A PEN-BASED INTERACTIVE ANIMATION ENVIRONMENT FOR EFFECTIVE COMMUNICATION USING A TABLET PC

JEROME T. HOLMAN

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science Applications

Osman Balci, Chair
Manuel A. Pérez-Quiñones
Joseph G. Tront

May 5, 2005

Blacksburg, Virginia

Keywords: Animation, Collaboration, Ink technology, Modeling environments, Pen computing, Tablet PC, Visual simulation

A PEN-BASED INTERACTIVE ANIMATION ENVIRONMENT FOR EFFECTIVE COMMUNICATION USING A TABLET PC

JEROME T. HOLMAN

ABSTRACT

The human mind is a churning engine, processing all forms of sensory input. Information that reaches these senses is powerful, facilitating and deepening understanding. Animations access sight. They provide a rich visual model that portrays not only information about the system graphically, but they also present information about the relationships and interactions in that model over time. However, the complexity of today's software for constructing animations limits their use and isolates animations to a small set of scenarios. One such eliminated scenario is being able to create and manipulate animations in real time, while a conversation takes place. The human mind comprehends from manipulation, trial, and error. Animations are productions. Instructors cannot manipulate and recreate animations within the time constraints of a classroom. Developers cannot recreate animations within the time constraints of a meeting. This thesis explores the engineering of a modeling environment prototype for creating animated models. By building an environment designed to leverage the capabilities of Microsoft's Tablet PC operating system, this thesis contributes a new approach to creating animations and visualizations. This prototype demonstrates a pen-based user interface that decreases the time and effort required to create an animation by enabling users to create an animation by simply drawing a model and animating it. The Tablet PC hardware and software provide developers with the capabilities necessary to create software that works as naturally as the pen and paper. With the Tablet PC's Ink technology, users can draw models and simply draw the paths where the objects in that model should follow. This thesis explores the design of the environment, the research prototype named Model Pad, and explores its importance, its contribution, and its unique design based on the capabilities of the Tablet PC. As mobile computers become increasingly central to the workplace, the ability to create animations quickly and easily while talking with others, while presenting in front of an audience or a class, or while working with a team of engineers allows animation to become a natural part of the way we visually communicate information. The Model Pad environment is merely a first step in the direction of creating fully pen-based modeling and animation tools. The concepts it builds form the foundation of understanding how to express animations that are more complex. This thesis and the Model Pad environment provide the initial framework for building pen-based animation tools on the Tablet PC, allowing simple models to be expressed in methods as simple as using a pen and paper. This solution enables people to construct simple models in real time, enabling a rebirth of the power of animation for education and engineering design.

A PEN-BASED INTERACTIVE ANIMATION ENVIRONMENT FOR EFFECTIVE COMMUNICATION USING A TABLET PC

JEROME T. HOLMAN

GENERAL AUDIENCE ABSTRACT

The human mind is a churning engine, processing all forms of sensory input. Information that reaches these senses is powerful, facilitating and deepening understanding. Animations access sight. They provide a rich visual model that portrays not only information about the system graphically, but they also present information about the relationships and interactions in that model over time. However, today's software tools for constructing animations are complex, unnatural and limiting to for Students, Instructors, and Developers. This thesis explores the engineering of a modeling environment prototype for creating animated models with a natural form of input, a pen. By building an environment designed to leverage the capabilities of Microsoft's Tablet PC, a personal computer designed with a built-in interactive screen that enables users to draw on the screen to interact with it directly, this thesis contributes a new approach to creating animations and visualizations. This prototype demonstrates a pen-based user interface that decreases the time and effort required to create an animation by enabling users to create an animation by merely drawing a model and animating it. The Tablet PC hardware and software provide developers with the capabilities necessary to develop software that works as naturally as the pen and paper. With the Tablet PC's Ink technology, users can draw models and merely draw the paths where the objects in that model should follow. This thesis explores the design of the environment, the research prototype named Model Pad, and explores its importance, its contribution, and its unique design based on the capabilities of the Tablet PC. As mobile computers become increasingly central to the workplace, the ability to create animations quickly and efficiently while talking with others, while presenting in front of an audience or a class, or while working with a team of engineers allows the animation to become a natural part of the way we visually communicate information. The Model Pad environment is merely a first step in the direction of creating fully pen-based modeling and animation tools. The concepts it builds form the foundation of understanding how to express animations that are more complex. This thesis and the Model Pad environment provide the interaction primitives for building pen-based animation tools on the Tablet PC, allowing simple models to be expressed in methods as simple as using a pen and paper. This solution enables people to construct simple models in real time, enabling a rebirth of the power of animation for education and engineering design.

ACKNOWLEDGMENTS

I would like to thank Dr. Osman Balci, my advisor, for his help and inspiration in making this thesis possible. His experience in modeling and simulation along with his passion and vision for the future of computing are an inspiration. I would like to thank my committee who I also admire and respect for their time, patience, and expertise. They contributed tremendously to the ideas in this thesis and worked under extremely tight time constraints. I finally would like to thank my better half, Ines Khelifi. Finishing this thesis was possible because of her love and support throughout my years in graduate school. I thank you Ines for always understanding that being "almost done" never really means being almost done.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

API	Application Programming Interface
CLR	Common Language Runtime
DLL	Dynamic Linked Library
DPI	Dots Per Inch
GDI	Graphical Device Interface
HID	Human Interface Device
J2EE	Java 2 Platform, Enterprise Edition
OEM	Original Equipment Manufacturer
OS	Operating System
PC	Personal Computer
RPC	Remote Procedure Call
UML	Unified Modeling Language
VSE	Visual Simulation Environment
XML	Extensible Markup Language

Chapter 1: Introduction

The earliest records of human civilization can be traced to graphical models. Since the dawn of man, humans have created and constructed graphical representations of the world around them for art, for religion, but more importantly for understanding. These graphical models have represented some of the world's greatest feats. Ancient Egyptians planned the great pyramids intricately on papyrus with ink. Leonardo da Vinci sketched prophetically models for flying machines with ink and quill. Graphical models have continued to be the primary method in which we communicate ideas and build an understanding of the systems we attempt to create. Ink and paper have and continue to be the primary tool for creating graphical models.

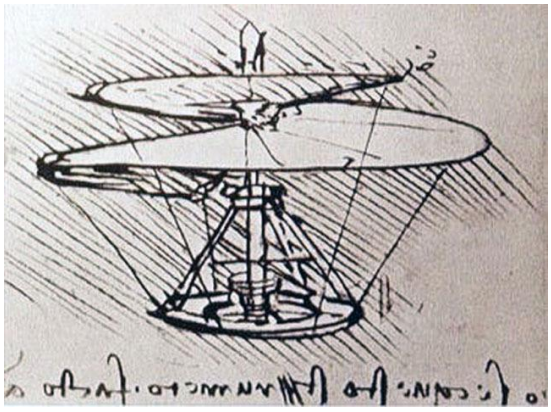

Figure 1. Leonardo da Vinci's Sketched Model [Fabricius 2010]

Even though graphical models can communicate a great deal of information, more can be expressed by animation. When the great minds like Joseph Plateau in 1826 and Pierre Desvignes in 1860 discovered that they could simulate motion by quickly moving images successively [James 2005], animation was born and as the 20th century progressed, animation became a powerful method for engineers to express how their graphical models operate over time. Today, animation can instantly convey how a system works by demonstration.

Commercial and academic software for constructing and animating graphical models pervades the market. Engineers of all disciplines use animation as a way to express rich information about their graphical models, creating simulations of how those models act over time when they are operational. These software packages provide engineers with the ability to use a variety of input devices, a mouse, a keyboard, and a plethora of various alternative input devices, to build and construct models for animation.

With the advent of the Tablet PC, a pen-based computer with mobility and a slate like form factor, the idea of pen-based computing has arisen again to the forefront of the computer industry. The pen is the original and most natural input device for creating graphical models. With a pen that can act as a digital input device for a computer and an LCD screen that can act as a sheet of digital paper, a new breed of pen-based modeling and animation applications can be created that make creating animated models simple. Through this simplicity, the time and the

learning curve for creating an animated model is shattered, and by lowering the barriers of time and knowledge, animated models can be created and used in ways that are not possible with today's tools. This thesis describes Model Pad, a tool created for leveraging the capabilities of the Tablet PC that enables users to create models by simply drawing the model components and connecting them by using paths. The Model Pad tool is one of the first environments for creating animated models simply and quickly using the natural strokes of a digitized pen.

1.1 Pen-based Animated Modeling Scenarios

1.1.1 Scenario I: A Professor in the Classroom

A computer science professor is giving a lecture on the insertion sort algorithm. The professor draws an empty array and informs the class about how the elements will be inserted into the array unordered. The professor then fills the array with random integer numbers and explains how the comparisons for the insertion sort algorithm work and how the insertion sort algorithm will swap elements based on those element values being less or greater than the current element. To demonstrate this, the professor draws the paths of the elements that will be swapped during the first iteration of the insertion sort. Using the Model Pad prototype tool described herein for all of the drawings, the professor simply presses play and the elements swap in and out of the array, demonstrating to the class by animating the operations of that iteration.

1.1.2 Scenario II: A Design Meeting Over Lunch

Two engineers sit down to discuss a design for a new traffic pattern for a new major interstate that they are creating. The discussion leads to an idea. One of the engineers quickly pulls his Tablet PC from his suitcase. The engineer quickly sketches the road and all of its branching entrances and exits. The engineer opens the model library, drags a few cars onto the sketch, and proceeds to draw paths for them all according to the new traffic pattern just discussed. The engineers sit back, watch the animation of the traffic pattern, and discuss their design.

1.1.3 Scenario III: Understanding a J2EE System

A student sits in class listening to a software engineering lecture. The professor is discussing a flow of message passing in a Java 2 Platform, Enterprise Edition (J2EE) system. The student has a hard time visualizing the message passings among the objects as the teacher describes them. The student quickly opens Model Pad, drags the picture of the J2EE system onto the model pad, sketches a circle to represent the message, and draws the path of the message through the system as the teacher has indicated. The student then watches the animation of the message passing through the J2EE system and comes to an understanding of how the message flow really works.

1.2 Animation as a Tool for Learning

Today, information is available in all forms through all mediums. The twenty first century will be remembered as the digital age where information flows through society as quickly as computers can transfer bits over the Internet. This information based society includes rich media

that touches all of the human senses, sight, sound, and through those, the heart. Because of this, children, and increasingly the adults of this digital era, learn using multiple forms of media [Tapscott 1998]. These forms of media include everything from television, interactive games, and the Internet. Studies have shown that this new generation is exceptionally curious, smart, and focused [Tapscott 1998]. This growing demographic of youth challenges society by completely changing the way that children and young adults gather, accepts and retains information [Tapscott 1998].

Classical education where a student sits and listens to a presented lecture is a very passive form of education. With this new, media savvy generation, passive techniques become ineffective and do not engage the student at the same level as if they were actively participating [Schank 1994]. When students are given the opportunity to actively participate in the information a teacher is presenting, they access more of their mind and actually learn by doing. Babies learning to take their first steps learn by actively trying and eliminating ways to hold their body up and move their feet. They actively manipulate the world around them to find a solution that works. By doing this, they learn at a level that could not have been obtained by passively watching another baby walk [Schank 1994]. Active learning obtained from interactive multimedia content, animations or systems that can not only be watched but also manipulated provide this same opportunity to learn, gaining a deeper understanding of the system or information in a natural way [Schank 1994].

Even though research continues to demonstrate the benefits of active learning, active learning and interactive content has not become an integral object of today's classroom. Part of this lack of progress can be traced to antiquated beliefs in today's educational system [Schank 1994], but a great deal of the reasons behind the lack of progress trace directly to the cost of renovated curriculums, training teachers, and developing the interactive content. Synapsys, an educational solutions provider, estimates that the cost of developing interactive content is at a minimum $15,000 per learning hour [Garing 2005]. For highly engaging and interactive learning simulations, the predicted cost is approximated to be as high as $42,500 per learning hour [Garing 2005]. These extremely high prices are prohibitive for most learning institutions to truly take the time and cost to develop the type of interactive learning models and simulations that would truly create active learning experiences for students in today's schools and institutions. There is a need to discover ways to increase the interactivity of the classroom experience while allowing students to use their senses, already used to graphical and media rich experiences, to interact and learn classroom content.

1.3 Review of Tools for Constructing Animated Models and Simulations

1.3.1 Overview

Before delving into the engineering of Model Pad, it is important to understand the landscape of modeling and simulation environments that include the ability to animate a modeled system. After a search of today's and yesterday's prominent literature, very few modeling and simulation environments allow for the ad-hoc creation of animations using a pen-based interface.

1.3.2 The Application Visualization System

The Application Visualization System is designed to be a framework for engineers and scientists to develop and construct programmatic models through visual programming [Upson *et al.* 1989]. Through the Application Visualization System, a scientist or engineer can build a high-level model out of building blocks that represent different objects, modules, and logic. The modeler drags these reusable blocks into the visual designer and creates a layout of a model for a system. That model can then be executed through an advanced analysis and rendering system. The interesting aspect of this tool that is relevant here is that it establishes this concept of starting the user with the ability to drag objects into a blank modeling space. In addition, since the model can be compiled and rendered, there is an ability to build a model and watch it render as a 3-D visualization that can be manipulated and viewed. This builds the relationship of blank space to model to rendered system that provides the user with the ability to interact with the product of that model. This style of user interface can be found in many modeling applications in this area of domain-specific modeling and visualization.

1.3.3 Visual Programming in a Computer Animation Environment

Understanding how software can present an interface for creating computer animations is extremely important to the core of this thesis. An early paper on a Visual Programming environment for constructing computer animations demonstrates a model for constructing software for building computer animations [Reeth *et al.* 1990]. Reeth and Flerackers [Reeth *et al.* 1990] establish an important definition for the different types of animation environments while establishing the generic steps required by any software tool for creating animations.

They define computer assisted versus modeled animation. Computer assisted animation is an animation created with a software environment that provides a means for specifying some aspect of how an object created in the system should be modeled [Reeth *et al.* 1990]. When the animation is rendered, the software acts as an interpolator, animating motion based on the aspects defined by the user. Modeled animation is an animation the motion of which is fully specified at each frame of movement [Reeth *et al.* 1990]. Modeled animations are created with sophisticated tools that allow the user to define movement through specific translations and movements that they specify.

Reeth and Flerackers [Reeth *et al.* 1990] indicate that there is a generic sequence of steps that must be defined by an animation environment to create an animation. An animation environment must create a set of objects, specify their motion, and render the images that will compose the animation [Reeth *et al.* 1990]. Even though it is extremely obvious, the clarity in which it is defined is important to understand when creating an animation environment. Most environments for constructing animations follow this model.

1.3.4 A Picture-Based Object-Oriented Visual Simulation Environment

Balci et al. [Balci *et al.* 1995] introduces the concept of using pictures to represent the objects of a visual model. This concept is fundamental to providing a richer experience for an animated

model. Other systems use boxes and symbols to represent the objects of the system, forcing the user to visualize what these objects actually represent in the real world. The unique concept of using pictures, pictures that can represent richer visual information, possibly even being depictions of what an object may be in the real world, adds to the richness of interactive experience. The Visual Simulation Environment (VSE) developed by Balci et al. [Balci *et al.* 1995] highly influenced the direction of this thesis research. Images provide more information than symbols. Any modeling environment that does not provide the ability to use images as a representation of a object will not be able to provide the same depth and connection with the user as a system that does have this ability. This is integral to learning and constructing interactive animations that could be used for education.

VSE provides a model for designing a modeling application with simulation and animation capabilities. VSE provides a visual editor that enables the user to have an open space for graphically constructing a model, and a library for accessing reusable objects that can be used in the construction of a model [Balci *et al.* 1995]. VSE also contains a model analyzer and tester for assisting in the accuracy assessment of a model, a model simulator for running and animating the model, and several other support tools [Balci *et al.* 1995]. This thesis focuses on creating an animation environment for constructing a visual model. This environment logically parallels the purpose and functionality of the visual editor and reusable objects library of VSE.

1.3.5 JSIM: A Java Simulation and Animation Environment

JSIM provides a model and architecture for a modeling and animation environment. Fundamentally, JSIM provides the user with the ability to construct a model, execute that model, and watch an animation of a visual representation of that model, constructed from the model [Miller *et al.* 1997]. Through pointing and clicking, the user can construct a model and animate that model. JSIM provides a model for programmatically building a library of reusable objects that contain logic that can be used to construct a model. Each simulation or model has a built-in animation defined by connecting objects in the model designer. When an object is added to the model, a properties window is displayed that allows the properties for that object to be defined. The animation is defined through the user's construction of a set of entities and their flow throughout the model [Miller *et al.* 1997]. All of the information about the simulation is stored within the objects to be accessed during animation or when the simulation is saved.

JSIM provides an example of a visual modeling environment for creating animations also intended to run on the web. JSIM has the ability to connect to backend databases. This provides JSIM with the ability to provide simulations based on real data and animations that correlate to those simulations.

1.3.6 The JOTSA Animation Environment

The popularity of creating animation environments for web based systems inspired Robbins [Robbins 1998] to delve into a deeper analysis of the implementation issues of constructing animation environments. Robbins [Robbins 1998] introduces several methods for redrawing objects to the screen as they are animated. He suggests several methods, including having each

object have its own independent thread that will redraw its respective object when necessary and alternatively, having a single master thread that repaints objects based on a virtual clock, independent of any individual movement [Robbins 1998]. Robbins [Robbins 1998] discusses the implications and performance issues of the various techniques. The knowledge gained from Robbins [Robbins 1998] is of techniques and approaches for implementing a virtual clock to have a master thread that can control animation. The Tablet PC API's inability to handle certain threading scenarios (refer to The Ink in section 2.5 for further details) exemplifies the importance of Robbins' work.

1.3.7 Tango – A Framework for Algorithm Animation

Tango introduces a framework for algorithm animation [Stasko 1990]. Algorithm animation is one of the core scenarios this thesis attempts to support. In the journal article, Stasko [Stasko 1990], provides a framework for using graphical objects and paths as a method for animating algorithms and data structures. What is unique about this article is that it introduces the idea of modifying and creating animations in real time, using the natural semantics of allowing a user to identify the objects in his or her system and simply define where those objects should move in time. This framework closely aligns with the principles used to design the Model Pad prototype.

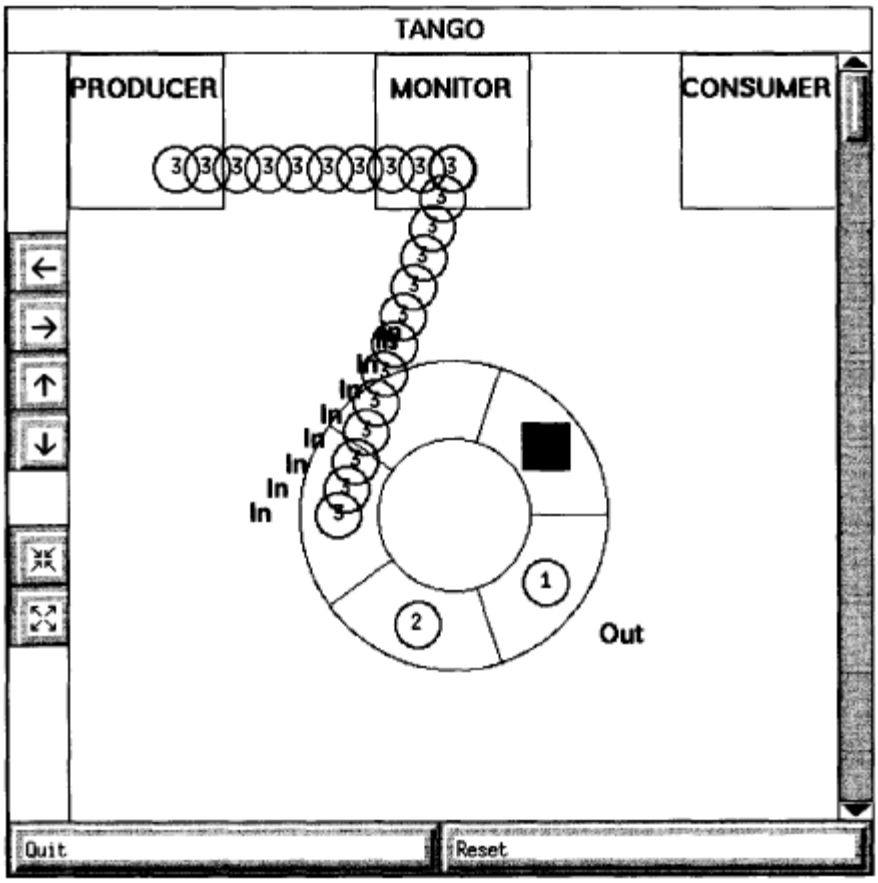

Figure 2. Tango Animation of Producer-Consumer Data Structure [Stasko 1990]

1.3.8 Visual Languages for Complex Design

A 2002 CHI paper provides one of the best modern examples of using the pen to draw and design complex systems [Lin *et al.* 2002]. In the paper, the pen provides the primary method for drawing the design of a web site with a tool named DENIM [Lin *et al.* 2002]. DENIM introduces a concept of drawing complex, reusable components to representing the components of a web site and saving those representations as reusable components. These components then form the basis of a visual language that can leverage the drawn representations to create complex systems. Figure 3 provides a screenshot of the DENIM tool. Uniquely, DENIM uses a side bar of reusable components as a basis of its interface with a general use toolbar. DENIM also allowed the representation of relationships between paths between the different components of the web site using drawn arrows. These choices closely align with the design choices used to construct the Model Pad prototype described herein.

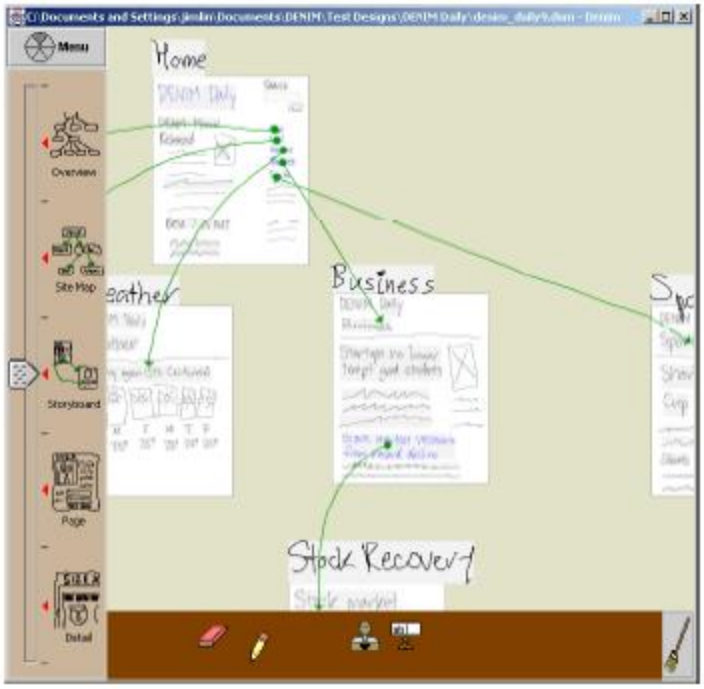

Figure 3. DENIM Tool for Drawing Web Designs [Lin *et al.* 2002]

1.3.9 SATIN – A Toolkit for Informal Ink Based Applications

SATIN is a framework that was also used in the previously discussed DENIM application [Hong *et al.* 2000]. SATIN is a foundational framework for creating ink-based applications. The greatest contribution from the SATIN paper is the framework it establishes for constructing ink-based applications. SATIN formalizes many of the concepts seen in the Ink API and the Tablet PC such as maintaining ink in the form of stroke objects and translating ink strokes into formal shapes [Hong *et al.* 2000]. For this reason, it is important to acknowledge their work in the area because it formalizes a user interface framework for ink applications even if the fundamental pen technology was created much earlier.

1.3.10 Conclusion

A survey of the literature revealed how animation and modeling environments have evolved over the years. Fundamentally, they have all followed similar principles in terms of user interface and design. However, unique techniques such as adding images to a model and supporting rich forms of animation show the variety of approaches that currently exist. Stasko's work [Stasko 1990] provided an innovative approach to animation, moving away from models based on how animations are implemented, instead creating an animation tool that allowed users to build animations naturally using the semantic reasoning of where they want objects in an animation to move using paths. This work and others moved animations tools towards simplification. With the increasing acceptance of alternative input devices like the pen, modeling, simulation, and animation environments can be revisited, leveraging the abilities of devices like the pen. This

raises the question whether or not a modeling approach [Reeth *et al.* 1990] to animation could simplify the creation of an animated visual model and fundamentally change the level of complexity a user experiences when creating a visual, animated model.

