M-BUFFER: A PRACTICE OF OBJECT-ORIENTED COMPUTER GRAPHICS WITH UML

YI MIAO

M-Buffer: A Practice of Object-Oriented Computer Graphics with UML

by

©Yi Miao

A thesis submitted to the School of Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

April 2000

St. John's Newfoundland

Abstract

In many applications such as virtual reality and computer-aided design (CAD), time-critical rendering of pictures and adequate display of information are fundamental to real-time human-computer interactions. In compassion with shaded pictures that present a simulated level of photo-realism, line drawing (wireframe) pictures are commonly used to show the shapes and movements of three-dimensional objects during the modeling and testing stage of computer-aided design/manufacturing. This thesis presents a practice of object-oriented computer graphics. It applies object-oriented methodology in the development of a mixed-buffer hidden-line rendering algorithm, namely the M-Buffer algorithm. The developed M-Buffer algorithm combines the Z-Buffer algorithm, which is a traditional hidden-line removal algorithm in image-space, with the D-Buffer algorithm, which is a modified hidden-line rendering algorithm. In addition, as it has been constructed by means of object-oriented methodology, the M-Buffer algorithm has the unique capability of rendering pictures quickly in the image space and revealing hidden structures according to users' attention.

Acknowledgements

I would like to express my highest gratitude to my supervisor, Dr. Xiaobu Yuan, for his continuous advice and encouragement in pursuing this research. Without his support, guidance and challenges, this thesis would not have taken this final form.

Thanks are due to the graduate students and postdoc in the Department of Computer Science for vigorous and helpful discussions, e.g. Boting Yang, Vasantha Adluri, Xiaoming Dong, Kaleem Momin, Dr. Guangda Hu, and many more. Many thanks to the faculty and staff of the Department of Computer Science for their assistance and support throughout my program.

Last but not the least, I would like to thank my mother and father for their unlimited encouragement and support. I am also very grateful to my husband, Ning Zhong, and my brother for their great help and understanding.

Contents

List of Figures

Chapter 1

Introduction

In the past thirty years, computer graphics has proven to be beneficial in a large range of applications such as electrical and mechanical computer-aided design (CAD), computer animation, medical imaging, and chemistry. In fact, it would be very difficult to solve many of today's problems in these fields without the assistance of computer graphics.

Rendering images from models is an important part of computer graphics. Two of the most frequently used rendering techniques for three-dimensional objects are solid-object (surface-rendering) mode and wireframe (line-drawing) mode [12]. The solid-object mode provides more information about the shape and surface of objects. It also generates more realistic images than the wireframe mode does. The solid-object

mode may be implemented in either software or hardware. The software implementation of solid-mode is time-consuming due to the complexity of shaded areas. Though researchers have developed different ways to deduce the rendering time, such as level of detail and focus of attention, the problem of time consumption is still an inevitable weakness of the solid-object mode in time-critical interactive applications. Even with hardware support, images rendered by solid-object mode can display only those surfaces that are close to the viewer, concealing the information of the objects' internal and rear structure. Hence, the computational inefficiency and inability of visualizing the rear and internal structure of objects become two major disadvantages of the solid-object mode method.

In comparison, the wireframe mode is able to provide sufficient information of three-dimensional objects at a significantly lower computational cost. The conciseness of data representation and accuracy of boundary description make wireframe mode particularly useful in many circumstances where internal structures are more important than surface shapes for the tasks of object manipulation. When pictures are rendered with the wireframe mode, users can "see through" objects. Thus, people can visualize the internal structure and shape of normally invisible surfaces. In addition, the wireframe mode significantly reduces the need of computing power. Its rendering time is much faster than that of the solid-object mode. Therefore, the wire-

frame mode is a better choice than the solid-object mode to meet the requirement of time-critical rendering and to show the internal structure of objects.

However, when all visible and invisible lines are drawn in the same style, as what a typical wireframe picture presents, it is difficult for viewers to distinguish the back from the front because of the lost depth information. On the other hand, if the wireframe picture presumes opaque object surfaces and shows only those boundary lines or segments that are in the front, the result is similar to the solid-object mode that conceals the objects' internal structures. Hence, more elaborate treatments of hidden-lines are required for providing adequate information of three-dimensional objects while meeting the requirement of time-critical rendering.

1.1 Objective of the thesis

This thesis intends to develop a new algorithm that renders pictures not only quickly, but also with adjustable contents of object details according to the viewers' interest. This algorithm is called the M-Buffer algorithm because it mixes the classical Z-Buffer algorithm with a newly developed D-Buffer algorithm. The main objective is to apply object-oriented methodology to construct a prototype system of virtual assembly, in which users can control the style of hidden-line rendering by removing or displaying

hidden lines adaptively.

1.2 Outline of the Thesis

The rest of this thesis is structured in the following manner. Chapter 2 gives a brief description of the related techniques and research on time-critical rendering. Chapter 3 highlights the benefits of object-oriented computer graphics, and introduces UML (Unified Modeling Language) as a graphic, standard notation for object-oriented software development. Chapter 4 discusses in detail about the development of the M-Buffer algorithm with UML. Chapter 5 describes implementation and discusses experiment results. Finally, Chapter 6 concludes the thesis by summarizing the advantages and shortcomings of the proposed algorithm, and pointing out the direction for future research.

Chapter 2

Related Work

Interactive computer graphics systems require three-dimensional objects to be displayed smoothly and quickly. They make users feel as if they are working with objects in the physical world eventhough they are actually manipulating objects on the screen. The following sections will introduce methods used by interactive systems for picture rendering.

2.1 Level of Detail (LOD)

2.1.1 Introduction

The first concept of using multiple geometric representations of objects for improving performance was introduced by Clark in 1976 [7]. Since then, level of detail is a popular technique in time-critical virtual environments or real-time computer graphics. The general idea is the adoption of different representations with variable complexity for objects. Depending on a decision criterion, e.g. the distance to a viewer, the graphics system selects the most appropriate model to display. More detailed representations are used when an object is perceptually more important, and less detailed representations are used when the object is less significant. This method allows the graphics system to achieve higher frame rates while maintaining good visual realism.

Polygon reduction, texture mapping, and illumination models are three most common methods used to adjust the levels of detail for a polygonal object. Among these three methods, polygon reduction decreases the number of polygons by introducing a new model over the original object. The new model presents the general form and genus of the old one but with a simplified shape. For the texture mapping method, the essential is to substitute regions of high geometric detail with a single textured polygon such that, from a certain viewpoint, the polygon's texture is simply a ren-

dered image of that section. Moreover, the method of different illumination models controls the shading details of objects and thus the time to render a picture is different. For example, the computational complexity of flat-shading is much lower than smooth-shading [12], such as Phong illumination model and Gouraud illumination model, at the expense of picture quality.

2.1.2 Level of Detail Implementation Criteria and Their Implementation

There are a number of criteria to implement LOD, including distance LOD, velocity LOD, size LOD, eccentricity LOD, and fixed frame rate LOD. The first in the list is the original method in this category, and the rest are recently developed. The following discussion will introduce them in detail.

Distance LOD is a simple method. In this method, model selection is based upon the distance between the viewpoint and a predefined point inside the object volume. When the distance between the observer and the object is short, a high LOD model of more polygons is used. If the object progresses away from the observer, a cruder model of less polygons is invoked. The reason is that when the distance is longer than a certain value, the object can only be displayed with less accuracy and detail due to

the limited resolution of display devices. Some details become unavailable. Hence, it is useless to generate a highly detailed object.

Distance LOD has been extensively applied in applications because of its simplicity and efficiency. For example, almost all flight simulators [30, 43] and vehicle simulators [20] use the distance LOD technique.

Size LOD is particularly popular for optimizing digital terrain models in real-time. This method is employed when an object's representation is selected based upon the value of its projected size (or area) in screen coordinates. Size LOD provides a common and proper way of modulating LOD. The first reason is that it offers a measurement to determine the features' visibility within an object, regardless of the display device's resolution and object scaling. Secondly, it avoids the necessity of selecting an arbitrary point for the computation.

One of the typical implementations of size LOD is provided by Silicon Graphics, Inc. with its open inventor graphics toolkit. The toolkit can automatically select different levels of detail based on a screen area criterion. Also Lindstron *et al.* [22] introduced a digital terrain system which uses the size LOD technique.

