project was to design and implement a tool to create and visualize simple mobiles. The application was to be flexible and easy to use. So easy in fact that a child should be able to construct a mobile with this software, view it, and perhaps even construct it with real materials guided by a plan printed on an ordinary printer. Of course, as a requirement for this thesis project emphasis was placed on obtaining a realistic 3-D representational view of the finished mobile. The 3-D view was to include real-time animation of the rotating finished mobile. Additional features included phong shading, a directional light source, positional controls for zoom in/zoom out, and rotational adjustment about the x-axis. (providing views from above and below)

In addition, a great amount of effort was placed in developing an easy to use 2-D design environment. The design environment is, in a sense, the antithesis of the classic CAD package. Instead of allowing for a large amount of drawing possibilities, with an even larger learning curve, I constructed an environment that limits what can be drawn but more importantly reduces the learning curve dramatically. The program takes only minutes to learn and is ideally suited to people within a wide range of age and ability.

The Basic Idea

Typical use of this application begins with the 2-D design environment The user can select from a few simple drawing tools to create strings, sticks and simple geometric shapes. The drawing tools are extremely user friendly. Simple click and drag operations allow objects to be drawn in any size. A "move" tool is provided to "drag and drop"

objects in order to create a connected mobile. Objects are detached as easily as they are attached. At no time is the keyboard needed. Everything can be done with the mouse. In addition objects can be resized at any time with the 'resize' tool. The color of an object can be changed at any time as well. Standard editing features such as cut, copy and paste are also included. These basic tools allow for rapid creation of one or more mobiles.

The interface includes all the standard Windows features including file save and load, a tool bar with pop up help hints and a status bar for longer help hints. The interface also supports multiple views including the ability to display both 2-D and 3-D views simultaneously. The 3-D view can be selected at any time and work in the 2-D view can continue with the 3-D view active. The 3-D view will dynamically update itself as changes occur in the 2-D view. The 3-D view allows for zoom in and zoom out with a simple left click and drag with the mouse. A right click and drag rotates the viewing angle. Finally the 2-D view supports printing to any standard Windows compatible printer.

Planning and Research.

Aesthetic Considerations

A primary concern was to capture the serene almost floating nature of a mobile and the random spin caused by subtle wind currents. Control over color choices was of utmost importance too. The user of the software should be able to choose any color desired for any of the components of a mobile. The controls for the program should be simple as well. Mouse controlled camera movements for the 3-D view, and point and click drawing for the 2-D view were early decisions for the aesthetic feel of the software. These decisions influenced the project throughout its entirety.

For example, the camera controls for the 3-D view yield more than adequate control and are fun and easy to operate. A simple click and drag with the left mouse button zooms the view in and out. A simple click and drag with the right mouse button tilts the view

up to 360 degrees in rotation. Changing the color of an object is quite simple as well: Select an object and choose "change color" from a menu. A color tool appears allowing the user to choose any color the computer is capable of reproducing.

Another area of importance was creating the illusion of balance. It was decided early on that the software would literally know how to "redraw" a mobile that a user created, to accurately show how it would look if it were balanced correctly. This is very important in creating the final image of a "real" mobile. Accomplishing this goal was paramount to the success of the overall project. Indeed it can be stated that this portion of the software was successful.

Balance, color, shape, and motion all are important in the overall illusion. The ultimate challenge was to make them all happen at once. This could only be done with what is commonly called "real-time" 3-D animation. Real time animation is live, there are no pre-planned scripts. Rendering of each frame of animation has to be done on the fly. This required much forethought as to how these aesthetic requirements could be practically implemented under these tight time constraints. Sacrifices in accuracy and realism would no doubt have to be made at some point in order for the software to work on today's computers.

Another item of importance was to engage the user in the overall process of creating a mobile. It was decided that the software should be a complete mobile development environment. This environment should allow for design in 2-D and viewing in 3-D. It should also allow for printing of plans, to allow the construction of a mobile with whatever materials the user wished. To make this all practical the components of the mobile should be "flat". Complex objects of a 3-D nature would limit the flexibility, speed and functionality of the application, and hinder the creative process of the average user. Another major sacrifice in the interest of project development was that only simple polygons would be allowed as shapes for the first version. Advanced features such as

free-hand drawing and spline-assisted drawing would have to wait for future versions. All in all, even with these major sacrifices, an exciting project did unfold.

Software Considerations

Much of the foundation work for this project took place in the Macintosh environment. But commercial foresight gave me the impetus to complete this project in the Microsoft Windows environment. Early planning placed this prototype in the Windows 3.1 Environment. Later on, this changed due to the impending migration of most commercial software to the windows 32-bit environment (Windows 95). The general software components for this project consisted of the following: 2-D and 3-D vector graphic algorithms introduced in the "Programming for Computer Graphics" course sequence for the Computer Animation degree program, (See "Under The Hood" later in this paper) the OpenGL 3-D rendering engine and the MFC application framework as supplied by Microsoft.