1.4 Review of Tablet PC Tools for Modeling and Animation

1.4.1 Overview

The Tablet PC remains a new device in the computer science community. With only a life span of a little over three years, developers and researchers are still attempting to understand how to take advantage of the device to redefine application design around the pen and take advantage of the capabilities offered by the operating system and the Tablet PC Ink API. This is a review of the prominent commercial and research applications and their approaches to constructing environments for modeling and animation.

1.4.2 Ink AniEd

Geselowitz [Geselowitz 2005] introduces an animation editor for the Tablet PC. This editor has the ability to take the drawn handwriting (known as Ink, refer to 2.4 for further information) of a user and convert it into an animation. AniEd creates the animation by allowing the user to move a time line and redraw the position of the Ink strokes at the new frame. AniEd calculates the translation of the strokes per each drawn frame and creates a smooth animation in SVG that can be replayed [Geselowitz 2005]. AniEd's contribution to the pioneering field of Tablet PC animation environments is the ability to interpolate where strokes translate over an arbitrary timeline and its ability to convert Ink stroke data into SVG, allowing it to be rendered and used in other applications and on the web.

Figure 4. Ink AniEd Screenshot

1.4.3 Drawing Animator Toy for Tablet PC

Microsoft created several technology demos after they released the Tablet PC to demonstrate the capabilities of the Tablet PC and more importantly the Ink API. One of these technology demonstrations was a small application released as a Power Toy called the Drawing Animator Toy [Microsoft 2005]. Even though it is like its name, merely a toy, and should not be considered a serious animation environment, its contribution to demonstrating the capabilities of the Tablet PC and the Ink API are important. The Drawing Animator Toy takes handwritten Ink strokes and translates them into polygonal figures drawn by the Windows drawing system, Graphical Device Interface (GDI). That means that the Ink data is completely lost, however, by doing this, they were able to create a faster, more memory efficient animation. The calculated polygons simply bounce around the window until the application is closed [Microsoft 2005]. This application also implies the intent for Ink strokes not to be used directly for animation because of the design of the current version of the Ink API. This will later impact the design and development of Model Pad as discussed later in the thesis.

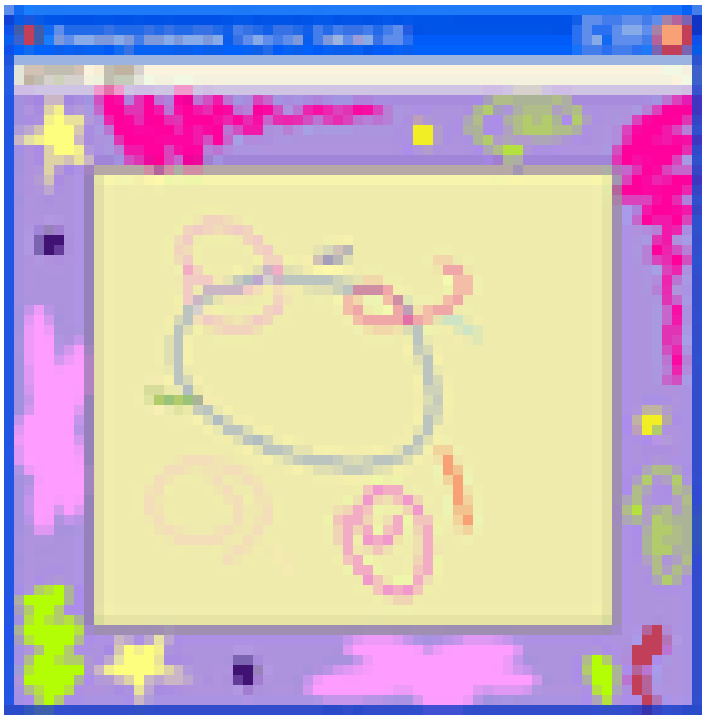

Figure 5. Drawing Animator Toy for Tablet PC Screenshot

1.4.4 Physics Illustrator

Physics Illustrator was created as a joined research effort with Microsoft and the University of Washington [Microsoft 2005]. Physics Illustrator builds on the concept of translating Ink strokes to polygonal figures as found in the Drawing Animator Toy. The Physics Illustrator just uses this for purpose other than speed. In Physics Illustrator, everything drawn is translated. Shapes that are drawn circular are translated to circles. Shapes that are drawn as rectangles are

translated to rectangles. Physics Illustrator takes this a step further by introducing the concept of gestures. Gestures are invisible strokes that instead of being drawn on the screen are recognized by the computer as a set command [Rob Jarrett 2002]. In Physics Illustrator, a check drawn inside of a constructed polygon will turn it into a solid object [Microsoft 2005]. Each object is assigned a set of physical properties and the user can interact with the drawn model in a live, interactive animation based on the properties that are set. This is again, another demonstration of translating ink strokes to polygons and a demonstration of gestures.

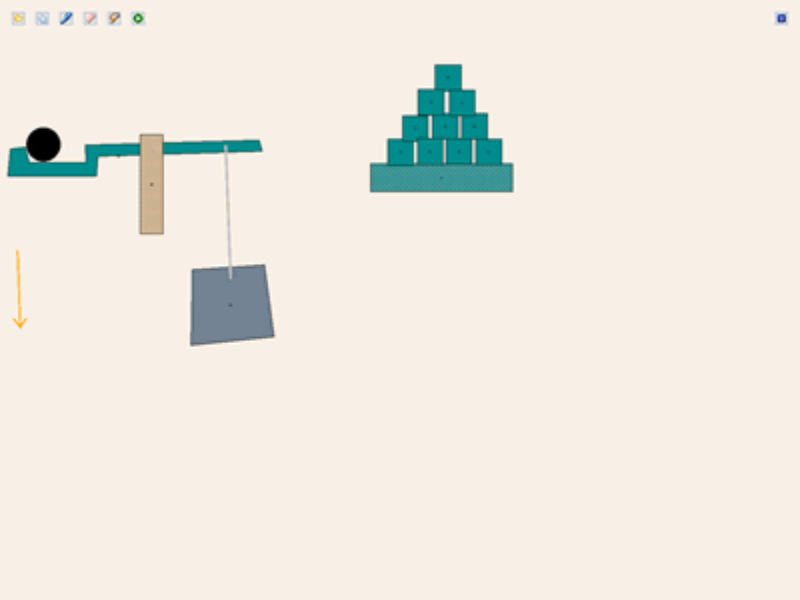

Figure 6. Physics Illustrator Screenshot

1.4.5 Tablet UML

Tablet UML (Unified Modeling Language) is a UML modeling tool that translates handwritten models into UML models [Shoemaker 2005]. If an object is drawn that closely resembles the semantics and graphical syntax as defined by the UML standard, Tablet UML will translate that handwritten object into a defined UML object with all its drawn relationships.

Tablet UML has no animation object; however, it stands uniquely as the only modeling tool for creating software systems. Tablet UML provides a user interface that maintains a side bar that shows all of the currently defined objects on the canvas. This UI approach creates a virtual palette of objects that the user can then use to navigate the objects defined on the canvas, delete them, or duplicate. This unique approach has its advantages when building a model using the pen.

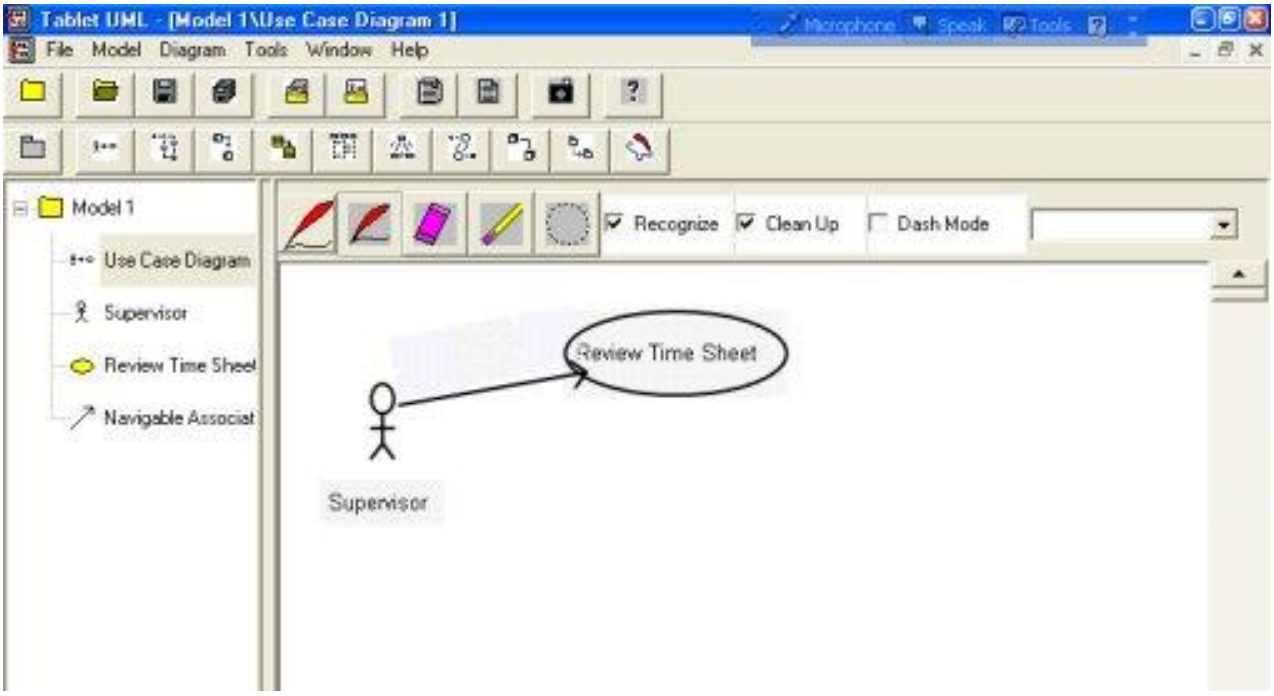

Figure 7. Tablet UML Conversion Screenshot

1.4.6 Alias SketchBook Pro

Alias Wavefront, a company known for their graphical products such as Maya, released a commercial application for the Tablet PC called SketchBook Pro. SketchBook Pro is not specifically a modeling tool even though it can be used as one due to the ability to have named layers like Photoshop [Alias 2005]. SketchBook Pro does not have the ability to animate a drawing. SketchBook Pro's contribution to software on the Tablet PC is its unique user interface.

The SketchBook Pro user interface consists of a radial control that is located at the bottom of the main window. This radio dial contains all of the controls for the application. To access the menu, the user simply clicks the icon that he or she wants, located on the other edge of the circle. That icon then expands, allowing for the user to simple draw a line from inside the circle to outside and through the control he or she would like to accept [Alias 2005]. This demonstrates the unique controls Tablet PC applications can design to take advantage of the pen. More importantly it demonstrates that today's menu systems are not adequate for the needs and demands of Tablet PC users.

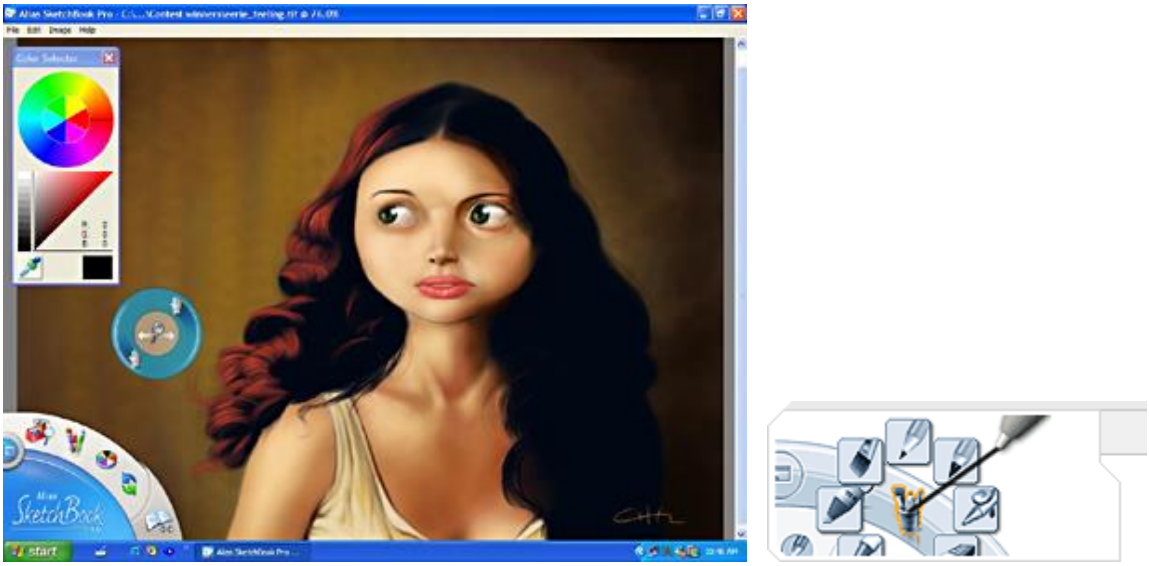

Figure 8. Alias SketchBook Pro Screenshot and Pen-based Control UI

1.4.7 ArtRage

ArtRage is an application that takes a different approach to creating a drawing application than those discussed previously. ArtRage, like SketchBook Pro, is not an animation tool or specifically a modeling tool even though it can be used as one. ArtRage is simply a unique application for the Tablet PC whose contribution is the elimination of the standard Windows UI toolkit and specialized Ink made to resemble actual paint by taking advantage of pressure sensitivity (see 2.4.2 for further information).

ArtRage's unique user interface is large and optimized for the pen. All clickable menus are easy to access with a simple stroke. ArtRage inherently makes the knowledge of the fact that the accuracy of the pen will be difficult by not conforming to the Windows UI. The success and pleasure that users state with Art Rage exemplifies the importance of menus designed for the inaccuracy of clicking with the pen.

ArtRage's use of custom Ink shows how the Ink of a stroke can truly be used to add to the user experience of the application. ArtRage generates photorealistic paint, tapering paint strokes off based on pressure sensitivity and other conditions. This functionality can and will be applied to animation environments, separating modes and functions.

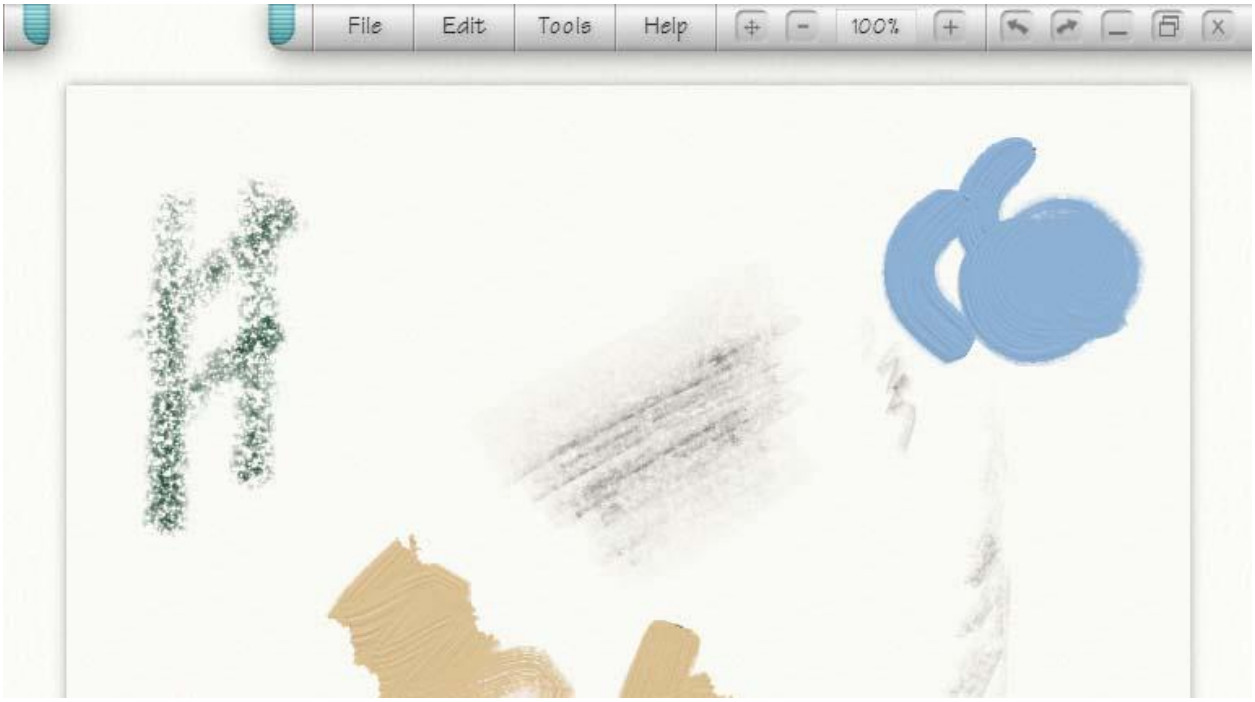

Figure 9. Art Rage Screenshot

1.4.8 Conclusion

In conclusion, today's existing software applications are just beginning to take advantage of the power and unique capabilities of the Tablet PC. There are applications for animating strokes by converting them to polygons and there are applications for creating modeled drawings in ways that resemble what is possible with paper and pen. The positives and the negatives of each of these attempts must be taken into consideration in the creation of a Tablet PC animation environment.

1.5 Statement of the Problem

Today's applications for creating visual models and animations often require overcoming high learning curves and even after passing the hurdle of learning the application, these software tools require time and several steps to build even the simplest of animations. Animations are effective tools for education and communication. As systems become more complex, their need becomes even greater for communicating ideas and concepts.

If the animation itself is conceptualized as the medium for communication, communication of an idea or concept between two people, then there is an immediate problem with today's traditional forms of animations. For example, words are conceptually a medium for communication of ideas and concepts as well. However when humans communicate ideas using words, the words can be modified, manipulated, and repaired in real-time, allowing the words to model the idea based on the real time feedback of the receiver [Koshik 2005]. Because of the time that it takes

to construct an animated model because of the nature of the software, animations are demonstrated, shown, but not able to be modified quickly in real-time based on feedback. Therefore, today's animations are conceptually comparable to a lecture of words without the ability to change those words by asking questions. In a real world scenario, a presenter would use the medium of words to compensate for misunderstandings an animation he or she is presenting.

1.6 Drawbacks of Today's Modeling and Animation Software

1.6.1 Timeline Based Systems

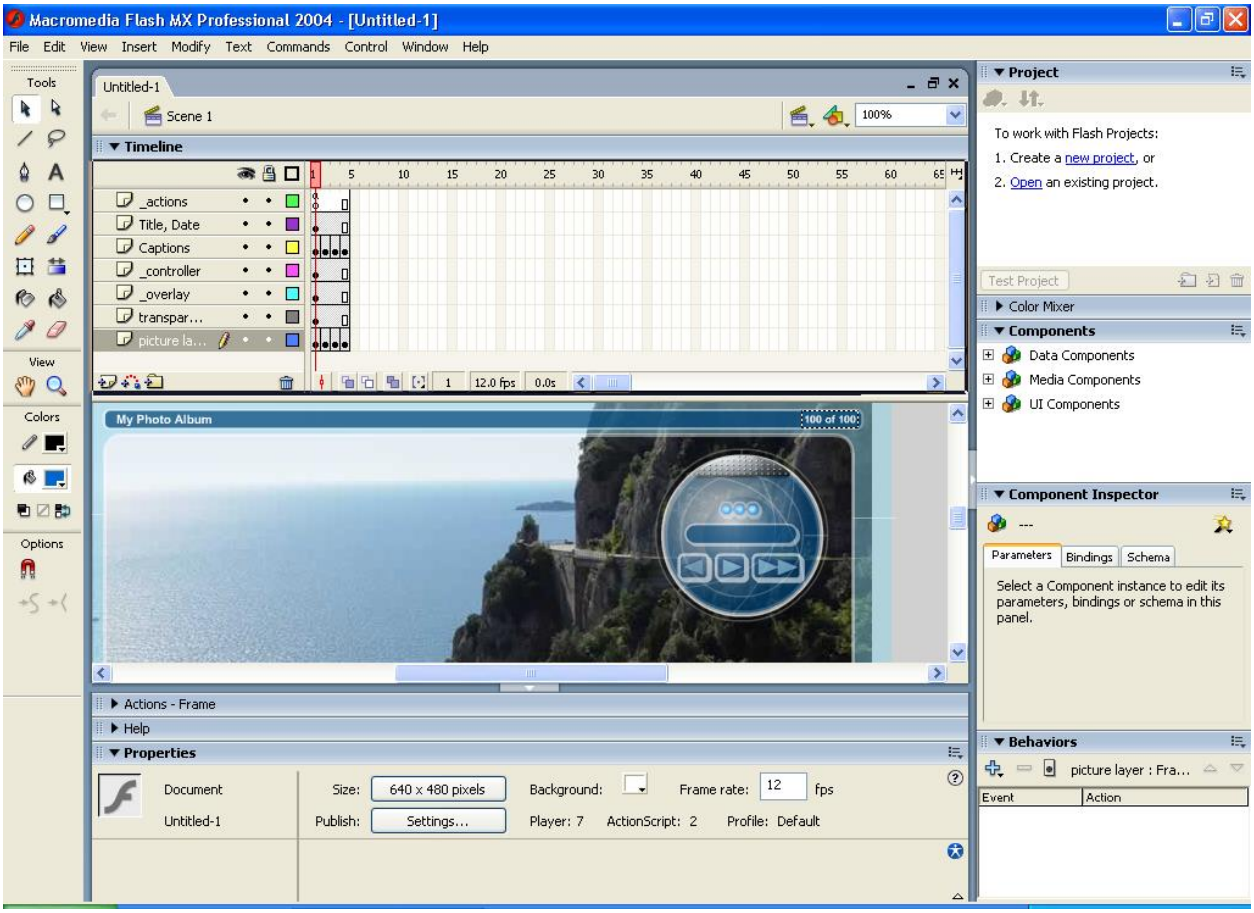

Figure 10. Macromedia Flash's Timeline System Screenshot

Most commercial animation tools are constructed around the concept of a timeline. By manipulating how an object moves or changes over a time line, specifying each frame of movement, an animation is created. This however is the point where complexity begins. By forcing the user to content with a complex time line no matter what type of animation they are attempting to create, the software tool is adding time and complexity to the creation of the animation.

1.6.2 Behavioral Based Systems

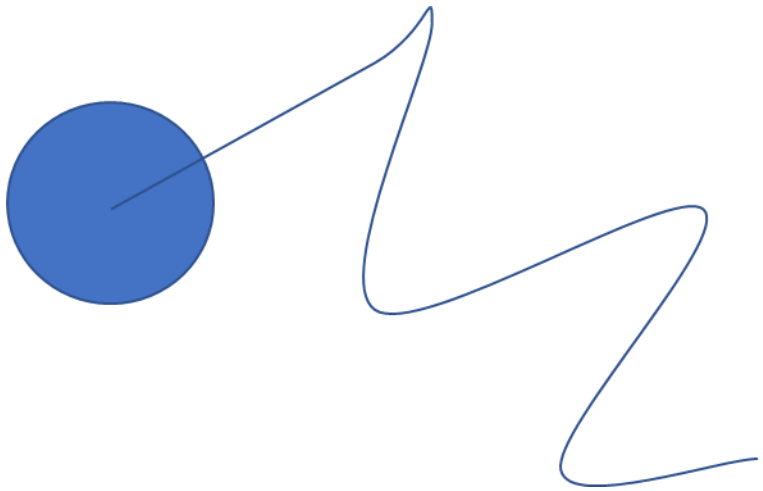

Figure 11. Example Visual of Apple's Motion Animation Environment

Another approach emerging in the academic community and commercially is the concept of behavioral based animation [Apple 2005]. Apple has one of the first commercial products on the market to coin this terminology, Motion, describing behavioral animation as allowing designers to create animation without fighting the time consuming nature and the complexity of key frame based animation that controls every finite detail of an animation [Apple 2005]. With behavioral animation in Apple's Motion application, animation is defined by dragging and dropping predefined movement objects onto text or an image or by simply specifying the path of an object graphically. This approach eases the creation of animation by allowing instant manipulation of a working end animation.

1.7 Statement of Objectives

The objective of this thesis is to annihilate the barriers of time for animation tools by leveraging the pen as an alternative interface to the modeling and animation tool. By decreasing the learning curve for the modeling and animation tool and by decreasing the time it takes to construct and modify an animation, animations take on the ability to be a dynamic medium for communication, allowing animations to be created on the fly in real-time, adapting to questions and people at hand.

The Model Pad application takes advantage of the simplicity of the Tablet PC and the fact that users immediately understand what has been a fundamental form of communication since the dawn of humankind, the pen and paper. By using the pen and paper paradigm as a means of interfacing the user with a graphical model, Model Pad eliminates the barriers preventing visual models to be used in a wider variety of scenarios, opening the possibility that visual models may be used in a real time scenario. Hence, instead of spending an hour or more constructing a visual model, time can instead be spend thinking and understanding the model. Editing that model and deleting it is as simple as paper because the pen is flipped and scrubbed to erase a model as if it

were paper. Being forced to construct such a model with merely a keyboard and mouse lessens the opportunities for accomplishing this task in real-time.