Eccentricity LOD is based on the level to which the object exists in the visual periphery. The technique assumes that a user would be looking at the center of the display when provided without any suitable eye tracking technology. Objects

therefore should be degraded according to their displacements from the center point.

Watson *et al.* performed a user study to evaluate the perceptual effect of eccentricity LOD in head-mounted displays (HMDs) [37], and Funkhouser *et al.* exploited eccentricity LOD in their architectural walk through of Soda Hall [14]. Other practices of eccentricity LOD include NASA Ames Virtual Planetary Exploration (VPE) tested by Hitchner and McGreevy [17], and head-tracked desktop system by Ohshima *et al.* [26].

When an object's relative velocity to a viewer's gaze determines the rendering detail of this object, this type of LOD is called velocity LOD. Similar to the eccentricity LOD, when there is not a suitable eye tracking system, the velocity of an object refers to the display device.

Unlike the other methods, the original velocity LOD received little attention. However, there have been more investigations in this direction recently, such as the ones made by Hitchner and McGreevy [17], Ohshima *et al.* [26], and Funkhouser *et al.* [14].

Fixed frame rate LOD degrades the LOD of objects in a scene in order to achieve a desired frame rate. It focuses more on computational optimization, instead of perceptual optimization.

To implement fixed frame rate LOD, a system must contain a scheduler to analyse

the system's load and allocate a LOD rating to each object. Examples of reactive system are viper systems developed by Holloway [18] and architectural walk through system by Airey *et al.* [1]. Wloka's update rate system [41] is an example of predictive models in the scheduling system.

2.2 Focus of Attention

In addition to level of detail, there have been wide researches on other time-critical image rendering techniques, such as focus of attention. The basic idea comes from biological considerations, i.e., from the analysis of a phenomenon observed in the Human Visual System (HVS). It is known that the human visual field does not have a uniform resolution. The parts of a scene which are focused upon are referred to as Focus of Attention (FOA). Several stimuli drive the position of the FOA in the human visual field. For example, a viewer's attention is attracted to special colors or combinations of colors. Human visual acuity is the highest towards the center of the retina which is called *fovea*. This is partially relied on the physiological reasons. Based upon this biological phenomenon, focus of attention technique uses a different method to depict objects or parts of objects that are of particular interest. In such a way, the interested objects or interested parts of objects stand out from the others,

and a viewer's attention will be caught right away.

Focus of Attention may be applied to eye movement system to track a viewer's attention. Model based gaze tracking system developed by the interactive systems laboratories located at Carnegie Mellon University and University of Karlruhe employs this concept. Everything that moves in a viewer's visual field strongly attracts the viewer's attention. In fact, motion information is treated fastest by a viewer's visual system, and may control directly viewer's actions even before any high level understanding of the scene has been performed. These reflexes push the viewer to turn his head towards the moving object so as to focus his attention on it. The model based gaze tracking system can identify where the viewer is looking at, and what he is paying attention to according to the orientation of the viewer's head and the orientation of his eyes.

2.2.1 Intent-Based 3D Illustrations

Seligmann and Feiner [32] developed IBIS (Intent-Based Illustration System) which is an automated intent-based illustration system. The key idea is to apply the knowledge of visual effects generation and evaluation (generate-and-test) to achieve goal-driven illustration process.

This system renders objects with different levels of detail according to a user's

requirement of information about objects' locations, relative locations, properties, and states. Some information is provided with the focus of attention methods, such as highlighting part of an object to draw attention to this part. It also cuts off parts of an object's outside shell to reveal the object's inner structure.

2.2.2 Focus of Attention Applied on Buildings in an Automated System

Another practice of focus of attention is in an automated building recognition system. In such applications, the general strategy is to apply a rapid focus of attention algorithm to identify areas that are likelly to contain objects of interest.

R. Collins [8] utilized the focus of attention technique to develop a multi-image focus of attention mechanism which can quickly distinguish raised objects such as buildings from structured backgrounds. The features from multiple images are back-projected onto a virtual, horizontal plane that is methodically swept through the scene. Back-projected gradient orientations from multiple images are highly correlated when they come from scene locations containing structural edges that are roughly horizontal, like building roofs and terrain. These observations are used to define a structural salience measure that can determine whether a given volume of

space contains a statically significant number of structural edges.

Another example is the FOA mechanism developed by Grimson et. al [16] for object recognition. It uses binocular stereo to determine groups of line segments that are near each other in three-dimensional space, and therefore likely to belong to the same object.

2.3 Transparency

In comparison with the methods mentioned in the previous sections, transparency is useful for depicting multiple overlapping surfaces in a single image. The principle of transparency is to allow viewers to see not only surfaces behind a transparent medium but also the transparent medium or object itself [23]. Therefore, an image region is transparent when it is perceived in front of another region and has a boundary that provides information of that object visible through this region.

When there are several objects with irregular shapes in a scene, an overlap exists among objects; level of detail or photorealism cannot accurately describe the object's structure and complex spatial relationship between each. On the contrary, by using transparency, each surface is visible in the context of the other, and the three-dimensional structure of a scene can be more accurately and efficiently appreci-

ated. Moreover, objects and the spatial relationships among objects can be precisely interpreted when the layered elements are displayed in their entirety.

V. Interrante etc. [19] used transparency technique and added opaque texture elements to a layered transparent surface to describe overlapped objects. In such a way, not only objects' three-dimensional shape can be readily understand, but also their depth distance from underlying structures clearly perceived.

2.4 Hidden-Line Algorithms

The wireframe mode has received a number of research interests as wireframe images are widely used in time-critical systems. This section introduces algorithms in this category.

2.4.1 Introduction

To determine the visibility of an object from a particular viewpoint is one of the principal problems in computer graphics. Usually, this process is defined as *visible-line* or *visible-surface determination*, or *hidden-line* or *hidden-surface elimination*. Objects are assumed to be opaque. The surfaces that are closer to a viewer may obscure the edges or surfaces farther from the viewer. In wireframe mode, an object's

boundary edges or silhouette lines of surfaces are presented as lines. *Hidden-line elimination* is usually used in this problem.

Due to the requirements of computing time and processing power, *hidden-line elimination* has been developed into numerous solutions. These solutions, in general, fall into two major categories: image-space method and object-space method.

The image-space algorithms, such as Warnock's area subdivision algorithm [34, 35], Z-Buffer (Depth-Buffer) algorithm [6], and scan line algorithms [4, 5, 36, 42], determine, at each pixel, the visibility of each object according to the object's distance along the viewing ray through that pixel. On the other hand, a direct comparison of any two of the objects in the space is used by object-space algorithms. These algorithms work in the physical coordinate system in which the objects are described. They use geometric computation to decide visible lines or line segments in a picture. Methods like the back-face culling algorithm and the list priority (depth sorting) algorithm are object-space algorithms.

Theoretically, the computation complexity of an object-space algorithm increases with the number of objects. The computation complexity is slow since each object in the scene must be compared with every other object. On the contrary, image-space algorithms are fast because these algorithms do not need the time-consuming object-object comparisons, sorting, and intersection checking. This advantage of image-space

algorithms leads to a wide application where rendering time is critical, such as in the modeling and testing stage of virtual reality and computer aided design (CAD).

2.4.2 Z-Buffer (Depth-Buffer) Algorithm

The Z-Buffer algorithm is a typical and famous image-space method. It can be implemented in either software or hardware. The Z-Buffer algorithm employs not only a frame buffer to store color values, but also a depth buffer (Z-Buffer) to keep depth information. These two buffers have the same number of entries. The depth buffer is used to accumulate the z value or the depth of each visible pixel in an image. In the initialization stage, the depth buffer is set to zero which represents that the z value is at the back clipping plane. The frame buffer is set to the background color. In use, the z value of a new pixel to be written to the frame buffer is compared with the value stored in the depth buffer. If the comparison indicates that the new pixel's depth value is no farther from the viewer than the one whose depth value is already in the depth buffer, the new pixel is written to the frame buffer, and the depth buffer is updated with the new z value. The Z-Buffer algorithm goes over each pair of x and y values in the frame and depth buffers to update them with the objects that contribute to the largest value of z(x, y). By comparing to those pixels, objects can be rendered in a line drawing picture with their hidden lines being removed.