Porting of the vector code to the Windows environment was quick and easy. I credit the modularity of Professor Kurtz's C++ code to this. However, some drawing functions presented a minor problem. Drawing to the screen on the Mac is a little different than in Windows. In an effort to accelerate the project schedule, I squeezed the Windows 2-D drawing commands into a few places that appear to be a bit awkward. But it works, and a rewrite may be in the cards for some of this code.

The choice for a third party rendering engine for the 3-D view was based on two main factors. Firstly, the drawing portion of the 3-D software inherited from the Computer Animation course curriculum was in it's infancy. (in the development cycle terms) Development of a 3-D rendering engine can be a whole thesis project in itself. Second I needed a rendering engine that could assure me of at least 15 frames per second real-time rendering rates. This led me to explore some of the 3rd party software solutions that were available. I spent several months evaluating various packages. I settled on the OpenGL

engine for several reasons, one being that it was an industry standard and had a history of successful applications. In addition it was then being ported the Windows environment.

Because of the choice to target Windows 95, and use the OpenGL engine, the development platform of choice (and necessity) was Windows NT workstation 3.5. I found this to be an excellent "crash free" environment. I credit Microsoft for developing an excellent compiler environment (VC++ 2.0) and a solid operating system for using it.

Initial consultation with various experts in motion studies and mechanical movement indicated that accurate representation of a hanging mobile object could become quite mathematically intensive. Therefore a simplified balancing algorithm was developed, by the author, that accounts for proportional representations based on a simplified equation, that does not account for complex motion (e.g. inertia, friction etc.) The equation is accurate enough to yield a visible mobile that reflects a "close approximation". Actual construction of a real (physical) model based on a mobile constructed with the software revealed that the simulation is accurate enough for the "artistic eye". In other words a simulated mobile looked just like the real thing as long as the real thing was peacefully spinning around. What the simulated mobile could not do is wiggle, tangle, bounce or anything that required complex motion studies.

Some "look and feel" elements found in the 2-D drawing environments are also original ideas. The user interface allows for fool-proof constrained drawing. In other words all the user need do is click and drag., and they can't really make a mistake. For example, the tool that is used to create polygon shapes has the following modes of operation; After pointing to a location on the screen the user then clicks the mouse and drags. Dragging the mouse in an up and down direction increases or decreases the relative size of the shape. Dragging the mouse left and right increases or decreases the number of sides the polygon has. By a simple click and drag the user can quickly create a piece of various size and shape. This theme of power combined with simplicity is prevalent throughout the design of the interface.

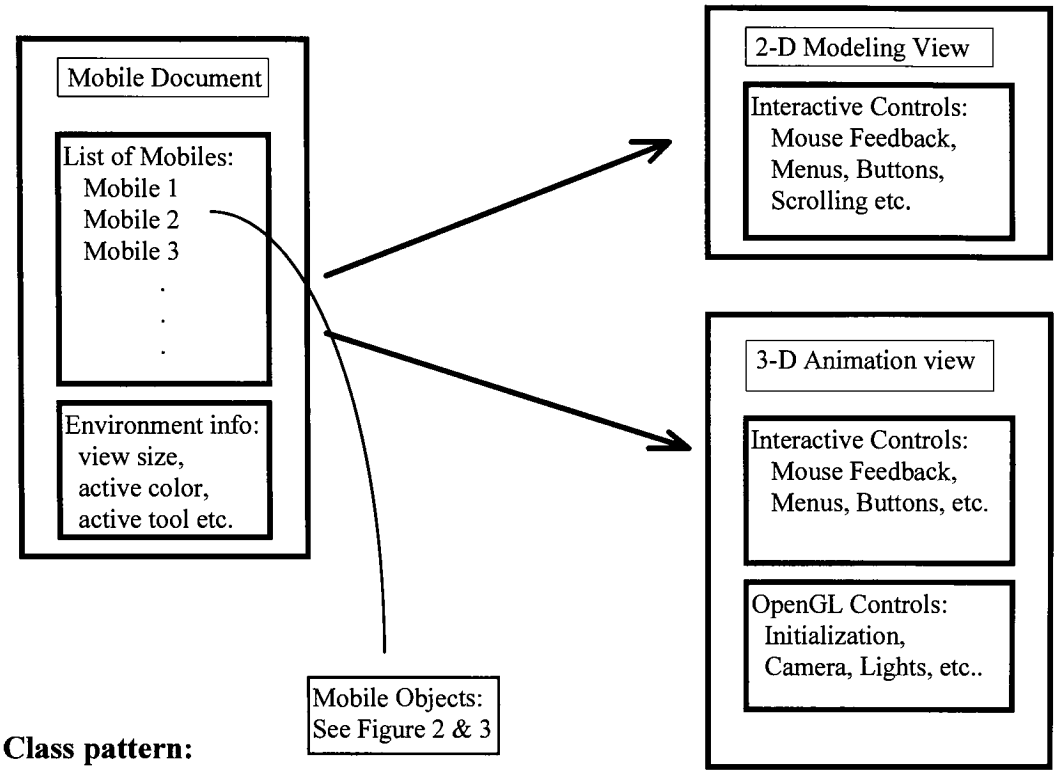
The simplicity of operation allows the user to concentrate on the overall configuration of the mobile (color, size, shape, etc.) so as to free the creative process as much as possible. The concept as a whole I also feel is an original. Oddly enough not a lot of time was spent developing these ideas, as they seem to have come intuitively. Someone observing the creation of this project might even think they were merely an afterthought. However these interface concepts are extremely important and contribute to the overall success or failure of any piece of software. Their implementation was not a trivial matter.