1.8 Overview of Thesis

Model Pad is a solution, but is only the first step in the right direction towards the final realization of a pen-based modeling and animation tool. This solution supports simple translation. As the environment adds rotation, sequences, and other advanced modeling and animation scenarios, much of the design expressed in this thesis would need to be expanded to enable support for this functionality.

In the remaining content of this thesis:

- Chapter 2 introduces the logic and concepts behind the design of the Tablet PC and the Ink Application Programming Interfaces (API) that allow for the creation of Ink-based applications like the software prototype discussed in this thesis.

- Chapter 3 provides information about the informal usability study conducted to construct an understanding of the targeted user for this thesis' software prototype and attempts to outline requirements for the prototypes eventual construction.

- Chapter 4 establishes the framework used to determine user interface considerations for the software prototype.

- Chapter 5 provides a high-level overview of the architecture for the software prototype, designating the prototypes name as Model Pad. This chapter also provides detailed design information of how the software prototype was constructed.

- Chapter 6 concludes the information discussed in the previous chapters and offers forward looking perspectives of where the research covered in this thesis can continue in the future.

This thesis concludes that using the pen and paths, as a model for expressing animation, is a more natural and simple approach then existing solutions. This solution enables people to construct simple models in real-time to demonstrate to others or for personal education.

Chapter 2: The Tablet PC and Ink API

2.1 History of the Tablet PC

The Tablet PC is the culmination of years of research and development in the area of pen-based computing.

2.1.1 The Beginnings of Pen Computing

Pen-based computing is not a new concept. Alan Kay in 1968 outlined his vision for a computer that was light, portable and simple to interact with because its interface would be completely accessible with the pen. This vision led him to create a prototype cardboard model of what he named the Dynabook. The Dynabook influenced many and became a central dream as Kay went on to carry his vision to Xerox Palo Alto Research Center (PARC) in 1972. There he helped influence and craft much of what we have come to know as the personal computer as he and his colleagues invented object oriented programming and graphical user interfaces at Xerox PARC. Much of what happened during those years has become a part of computer history.

It is well known that Kay and his colleagues, Adele Goldberg, Larry Tesler, and others, influenced the beginning of the personal computer industry, inspiring then personal computer industry leader Apple Computer to adopt and adapt many of the concepts and principles established at Xerox PARC to create the Macintosh computer. The Macintosh was the first commercial product to usher the personal computer industry into the graphical user interface revolution.

However, Kay's passion for pen-based computing and his prototyped Dynabook also inspired Apple. Apple began to create prototypes of a pen-based computer that would encompass the principles and concepts of Kay's cardboard Dynabook model. In 1987, they demonstrated to the world a video of their prototype that they called the Knowledge Navigator, but it took a few more years for Steve Sackoman, a leader in Apple's personal interactive electronics department to bring those ideas into a real product in 1992 with the Apple Newton. Even though there was continuing research in the area and other commercial products such as Robert Carr's PenPoint operating system, GRiD system's DOS based GRiDPAD, and the continuing research at Xerox PARC by Mark Weiser's ubiquitous computing division, the Apple Newton was the first commercial product with significant sales that began the era of pen-based computing. The Newton defined the capabilities that we have come to expect from the hardware and software of a pen-based computer. The Newton had the ability to track the position of the pen on the screen. The Newton had the ability to display ink strokes made by the pen on the screen with no constraints other than those created by the application. The Newton pioneered the ability to store handwritten ink, search ink, and translate hand written ink into shapes and text using an advanced handwriting recognition system.

This thesis does not discuss pen-based computing on handheld sized device like the modern personal digital assistant (PDA). Even though the PDA has been a central part of the history of pen-based computing, the focus of this thesis is on fully functional computers that have extended

to include pen functionality or on dedicated pen-based devices that attempt to be the size of a sheet of paper. The interactions and the purpose of full computing devices that attempt to be thin and the size of a sheet of paper are completely different then PDAs. PDAs have taken a minimalist perspective that has made most ink input only an alternative for a keyboard on a small device, not full drawing and design spaces.

2.1.2 The Apple Newton

The Apple Newton's user interface consisted primarily of sheets of paper that could be filed into folders. These sheets of paper were manipulated and edited with the pen and could contain graphics to represent fields and other types of user interface elements. A toolbar at the bottom provided options and access to applications, menus, and other options for applications running on the Newton. The Newton's operating system was called "Newton Intelligence." It was a fully 32-bit system with multi-tasking that was designed from the ground up to be for a handheld, pen-based device. This allowed Apple engineers to completely free of the legacy and baggage it would take to be based on the Macintosh operating system. This also provided Apple with the opportunity to create completely object-oriented API for the Newton using Newton Script, a language based heavily on Pascal and Lisp, enabling developers to create powerful Newton applications quickly and easily. The Newton also included full handwriting recognition that could interpret both print and cursive handwriting.

Error! Reference source not found.Error! Reference source not found. shows the final Ne wton form factor compared with the original form factor and that of a standard Palm based PDA (in that respective order). It is important to note that the later Newton models were larger than that of the original. Apple saw the Newton as a slate form factor device that could have full screen applications that could attempt to replicate the space and sophistication of paper. For this reason, it is one of the first commercially successful tablet form factor devices.

The Newton failed to become a market success even though it was more successful then other pen-based computing products of its time until the creation of the Palm Pilot. The Newton failed because of its horrendous handwriting recognition, software malfunctions, and a lack of an ability to easily connect the Newton to a desktop PC. Despite its failure in the consumer market, its legacy will be its definition of many of the concepts behind today's modern pen-based computer.

2.1.3 Jeff Hawkin's GRiDPAD

Jeff Hawkin's, whose fame would eventually be defined in his founding of Palm, originally was the driving force behind the Tablet PC's closest original kin, the GRiDPAD from GRiD Systems. The GRiDPAD provided no overall graphical operating system. Instead, it was constructed on DOS and GRiD Systems provided developers with development kits that allowed them to create full screen applications that would take advantage of the pen capabilities provided by GRiDPAD. Even though the GRiDPAD was never a major market success, its contribution to pen-based computing was vision. It identified a vision for the vertical markets that needed a full, paper sized, slate like computer that would enable them to work portably. GriD targeted their field

data collection warehouses, transportation applications, police, census takers, nurses, basically anyone who needed to use forms to collect information. Forms are the driving force between creating a device that would be the size of a sheet of paper. This device defined the market and the central inspiration for what later would become the Tablet PC. The keyboard was detachable from the display, allowing for mobile workers to take the screen which weighed approximately 4.4 pounds. With its size and weight, it was able to cater to the vertical market it identified even if it was not a commercial success.

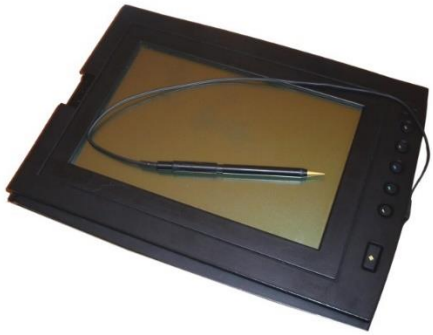

Figure 12. The GRiDPAD [Khadkikar 2011]

2.1.4 Microsoft Windows for Pen Computing

Microsoft's first entry into the realm of pen computing was with a product called simply Microsoft Windows for Pen Computing in 1992. Microsoft's original vision was to take their existing Windows 3 product, the graphical shell that ran on top of their popular DOS operating system, and extend it with DLLs that would add the ability for the operating system to recognize pen input as another system cursor with pen enabled drivers, alternatively to the mouse. They also created special applications such as PenPalette that would allow for all editable fields in existing applications to accept pen-based input. Microsoft also created a Pen API that mostly allowed developers to create larger text fields that could support handwritten input that would be translated to text.

Microsoft Windows for Pen Computing failed for many reasons. First and foremost, Microsoft's emphasis on enabling pen input for existing application by allowing ink to be written and translated in text fields was flawed because most text fields were designed to be the size of the characters being typed. This forced users to attempt to write in text fields that were too small to write in effectively. The other problem with Microsoft Windows for Pen Computing was the resource requirements forced hardware manufactures to build bulky, expensive systems that in no way matched the level of portability required for those users who would find pen input useful. Even then, the processing power available on the market could not process handwriting efficiently leading to inaccurate and troublesome translation.

Microsoft delayed upgrading their original release until they released Chicago, the code name for what would become Windows 95. After slipping their schedule with other feature work on

Windows 95, the pen features planned for Chicago were moved to a separate product renamed as Pen Services for Windows 2.0. Pen Services for Windows released for Windows 95 received little hardware manufacturer support or consumer interest [Rob Jarrett 2002]. The product eventually was cut and its core technology would be reused in future versions of Windows CE and the Pocket PC, but a fully pen-based Windows operating system would not reappear until 2001 with the release of Windows XP Tablet PC Edition.

2.1.5 The Birth of the Tablet PC

Bert Keely, a veteran of SGI, led an internal project in SGI to create a pen-based computer much like what would become the eventual Tablet PC [Rob Jarrett 2002]. SGI in the late 90s however was facing serious competition in the market, and was slowly moving away from low end workstations, preferring to focus their company's energy on maintaining their high end server customers. Keely knew that SGI would eventually forget and cut his Tablet project, so he left SGI in search of another company to back his ideas. He found a backer with Bill Gates who in 1998 asked Keely to join Microsoft's ranks to continue his work. Gates, an adamant supporter of pen-based computing, wanted to bring back to life the failed vision that ended with Microsoft Windows for Pen Computing. At Microsoft, Keely joined forces with Chuck Thacker, formerly of Xerox PARC, Butler Lampson, Microsoft Distinguished Engineer, and Charlton Lui and Dan Altman of Aha Software which recently joined Microsoft. These pioneers would become the driving forces behind the creation of the Tablet PC team [Rob Jarrett 2002].

Microsoft's approach to the Tablet PC was different than their approach with the original Microsoft Windows for Pen Computing release. Microsoft learned that in order for the software to be successful, there had to be standards that the hardware would meet to provide a consistent and capable interface for users. Some of the main aspects of these guidelines were that all OEMs would require hardware support for a digitizer, power states, dynamic viewing modes, and CTRL-ALT-DEL hardware functionality that is always accessible [Rob Jarrett 2002]. The digitizer is a touch sensitive screen that can detect hovering of a pen and support a sample rate of 100 times per second with a resolution of 600 points per inch, much higher than today's screen resolution. The power states that Microsoft stated needed to be supported was the ability to resume and standby in less than two seconds. Dynamic viewing mode requirements specified that the user be able to change from landscape to portrait screen orientations without requiring a reboot. And the final requirement for CTRL-ALT-DEL functionality through an always accessible hardware interface ensured that users could access Microsoft's secure authentication keystroke even without an actual keyboard present.

With the Tablet PC, Microsoft also took a different approach to software. The Tablet PC would have a richer Ink API that would not only allow for the creation of user controls that could receive and translate handwritten Ink, but it would also allow for developers to easily leave ink as a data type without translation. The benefit of this is that handwriting itself was able to be left as it is without forcing users to translate that handwritten ink to text to be useful to applications. This is a significantly different approach then previous attempts to create pen-based computers. Microsoft also created an Ink Input Panel that would allow the user to have a larger, floating panel that could be used for inserting ink into a text field as seen in Figure 13.

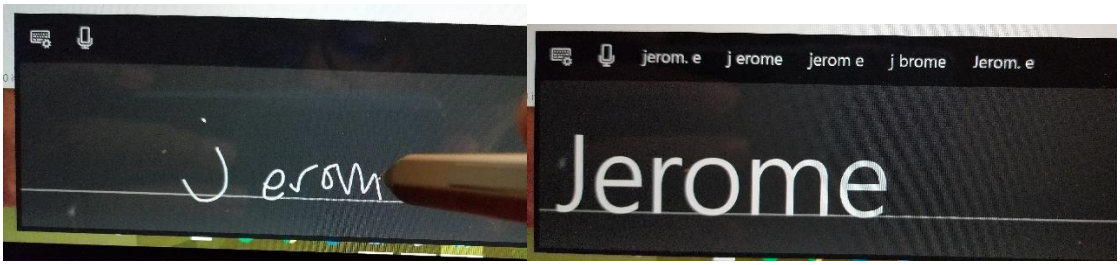

Figure 13. Photo of the Tablet PC Ink Input Panel

The input panel eliminates the interoperability problems with existing applications that Microsoft experienced with the original release of Microsoft Windows for Pen Computing when they forced the user to enter their handwritten input in the actual field.

The Tablet PC was released in 2001 as Windows XP Tablet PC Edition. Microsoft released the Tablet PC Operating System (OS) along with several hardware partners including Toshiba, Motion Computing, Gateway, Acer, and many more.

2.2 Tablet PC Hardware

The difference between Microsoft's previous efforts in the area of pen-based computers; the Tablet PC was a software product and a hardware specification. By Microsoft forcing Original Equipment Manufacturer (OEM) partners to follow a hardware specification, Microsoft ensured that the Tablet PC hardware would have consistency required to grow a software market around the Tablet PC platform. The core parts of this hardware specification were the specification of a digitizer, standby requirements, display modes, docking, and legacy free motherboard architectures.

2.2.1 The Digitizer

Microsoft required that all Tablet PC systems have a hardware digitizer for translating pen input into digital data. The digitizer is discussed at greater length in section 2.3 below. The hardware digitizer must support hovering and is required to have a sample rate of 133 samples per second or higher at 1,000 dots per inch (dpi) resolution. This high resolution requirement allowed Microsoft to ensure that pen input looked as realistic as possible on the screen [Microsoft 2004].

2.2.2 Standby Requirements

Microsoft's visions for the Tablet PC included that it should be turned on and off like an electronic appliance, instantly. Unfortunately, today's computers do not technically have that ability. Microsoft consequently required that resuming from standby require less than five seconds. By mandating this requirement, Tablet PC manufactures would enable users to use standby mode to closely mimic the instant on and off behavior of dedicated electronic devices with a computer system.

2.2.3 Display Modes

Whether the form factor of the Tablet PC was a slate without a keyboard or a convertible with a swivel display and keyboard, Microsoft wanted the Tablet PC to be able to easily switch back and forth between displaying things horizontally, landscape mode, and vertically, portrait mode. This requires hardware and software support. Microsoft required that this feature be supported on all Tablet PCs to ensure that all users would be able to use their Tablet PCs in the most comfortable orientation.

2.2.4 Docking

Many notebook computers running Windows that support being docked in a docking station require that the user select an undock option from the task bar before separating the notebook from the docking station. Microsoft recognized that the most likely dock form factors for Tablet PCs would be stands. Microsoft's Tablet PC specification specifies that all Tablet PCs must support "Grab and Go." "Grab and Go" is a standard for being able to quickly disconnect the Tablet PC from a dock, from the network, or external hard drives without warning. That way, literally, a user can grab a Tablet PC and run to a meeting without having to worry about errors in I/O operations.

2.2.5 Legacy Free Motherboard

The final specification that Microsoft requires for Tablet PC manufacturers is that all Tablet PC systems must follow the PC 2001 System Design Guide that Microsoft created for all mobile computers. For the most part, this ensures that the system will support USB technology, USB drives, and USB booting.

2.3 The Tablet PC Digitizer

The Tablet PC Digitizer is the technology used to convert pen strokes into digital input that can be used to "write" ink to a screen. Digitizer technology for the Tablet PC was designed to consume as little power as possible as well as make sure that the user can write directly on the Tablet PC screen without the risk of damage to the LCD. Handheld computers and notebook touch pads have popularized many digitizer technologies on the market. The electromagnetic digitizer is the predominant choice amongst Tablet PC manufacturers.

2.3.1 The Electromagnetic Digitizer

Most Tablet PCs on the market use a variant of the electromagnetic digitizer technology. Electromagnetic digitizers embed a coil under the LCD, receiving magnetic or radio frequency signals from the pen that allows it to detect the pen when it comes within a certain proximity and as it presses against the screen. This coil is very low power and cheap to manufacture. Because the coil can be placed under the LCD, the coil does not effect the screen's brightness or clarity in any way [Rob Jarrett 2002]. By being a coil that picks up on magnetic or radio frequency signals, this digitizer technology also solves the problems of false touch signals that could occur

with technologies that simply read pressure from the LCD screen. With electromagnetic digitizer technology, a user's hand pressing against the screen could never be misinterpreted by the computer as a form of input because the hand does not emit the proper magnetic or radio frequency signals. Figure 14 shows how the coil is embedded into the screen.

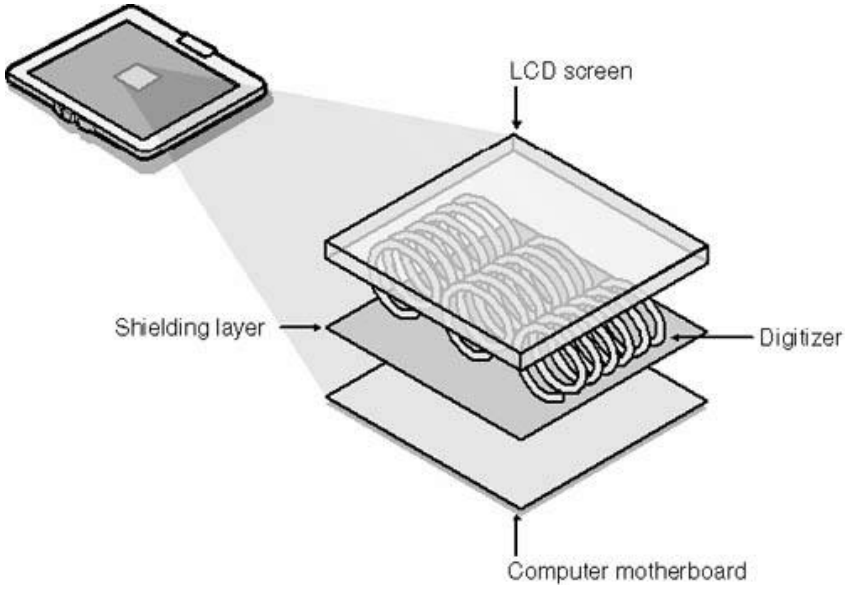

Figure 14. Cross Section of an Electromagnetic Digitizer [Rob Jarrett 2002]

As you can see in Figure 15, the pen's radio frequency or magnetic signal provides the digitizer with several pieces of information. The digitizer can detect tilt of the pen, the distance the pen is hovering from the screen itself, rotation of the pen (distinguishing the front of the pen from the back of the pen), and finally, pressure of the pen applied to the screen. This information combines to form a powerful set of input capabilities that allows Tablet PC applications to truly take advantage of not only the pen strokes themselves, but how the pen is being used at any given time.

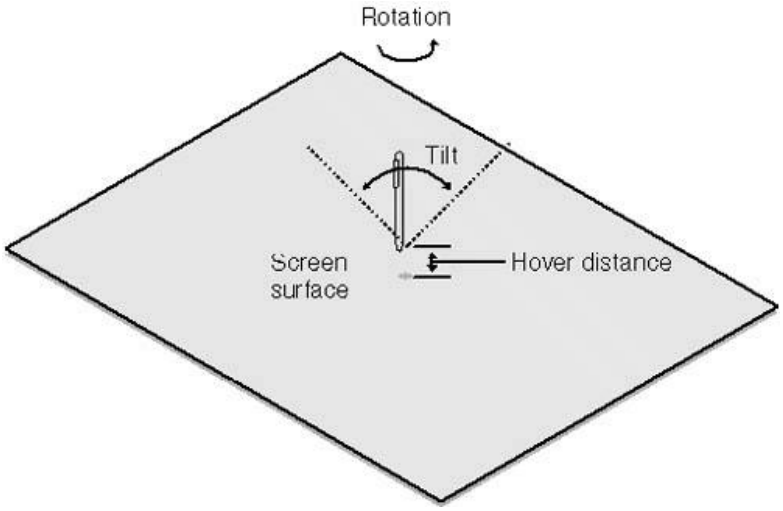

Figure 15. Four Axes of Pen Movement [Rob Jarrett 2002]

2.3.2 Digitizer Distortion and its Effects

There is a negative to using an electromagnetic digitizer. All electronics generate magnetic fields. That means that as a consequence, the internal circuitry of the Tablet PC itself generates enough electromagnetic fields to distort the digitizer's input system. All Tablet PCs are equipped with shielding that sits between the screen and the internal objects to compensate for this distortion, however, this still decreases the accuracy of pen input. The Table PC operating system corrects for this through calculating the anticipated distortion and compensating in software for what input should be received when the electromagnetic field distortion is accounted for. The user can tweak this calibration through using an alignment wizard that is included with every Tablet PC. Figure 16 shows a visualization of this distortion.

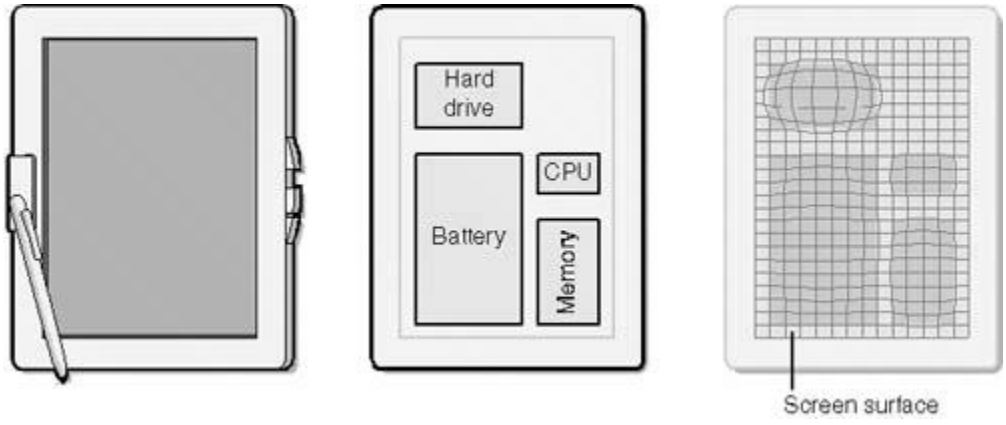

Figure 16. Distortion from EMF [Rob Jarrett 2002]

It is important to discuss this distortion effect because it directly correlates to the limitations Tablet PC applications have in the size and types of inputs that are acceptable. Icon size, click

target size, and button size all must take into account parallax. Parallax is the visual difference between where an object seems to be when viewed at an angle and where that object actually is. This visual phenomenon is only more important due to distortion caused by EMF and the fact that all Tablet PC hardware has a protective layer of glass or plastic that protects the LCD from pressure and the pen. Therefore, when you add light refraction from the fact that you are viewing an object displayed on the screen through a protective layer with parallax and the natural distortion of the screen surface due to EMF, then there are definite thresholds to how large things must be before they are impossible to click or differentiate with the pen. You can see a graphic demonstrating these factors in Figure 17.

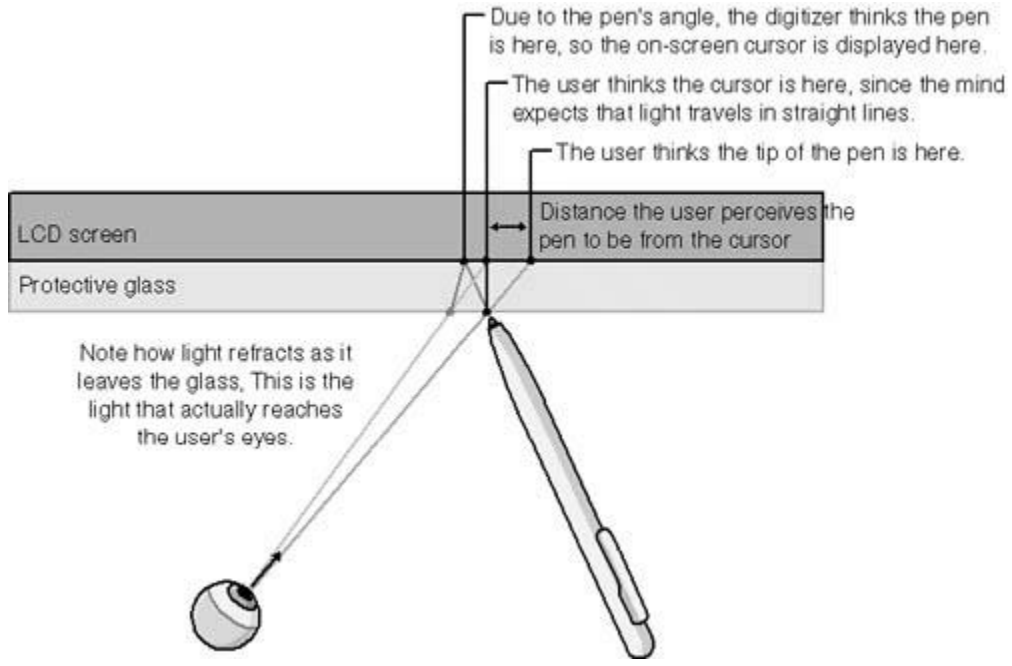

Figure 17. Visual Demonstration of Parallax [Rob Jarrett 2002]

2.4 Ink Technology

2.4.1 Overview of Ink

The philosophically different approach that Microsoft took with the Tablet PC compared to previous pen computing devices is that they made handwritten input into its own data type, Ink. With previous systems, handwritten input was used only as a means of translating that handwriting into text. If the actual handwriting was kept, it was for graphical applications only. With the Tablet PC, Microsoft came to the realization through usability studies that the handwriting itself should remain a form of input. This enables the user to create papers in handwritten form and then search their handwritten Ink for text strings without having everything converted from its handwritten form. This also enables a plethora of scenarios such as annotations that can be searched, Ink that can be copied from one application, pasted, and

translated to text in another application, and Ink that can be universally accepted as a form of input to any type of application.