The most powerful aspect of the Z-Buffer algorithm is its simplicity. It can be used to render complex surfaces without the necessity of explicit intersection algorithms. It therefore does not have the problem of computation time associated with the object-object comparisons and depth pre-sorting. However, the Z-Buffer algorithm discards all the information except the "closest" surface, thus it is unable to depict the rear and internal structure because the surfaces and lines that belong to the rear and internal structures are eliminated. The following two algorithms, i.e. the P-Buffer algorithm and D-Buffer algorithm, provide a solution to this problem.

2.4.3 P-Buffer Algorithm

The P-Buffer algorithm was introduced by Yuan and Sun [44]. It is an image-space algorithm based upon the Z-Buffer algorithm. In addition to the frame buffer and depth buffer used in Z-Buffer algorithm, the P-Buffer algorithm employs an additional buffer called pattern buffer to keep a two-dimensional grid of filtering pattern. The size of this buffer is the same as the depth and frame buffer. In the pattern buffer, the value of each element can only be either "1" or "0". For a pixel (x, y) on an interior boundary line, the frame buffer element at (x, y) is updated only if the patter buffer element at (x, y) is equal to 0. Otherwise, the content of the frame buffer (x, y) will be left unchanged. In such a way, the hidden lines are displayed with "1"s while also

dashed with "0"s.

The selection of a filtering pattern is the key to the final result of the line-drawing image rendered by the P-Buffer algorithm. Usually, complicated objects have a wide range of shapes. This pattern must be able to handle all kinds of boundary lines. However, no one filtering pattern can guarantee that it may draw hidden lines with evenly dashed and spaced lines.

2.4.4 D-Buffer Algorithm

The D-Buffer algorithm is also an image-space algorithm, it can generate dashed hidden lines with adjustable length of dashes and spaces for any three-dimensional shapes. The D-Buffer algorithm [11] is an improved algorithm based on the P-Buffer algorithm.

The basic idea of the D-Buffer algorithm is simple. It applies *neighborhood operations* to trace and dynamically generate the hidden lines in the image plane. Neighborhood operations combine a small area of pixels or neighborhood to generate an output pixel. Unlike the Z-Buffer algorithm, the D-Buffer algorithm employs three buffers. It has a frame buffer and a depth buffer that are the same as the ones in the Z-Buffer algorithm. In addition, it uses an extra boundary buffer. All the three buffers have the same size as that of the image. The boundary buffer contains all the

boundary information as well as their depth values. Pixels for boundary points are set to their closet z-values, while the non-boundary pixels are set to a background value. As in the Z-Buffer algorithm, the depth buffer maintains the closest depth value of every pixel of the image, and the final content of the frame buffer is the created picture. In initialization, the frame buffer is initialized to UNDECIDED, the depth buffer and boundary buffer are both set to background color. For each pixel of every object, the D-Buffer algorithm updates the depth buffer if the new value of a pixel is in front of the value stored in the depth buffer. It also records the new depth value into the boundary buffer if that pixel is on an object's boundary line. The result is a boundary buffer whose elements distinguish boundary points from non-boundary points with their corresponding depth value or a distant value. The D-Buffer algorithm then recursively traces the boundary line to determine if the line should be solid or dashed. To adjust the distance between dashes and spaces on a hidden line, the algorithm uses a threshold to control the length of the dashes and the spaces in between. As a result, different effects can be produced by simply adjusting the threshold value.

The greatest advantage of the D-Buffer algorithm is revealing the hidden lines or hidden surfaces so that the viewer knows the rear or internal structure of objects. Moreover, this algorithm can rapidly display sufficient information of objects with

complicated shapes at a low computational complexity as close as that of the Z-Buffer algorithm.

Chapter 3

Object-Oriented Computer Graphics and UML

Object-oriented technology has become one of the most heavily used slogans in Computer Science. Consequently, its influence on computer graphics is becoming more and more significant. In fact, object-oriented technology has been used in three different contexts of computer graphics applications: in describing user interfaces, in modeling systems, and in connection with object-oriented programming languages and environments. This chapter introduces the evolution of object-oriented computer graphics, the benefits of the object-oriented technique, and the object-oriented modeling language UML.

3.1 Object-oriented Graphics

In 1963, Sutherland [33] introduced, as forerunners of object-oriented technologies, the concepts of creating objects by replication of standard templates, hierarchical graphics structures with inheritance of attributes, and programming with constraints. In the 1980s, objects began to move away from the research labs and took their first steps toward the "real" world. Meanwhile, a hardware and software system called *Dynabook* was created at the Xerox Palo Alto Research Center (PARC), and the software portion of Dynabook became the language Smalltalk [15]. Since then, almost all workstations come with some sort of object-oriented user interface toolkit.

Like the identification of Human-Computer Interaction, an object-oriented user interface has the features of visibility and affordance which is defined as a 'technical term that refers to the properties of objects – what sorts of operations and manipulations can be done to a particular object [25]. Controls need to be visible, with good mapping of their effects, and their design suggest their functionality. An object-oriented approach may produce a straightforward interface which is closer to the "real world". An object-oriented user interface allows the direct manipulation of objects which represent components of the entire on-screen information, using overlapping windows, pulldown menus, buttons, icons and other graphical techniques to repre-

sent "real world objects," such as documents, to the end user. Object-oriented user interface is an improvement on the kinds of "user-hostile" interfaces, for example, command line interfaces, and can thus be considered user friendly.

In this thesis, the virtual assembly system, a prototype of the M-Buffer algorithm, is an application of object-oriented user interface. The overall structure of this system is immediately visible to a user. There is no requirement for the user to understand the command syntax. A user simply presses the menu button to select a function, and the feedback is immediate and clear. Because of the graphic user interface, users may feel close to the task, and devote more attention to manipulating models, and studying the model's structure.

However, friendly user interface is not the unique reason for developing object-oriented computer graphics. Defects in traditional graphics systems are one of the reasons that lead to the necessity of object-oriented techniques. For example, working with traditional graphics libraries, a programmer draws a square by calling one of the square drawing routines, which has the immediate effect of displaying the square on the screen. However, the effects can only be shown on the screen, for the square is composed of an array of dots. In a sense, the image is bitmapped, and the system itself has no record showing that the square exists. Therefore, in an event-driven system, subsequent events do not know about the existence of the square. It becomes the

responsibility of the application programmer to explicitly take care of and maintain this information.

By contrast, in an object-oriented system, a square is drawn by first creating a square object and placing it in the graphical hierarchy, and then issuing a "redraw" command. In this way, the graphical state of the system is always known from one event cycle to the other: every visible figure on the screen (geometric figures, windows, or widgets, etc.) is an object fitted into a single graphical database. All details of the object's appearance, such as color, dimensions, position, etc., are maintained as state variables within the object's data structures. In such a case, once the screen needs to be refreshed, this can be done by traversing these data structures.

In addition, the idea of objects with associated methods provides a direct description of the items relevant to graphics in their visual appearance, such as geometric data, and in their behavior, such as reaction to graphics input. Intuitiveness of applying object-oriented concepts to graphics becomes a bonus of using the object-oriented graphics system.

3.2 Function-oriented Design and Object-oriented Design

Like many developments in software, since the 1980s, objects have been driven by programming languages. Currently, from the viewpoint of programming, there are two kinds of software design strategies: function-oriented design and object-oriented design.

3.2.1 Function-oriented Design

Function-oriented design involves decomposing the design into functional components. Functions have local state, but shared system state is centralized and accessible by all functions. Usually, it leads to a top-down design style. A programmer starts with a high-level description of what the program does, then, in each step, takes one part of the high level description and refines it. Samples of functional design are Structured Design [9], SSADM [10, 38], and step-wise refinement [39, 40].

The function-oriented design conceals the details of an algorithm in a function but system state information is not hidden. Therefore, changes to a function and the way in which it uses the system state may cause unanticipated interactions with other functions. A functional approach to design is most successful when the amount of

system state information is minimized and information sharing is explicit. However, it is not a perfect way to implement large systems, because it focuses on the function of the program. This method works fine for small programs, but, may cause difficulties in designing and maintaining large systems.