Software Architecture

MFC Doc/View Model Comments:

The application framework for this project is the Microsoft Foundation Class 3.0. The Microsoft Visual C++ environment allows for rapid development of complex Windows applications. The compiler environment comes with an "AppWizard" that spits out a blank template for a Windows application. The heart of this template is an object oriented class structure known as the "Doc/View" model (figures 1 & 4). In depth discussion of this model is beyond the scope of this paper; however, it is important to note that use of this model results an effective framework in which to build an application that has multiple views of the same objects. The "2-D view" of the mobile is just one interpretation of the data stored in the "Mobile Doc". another interpretation is seen in the 3-D view. The following diagrams show a somewhat simplified view of the C++ classes involved. Detailed diagrams of the classes involved would be far too extensive for this paper. The point in presenting this is to illustrate the ease of use and control that MFC (or any application framework) provides. This was useful for the main application and will continue to be useful as future modules are added (object database, history, etc.).

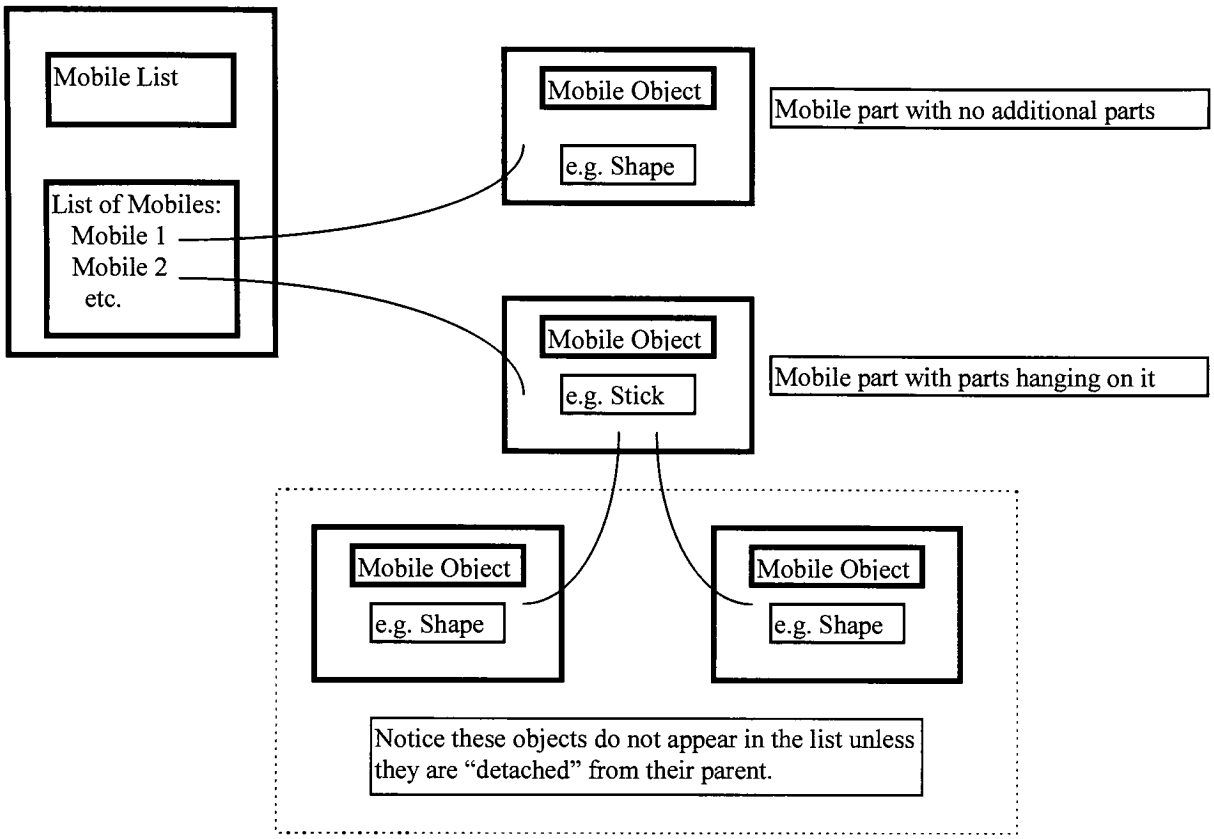
Figure 1: Mobile Application Overview



Mobile Class pattern:

The core technology central to the operation, creation, and functionality of the mobile created by the user is the "Mobile Class". String, Stick, and Shape subclasses all inherit this core class. This C++ structure is actually an inverse tree in nature. A "parent" mobile object has "child" mobile objects hanging from it. That is to say a string object might have a stick hanging from it which in turn might have other strings and/or shapes hanging from it. The inherit nature of this structure also leads to several recursive operations needed to calculate things like mass or motion control. The Mobile Document contains a list of all mobiles in the view (See figure two) and likewise each mobile contains a list of all mobiles it contains. These lists literally represent the inverse tree structure that makes up a mobile. The object oriented approach of managing these lists allows for ease in development of various mobile specific functions as well as non-mobile specific functions such as saving a file, cut, copy, paste etc..

Figure 2. Example Mobile List Relationship



The internal workings of the mobile object (See figure 3) contain links to specific vector based objects that control the position and motion of the graphic representation in its respective view. This "division of labor" creates an environment that is easy to maintain as well as debug. Future features will be easily added to their respective position in the "scheme of things" within the mobile software. The disciplined use of Object Oriented Software design techniques has yielded a truly sturdy, robust application that should surely withstand the test of time.

Figure 3. Mobile object details

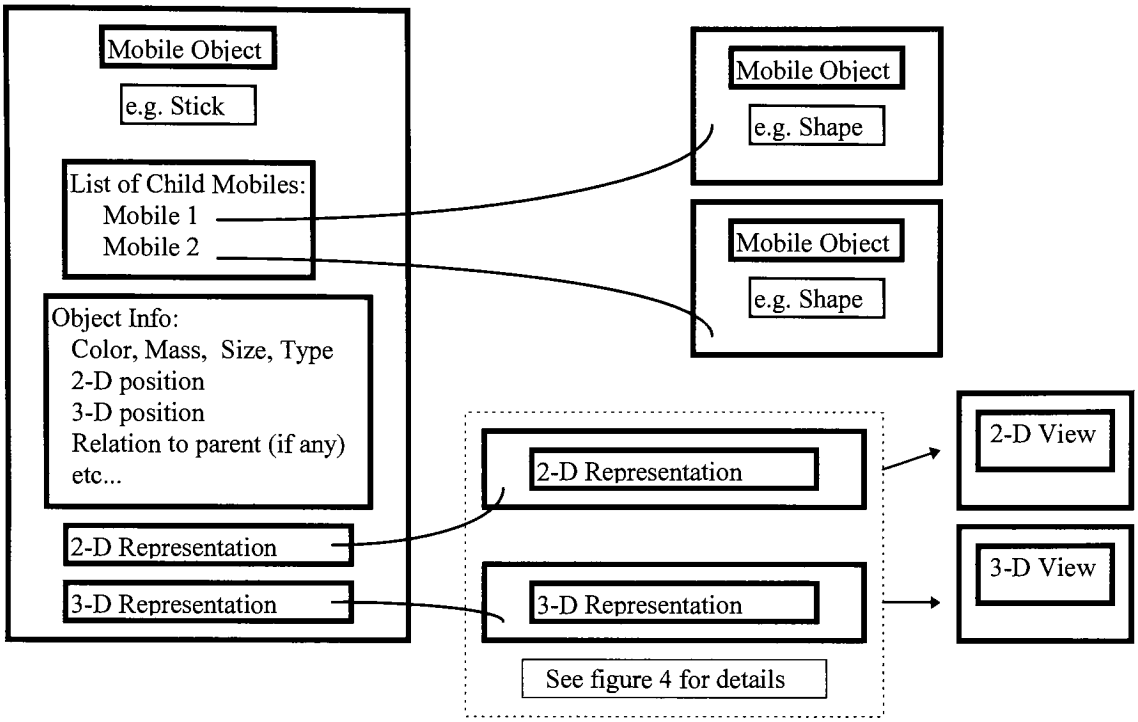
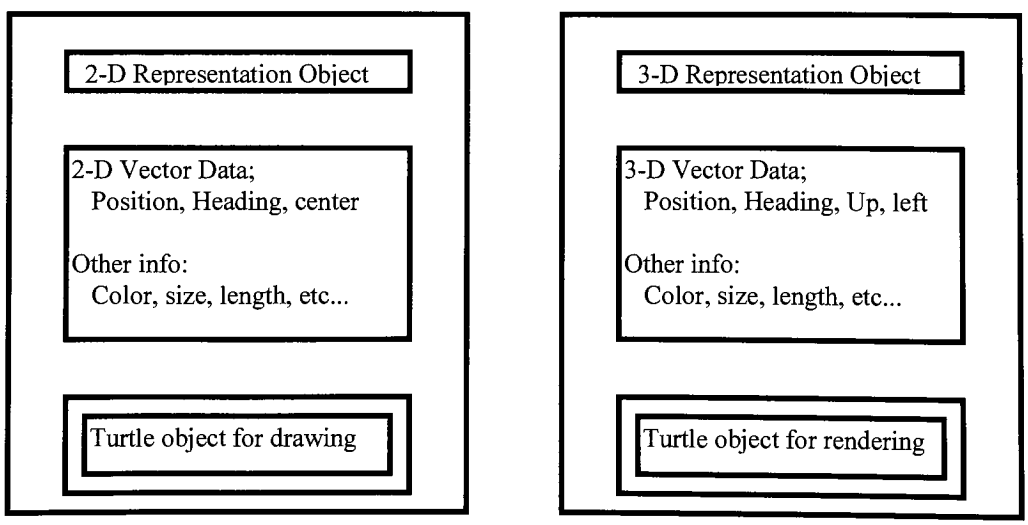