2.4.2 Ink Realism

Microsoft attempted to give several properties to Ink to make sure that its representation on the Tablet PC would be as realistic as possible.

Ink Responsiveness

Ink is generated using a special high priority thread. Microsoft did this to ensure that the second the pen touches the screen Ink is generated. This ensures that as various processes take over CPU time on your system, Ink for the most part would flow regardless of your Tablet PC's current load.

Pressure Sensitivity

Another property that Microsoft implemented for Ink on the Tablet PC is pressure sensitivity. As discussed previously in section 2.3.1, the digitizers included on all Tablet PC systems have the ability to send pressure information back to the Tablet PC operating system. This pressure can be represented in the strokes that the user makes when they are inputting handwritten Ink. For example, Figure 18 demonstrates how a signed letter taters off at the end as the user begins to lift pressure off the pen and end the stroke.

Figure 18. Pressure Sensitive Strokes

This adds to the level of realism and the types of data applications can collect for Tablet PC Ink input. Applications like Art Rage (See 1.4.7) take advantage of this pressure sensitivity to provide rich Ink data for the user within the application.

Bézier Curve

Microsoft greatly improved the power of the Tablet PC platform for Ink recognition and realism by providing Bézier curve fitting. A Bézier curve is defined as a best fit interpolation of a smooth fitting line, determined by calculation a set of control points that will best fit a given handwritten stroke [Rob Jarrett 2002]. An example line can be seen in Figure 19.

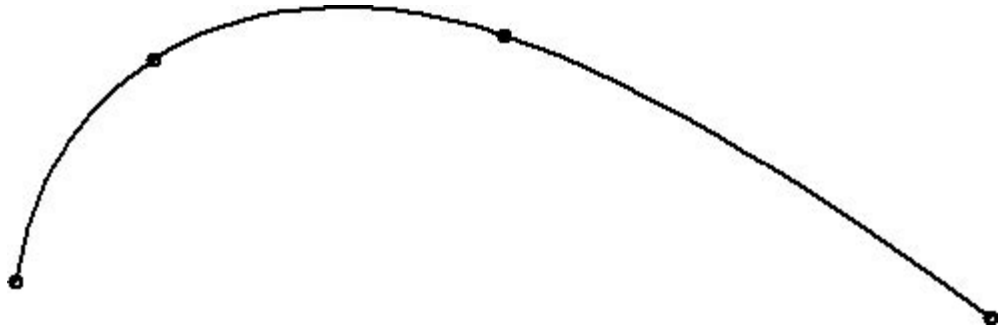

Figure 19. Bézier Curve

Bézier curves are important to understand because they are the fundamental basis of the Model Pad animation engine (5.4). Mathmatically, the Bézier curves can be determined given *n*+1 points P_i in R^3 a Bézier curve of degree *n* is a parametric curve

$$\mathbf{B} : [0,1] \rightarrow \mathbb{R}^3$$

composed of Bernstein basis polynomials of degree *n*

$$\mathbf{B}(t) = \sum_{i=0}^{n} \mathbf{P}_i b_{i,n}(t) \ , \ t \in [0,1]$$

with the Bernstein basis polynomials defined as

$$b_{i,n}(t) := \binom{n}{i} t^i (1-t)^{n-i} \qquad , i = 0, \ldots n.$$

P**i** is called control point for the Bézier curve. [http://en.wikipedia.org/wiki/Bezier_curve] This thesis does not go into an in-depth analysis of Bézier curves, however, it is important to understand the fundamentals of how they are determined.

Antialiasing

The importance of having smooth strokes displayed on the screen encouraged Microsoft to implement antialiasing for all Ink strokes. Antialiasing smoothes the edges of handwritten strokes and eliminates the jagged edges caused by the resolution of today's digital displays. This is extremely important for maintaining a level of realism with digital Ink.

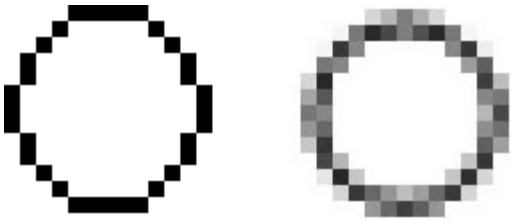

Figure 20. Antialiasing Example

Pen Tip Shape

One of the final properties of Ink that will be discussed in this thesis is Microsoft's creation of two different types of pen tips. Microsoft, understanding that users may want an assortment of pen tips to simulate highlighting and other alternative forms of handwriting, created to standard pen tips. They created the round and the rectangular pen tips. The round tip gives a simulated version of the ballpoint pen, and the rectangular pen, as mentioned above, gives the user a simulated feeling of using a highlighter pen tip. The interesting feature is that there is an understanding by the Tablet PC operating system of stroke inflection. This understanding of inflection uses the ability for the digitizer to understand angle to create a realistic turn of direction and angle with the round and rectangular pen tips [Rob Jarrett 2002].

Figure 21. Pen Tips

2.5 The Ink Application Programming Interfaces (API)

2.5.1 Introduction

As discussed in section 2.1, the creation of the first commercially successful object oriented languages closely aligned with the market craze for pen-based computing. Consequently, since the Tablet PC Ink API's origins begin with Microsoft Windows for Pen Computing, the API is heavily influenced by modern object oriented philosophies. Microsoft's purpose for the Ink API is to facilitate the creation of Ink-based applications that input and manipulate Ink data either as a form of input for applications, as a form of user interface manipulation, or as a gesture language to replace mouse or keyboard commands. This heavily influences the architecture of the Ink API. The Ink API consists of the Microsoft Tablet PC API DLL that can be accessed using Microsoft's Component Object Model or through the Ink API's managed interface [Gocinski 2004]. The managed interface allows the Ink API to be used in any of Microsoft's .Net languages using the Microsoft Command Language Runtime environment (CLR) [Gocinski 2004]. The focus of this thesis will be on the managed interface of the Ink API and primarily its use with the C# language.

2.5.2 Ink's User Defined Behavior

Because the Tablet PC software stack exists on top of the full Windows XP operating system, Microsoft ensures that developers do not require a large amount of programming effort to ensure their applications work on the Tablet PC. Ink manipulation and the ability to capture surface Ink, Ink that is written on a text a graphical area, requires the least amount of programming effort. As developers attempt to develop their own user interface controls or ink recognition and translation systems, the Ink API requires a great deal more work and code on the half of the developer to implement that custom functionality. Figure 22 provides a graphical visualization of the increased coding requirements of the developer for the different applications of the Ink API.

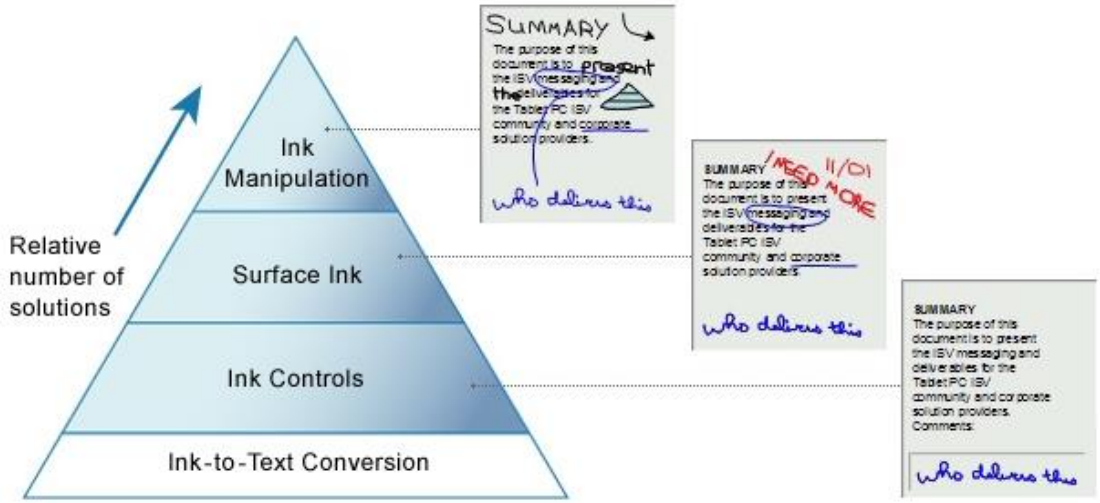

Figure 22. Levels of Ink Support and Degree of Effort [Microsoft 2004]

2.5.3 Ink Data

The basic operations that take place when the user begins writing in a Tablet PC enabled application begin with Ink collection. Each time the user touches the pen to the screen and begins writing, a stroke is created. A stroke is formally defined as a single uninterrupted movement by a writing instrument, such as a pen, that creates a mark [Mifflin 2000]. The Ink API upholds this definition and by starting to create a stroke when the user begins writing and ending that stroke when the lifts the pen for the first time, interrupting the stroke. A new stroke is created when the user starts another stroke. The Ink API defines stokes as a unique type is discussed further in section 2.5.4. For now, it is important to understand that a single handwritten line or a figure may consist of 1 to *n* strokes. The Ink API renders this set of strokes to the screen, applying all of the techniques to ensure its realism on screen. If the application collecting strokes is set to translate those strokes to text then the Ink API groups the strokes into individual letters using advanced Ink layout analysis algorithms [Microsoft 2004]. After the individual letters are recognized, the Ink API passes the information of the strokes determined to be a single letter to a built-in interpreter that will translate the Ink into a set of alphabetic or

numeric possibilities. This process is illustrated in Figure 23. If the application implements a custom Ink recognition system, then at the point where the Ink API attempts to analyze the layout of the written Ink, developers can instruct the logic of the Ink API's analysis so it can identify and create anything based on the Ink strokes such as shapes, musical notes, or whatever the developer prefers.

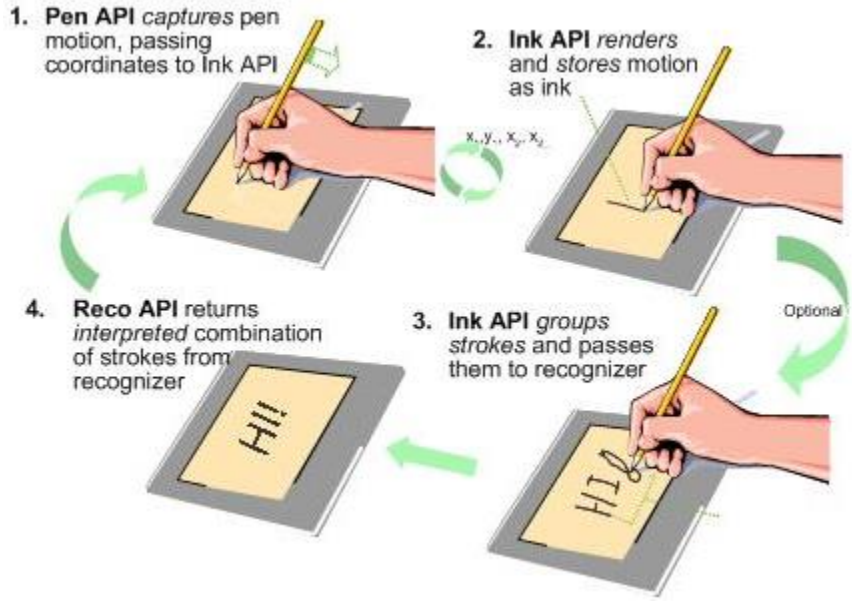

Figure 23. Process of Collecting and Recognizing Ink [Microsoft 2004]

2.5.4 Overview of the Ink API

The Ink API is divided into three main sets of functionality, Ink collection, Ink manipulation, and Ink recognition. This thesis focuses purely on Ink collection and manipulation since the current functionality of the Model Pad does not contain any advanced uses of Ink recognition. Figure 24 provides on overview of the managed objects that make up the Ink API.

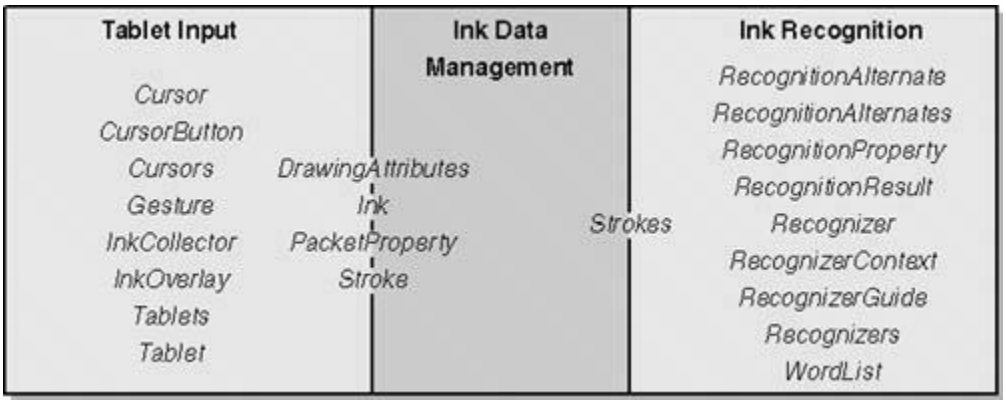

Figure 24. Overview of the Managed API [Rob Jarrett 2002]

2.5.5 The Tablet Input Subsystem

The pen, as it is represented by the underline operating system, is a real-time mouse. The pen is recognized by the Tablet PC operating system through its Human Interface Devices (HID) driver, like a mouse, running in kernel space [Rob Jarrett 2002]. Wisptis.exe is the heart of the input system, talking to the HID driver whether it be a mouse or a pen to broker input to Tablet aware applications [Rob Jarrett 2002]. Wisptis.exe gets its name from Windows Ink Services for Pen, the name of the second release of Microsoft Windows for Pen Computing [Rob Jarrett 2002], further proving that the Tablet PC is the evolution of decades of development.

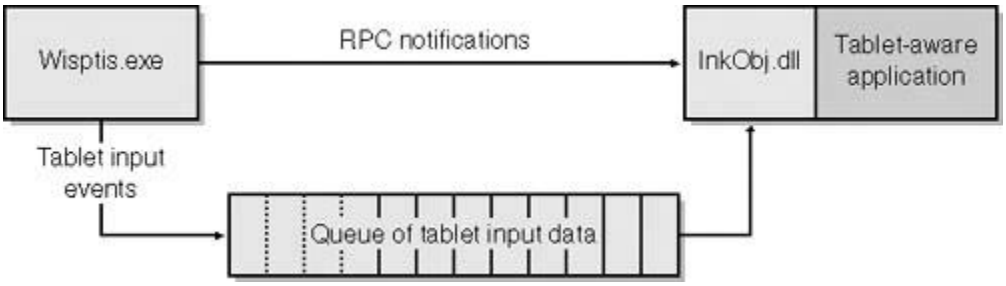

Figure 25. How Wisptis.exe Communicates [Rob Jarrett 2002]

Understanding Wisptis.exe is important to understanding how data packets are received by Tablet-aware applications. Implementing custom Ink stroke handlers like the one that this thesis presents in section 5.2.6 depends on being able to access this raw packet data. Raw packet data is transmitted from Wistis.exe that receives the data from the HID driver to the InkObj.dll that is running in the Tablet-aware application [Rob Jarrett 2002]. Wistis.exe accomplishes this by queuing values in shared memory using remote procedure calls (RPC). Wistis.exe then generates events so that the InkObj.dll knows that there is new data in the queue.

2.5.6 Ink Collection

InkCollector

Ink collection occurs as the user creates strokes using the Tablet PC. The *InkCollector* object is responsible for collecting incoming packets as *Strokes* [Microsoft 2004]. The *InkCollector* accomplishes this by attaching to a control or to a space and listening for incoming strokes from the user. Handwritten *Strokes* are collected in the *Ink* container object, a member of the *InkCollector*.

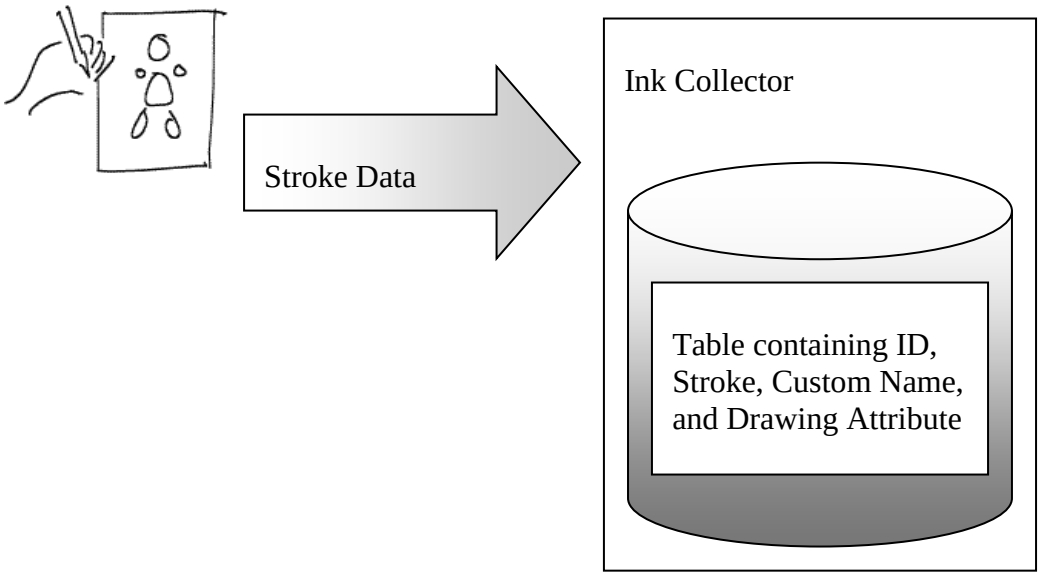

Figure 26. Ink Collector Diagram

The *InkCollector* is the central hub for all of the collected *Stroke* data. The user generates *Strokes* by writing on the Tablet PC. As the pen moves across the screen, *NewPacket* events are generated (*NewInAirPackets* if the pen were hovering without contact to the screen but in range of the Tablet PC digitizer). These *NewPacket* events contain the data that the *InkCollector* will use to create a *Stroke* object. This data is the data being sent to the application from Wisptis.exe.

Packets

Wisptis.exe fills the queue that exists in shared memory with integer values that represent the individual packets being generated from the moving pen. This queue is exposed as an array in the Ink API that is accessible through a *NewPacket* event.

The tricky part of parsing this data to use in an application is that because Wisptis.exe talks to any HID driver whether it be a pen, a mouse, or whatever other input device complies to the driver standards, the data is always different depending on the features that the HID device exposes. For instance, pens, in collaboration with their digitizers, may record and send pressure, roll rotation, and/or pitch rotation [Rob Jarrett 2002]. Because of the variability, it is the responsibility of the developer to specify the Desired Packet Description property to ensure that only the expected values are retrieved or the expected values are null when they have not been sent by the device. Figure 27 shows an example of the contents of the packet data that may be received in a typical Tablet PC aware application.

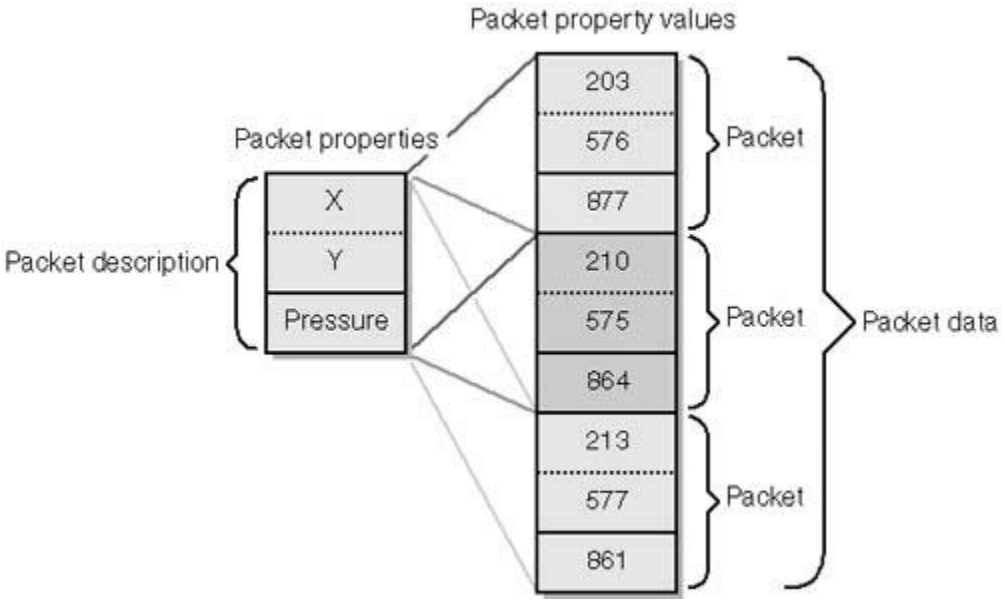

Figure 27. Packet Data Sent by Wisptis.exe [Rob Jarrett 2002]

Stroke Objects

A *Stroke*, as discussed earlier, is the fundamental, base component of Ink handwriting. An Ink drawn model or handwritten statement may consist of several tens of *Stroke* objects if not more. Strokes are stored in a container object, *Ink*, that the Ink API provides within the *InkCollector* class. Several classes extend the base *InkCollector* class to ease the development of Ink enabled surfaces. This thesis reviews one child of *InkCollector*, *InkPicture*, later on in section 5.2.7. *InkOverlay* is another popular child of the *InkCollector* class. *InkOverlay* allows Ink to be bound to any control such as a Windows Form or a Button in the .Net Framework.

When a *Stroke* object is created by the user, the Ink API assigns that *Stroke* a unique ID [Rob Jarrett 2002]. This unique ID is a positive integer starting at one that will continually increase by one for the existence of the *Ink* container. The deletion of a *Stroke* object does not have an effect on the ID of a new *Stroke*. The ID will still be the sequentially next integer after the last assigned ID. When a *Stroke* is divided by being partially erased, the half where the original *Stroke* was created will maintain its original ID while the second half becomes a new *Stroke*

object with a new ID of its own. Figure 28 shows an example application that actually writes the *Stroke* IDs to the screen.

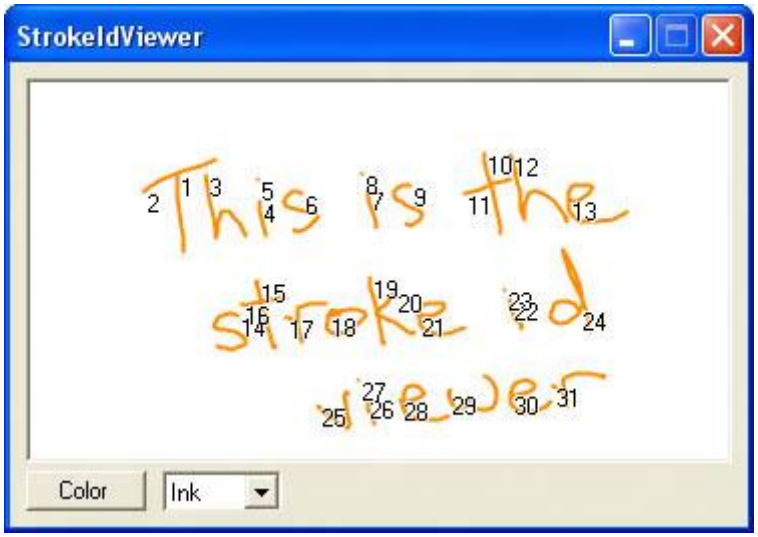

Figure 28. Screenshot Demonstrating Stroke IDs [Rob Jarrett 2002]

Strokes

At first glance, one may assume that this section is an error or somehow repeats the previous. That assumption is wrong. Microsoft has two completely different objects defined in the Ink API, the *Stroke* and the *Strokes* object. Where the *Stroke* object encapsulates and contains the data packets that make up a handwritten stroke of Ink, the *Strokes* object is a widely used temporary container that provides reference to a set of *Stroke* objects that have some relationship that the *Strokes* object represents. The easiest way to understand this is by looking at how the built-in *InkCollector* type handles user selection.