3.2.2 Object-oriented Design

Unlike function-oriented design, object-oriented design focuses on the data to be manipulated by the users. The basic idea of object-oriented design is information hiding [27]. Each object manages its own state information by grouping similar objects into classes, and by defining appropriate fields, such as attributes, which define the object's state, and operations, which act on the attributes, into each class. In practice, objects communicate by an object calling a procedure associated with another object.

Object-oriented programming allows easier creation and use of *abstract data type* (ADT). *Inheritance* mechanism allows programmers to conveniently derive a new type from an existing user-defined type. The attributes and operations of an object class may be inherited from one or more super-classes so that a class definition needs only to set out the differences between that class and its super-classes. Another benefit of object-oriented programming is *encapsulation*. It hides parts of the program that

do internal processing, and consequently, these parts of the program should not be of concern to anybody from outside. In this case, users of the code may know what can be used safely, and what is not of users' concern. Furthermore, programmers can change what is hidden without influencing the public interface. In addition to inheritance and encapsulation, *polymorphism* makes it possible for objects to act depending on their run-time type, and saves programmers a great deal of duplication of code. Polymorphism refers to a situation in which an entity could have any one of several types. It is a mechanism to deal with the awareness of the run-time class of the particular object.

These are the main reasons that lead to a wide application of and broad research interest in object-oriented techniques in the computer graphics domain. It is also based upon these reasons that this thesis uses the object-oriented approach to model M-Buffer algorithm and its prototype system. By defining machine parts which are samples of the virtual assembly system as objects and by using encapsulation mechanism, it is easy for users to modify or change one model without affecting the the other models and other object classes. Polymorphism allows M-Buffer to smoothly integrate the Z-Buffer and D-Buffer algorithms by reusing part of the code, thereby avoiding duplication of some codes.

3.3 Unified Modeling Language (UML)

In the past ten years, user interface design has been the most successful application of object-oriented graphics. Meanwhile, object-oriented modeling and architectures of graphics systems have undergone development too. Among the object-oriented analysis and design techniques that appeared in the late 1980s and 1990s, the Unified Modeling Language (UML), that is a commonly applied graphic, standard notation in system modeling, is the most favorable.

UML most directly unifies the methods of Booch, Rumbaugh, and Jacobson, but its reach is wider than that. It is a notational system aimed at modeling systems using object-oriented concepts [3, 29]. It is "a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems" [2]. It can be used to understand user requirements, to design components at different stages, and to browse, configure, maintain, and control information. It supports most existing object-oriented development processes.

The techniques in the UML were to some degree designed to help programmers do good object-oriented analysis and design; however, for different techniques, they still have their own different advantages. For example, *patterns*, which has become vital

to learning object-oriented concept, lets programmers concentrate on good object-oriented designs and learning by following examples. Iterative development lets programmers exploit object-oriented analysis and design effectively. Interaction diagrams which make the message structure explicit are very useful for highlighting over-centralized designs. These characteristics and advantages of UML allow it to be a favoring modeling language for object-oriented modeling.

In the analysis and design phases, the UML catches the static structure and dynamic behavior information of the M-Buffer algorithm and its prototype system. For example, use case diagrams help to catch scenarios, interaction diagrams help to capture the behaviors of use cases, and class diagrams help to catch objects and describe the types of objects involved in the the system. The diagrams work together to delineate and depict various aspects of the system. Because of these, the UML is used to model the M-Buffer algorithm and its prototype system.

To understand the requirements and capture the behavior of a system, its subsystem and its external environment, a modeler first uses the "*use cases diagram*" to describe the sequences of scenario for the processes requested by external actors. The use cases diagrams contain different use case scenarios, and these use case scenarios illustrate the system in terms of actors, actions and the relationship among them. Use cases diagrams are central to understanding what users want. They help with

communication about abstract concepts. Since they control iterative development, they also present a good way for project planning which gives regular feedback to the users about where the software is going.

Defining a conceptual model is the second step. The conceptual model is illustrated in a set of diagrams that describe objects. It involves an identification of the concepts, attributes, and associations in the problem domain.

In the meantime, interaction diagrams work on object interaction and message passing. These diagrams describe sequences of message exchanges among roles that implement behavior of a system. Typically, there are two kinds of interaction diagrams: sequence diagrams and collaboration diagrams. In sequence diagrams, a set of messages are shown in time sequence. Lifelines represent each classifier's role, and arrows represent message passing between different lifelines. Though a collaboration diagram allocates the responsibilities to objects and illustrates how they interact via messages, it also shows the roles as geometric arrangements. The messages are shown as arrows attached to the relationship lines connecting classifier roles, and a sequence of numbers which are prepended to message descriptions indicate the sequence of messages.

Finally, the specification of a system is then presented in forms of class diagrams. A class diagram not only shows a collection of static model elements, such as classes,

types, and their contents and relationships, but also contains certain operations which are expressed in other diagrams, such as collaboration diagrams. In most modeling processes, class diagrams are the final product.

Chapter 4

M-Buffer

4.1 Introduction

The purpose of this thesis is to develop a new algorithm by means of object-oriented methodology to render images quickly and adjustably.

Although Z-Buffer algorithm is the fastest approach among hidden-line removal algorithms, it cannot provide sufficient structure information of the displayed objects. The D-Buffer algorithm can reveal the concealed information by rendering the hidden lines or hidden surfaces with dashed/dotted lines. However, the rendering time of the D-Buffer algorithm is slower than that of the Z-Buffer algorithm, especially when the scene is composed of a large number of objects. In addition, If hidden lines of all

objects in a scene are displayed as dashed lines while a viewer's interest is merely in one or two objects, a viewer's attention will be easily distracted by so many lines, rather than focus on the interested objects. It is also a waste of computational power to display every object's geometric structure.

In time-critical applications, it is necessary to display objects with different level of details according to a viewer's interest to the objects. For instance, hidden lines can be removed or displayed as dashed lines. In such a way, not only is the computation time saved, but also the user's requirement of understanding the object's structure is satisfied. This leads to the necessity of M-Buffer that combines both the Z-Buffer and D-Buffer algorithms to convey the right amount of information for human-computer interaction.

4.2 Strategy of M-Buffer

The basic strategy of the M-Buffer algorithm is to use one more buffer, named status buffer, to record the status information of pixels on the boundary lines. Each pixel in the status buffer belongs to either a NORMAL object or a SPECIAL object. An object is defined as SPECIAL when a viewer is interested in its structure, or as NORMAL otherwise. For a SPECIAL object, therefore, its hidden lines must be

displayed as dashed; but for a NORMAL object, its hidden lines should be removed.

If the value of a pixel in the depth buffer is greater than or equal to the one in the boundary buffer, then, the status buffer is checked to see to which object this pixel belongs. If the pixel belongs to a SPECIAL object, the dashed hidden-line method is invoked. Otherwise, if the pixel is on a NORMAL object's boundary line, the hidden-line removal method is called. For those pixels whose value in the depth buffer are less than those in the boundary buffer, the hidden-line removal method is directly invoked without checking the status buffer.

4.3 Employing Use Cases to Define M-Buffer

To achieve the M-Buffer by means of object-oriented methodology, the objects to be used must be clarified. Therefore, it is important for object-oriented modeling to adopt a proper modeling process during system analysis and design.

In UML, a use case is a narrative document that describes the sequence of events of an actor using a system to complete a process [29]. Usually, use cases are scenarios or cases of using the system. They elucidate and indicate requirements in the scenarios. Thus, use cases capture the behaviors and functional requirements of a system, and define processes in terms of goals, responsibilities, pre-conditions and post-conditions.

Figure 4.2 shows the use case diagram of a virtual assembly prototype system. There are five use cases in the system. The *M-Buffer* use case is one of the system's use cases. Use case *Move Object* "uses" *M-Buffer* use case, which means that an instance of the *Move Object* use case will include the behavior as specified by the *M-Buffer* use case. Figure 4.1 shows the definition of the *M-Buffer* use case.

Use Case:	**M-Buffer**
Purpose:	Invoke different algorithms in different cases
Type:	Primary and essential
Description:	An instance of System retrieves each object one by one in the container, determines each pixel's projected position and depth value z, and indicates whether one pixel is on a boundary line. Then instance of Decider decides to invoke Z-Buffer or D-Buffer algorithm according to each pixel's depth buffer, boundary buffer value, and the pixel's status.