Figure 4. View Representation objects



Under the Hood: Vectors and Turtles

The Mobile class of Objects described utilizes vector math technology for keeping track positional information in 2-D and 3-D space. Much of this technology was derived from the course curriculum at RIT in the Computer Animation MFA sequence of graphics programming as presented by professor Steve Kurtz. An in-depth discussion of this topic is beyond the scope of this article. However some important concepts should be illustrated here.

Determining where an object "exists" in 2-D or 3-D space could be as simple as having a pair of variables to keep track of location (x,y) or (x,y,z). However, by using a robust C++ class we can create an object that knows much more than where it is in space. We can create an object that knows how it behaves in space. The "Vector" objects used to support the Mobile class are such objects. The fundamental 2-D and 3-D vector classes are used to construct 2-D and 3-D mobile parts that know how to move, and spin. in addition "Turtle" objects are used to draw the various views of a Mobile part. For example, the Turtle that draws a 2-D polygon knows only how to draw straight lines. However the Polygon object instructs the turtle to "draw a line", "rotate", draw a line again, rotate again, etc., until a whole polygon image appears.

Likewise the 3-D Mobile Polygon view object "has a" 3-D turtle that moves through space with methods such as yaw, pitch and roll. The 3-D polygon view object can also yaw, pitch and roll. This becomes possible because both classes inherit a common local coordinate system class that has this functionality. To further illustrate, a single frame of animation for a typical polygon object in the 3-D view goes through the following sequence of events.

First a "next frame" command comes down the "tree" of mobile objects touching each object. A polygon object will then "rotate" the number of degrees for this frame using the rotation method inherited from the local coordinate system class. (currently a random

number between .5 and 1.5 degrees). Next the polygon will prepare for rendering by sending its turtle out to circumnavigate a polygon in space. The turtle always starts out going "forward" relative to the polygon objects facing. Since the object has rotated the turtle will have automatically altered its course by this rotational amount. The turtle then uses its own inherited rotational function to alter its course as it makes each turn for each vertex of the polygon. The turtle then moves "forward" (another inherited function) to the next vertex. At each vertex the absolute x, y and z values for that object are "extracted" from the turtle objects position vector and sent to the OpenGL rendering engine in preparation for the final rendered image.

Using this Vector based technology turned out to be most useful in developing this application. It minimized bugs increased reusability and allows for easy expansion of functionality to the code. future features will no doubt take advantage of this existing code in the application.

OpenGL as a Rendering Engine

The success of the 3-D animation can be partially attributed to the use of the OpenGL rendering engine. A choice was made to use a "best of both worlds" approach by using only a portion of the OpenGL package. As illustrated earlier the 3-D vector based classes representing the mobile parts contain 3-D positional information that is passed to the OpenGL polygon rendering functions. All motion control is done using "non-OpenGL" vector technology. This is faster than using the matrix functions that are also included in the OpenGL package. Use of the matrix functions would result in computational penalties in calculation of the recursive motion of the mobile. Mostly this would be caused by additional matrix "pushes" and "pops" for every branch in the tree of a mobile. In addition bizarre code complexities would arise from over "OpenGLizing" the mobile classes. Therefore the OpenGL engine is used only for fast assembly level rendering and bit-map transfers to the physical device (the screen).