To ease in the creation of Ink based applications, Microsoft built the ability to lasso Ink to select it into the Ink API. The *InkOverlay* and *InkPicture* collection objects implement a property called *EditingMode* that allows developers to easily set the property to equal *Ink, Delete,* or *Select* to make all collected Ink strokes either write Ink to the screen, erase Ink, or write a special selection Ink that allows for anything contained within the drawn loop to be selected [Microsoft 2004]. When a set of *Stroke* objects are selected using this lasso technique, the selected *Stroke* objects can be referenced through a created Strokes object that is accessible through the *InkOverlay's* or *InkPicture's* Selection property. Figure 29 provides a visualization of the relationship between the *Ink, Stroke, Strokes,* and *InkOverlay* objects.

This thesis discusses the limitations of the built-in selection system further in section 5.2.6. It is useful to note here that the selection system that is a part of the *InkOverly* and *InkPicture* objects can be overridden. The Ink API provides this facility because currently there is no way to modify what the special selection lasso identifies as a selection. The current Ink API only allows

for the selection of Ink strokes and provides no facilities for selecting images like in the Model Pad environment.

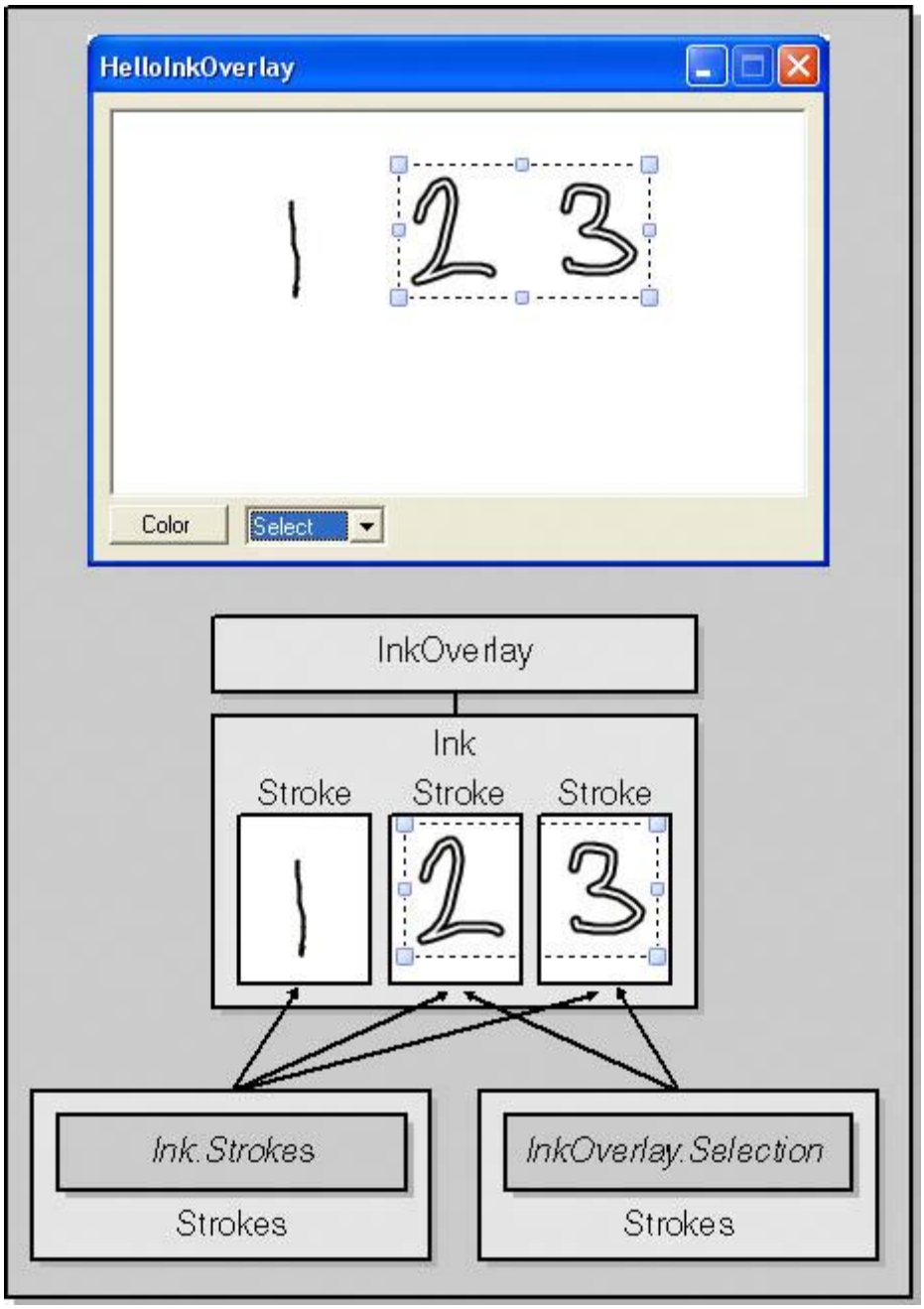

Figure 29. Demonstration of Strokes Object [Rob Jarrett 2002]

2.5.7 Ink Manipulation

Ink strokes can be manipulated programmatically. The two forms of manipulation that directly relate to this thesis is modifying the way the strokes are drawn to the screen and modifying a stroke's location.

A *DrawingAttributes* property is available in both the *Strokes* and *Stroke* object. Through this property, the appearance of *Stroke* objects on screen can be manipulated. The color, width, and transparency attributes are the only attributes that can be modified through this property [Microsoft 2004].

Strokes and *Stroke* objects expose transformation behavior as a part of their class definitions. Transformation allows for Ink strokes to be resized, translated to different locations, and rotated. Both classes implement *Scale*, *Move*, *Rotate*, and *Transform* methods that allow Ink strokes to be manipulated on screen [Microsoft 2004].

There is not a great amount of information that needs to be discussed to understand the purpose and application of this portion of the Ink API. However, it is extremely important to understand that manipulating Ink strokes is a relatively expensive operation because of the level of detail expressed within each Ink stroke displayed on the screen.

2.5.8 Ink Coordinate System

Ink on the Tablet PC has its own coordinate system. As this thesis discusses, the resolution of the Tablet PC digitizer is far higher than that of today's displays. Microsoft maintains that high-resolution data to approximate the precision on screen as closely as possible. The high resolution is also extremely important for Ink handwriting recognition [Microsoft 2004]. To do so, the Tablet PC uses what is called the HIMETRIC coordinate system. A HIMETRIC unit is equal to approximately 0.01mm [Microsoft 2004] based upon the current screen dpi.

The Tablet PC uses the same coordinate space as Windows where the origin is located in the upper-left corner of the coordinate space with the positive direction being right and down. Figure 30 shows a graph of this coordinate system.

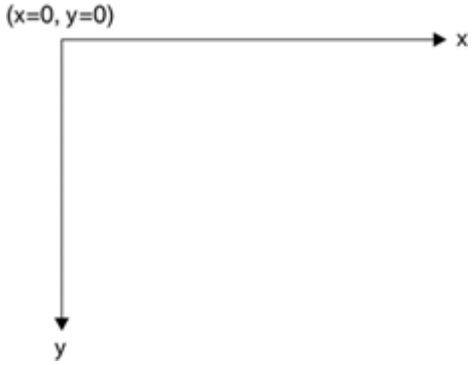

Figure 30. Microsoft Window's Fourth Quadrant Coordinate System [Chand 2003]

The Ink coordinate system is used for most of the methods implemented in the Ink API that transform and manipulate Ink. This coordinate system is used extensively in the implementation of Model Pad. The Ink API provides functions such as *InkToPixelSpace* and *PixelToInkSpace* to ease in the developer's translation of the two coordinate systems [Microsoft 2004]. Figure 31 provides an overview of the operations that take place when Ink is rendered to the screen using the two coordinate systems.

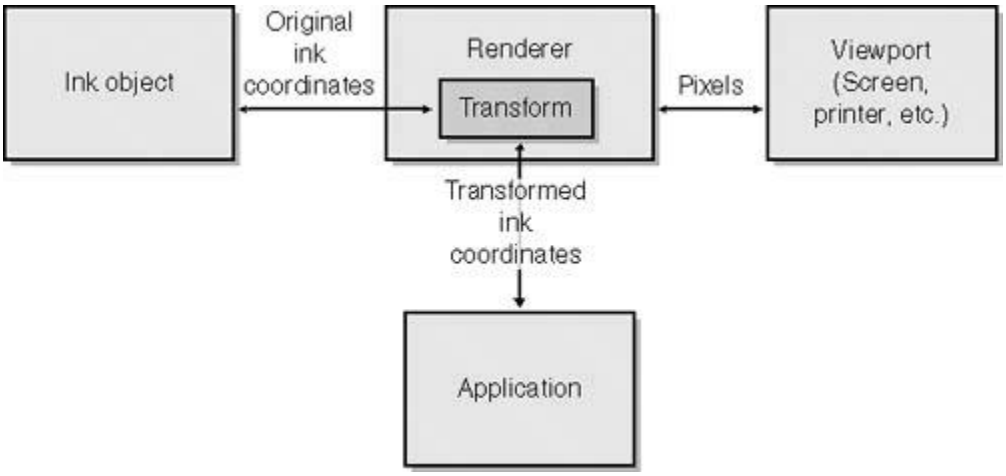

Figure 31. Renderer for Ink [Rob Jarrett 2002]

Chapter 3: Usability Study

3.1 Introduction

Before finalizing the design of the Model Pad prototype tool, a usability study was administered to better understand how people use drawing and modeling tools to build a simple system. Modeling tools have a plethora of applications. To isolate a specific area to investigate, the study only focused on observing graduate students attempting to use a modeling application to design a software user interface. The study took place as a part of a two-week CSCW (Computer Supported Collaboration and Work) course. The graduate students were selected randomly through asking for participants from the Computer Science student body here at Virginia Tech. On entrance to the study, students were asked to complete a survey that attempted to identify their background in drawing and modeling tools used for design and how they use those systems in their daily work. The student participants were then asked to use the Model Pad prototype tool as a design tool to design and develop prototypes for a user interface for a fictitious music player application. Their experiences were recorded and after the design was completed, the participants were asked to answer a follow up survey. The importance of this study is to understand insights into what is expected in a modeling application using today's technology and to gain insight on how the Tablet PC can be leveraged to improve the usability of this type of application. Because many of the scenarios this thesis defines as uses for the Model Pad prototype tool, a particular observation and interest in participant collaboration was also noted.

3.2 Experimental Setup

The participants were given the Model Pad prototype tool. Model Pad was installed on a standard Windows laptop with a mouse. The purpose of providing the users with a mouse rather than with a pen and Tablet PC was to clearly isolate and understand the current usability issues that are raised with modeling applications designed to be used with a mouse and keyboard. The groups were given an introduction to Model Pad, which at the time contained the ability to draw objects and define them, and the groups were then allowed to explore the tool for a few minutes to gain a familiarity with the application and its functionality. The participants were also provided and informed that they could and should use paper to complement their use of Model Pad. This was done with the intent of observing how a Tablet PC device might fit into a modeling and design scenario such as the one created for this study.

3.3 Task Definition

The groups were given the task of designing a user interface for a software media player that would play music files in the MP3 format. The time allowed for completion of this task was 20 minutes.

3.4 Participants

A total of four groups were observed. For the purpose of the study and to maintain anonymity, groups were named 1 through 4. Groups 1 and 2 had three members. Groups 3 and 4 had two members. The members of the groups were identified A through J.

3.5 Experimental Artifacts

Participants were asked to fill out an entry and an exit survey that would give us insights about their background and their experience working with graphical modeling and design applications. The entrance survey asked participants to identify whether or not they used modeling applications for design in their previous course work and their experience with drawing applications. The exit survey concentrated on the users' evaluation of their work on the assigned study task. The design sketches and any items produced by them during the session were collected. Their dialogue and gestures were noted down in the form of text transcripts of interesting behavior.

3.6 Observations

The most significant observations recorded dealt with how the groups collaborated to accomplish the design task and how in most instances they fought the tool to model their design in software.

3.6.1 Collaboration

Collaborative design mostly started by the participants mutually defining the roles that each would play in accomplishing the task. The participants would then identify the problem, elaborate the problem through brainstorming, and then rapidly prototype designs on paper. The participants would then use Model Pad to build a model of their final design. Usually one participant would be elected to handle all of the design work in Model Pad to translate a final paper prototype into a graphical model.

3.6.2 Draft Prototyping and Model Pad Interactions

Various issues arose while observing the participants work. It was extremely obvious that many teams chose to use paper and pencil to draft most of their prototypes. Even with encouragement to use the Model Pad application, the inherent complexity of drawing with a mouse led them to instead choose the paper and pen. The groups usually would quickly draw a sketch of the completed music player design, and then they would verbally go over the different parts and their functionality. From that, they would determine whether or not they were satisfied with the existing model and either erase it and begin again or consider it final and begin transporting the prototype into the Model Pad prototype tool. When using Model Pad, the groups struggled with creating precise components as they had drafted with the mouse. The error rate for drawing a straight line was quite high. There was also a high error rate of accidentally moving too far and drawing over a currently drawn part of the model. Attempting to correct that by erasing either led to more errors or extremely slow, tedious editing by using small movements of the mouse.

3.7 Interesting Observations

During the course of the experiment, many unique situations occurred. This section simply outlines those situations and their effects on the group's collaboration and design.

3.7.1 Group 1

This was the only group that did not use any media for prototyping other than the Model Pad prototype tool. In addition, the group produced the design that had the maximum number of features and defined objects.

3.7.2 Group 2

A different role-playing was observed in this group where the participant F who was handling the mouse contributed very little to the brainstorming, as compared to the other groups where the participant handling the mouse contributed most to the design features while brainstorming.

3.7.3 Group 3

The participants in this group had difficulty communicating verbally. They had to write down some of the words and figures on paper to make each other understand the ideas. This group also took maximum time to complete the task.

3.7.4 Group 4

This group took the minimum amount of time to complete the task. In addition, the final design had the least number of features. The participants mentioned that the task was of very low fidelity.

3.8 Survey Results

Participants were asked to answer a survey before and after their participation in the study. The following charts outline the responses to that survey.

Most of the participants responded positively to the survey question asking, "Out of the assignments you estimated above, qualify how successful you and your group were with accomplishing the assignments in general?" Out of choices from complete failure to extremely successful, all of the participants stated that they were moderately to extremely successful with their group projects.

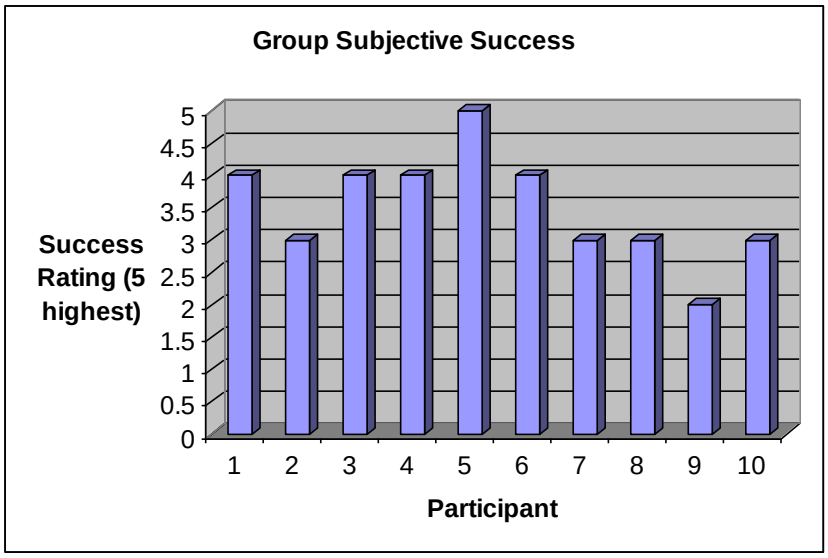

Figure 32. Subjective Group Success

There was an overwhelming positive response to the group work. Even though many of the participants had no previous relationship with each other and time was constrained, participants responded in the majority that they felt that their group assignment for the experiment was a success.

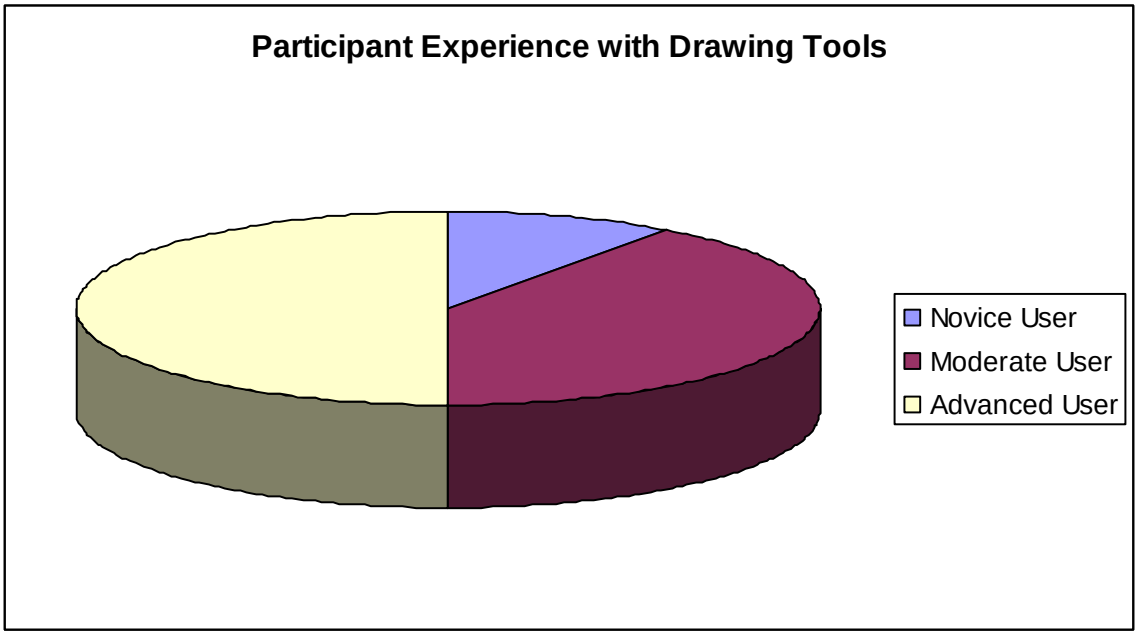

Figure 33. Participant Experience with Drawing Tools

The study collected information about the participant's experience with computer-based drawing tools to understand whether or not this may have had a correlation to how effectively they used the tool. Groups with members who were more experienced with drawing tools usually created better end designs using the Model Pad prototype tool.

3.9 Design Implications

The study demonstrates that paper and pencil artifacts are the primary way in which ideas are recorded, presented, and reviewed by a group. Without a question, whether it be through a computer software tool like Model Pad or with regular paper and pencil, it is extremely important for a group to share their ideas through drawing. All of these factors determine the success of collaboration and of the design. These results raise the question of whether or not a software application or a hardware device can support or help these factors to improve design collaboration. Can the Tablet PC provide the simplicity and elegance of paper while providing advanced modeling and animation techniques that would be useful to improving group collaboration and communication? Design and collaboration, as demonstrated here, is a complex and dynamic beast. For software to support design and collaboration in the best way possible, the software must provide an intuitive interface that promotes the simplicity of paper. The software must also allow experimentation by allowing models to be created quickly, modified quickly, and shared with others quickly.

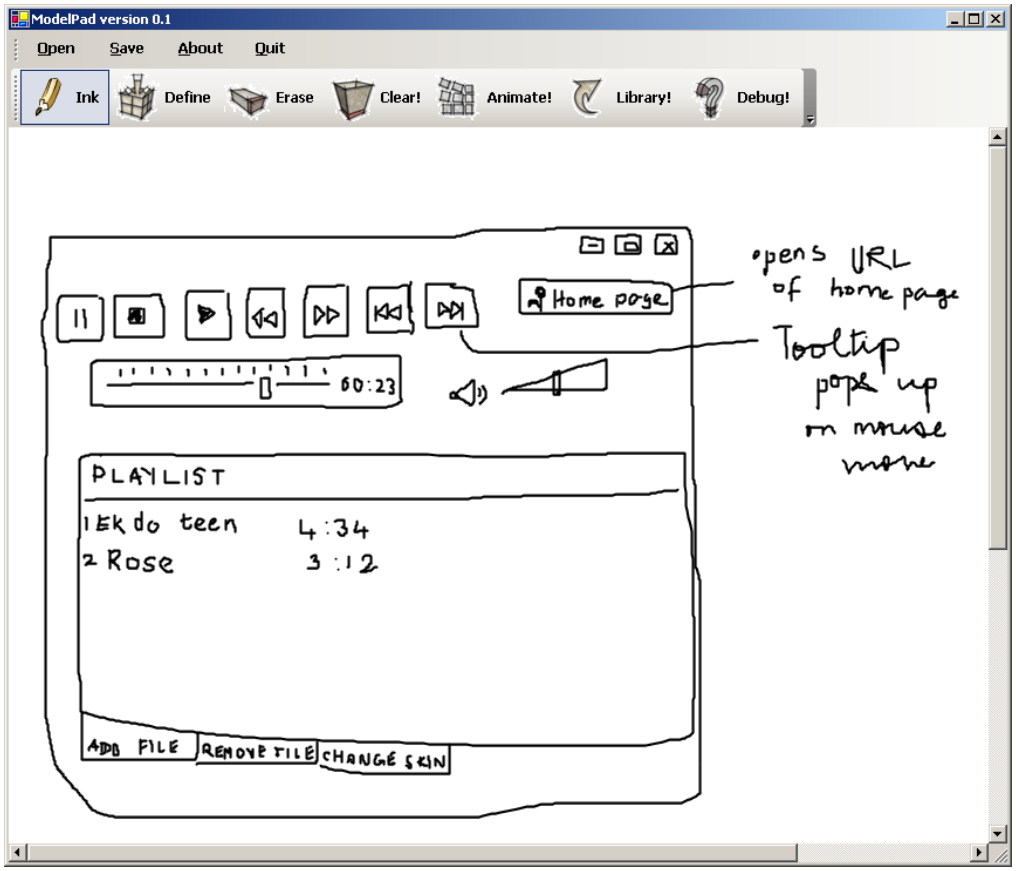

Figure 34. User Design Artifact from the Study

Chapter 4: Interface Design

4.1 Overview

The usability goal is to design a user interface for a modeling application environment that maintains the simplicity of pen and paper. Along with beginning usability evaluation, existing research and requirements heavily influence the eventual design of Model Pad. The following informal usability requirements form the basis for the design.

4.2 Usability Requirements

The environment shall leverage the pen tip and the eraser of the Tablet PC pen to ensure interaction with the application would never require more than simply writing, touching, dragging, and erasing. Using both the pen tip and the eraser also creates a sense of familiarity with the user, simulating the understood relationship between pen and paper.

The environment shall contain no action that will require the right click mouse operation. Right clicking with the Tablet PC pen is an awkward, unnatural interaction because most Tablet PC pen's implement the functionality with a small button on the side of the Tablet PC pen. Supporting right clicking would only increase complexity and would not add additional functionality to the application that could not be implemented with other means.

The environment shall not contain drop down menus that the user will have to select. Menus on the Tablet PC are complex to use for several reasons. The first is an issue called handiness [Rob Jarrett 2002]. Handiness represents the phenomenon of whether or not the user's hand will obstruct his or her view from the menu because to write on the screen, the hand must cover a portion of the screen. This issue is demonstrated in Figure 35.

Figure 35. Hand Obstructing a User Menu

Even though the Tablet PC corrects for this problem automatically, by drawing the menu on the opposite side of the user's set writing hand, the remaining menu is still difficult to target. This is because of the size of menu items and the ease in which the pen can slip into an area that Windows considers disinterested, consequently closing the menu automatically. This creates frustrating situations that often times require the user to click the menu several times before accomplishing a task.

The final requirement is that the environment shall have user controls that all have sizes greater than 6 millimeters. The Tablet PC SDK documentation [Microsoft 2004] reports that the smallest level of accuracy that is easy enough for the common user to target and hit successfully is approximately 3 millimeters [Microsoft 2004]. This fact combined with countless studies that have shown that using the pen, because of the nature of the way we write, leads to difficulty hitting small target points for menus and buttons. It has been shown through using children subjects that there are inherent difficulties hitting small target points [Read *et al.* 2004] and that the only solution is to ensure that an application design truly has targets that can be hit easily by the user. Six millimeters, twice the size of the smallest target size, is a conservative estimate and requirement to ensure that all buttons are accessible with ease using the pen.