Figure 4.1: **M-Buffer Use Case**

4.4 Preparing for Conceptual Models

These use cases show how users interact with the system. Capturing the objects involved in the system is one of the major works. Sequence diagrams illustrate user-system interactions and help designer clarify the object involved in the system. Hence, sequence diagrams contain the basic information for the construction of conceptual

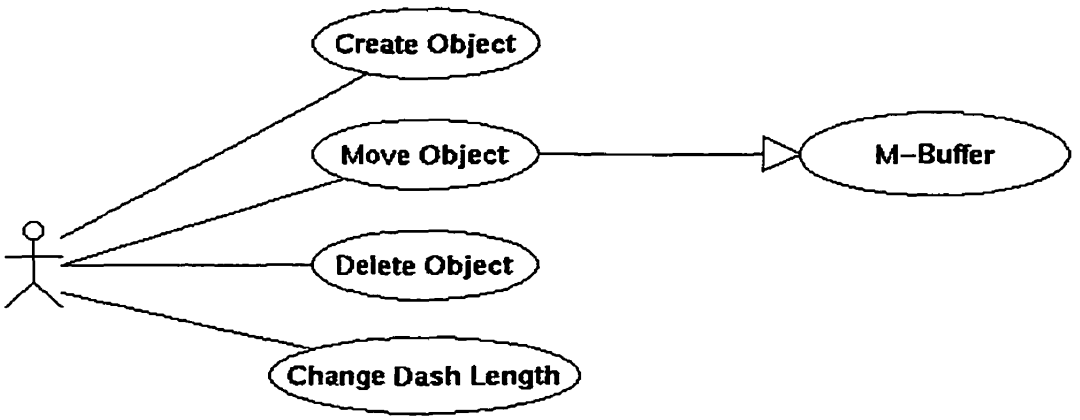
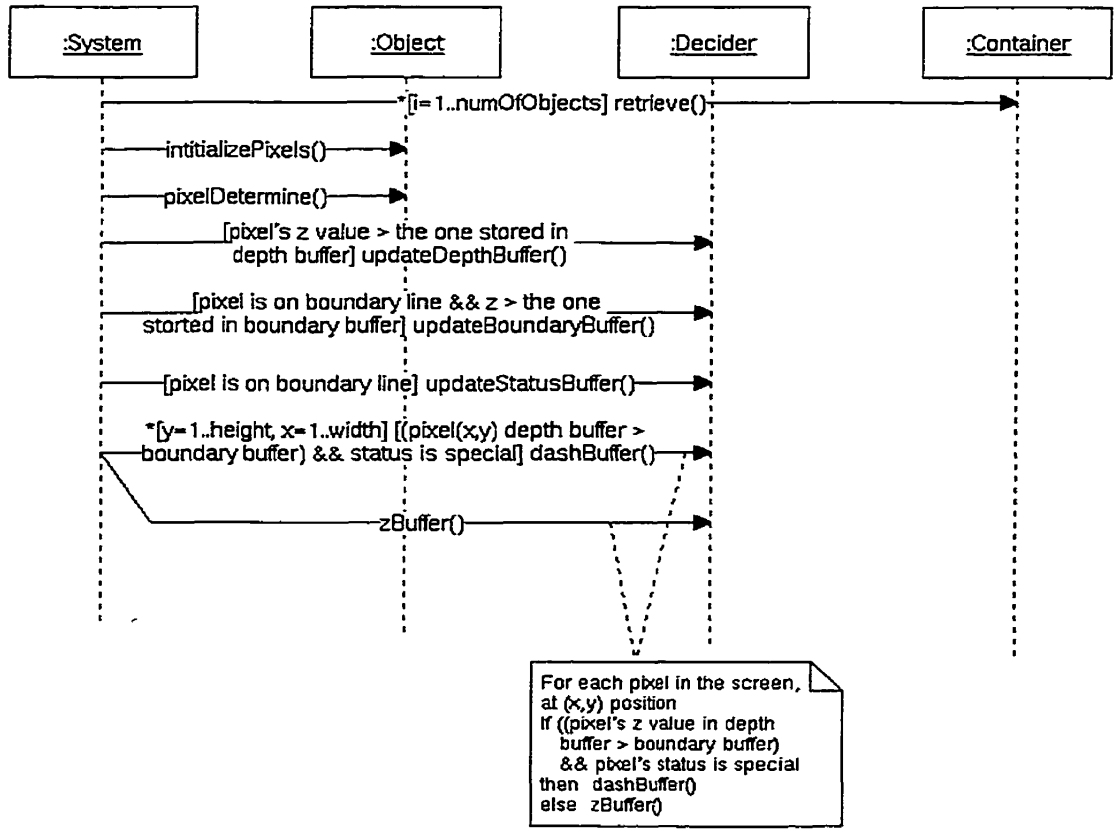


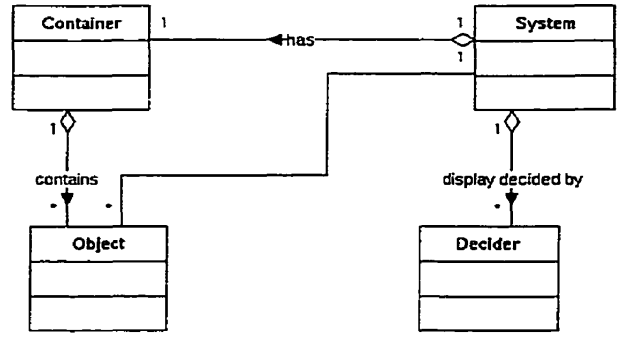
Figure 4.2: **A Use Case Diagram of the System**

models. A conceptual model is a preliminary form of a class diagram. At this stage, conceptual models can be understood as sets of static structure diagrams without operations of the concepts. Figure 4.3(a) is a sequence diagram which shows a particular scenario of the M-Buffer use case.

As can be seen from Figure 4.3(b), in the problem domain, the *M-Buffer* contains several concepts which may be potential classes of the system. They are concepts of **System**, **Container**, **Decider**, **Object**. The **System** is a system manager that controls the whole process and connects users with the system. It is responsible for creating instances of classes, communicating with external agents, etc. The **Container** stores the graphics representation of physical objects, keeps track of the number of objects, and provides services for accessing its elements. The **Decider** is a key



(a)



(b)

Figure 4.3: **A Sequence Diagram and Conceptual Model**

concept that decides which algorithm to invoke according to the depth position and status of an object. The last is the **Object** concept whose instances are machinery parts in the virtual assembly system. They are stored in **Container**, and a user operates on the object instances.

4.5 Constructing Activity Diagrams

An activity diagram shows the flow from activity to activity within a system [3]. It illustrates the dynamic aspects of the system, since it emphasizes the flow of control between activities. Figure 4.4 is such a diagram which illustrates the control flow of the M-Buffer algorithm. Each round-corner box contains an execution of a statement or activity. For example, the activity diagram starts at the initialization of each buffer. After the completion of initialization, an arrow leads to the next step that retrieves objects in the container one by one. The flow in the activity diagram provides important perspectives to complex operations. With proper notation, such as a start state sign, a final state sign, arrows, and diamond decision signs with multiple labeled exit arrows, the flow of control and data can be directly depicted by the activity diagram.