The Balance Algorithm

The simplified balance algorithm used helped to produce a close approximation of what the mobile would look like in the real world. The following equilibrium formula is the primary function used to calculate the length of each side of a stick from the string attachment point. (Lm * Ld = Rm * Rd) where Lm and Rm are the left and right total mass of each side of the stick and Ld and Rd are the distances from the string to the end of the stick where the mass is hanging. The mass of a polygon is simulated by calculating the area of the polygon. The mass of a stick is proportional to the length of a stick. And finally the mass of a string is assumed to be nil.

How the project changed over time

The groundwork for this project took place during the Computer Animation Graphics programming course work. Some fundamental software concepts for the project were derived directly from the course material. Two major assumptions were undertaken at the onset of this project:

The first assumption was that the 3-Dimensional view code would be very challenging to write. And to create a full fledged 3-D rendering engine from scratch would be both difficult and so time consuming that it could be a thesis project in itself. Since the point of this project was to make a useful (and hopefully interesting) application, it was decided early on to use an "off the shelf" rendering engine. This would save considerable time and allow the project to unfold in more of a creative direction.
The choosing of a 3-D rendering engine was finalized in the fall of '94 when the release of the new 32-bit Windows operating systems became a reality. The inclusion of the OpenGL rendering engine in the new operating system was a much celebrated event as it meant compatibility with an established industry standard that could be obtained on the PC, as well as the fact that working with this 3-D rendering engine was known to be quite easy compared to some of the more arcane systems available. As it turned out, reports

received as to its ease of use were very true. Enabling the engine to render an image was simple and yielded pleasing images in the first few attempts. The true challenge now lay in creating 3-D objects that lived in both worlds (2-D and 3-D).

The second assumption was that writing the code for constructing the 2-D view would be made less difficult by "re-using" some source code developed earlier. As it turned out this was true except for the implementation of some direct drawing functions. As mentioned elswhere drawing to a window on the Macintosh is done somewhat differently than in the Windows environment.

The majority of the work was now focused on creating a C++ class system that would have to keep track of the following problems: What a mobile was attached to, where it was in 2-D, how to behave in 2-D, where it was in 3-D, and how to behave in 3-D. After much experimenting it was decided that there should actually be two levels of classes to create the existence of a mobile. The first and most important was an underlying core class that represented the "soul of a mobile". This class knew about the where's, what's, and why's of a mobile but it didn't know the "how's". The second class of mobile existence was tightly woven into the first. This second class structure determined the how's: How a mobile was drawn in 2-D and how a mobile was drawn in 3-D.

As time went on this system became formalized and subclasses for strings, sticks and shapes naturally fell into place. Initial alpha testing of the software began in late winter/early spring of '95. User feedback revealed general acceptance of the software with minor improvements to be made in the area of the user interface in the 2-D building environment. By late spring of '95 the project was completed and shown to the public.

Major problems that were encountered.

The first attempt at this project actually began in the spring of '94. At the time it was decided to use 3-D Graphic Tools (TM) as a rendering engine. This proved to be a very arcane and difficult package to learn, although it was quite powerful in its speed and ability. The documentation was good for getting started. However it lacked a seriously in-depth tutorial, something I needed at the time.

Also at this time it was decided to use the Borland C/C++ and the OWL class library system for application development. Early work with the Borland environment revealed its power. But along with this power came a complexity that seemed more trouble than it was worth. The final straw came when it was learned from some fellow software developers that long term problems would occur from continued use of the Borland tools. These problems led the project to be delayed until the Fall of '94 when the new Microsoft tools became available.

Once the project was underway in the fall, new problems reared their ugly heads. One major problem was porting some Macintosh Drawing functions to Windows. The Windows methods of graphics drawing are very different than the Macintosh, the primary difference being the use of device contexts in Windows. This caused some very "inventive" code to be written to patch up the differences which I am still not completely comfortable with today. Needless to say, it works.

Another area of difficulty lay in developing the 2-D graphic interface and making it truly user friendly and interesting. This turned out to be much more time consuming than expected and at times difficult to debug. However this also proved to be some of the most interesting work, and continues to be an ongoing project.

Expected problems that did not occur.

There was only one area of concern that never became a problem. As it turned out porting the bulk of the object oriented vector graphics code from the Mac to Windows was quite easy and bug free. (Other than the platform specific drawing functions as noted above) This code is very portable and I was able to reuse the code again after the the abandoned spring of '94 (Using the borland environment) attempt and use it in the fall in the Microsoft environment without a hitch. In other words this code stayed very portable as the project evolved.