4.3 Interaction Design

4.3.1 Modal Toolbar

Artists often use palettes to hold their oils that they then paint onto a blank canvas. Model Pad leverages that simple paradigm with the creation of a single universal bar. This toolbar that aligns across the top of the Model Pad environment serves the same purpose. The difference is that instead of a palette of colors, the tool bar acts of a palette of modes and behaviors that the

user can use while constructing a model within Model Pad. Each mode changes the way Ink operates and ends when the pen toggles another mode through the toolbar or moves out of the range of Model Pad.

There is another menu bar above the main tool bar that acts as the point of access for environment commands. As stated in the requirements (4.2), there are absolutely no drop down menus in the Model Pad environment. Simply clicking the text in the menu bar will initiate the operation. For operations that drastically alter the state of the environment or the model, Model Pad displays a confirmation dialog to ensure that the menu item as not selected by accident.

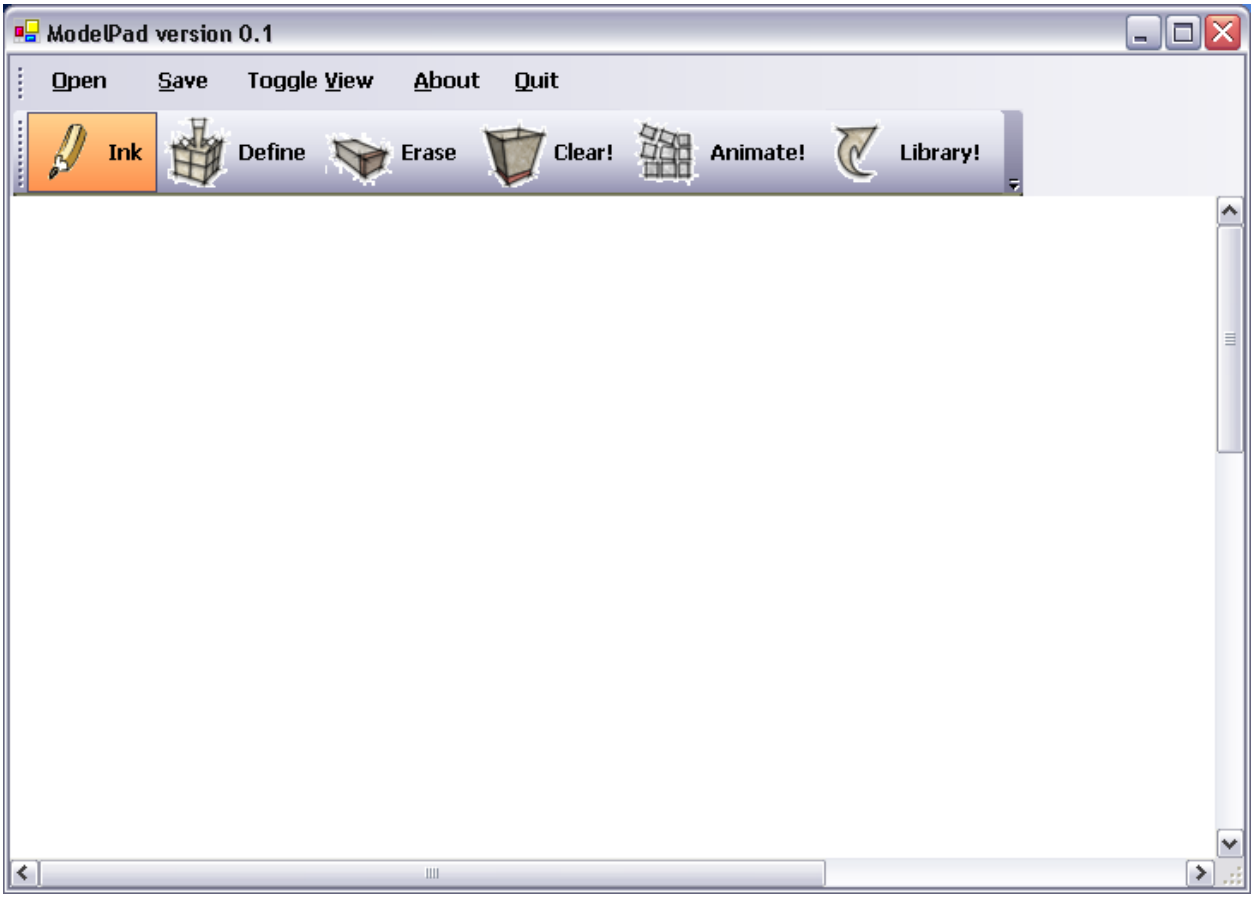

Figure 36. Main Model Pad Window

4.3.2 Ink Selection

Model Pad generates a dashed semi-transparent Ink when in definition mode. This mode represents the selection mechanism of the application. From this mode, Ink strokes and image objects can be moved, defined, and edited. The user selects an object by drawing a circle around the Ink strokes or image. When the stroke is complete, any image or Ink stroke within the circle will be automatically selected. To edit an object, the user clicks the context button (4.3.4). To

move an object, the user simply can press the pen down anywhere within the selection and drag the images or strokes to a new location.

4.3.3 Drag and Drop Duplication

As it is discussed in section 5.3 of this thesis, objects can be defined, becoming a component in the model. Their definition adds them to the definition bar located at the right of the screen. This definition bar acts as a second palette that serves two purposes. For one purpose, this bar allows for defined components to be quickly selected without requiring the user to always draw a line around a component to select and edit that component. This maintains the goals of simplicity and speed. To duplicate a defined component, the users only has to drag the representation of that component from the definition bar into the model. This creates a new set of strokes or a new image that can then be used in the model and later defined.

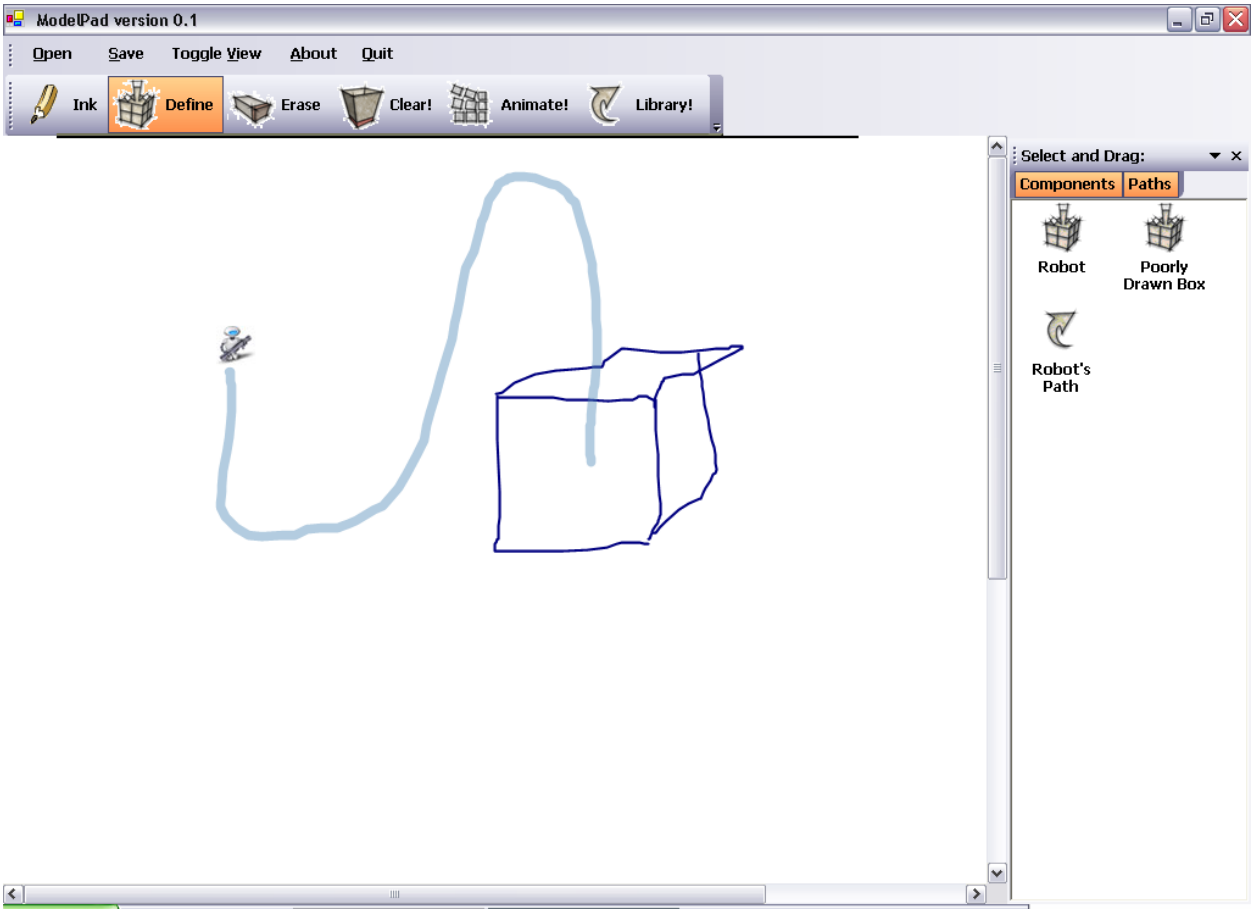

Figure 37. Example of Model Pad Definition Bar

4.3.4 Context Button

The context button provides the ability for an action to take place on any selection. Whether it be defining a component or path, or editing a component or path, the context button shows the possible action for each selection. This provides a simple method for having actions that take place in a given selection context that are easy to access and immediately apparent of their purpose. This also satisfies the requirement of not using the right click operation.

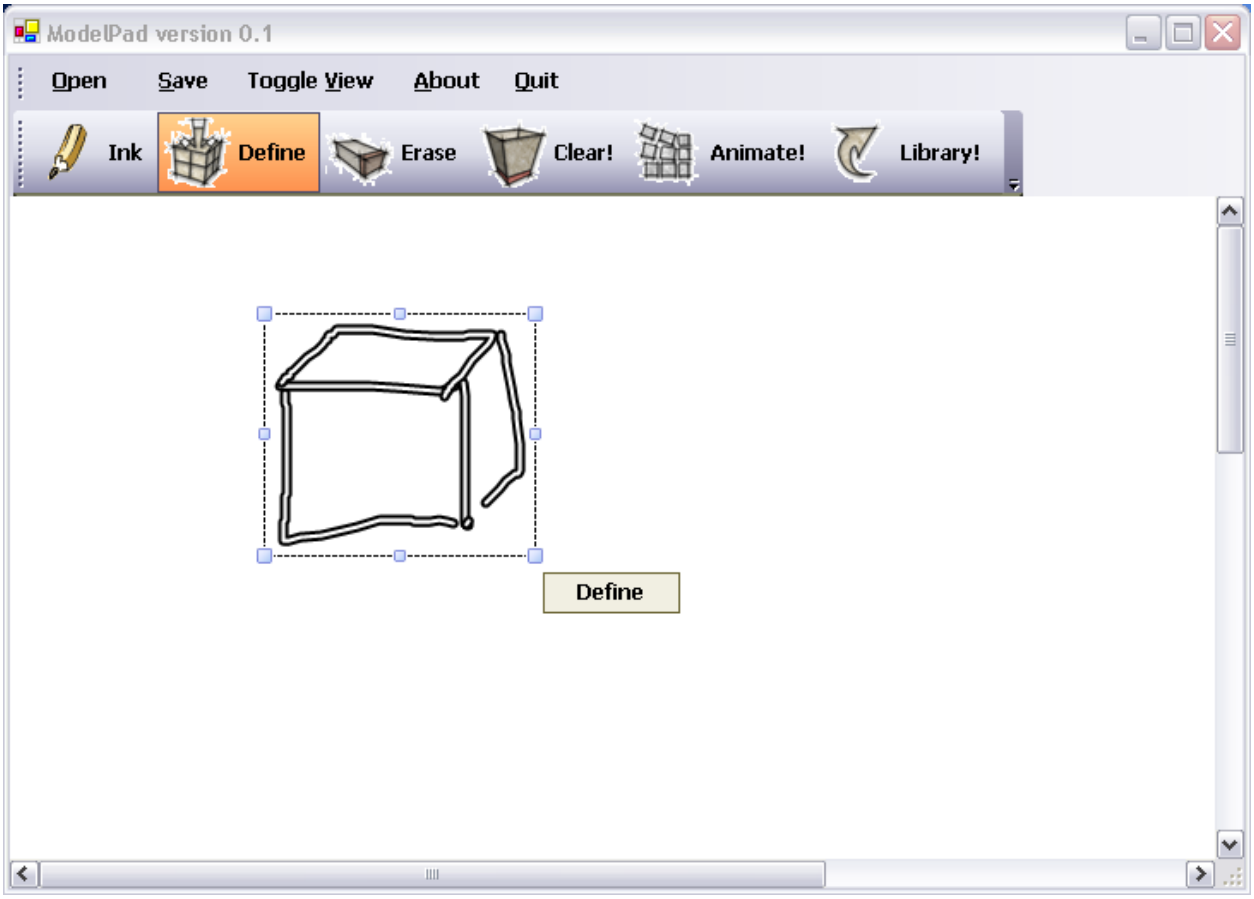

Figure 38. Selection Context Button

4.4 Path Based Animation

4.4.1 Overview

The goal in the design of the Model Pad prototype is to create a tool that allows a user to animate a drawn model quickly and simply. To accomplish this, the previous paradigm used in today's animation tools of creating each frame of animation and showing how it moved over time did not suit this goal. The concept of drawing a path and animating the model based on that path allows for the creation of a design that meets the goals of quick and simple.

4.4.2 Previous Research

Stasko [Stasko 1990] revolutionized thought behind how to construct animations with his paper on the creation of a path-transition approach to creating animations. His work outlined a way to simplify tools for creating animations by using formal semantics as the basis for the interface. His paper described how to use locations and images as a form of objects that move along transitionally using paths defined by the user [Stasko 1990]. Paths defined the semantics of the animation. Simplifying logic into this simple set of structures allowed for the creation of precise

semantics based on the user's intention of movement that could then be interpolated by the computer to form an animated sequence. His worked formed a framework for understanding how to create animations based on paths that naturally aligned with drawing paths using Ink and the Tablet PC.

4.4.3 Application

The Model Pad described in this thesis, influenced by Stasko's [Stasko 1990] work, allows the user to draw a model, create a model from a reusable image library, or create a model from a combination of drawings and images. Each Ink stroke or image or set thereof can be defined to create a component. These defined components are the moving objects in the animated model. After the model is created and components are defined, the user may draw paths that describe where these components move in the model. By simply drawing these paths and binding the paths to components that exist in the model, the user creates an animated model that can be played back.

Chapter 5: Model Pad Design and Implementation

5.1 Hardware and Software Environment

Model Pad was designed using the Toshiba Protégé 3500 Tablet PC. This convertible laptop provided the ability to test in landscape and portrait mode, with and without a keyboard.

5.2 Model Pad Architecture Design

The design of Model Pad took advantage of the Model-View-Controller design pattern. A set of Windows forms control the user view. Those forms communicate to the Object Model in the backend that stores all of the data and relationships contained within a given model. The controller acting between the model and the view is the Model Manager.

5.2.1 Model Manager

The Model Manager is the controller class that connects the view, the main windows of the Model Pad application environment to the backend data model. The Model Manager maintains the behavior that takes place as events are generated on the Main Window, sending data and commands to their respective parts as necessary.

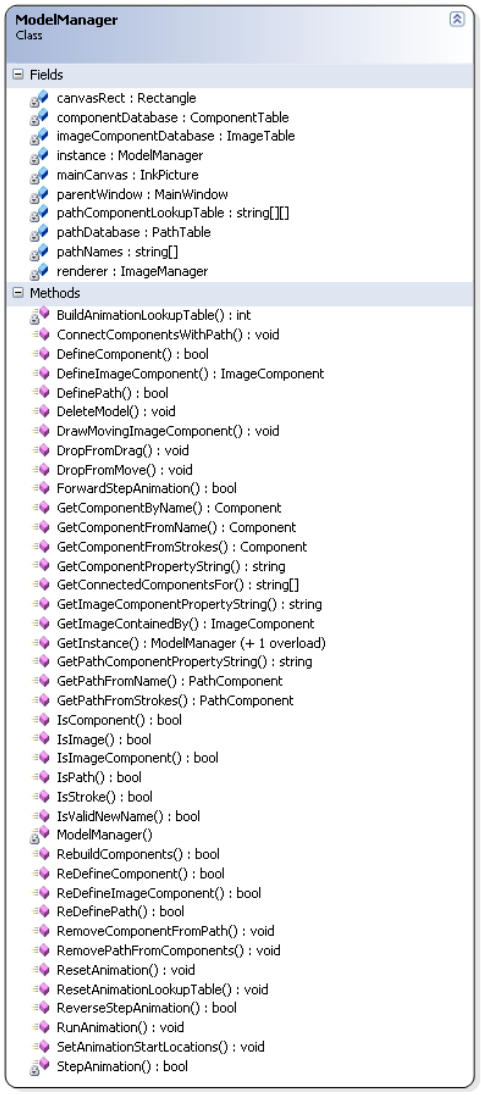

Figure 39. The Model Manager

5.2.2 The Windowing System

The windows that make up the Model Pad environment all derive from the Windows Form class library. These windows are responsible for receiving and validating end user input through events and passing those events on to the Model Manager. Each window uses the Singleton pattern to gain access to an instance of the Model Manager. From that instance, the windows pass data and events to the Model Manager for it to handle accordingly.

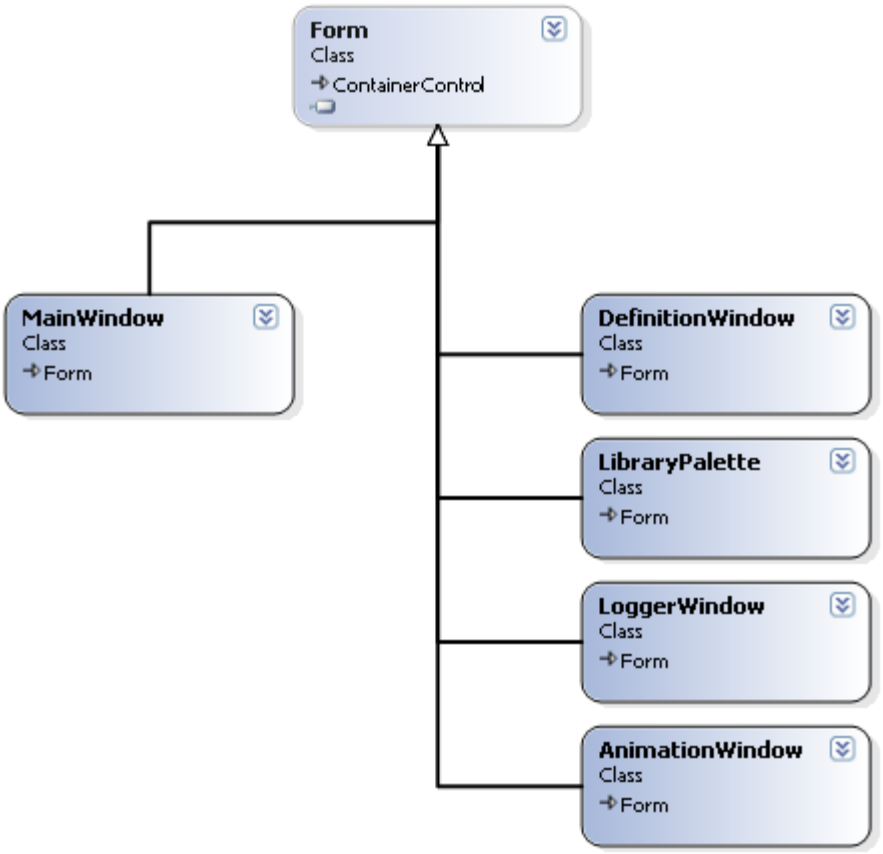

Figure 40. Windows that Compose the Model Pad Environment

5.2.3 Backend Data Model

The backend data model for Model Pad is based on a hierarchy of classes. A universal hash table is maintained in the Model Manager that provides references to the data instantiated throughout the lifetime of the Model Pad tool. This data is contained within the Component classes. Every defined object in Model Pad are represented by Component objects. This simplified representation allows all defined components and paths to be manipulated and represented in the same way. This eases lookup facilities and unifies storage.

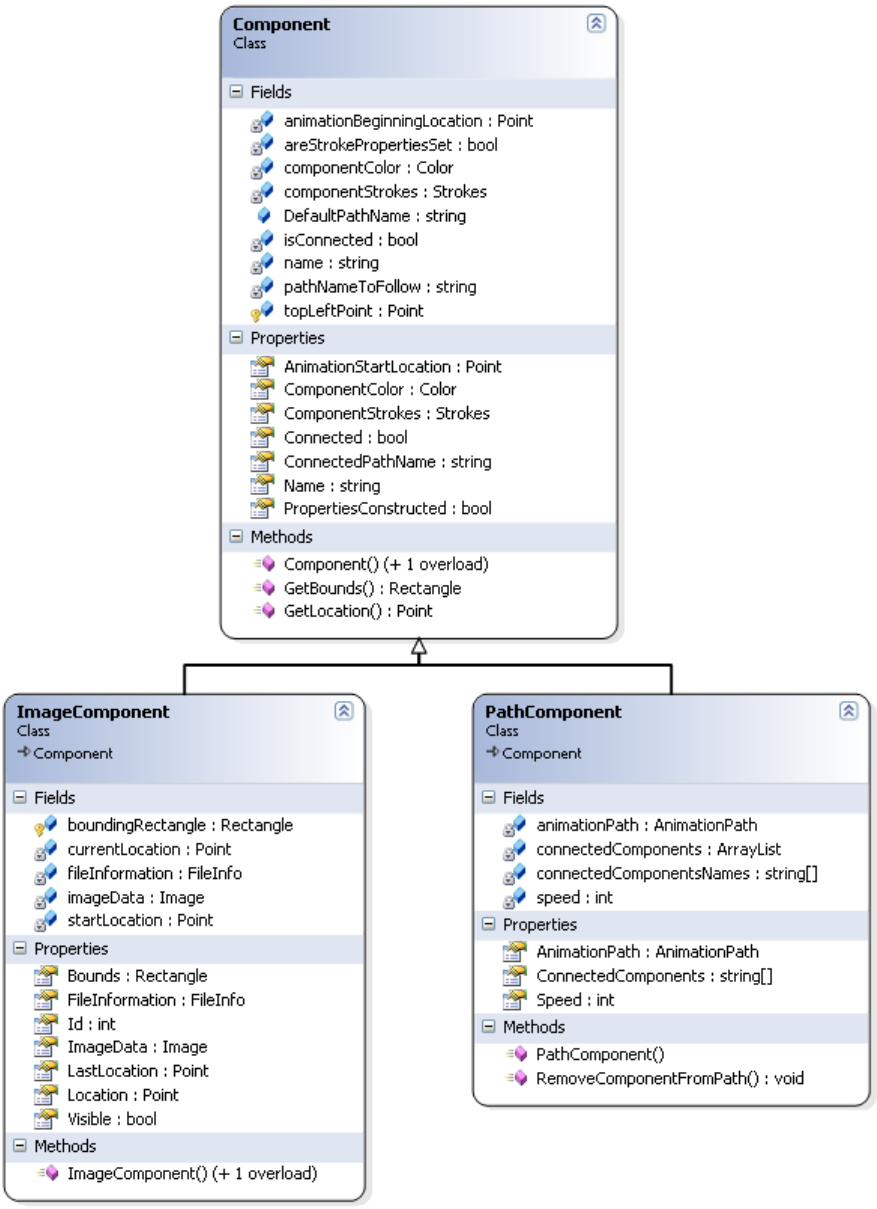

Figure 41. Backend Component Data Model

5.2.4 Backend Data Model – Hash Lookup System

The hash table described in section 5.2.3 actually consists of an orchestration of hash tables that occur in the Model Pad Manager. These hash tables isolate the different types of defined components even though the components maintain their own unique types. There are three hash tables that maintain all of the data within the application. There is a hash table for image components, a hash table for stroke based components, and a hash table for paths. This was done

to ease lookup load and compartmentalize the types with extremely different representation and behavior in the system.