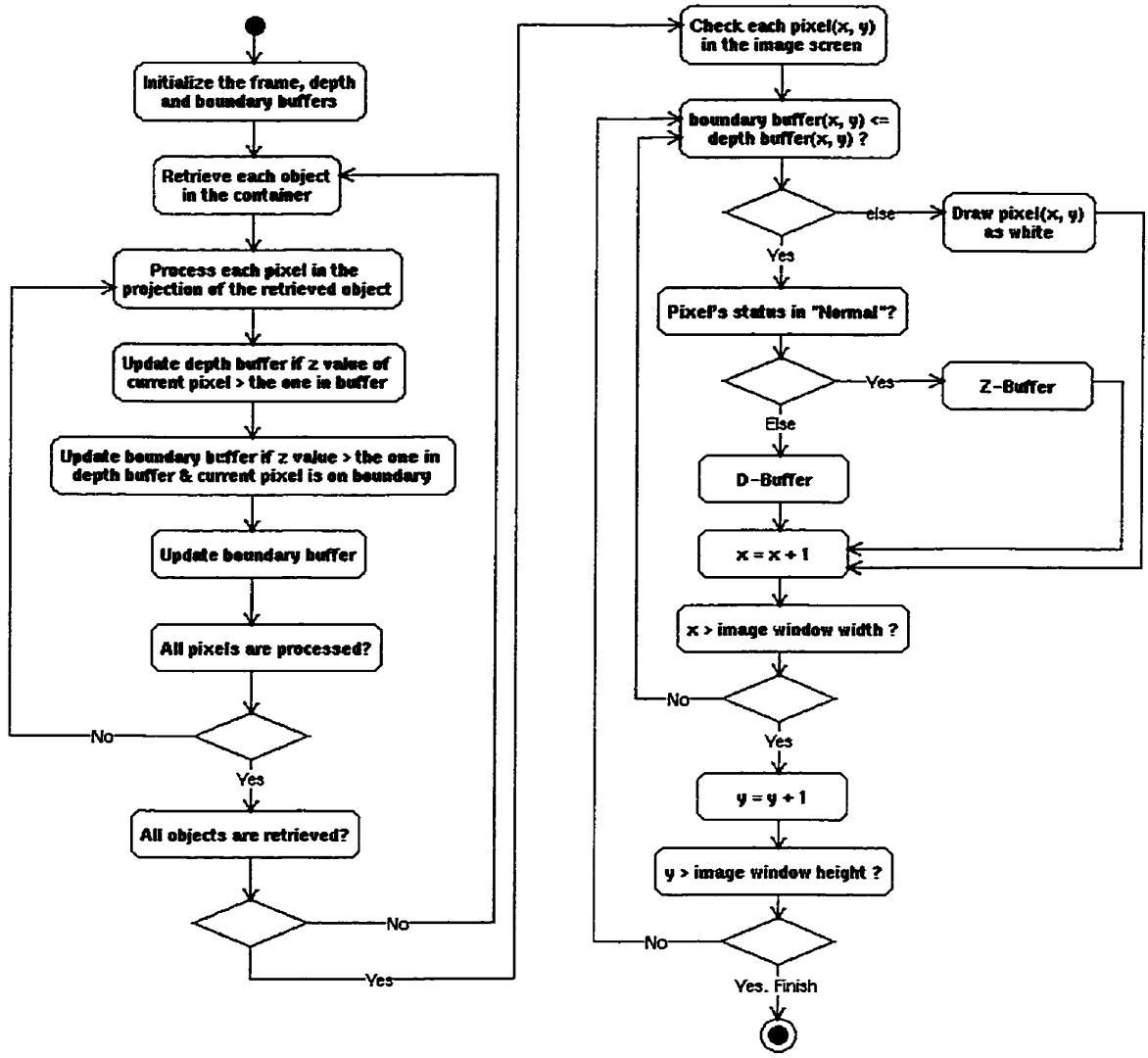


Figure 4.4: **An Activity Diagram**

4.6 Creating Interaction Diagrams

Both interaction diagrams and activity diagrams deal with the dynamic aspects of a system. Interaction diagrams focus on the interactions between objects and their message passing relationships, whereas activity diagrams are flowcharts that emphasize the activity taking place over time. Both of the two types of interaction diagrams, sequence diagrams and collaboration diagrams, describe a system's dynamic aspects, but they have different emphases. A sequence diagram focuses on the sequential occurrence of messages in time. For example, Figure 4.3 shows messages passing one by one in time ordering. A collaboration diagram, as shown in Figure 4.5, describes behaviors between different objects. Collaboration diagrams are often used in the design phases of software development, since they contain more contextual information, such as visibility between objects and exception handling. It is also easier to express conditional logic using collaboration diagrams than using sequence diagrams.

Figure 4.5 is a collaboration diagram that contains several iterations and conditional logics. Four instances are involved in this diagram: **System**, **Object**, **Decider** and **Container**, and all of them are unnamed instances. The action begins with a request of retrieving object (*"retrieve()"*) from the **System** instance to the **Container** instance. The star sign following the sequence number represents an iteration call.

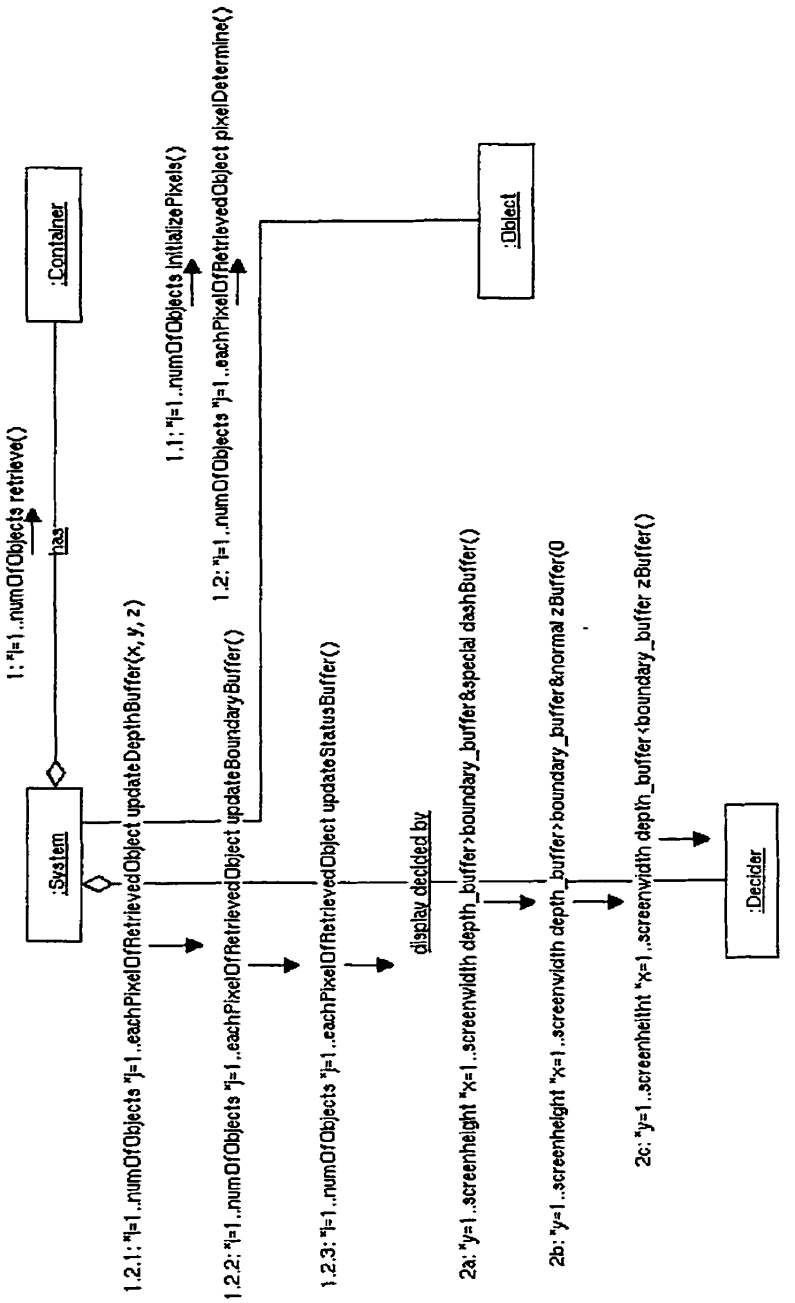


Figure 4.5: **A Collaboration Diagram of M-Buffer**

The iteration clause indicates the recurrence value, which starts from the first object in the container to the last one. The iteration clauses on each of the subsequence messages indicate that those messages happen within the same iteration. After this call, **System** sends *"initializePixels()"* signal to each **Object** instance to reset an iterator to the first pixel of each object. Procedure *"pixelDetermin()"* then starts to iterate through each pixel in the projection of the retrieved object. This call makes available all the pixel information of every object, including the x, y, z values, and whether a pixel belongs to a boundary line. This information is compared with the values in the frame buffer, depth buffer, and boundary buffer. For each pixel on the retrieved object, after a comparison of it's z value and the value of the one in the depth buffer, if the current pixel's depth is greater than or equal to the one in the depth buffer, the **System** instance sends an *"updateDepthBuffer()"* request to **Decider** so that the greater value is stored in the depth buffer. If the pixel is on a boundary line and its z value is greater than or equal to the value in the boundary buffer, then **Decider** updates the boundary buffer's value by calling *"updateBoundaryBuffer()"*. Moreover, the **Decider** instance gets the pixel's status and sets pixel to either "NORMAL" or "SPECIAL" by calling "*updateStatusBuffer()*". After all objects in the container are retrieved, the iterations terminate their operations of retrieving objects and pixel. Then every pixel on the image screen is analyzed to decide whether a pixel on an

obscured line should be drawn as white dot, or black dot. Furthermore, if the z value of a pixel in the depth buffer is greater than the one in the boundary buffer, and this pixel's status is special which indicates the viewer has special interest in the object's structure, the **System** instance then sends a *"dashBuffer()"* message to **Decider** to draw the hidden lines or hidden surfaces with dashed lines. Otherwise, the *"zBuffer()"* will be invoked to draw the pixel in black, which removes the hidden lines or hidden surfaces.