Future Work:

Here are some future things to be added to the mobile program:

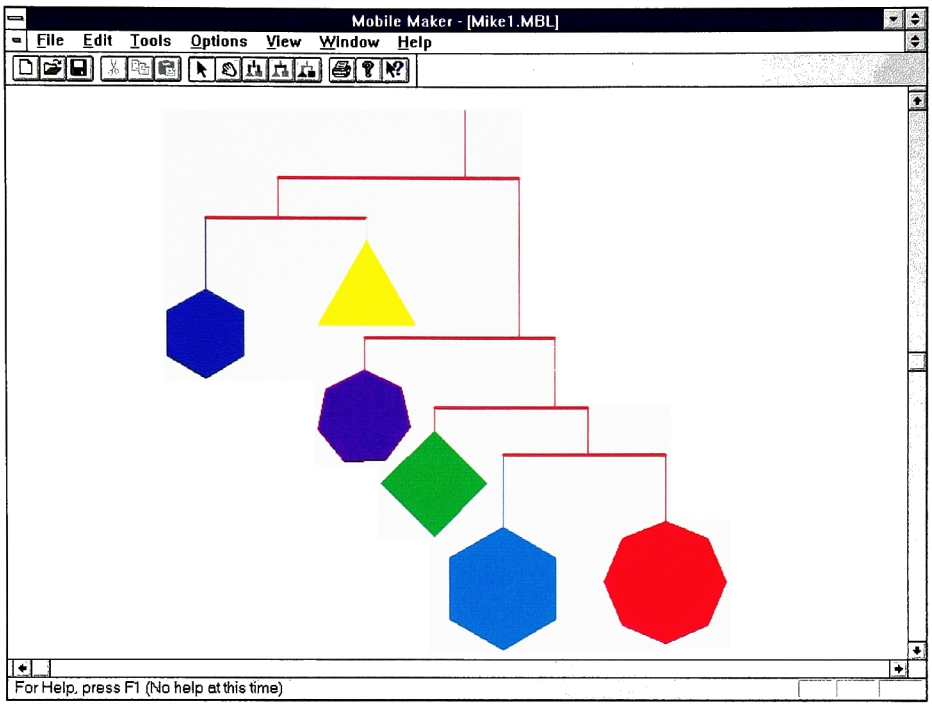
- Realistic "wobble" of hanging objects
- Freehand drawing of hanging objects
- User control of rotational velocity and direction, density, and color of each object
- Improved color chooser window
- Informational section on mobile history etc.
- Famous mobiles from art history to be included as pre-built mobiles.

Conclusion

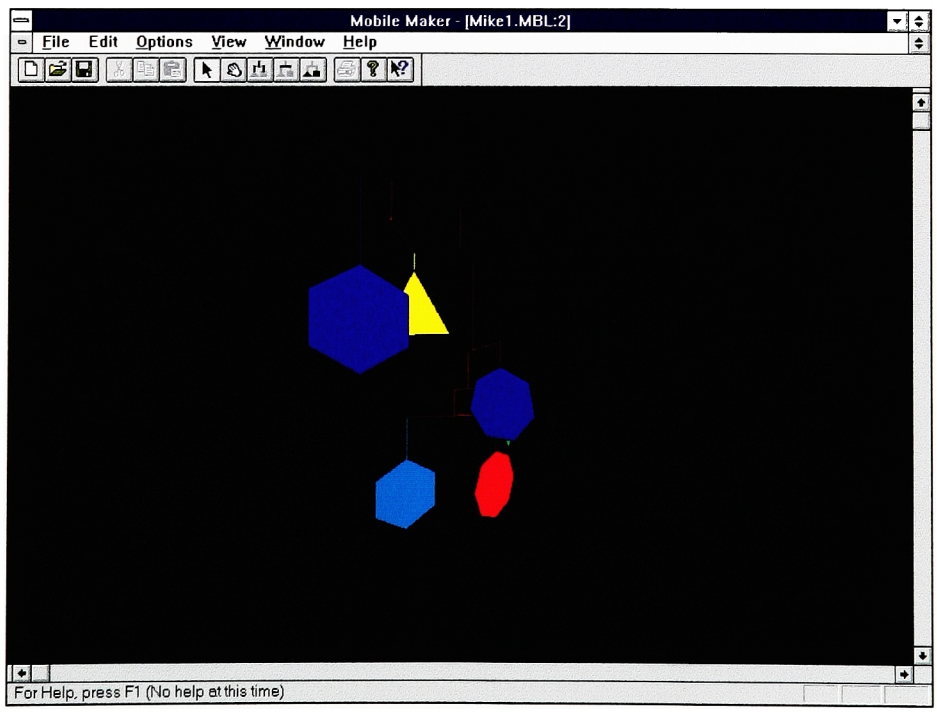
The project in its completion is now just a beginning. The future of this program depends greatly on the success of future enhancements and effective marketing. Perhaps the reader of this paper will someday see "Mobile Maker" or whatever it may be called on the retail shelves. Comercial success or not, it will certainly grow to become something greater than it is now. And it has certainly been a joy to create and to use.