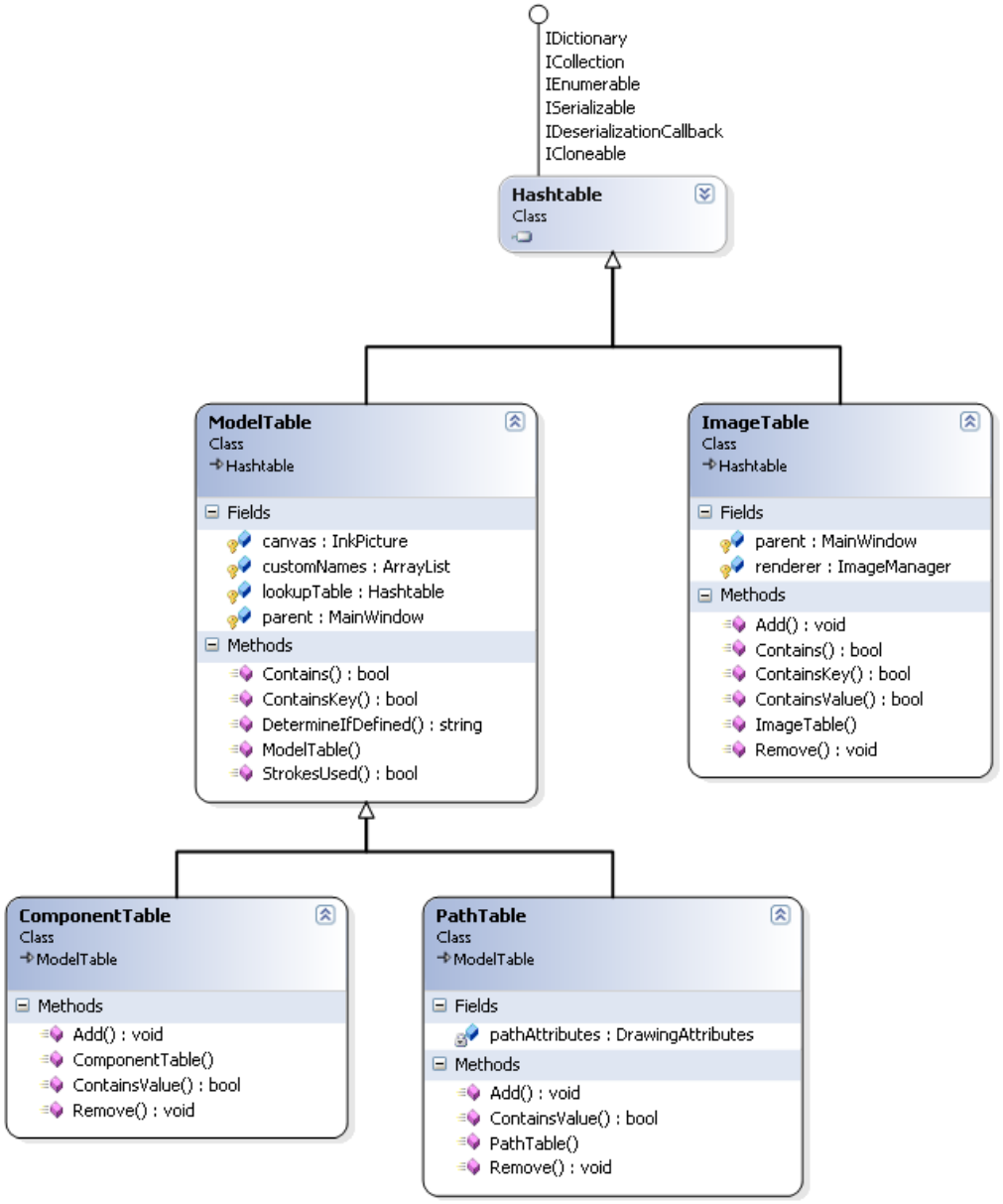

Figure 42. Hash Storage System

5.2.5 Animation Paths

Animation paths are the basis of movement in Model Pad. Animation Paths base their path on stroke data, calculating the Bezier line that represents the Ink stroke. From that line, they

establish a set of points that are then interpolated between to represent the track that the components bound to that path will follow.

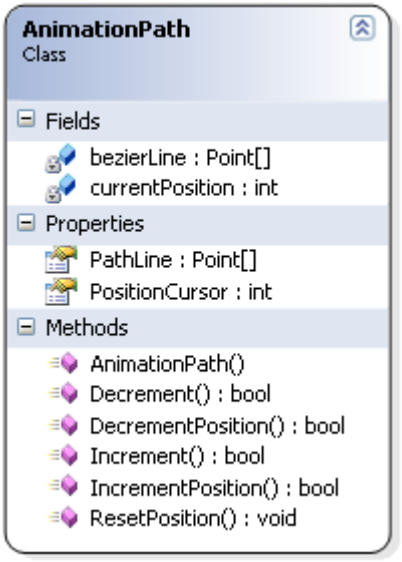

Figure 43. Animation Path

5.2.6 Selection Engine

Model Pad implements its own custom selection engine because of the lack of a facility to select images in the built-in selection engine. Model Pad accomplishes this by intercepting incoming packet data, manually drawing the lasso for selection, and calculating the bounds of that lasso and determining if any of the strokes or images in the system fall within those bounds. Once that is determined, if something is within those bounds, then it is assigned to the Model Selection type. The Model Selection type provides a simple, unified way to represent whatever user data is selected. It contains two separate types for when it contains strokes versus when it contains images. The Model Selection type expose methods that allow the application, mostly the Model Manager to understand the type that is selected and handle it accordingly.

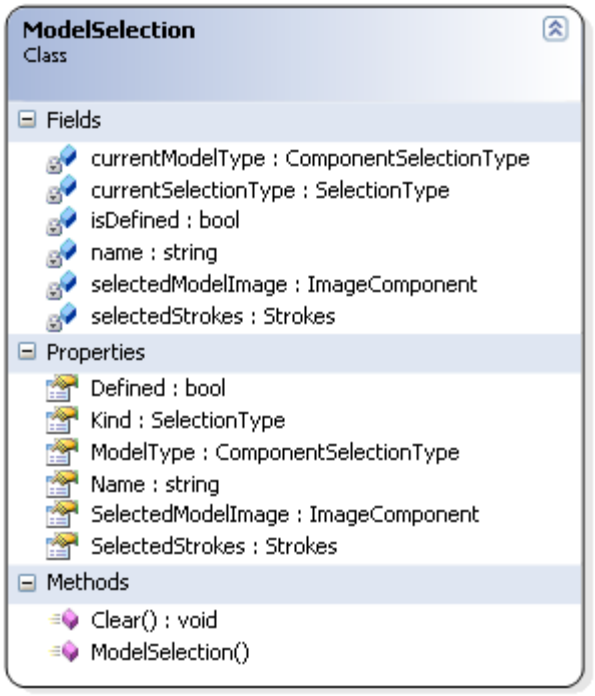

Figure 44. Model Selection Type

5.2.7 Ink Picture Control

Ink picture control is the Ink Collector type used by the Model Pad prototype tool. Ink Picture is the preferred class for collecting Ink because of its built-in facilities for saving Ink as a JPEG which is extremely useful for building a reusable library based on image data.

5.2.8 Rendering and Off-screen Buffering

The amount of data that must be manipulated each time Ink or images are transformed on screen required a buffering system for having a buffer that is manipulated under these stressful calculations that is then swapped to the screen. Writing directly to the screen only produces choppier animations because of the time complexities of manipulating Ink in real-time.

The Off-screen Buffer basically maintains a reference to an off-screen buffer that is used for Ink and image manipulation. When it is time to render that Ink or image data to the screen, the Model Manager passes a GDI+ reference to the Off-screen Buffer through the Render method.

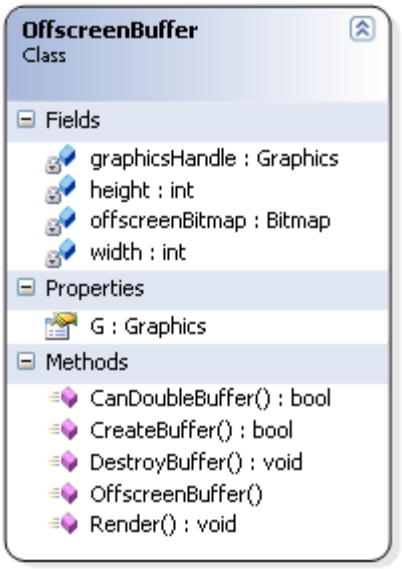

Figure 45. The Off-screen Buffer

5.2.9 Image Rendering

Image rendering is a complex task. There is no facility for maintaining image data in Ink controls. The Image renderer implements a homegrown solution for accomplishing this behavior. By instantiating a false JPEG background image to the Ink Picture Control, the Image Renderer takes a reference to that false JPEG and uses it as an output mechanism, rendering to the Ink Picture Control background image. This has several advantages because background images are saved with the built-in Ink serialization system. However, maintaining the original image data is a challenge because that data is lost on saves. For this reason, the disadvantage of rendering images this way is that the saved model will always be directly tied to the file paths of the originally imported images.

The image renderer used a stack-based algorithm to redraw images. Based on first come first serve order, the image renderer would redraw the regions that were to change during a repaint of the Ink Picture control based on this stack order. Movement is also based on this stack order. When an object moves, it is lifted out of the stack. When an object is dropped, it returns to its location on the stack and renders accordingly.

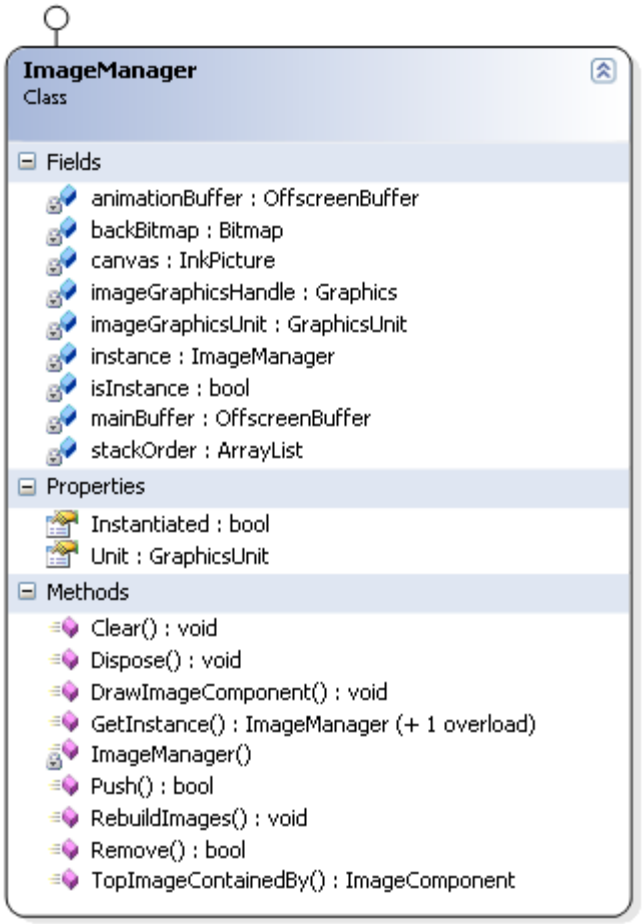

Figure 46. Image Manager

5.3 Model Pad Designer

The model designer is a composition of the above parts that provides the end user interface for Model Pad. The designer is composed of the main drawing space where the user is able to draw a model, the definition window that allows a user to define a component or path, the definition side bar that shows the user the currently defined objects in the model, the animation controls that allow the user to animate the model, and the model library that contains all of the images that the user can add into his or her model. Model Pad is not multi threaded due to limitations in the Tablet PC SDK that highly recommend allowing the Tablet PC SDK to handle all threading because of the high priority and aggressiveness of the threads that collect Ink data from the digitizer.

5.3.1 Definition Window

The definition window provides the user with an interface for defining a component or a path from the model. The window is reached by entering the definition mode of Model Pad and

drawing a lasso around the ink or image to be defined. After the lasso collects a set of Ink strokes or an image to define, the context button is displayed which allows the user to define or edit the selection. This request is fulfilled by the definition window. The definition window is black and semi-transparent, allowing the user to easily see the model beneath the window to understand the context of the action.

The definition window shown in Figure 47 allows the user to enter a name to define the component and select a color for that component. If an image component is selected, then color is ignored.

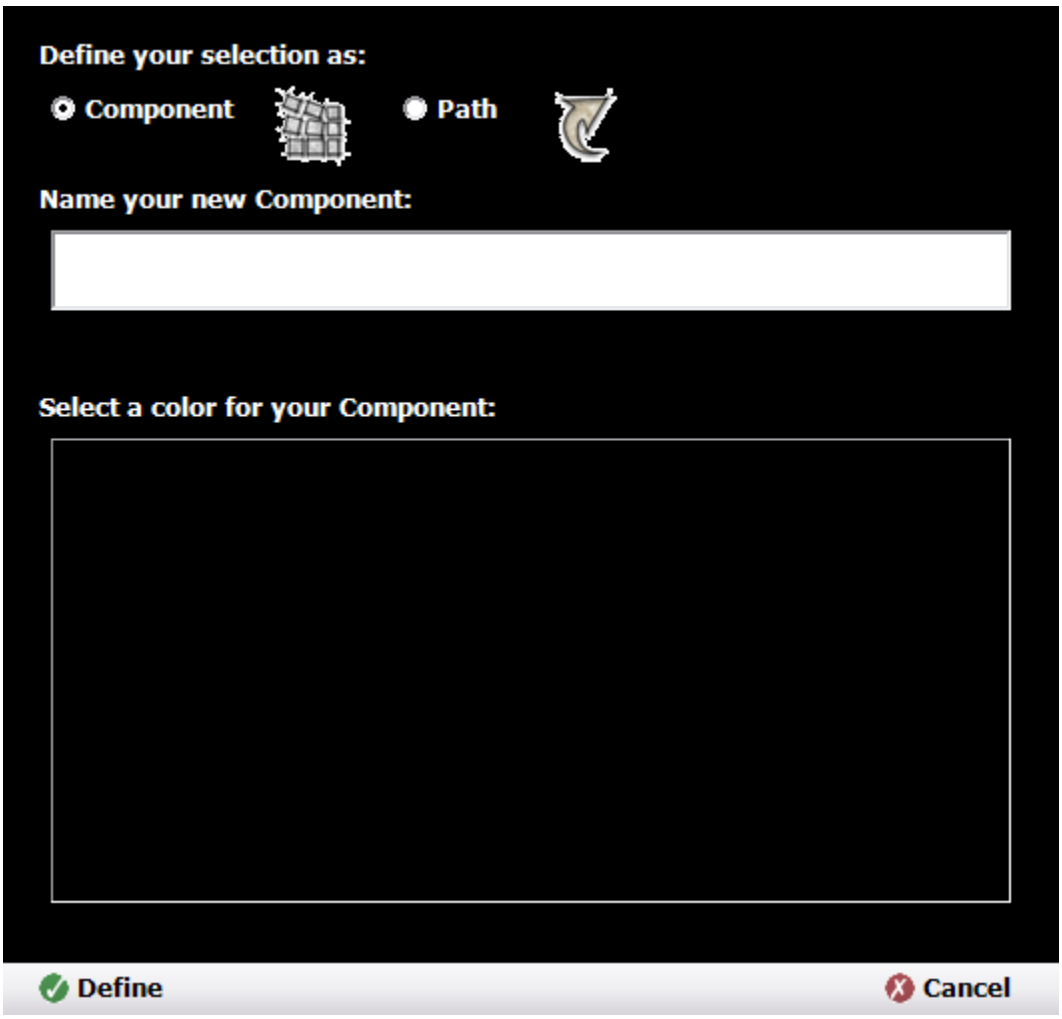

Figure 47. Component Definition Window

Figure 48 depicts the path definition window. This is the interface provided by the context button for defining an Ink stroke to be a path. Every path must have at least one component that defines it or it cannot be defined. Any components set to follow the path will move to the beginning of the path when the animation starts and follow that path to the end. Model Pad

currently does not support two components to be bound to the same path. The path can be set to move a certain number of pixel units per tick of the animation engine. This means that speed of traversal can be controlled by manipulating how many pixels on the Bezier line that defines the path will the objects that follow that path move per tick of the animation engine clock.

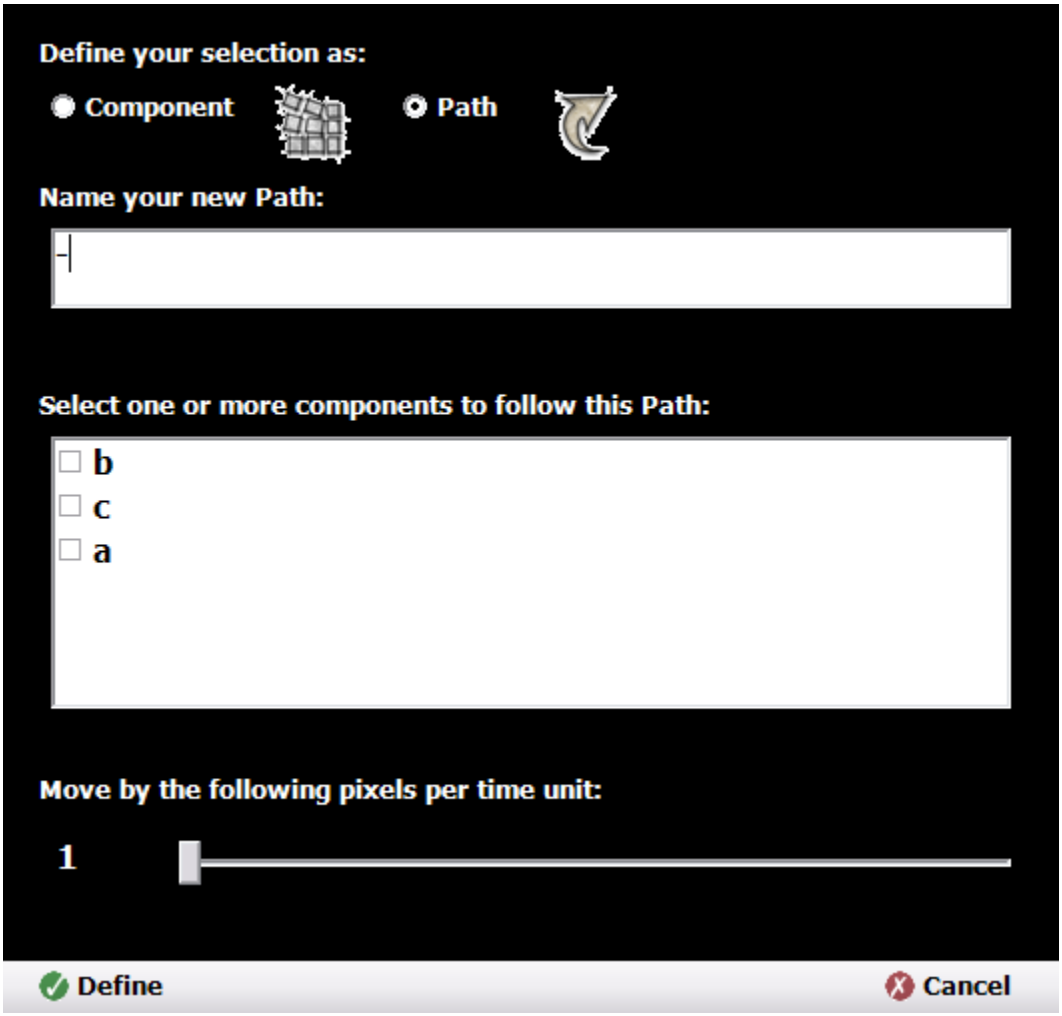

Figure 48. Path Definition Window

5.3.2 Definition Sidebar

The definition side bar allows users to view an overview of the components defined in their model. Model Pad does not support copy and paste, so to duplicate an object drawn in the Model Pad, users must define the object and then drag the component or path that represents that object into the space. The definition side bar also allows the user to display or hide paths or components that are defined. This feature enables models with complex paths to be viewed without the paths obstructing the view of the model.

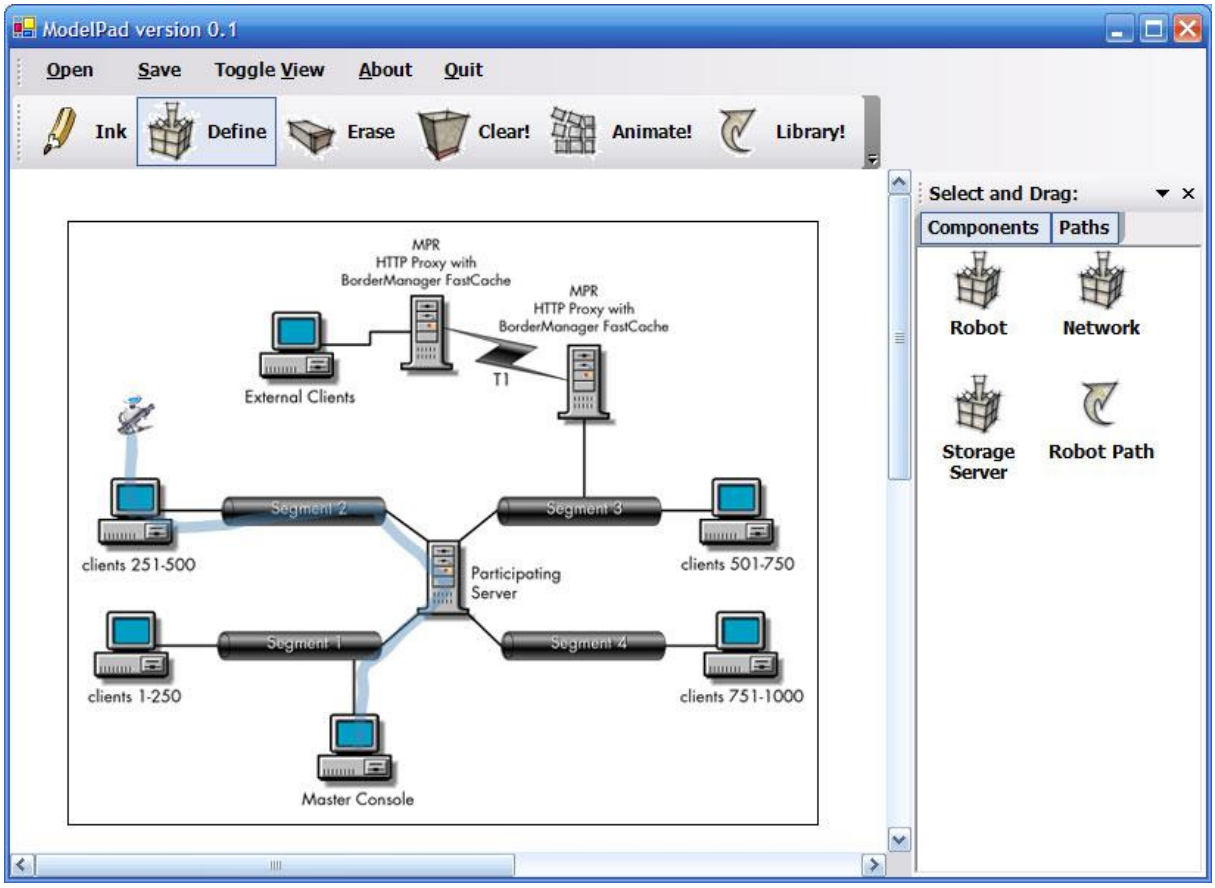

Figure 49. Definition Sidebar

5.3.3 Animation Controls

Animation is controlled by a single toolbar that is displayed modally when Model Pad enters the Animation mode. This modal operation enforces the context of mode. The animation controls allow the user to start an animation or step through an animation step by step with the forward and back buttons. Two sliders allow for control of the number of pixels objects move for each animation tick on the Bezier lines that represent their paths and the number of seconds a component will wait before moving to the next path, completely independent of animation ticks.

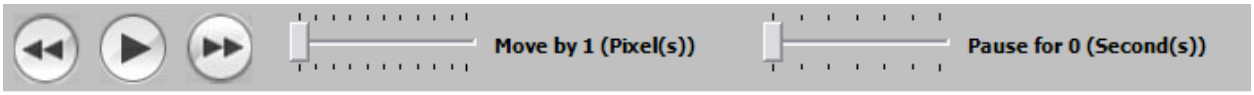

Figure 50. Animation Bar

5.3.4 Model Library

To provide a mechanism for reusable component libraries, the Model Pad environment implements a model library. This library, pulled from the default Application Data directory defined in Windows, will load folders and images and define them as library names and images that can be dragged into the drawing space to define an image component. This is currently the only method for adding an image to a model. The Model Library uses the folder names and file names to name the library names and image names. The Model Library supports drag and drop and like the definition bar becomes a reusable palette for creating new objects in the model.