4.7 Modeling Other Use Cases

As shown in Figure 4.2, *M-Buffer* is just one of the five use cases of the virtual assembly system, which is a test-bed of the *M-Buffer* display scheme. The other use cases include *Create Object*, *Move Object*, *Delete Object*, and *Change Dash Length*. To build up the whole system, all the use cases need to be analyzed so that all the objects required by the system can be identified. This section describes the modeling procedure of several use cases, including *Move Object*, *Create Object*, and *Change Dash Length*. Since the *Delete Object* use case is similar to the *Create Object* use case, it will not be introduced in this section.

4.7.1 Modeling Move Object

The *Move Object* use case depicts how a viewer operates on an object and interactions between the viewer and the system.

The action starts with a user pressing the MOVE button, which invokes *"moveButtonPressed()"* and returns a true value to notify **System** that the viewer intends to manipulate objects on the screen. If the viewer presses down a mouse button, **System** sends *"isButtonPressed()"* signal to inform **Mouse** that an object has been selected. Then **System** sends *"getX()"* and *"getY()"* methods, respectively, to get the x, y values of the object's origin. After these calls, **System** invokes *"pickObject()"* to decide which object has been selected by the viewer. After the object is determined, *"isDragged()"* is sent to **Mouse** to check if this object has been moved. If the selected object is moved, the **System** needs to know object's new position. This process is finished by the *"setX()"* and *"setY()"* methods. In this way, the **System** updates an object's position with its new origin. Then the moved object will be displayed by a *"display()"* message sent from **System** to **Screen**. During the manipulating process, if one object is moved by the viewer, its hidden lines are displayed with dashed lines, but the hidden lines of all the other objects are removed. If the moving object overlaps with another one, both of the two objects are displayed with dashed hidden lines. It allows the viewer to see their geometric structure.

This process requires two other classes: **Screen** and **Mouse**. A **Screen** object is used to display objects on the window, and a **Mouse** object detects the movement of the mouse, such as button pressed, release, or drag, and reports its status to the **System** object. Figure 4.6 is a collaboration diagram of this use case.

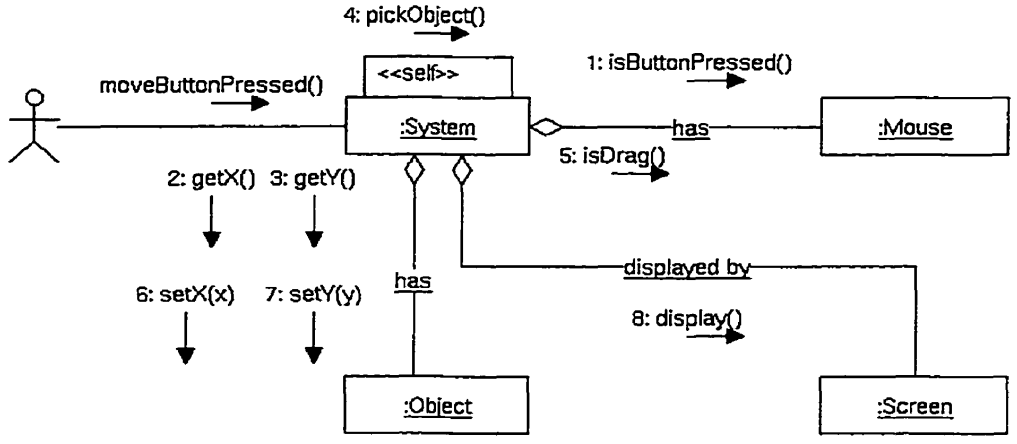


Figure 4.6: **A Collaboration Diagram of** *Move Object*

4.7.2 Modeling Create Object

The *Create Object* use case takes care of the creation of machinery parts.

Once a viewer clicks the PIECES button, a pull down menu appears with all the necessary machine pieces, so that the viewer may select the piece he wants. When the PIECES button is pressed (*"pieceButtonPressed()"*), **System** sends a *"requestDepth()"* message to retrieve the object's depth value along z axis in a left hand coor-

dinate system. The depth value is then written into the system with a return message from **Requester** to **System**. After determining the machinery part chosen by the user with *"determinePieceType()"*, **System** sends this information to **ShapeAdder** so that the selected object can be put on the screen. Afterwards, **ShapeAdder** sends *"setX()"* and *"setY()"* messages to **Object**, respectively, so that an instance of object is created. To keep track of each object displayed on the screen, **Container** needs to have a record of this object. **ShapeAdder** sends an *"add(obj)"* message to the **Container** class and asks it to add a new object to the object list. If updating the object list is successful, **System** calls *"display()"* to require the **Screen** class to display the new object. Figure 4.7 is a sequence diagram of this scenario.

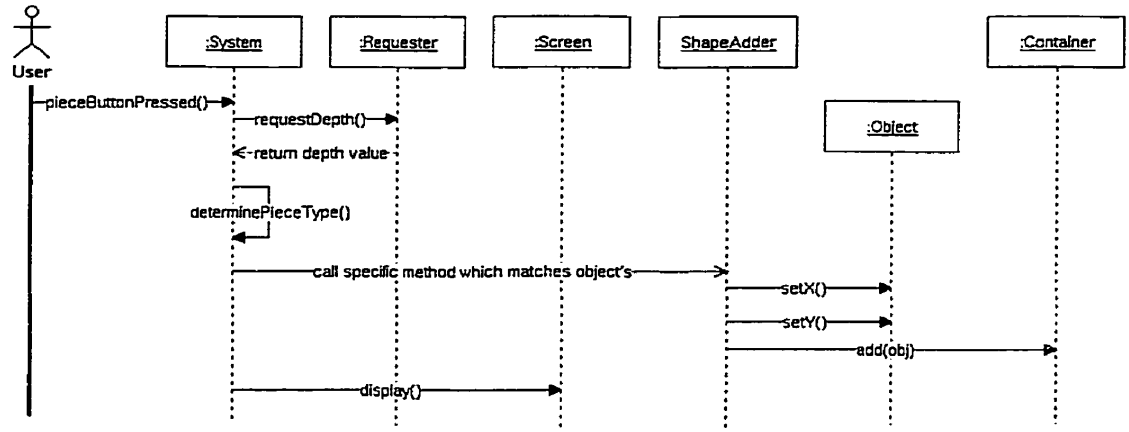


Figure 4.7: **A Sequence Diagram of** *Create Object*

4.7.3 Modeling Change Dash Length

As mentioned in the previous sections, the distance between the line segments and spaces is adjustable by controlling the threshold. This is completed by *Change Dash Length* use case.

Usually, for dashed hidden lines, the default space of black dashes and white space is set to five pixels, respectively. To increase or decrease the distance, the user first presses a DASH LENGTH button. Once **System** receives this message from (*"dashButtonPressed()"*), it sends a request to **Requester**. **Requester** then calls the *"requesDashLength()"* method to pop up a window for the user to input the new value. After **System** gets the new dash length, **Decider** invokes *"setDashLength()"* to reset dash length threshold to the new length. Finally, **System** sends a signal to **Screen**, and **Screen** invokes *display()* to redraw the object. Figure 4.8 is the collaboration diagram that interprets this process.