Appendix A - Some Screen Shots of the Application

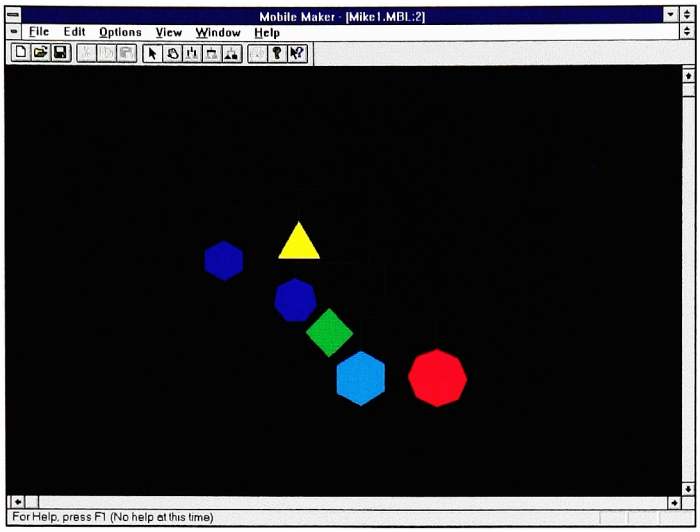
Screen 1: The 2-D Design Environment:

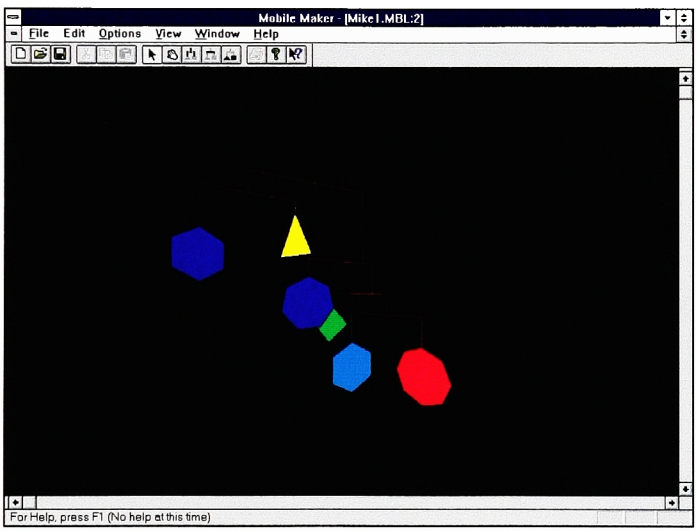


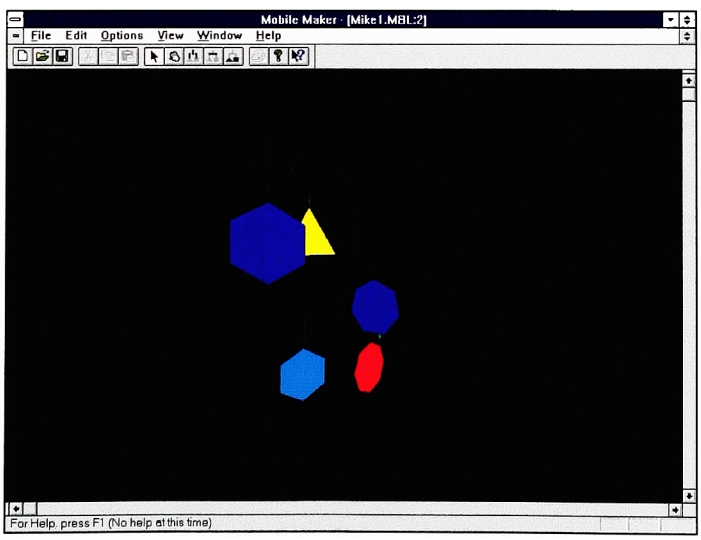
Screen 2: The 3-D View:



Screens 3 - 6: A Sequence from the 3-D View







Screen 7: MDI Interface

This screen illustrates the applications ability to allow design changes in the 2-D view while the 3-D view continues to animate. The 3-D view shows any changes made by the designer as soon as they are made. This instantaneous updating of views shows some of the real power behind a well designed object oriented modeling approach.