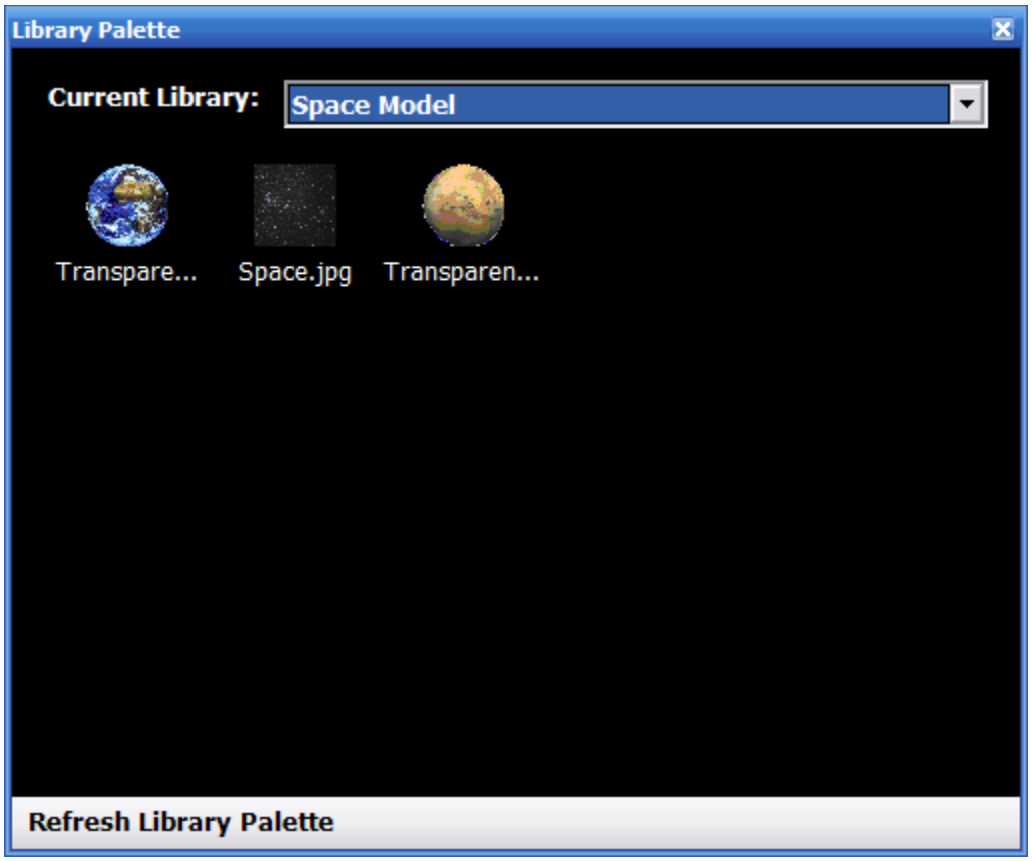

Figure 51. Model Pad Library

5.4 Animation Engine

The animation engine is based on paths. The Ink strokes or images follow the paths by following the calculated Bezier lines contained within the Animation paths. Users can speed up how fast the components follow a path by setting the number of calculated points to move per animation time unit or slow down an animation by setting the number of milliseconds to sleep and not move after moving one animation time unit. These controls provide a level of granularity to the control of the animation without requiring a complex timeline. These properties can be set on an individual basis for each of the components, allow components to move at different speeds on the same clock.

Time is recorded internally as integer ticks. A tick is merely a numbered cycle where the animation engine traverses the current model, determining everything that must move at the given tick. The animation engine moves images and stroke data using the off-screen buffer to improve performance.

5.5 Technologies

5.5.1 C#

The C# language was used throughout the creation of the Model Pad environment. C# provides facilities like properties, delegates, iterators, and meta data that greatly simplified the complexity of the code.

5.5.2 .Net Framework

The .Net framework provides the core data collections used extensively throughout the application. The ability to use indexers to access data directly in a hash or list collection type greatly simplified the code necessary for creating the Model Pad.

Chapter 6: Conclusions and Future Research

6.1 Conclusions

Model Pad lays the foundation framework for creating real-time, behavioral-based animation tools. This thesis is a unique survey of how pen computing has evolved from its earliest beginnings. This thesis explores how ideas behind animation systems evolved and how ideas behind pen based systems developed. Uniquely, this thesis bridges two strains of thoughts that so naturally create synergy when together. Stasko [Stasko 1990] introduces the ideas of creating animations based on the way that users actually semantically think of their movement using paths. With the introduction of the Tablet PC, pen-based computing technology has entered the mainstream market, planting the seeds for it to possibly one day be one of the primary methods for interacting with the modern computer.

The conclusion of this thesis is that these two distinctly different areas, drawing and animation, be brought together to create a tools like the prototyped Model Pad application that leverage the natural semantics of user intention expressed with the pen. By creating simple, easy to use, animation tools that allow a user to create animations in real-time, animations move from being something produced and to becoming a primary method in which ideas are learned and communicated. The ability to create animations in real time using Ink technology transforms animations into another primary form of communication much like drawing and human speech. Enhancing the way that animations are created with ink technology is much like how telephones and computer audio tools today enhance human speech. This ability to use animation tools in real-time opens the doors for where and how animation can be used in engineering design and education.

6.2 Contributions

This thesis contributes to an understanding of the Tablet PC architecture, its history, its capabilities, and its practical applications. The Model Pad prototype described in this thesis distinctly bridges two areas of computing that so naturally support each other, drawing and animation. Researchers continue to study drawing and pen computing. With the exponential growth in computer animation during this decade, researchers continue to grow their understanding of animation and animation tools. It is this thesis' unique contribution to bridge these before separate areas of research. This thesis attempts to challenge the community to investigate how pen computing can merge with modern computer animation tools to simplify the way that animations are created using the computer. The hope is that this thesis breaks the ground on eventually enabling developers to create tools for constructing animations that are as simple and as powerful as drawing on a sheet of digital paper. The Model Pad prototype establishes design patterns that can be leveraged throughout other types of applications designed for the Tablet PC. The Model Pad prototype also establishes a model for the types of user interfaces that are possible for pen-based animation environments.

6.3 Future Research

Future research should expand the capabilities of the animation capabilities, adding advanced behavioral animations that would allow more expression like rotation, etc. A library of animations as well as an expansion of the properties for animation paths would make this possible. Animation in Model Pad is slower than preferred because of the limitations of GDI+. The Model Pad research prototype should be reconstructed so that all on screen animation is handled using Direct X hardware acceleration or the upcoming Avalon drawing system that replaces GDI+ in Microsoft Windows Longhorn.

Model Pad is a new form of Visual Simulation Environment. Because of the recent growth in Model driven architectures, a complete Model Pad could be turned into a domain specific language tool for not only creating accurate simulations based on real time data input from databases or web services, but more importantly, this visual simulation could be bound to actual generated programmatic code.

The Visual Studio 2005 domain language SDK allows for visual representations to be bound to the modeling engine in Visual Studio. The interesting thing about this is that Visual Studio breaks the lines of only tying a model to classes of code. The Visual Studio environment allows for a visual representation of the physical systems that is tied in real time to their control and settings. Future research that could marry the Model Pad environment with the Visual Studio environment could create a new level of sophisticated modeling and simulation. Imagine being able to drag in reusable images that represent complex web services and connect animated behavior that is bound to actual application behavior. Spinning on a database could equal storing all the state data into the database. Drawing a path for a user coming from a web service component could bind and generate code that would transport the user securely to another system, providing them with a different UI.

This also enables the completion of the simulation behavior of the original VSE environment. Complex data models could precisely define the movement of components in the Model Pad environment.

APPENDIX A: EXAMPLE MODEL I – BUBBLE SORT

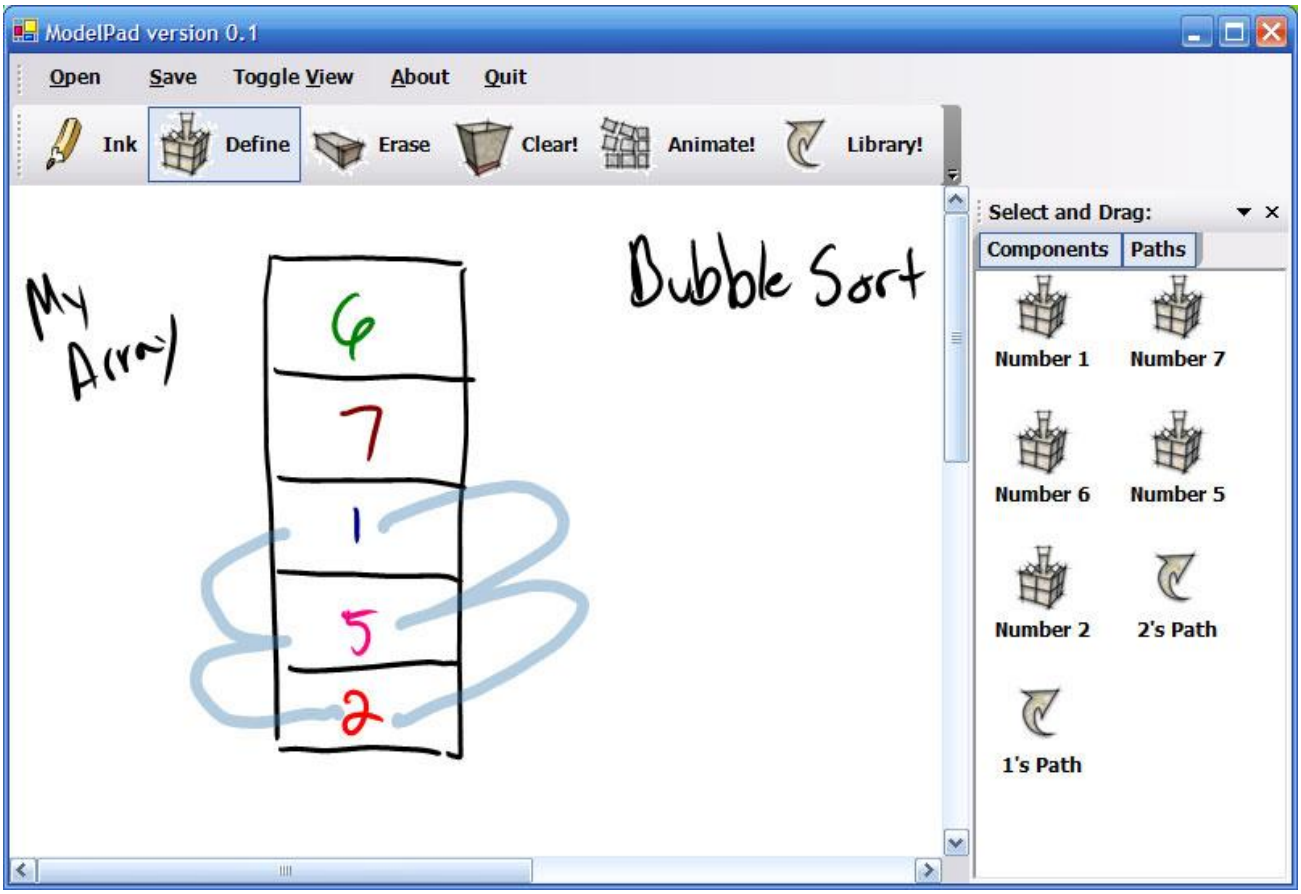

Figure 52. Screenshot of Bubble Sort Model

This example demonstrates using the Model Pad environment as a tool for teaching simple data structures. This model contains a drawing of an array and handwritten labels. Each number in the array is defined as a component. This enables the numbers to be animated. For demonstration, the model contains two paths, one for the number 1 and one for the number 2. When the user initiates the animation, the number 1 will leap down the array, ending in the array position that now contains a 2. The number two will swap with the number one, leaping upwards, following the path, until it rests in the current position of number 1. This simple demonstration depicts how useful Model Pad can be in expressing the movements of a simple data structure. This demonstration could be created in a matter of minutes by a professor to demonstrate in his or her office hours how the data structure works to a student who may not understand.

APPENDIX B: EXAMPLE MODEL II – NETWORK TOPOLOGY

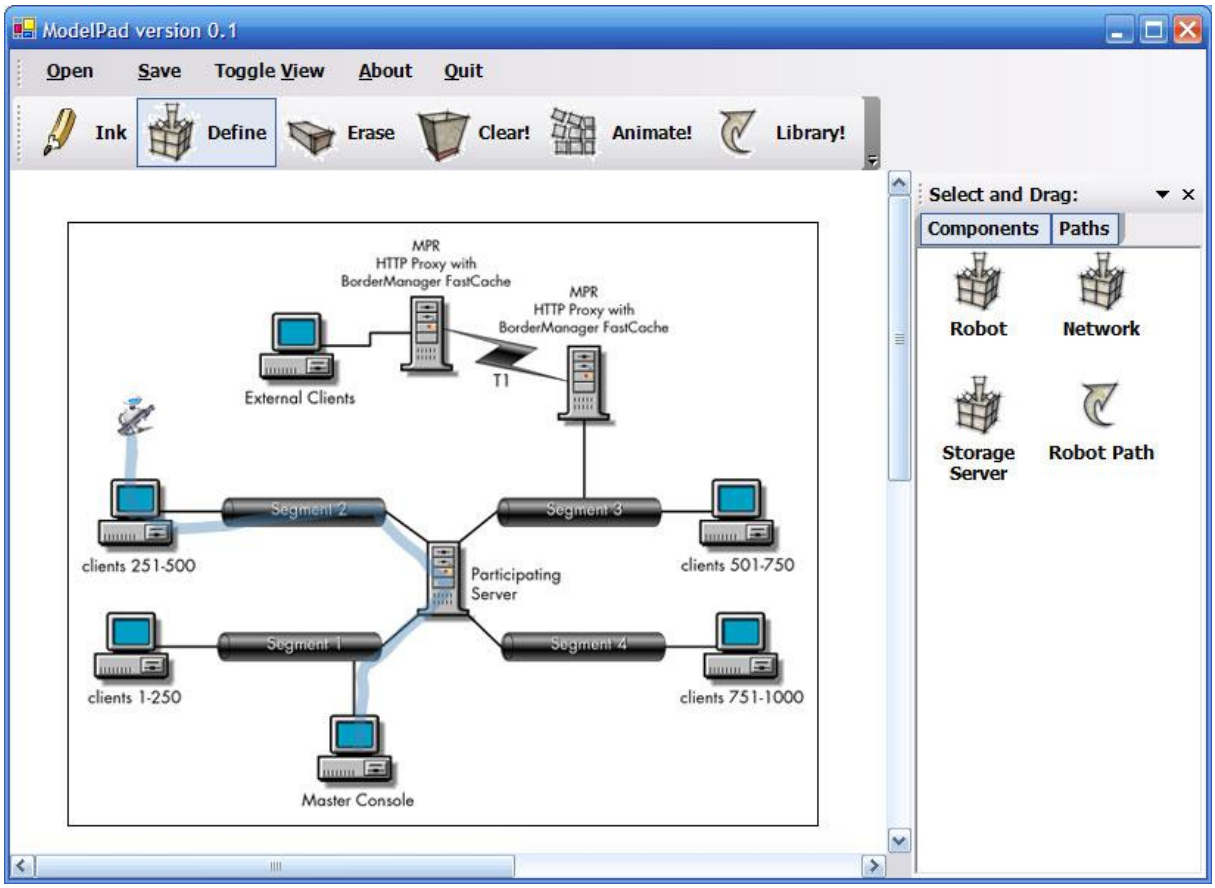

Figure 53. Screenshot of Network Topology Model

This example demonstrates the application of the Model Pad to a network topology. The supposed situation is that you are a network administrator attempting to demonstrate to a college the path of a login user on the current server configuration. Instead of merely showing the picture, one could create a model like this example. This example model uses an image of a user, represented by a robot, to traverse the servers, visually depicting the location of user data as it moves through the system. The robot is defined as a component. The robot is bound to a path drawn on top of an image of the network topology. When this model animates, the robot will move through the system according to the specified path. This example demonstrates a real world example of visualizing a simple concept in a matter of minutes that may not have been animated because of the time necessary to create such an animation. However, because of the minimal time required to animate the model using Model Pad, animation becomes a possible richer model for demonstrating the concept.

APPENDIX C: EXAMPLE MODEL III – SOLAR SYSTEM

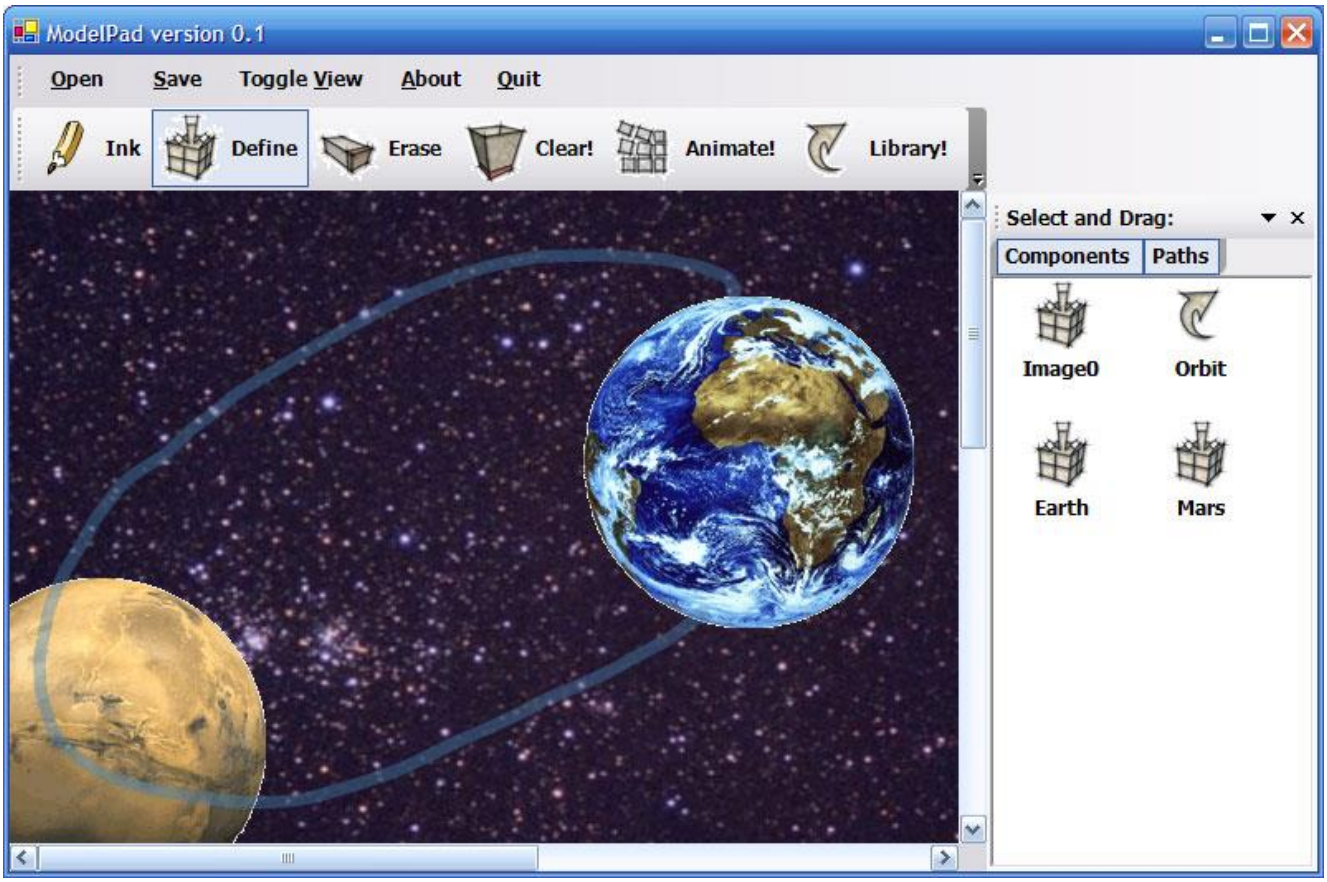

Figure 54. Screenshot of Solar System Model

This final example demonstrates applying animation to understanding a concept common in today's elementary education classes, solar orbits. By defining the image of the Earth and of Mars, the objects can be drawn an orbital path to follow and then animated. This provides a unique learning experience for students learning solar systems because they can easily construct an animation to illustrate the concept of a planet moving around another. Even though this model only demonstrates one orbital path, one could imagine being able to model the orbits of all the planetary stars in the solar system. This example again illustrates the wide range of applications for being able to create an animation based on images or Ink strokes in real time.

BIBLIOGRAPHY

Alias (2005), "Alias SketchBook Pro," http://www.alias.com/eng/products-services/sketchbook_pro/index.shtml

Apple (2005), "Real-time Design Engine," http://www.apple.com/motion/creative.html

Balci, O., Bertelrud, A.I., Esterbrook, C.M. and Nance, R.E. (1995), "A picture-based object-oriented visual simulation environment," In *Proceedings of the 27th conference on Winter simulation*, ACM Press, Arlington, Virginia, United States, pp. 1333 - 1340.

Chand, M. (2003),*Graphics Programming with GDI+*, Addison Wesley.

Garing, P. (2005), "Cost of developing interactive courses," http://www.synapsys.co.nz/blog/archives/000107.asp

Geselowitz, L. (2005), "Ink AniEd 1.2 - Ink based Animation Editor," http://www.leweyg.com/anied/index.html

Gocinski, F. (2004), "Tablet 101 Column 1: Getting Started," http://msdn.microsoft.com/mobility/tabletpc/default.aspx?pull=/library/en-us/dntab101/html/tab101c01.asp

Hong, J.I. and Landay, J.A. (2000), "SATIN: A Toolkit for Informal Ink-based Applications," In *Symposium on User Interface Software and Technology*, San Diego, California, USA, pp. 63 - 72.

James, P. (2005), "History of Animation - The Early Years: Before Disney," http://www-viz.tamu.edu/courses/viza615/97spring/pjames/history/main.html

Fabricius, K. (2010), "A 1923 Helicopter From Llucmajor," https://mallorcaphotoblog.com/2010/07/27/a-1923-helicopter-from-llucmajor/

Khadkikar, A. (2011), "Opening a galaxy of possibilities," http://openionate.blogspot.com/2011/08/opening-galaxy-of-possibilities.html

Koshik, I. (2005), "Alternative questions used in conversational repair," *Discourse Studies 7*, 2, 193-211.

Lin, J., Thomsen, M. and Landay, J.A. (2002), "A Visual Language for Sketching Large and Complex Interactive Designs," In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI, Minneapolis, Minnesota, USA, pp. 307 - 314.

Microsoft (2005), "Microsoft Physics Illustrator for Tablet PC," http://www.microsoft.com/windowsxp/tabletpc/downloads/powertoys.asp

Microsoft (2004), "Microsoft Windows XP Tablet PC Edition Software Development Kit 1.7," http://www.microsoft.com/downloads/details.aspx?familyid=b46d4b83-a821-40bc-aa85-c9ee3d6e9699&displaylang=en

Mifflin, H. (2000),*The American Heritage® Dictionary of the English Language*, Houghton Mifflin Company.

Miller, J.A., Nair, R.S., Zhang, Z. and Zhao, H. (1997), "JSIM: A Java-Based Simulation and Animation Environment," In *30th Annual Simulation Symposium*, IEEE, Atlanta, GA, pp. 31-42.

Pickrell, J. (2005), "Flying Machines," http://www.leonardo.net/flying.html#p2

Read, J.C., MacFarlane, S. and Casey, C. (2004), "Pens Behaving Badly - Usability of Pens and Graphics Tablets for Text Entry With Children," *University of Central Lancashire*, http://www.uclan.ac.uk/facs/destech/compute/staff/read/Publish/ChiCi/docs/UIST2002.pdf

Reeth, F.V. and Flerackers, E. (1990), "Visual Programming in a Computer Animation Environment," In *Proceedings of the 1990 IEEE Workshop on Visual Languages*, Skokie, IL, USA, pp. 194-199.

Rob Jarrett, P.S. (2002),*Building Tablet PC Applications*, Microsoft Press.

Robbins, S. (1998), "The JOTSA animation environment," In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, IEEE, Hawaii, USA, pp. 655 - 664.

Schank, R.C. (1994), "Active learning through multimedia," *IEEE Multimedia 1*, 1, 69 - 78.

Shoemaker, M.L. (2005), "Introducing Tablet UML," http://www.tabletuml.com/IntroducingTabletUML.aspx

Stasko, J.T. (1990), "The Path-Transition Paradigm: A Practical Methodology for Adding Animation to Program Interfaces," *Journal of Visual Languages and Computing 1*, 3, 213-236.

Stasko, J.T. (1990), "TANGO: A Framework and System for Algorithm Animation," *IEEE Computer 23*, 9, 27-39.

Tapscott, D. (1998),*Growing Up Digital: The Rise of the Net Generation*, McGraw-Hill Companies.

Upson, C., Thomas Faulhaber, J., Kamins, D., Laidlaw, D.H., Schlegel, D., Vroom, J., Gurwitz, R. and Dam, A.v. (1989), "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications 9*, 4, 30 - 42.