4.8 Modeling Class Diagram

The procedure of developing use cases, conceptual models, activity diagrams, interaction diagrams is a congenial and seamless modeling process. The purpose of creating these diagrams is to generate class diagrams, so that they could eventually

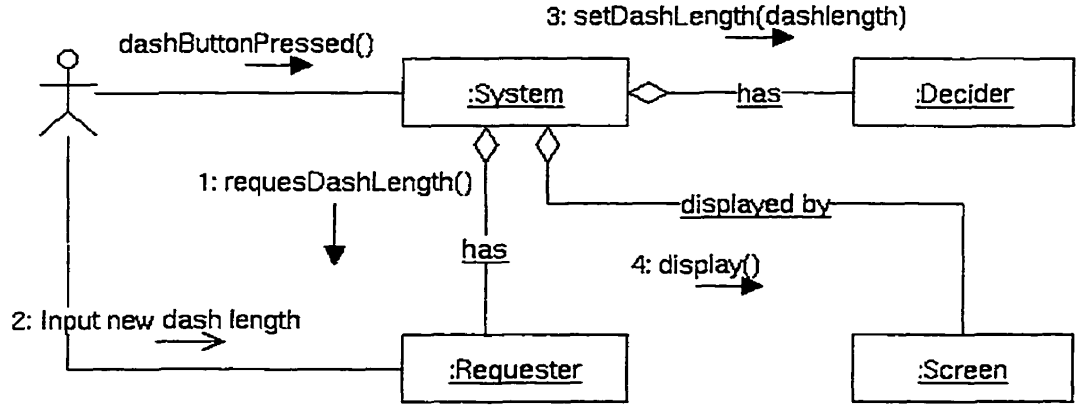


Figure 4.8: **A Collaboration Diagram of** *Change Dash Length*

produce codes. Class diagrams not only contain the static structure and behavior of objects, but also indicate the dynamic interactions between objects at run-time. Figure 4.9 shows a class diagram of the virtual assembly system. This diagram contains all necessary classes and methods for the virtual assembly system. Each class has characteristics (attributes) that describe the class's static structure, operations that are design details, and associations that are significant connections between different classes.

Except for the **Object** and **ShapeAdder** classes, all the other classes have an aggregation relationship (an open diamond at the end of a line) with **System**. It indicates that the instances of these objects are part of the whole system. At the bottom-left of this diagram, class **Object** has a many to one association with **Con-**

tainer, specifying that each instance of class **Container** may contain many instances of class **Object**. Therefore, a star sign appears at the far end of the association of **Container** and **Object**. In the attributes field of **Object**, object's origin is described by x, y whose data types are float. Another attribute is *status* whose data type is *integer*. The instances of the **Object** class are created by the **ShapeAdder** class, which is given a role name as *adder*. For class **Container**, the data type of its attribute *numOfObjects* is *integer*, and the default value of *numOfObjects* is zero. Class **Decider**'s attribute field has frame buffer, depth buffer, boundary buffer and status buffer. These buffers have necessary information for **Decider** to decide which algorithm to invoke according to the pixel values in the four buffers. Class **Shape** is an enumeration class. It has neither behaviors nor relationships with the other classes. In the attributes field, it's data type can only be *enum*. A *"$"* sign indicates that this attribute is static. Its value applies to the entire class of objects, rather than to each instance.

In the operation fields of each classes, some methods have a "+" sign and some others have a "-" sign. "+" sign identifies public operations and the "-" sign identifies private operations. In addition, data types following the operations indicate the data types of their return values. Moreover, the data types within "(" and ")" of operations indicate the data types of parameters.

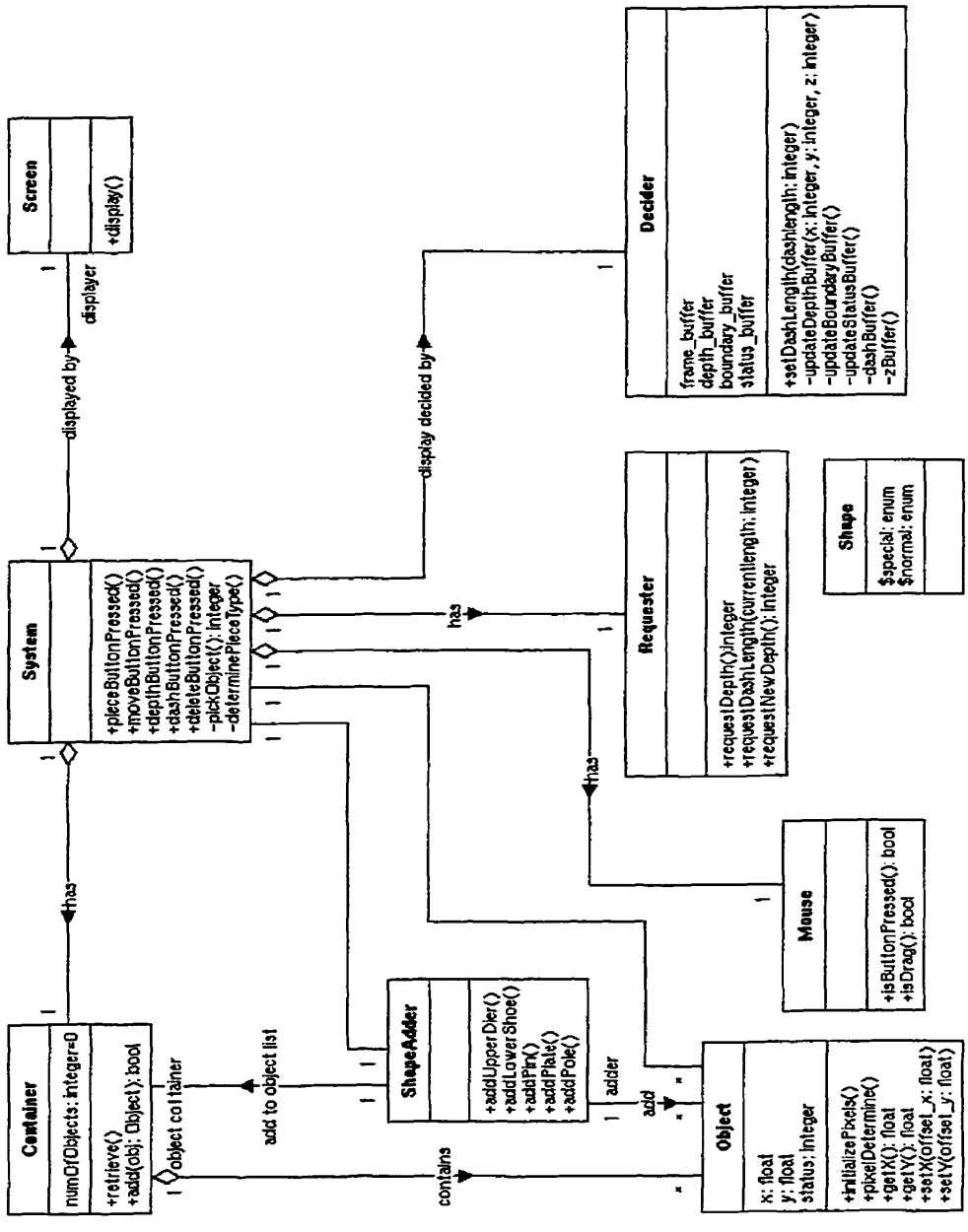


Figure 4.9: **A Class Diagram for the System**

Chapter 5

Implementation and Discussions

This chapter describes the architectural design and the implementation of the prototype system, and then discusses experimental results tested on the system.

5.1 Implementation

The virtual assembly system has been analyzed, designed with COOL:Jex, which is a Computer-Aided Software Engineering environment developed by Sterling Software Inc. It supports UML and allows models-to-code generation.

The system development has gone through four phases using the UML diagrams: analysis phase, package design phase, object design phase, and implementation phase.

Diagrams are created and modified in the first three phases, adding more details at each phase until the application codes in the final phase are generated. At the analysis phase, system requirements are analyzed. Use case diagrams, interaction diagrams are created to describe scenarios in the virtual assembly system. In addition, concepts of the system are identified and a conceptual model containing attribute information is achieved. Upon finishing the requirement analysis, system development goes to the package design phase. At this stage, application architecture is defined. The third stage is for object design, in which class diagrams are refined and enhanced to generate codes. Figure 4.9 is an actual diagram refined in the stage. The final stage deals with implementation and code generation. Using the class diagram created in the object design stage, COOL:Jex generates codes to implement the classes and their relationships, attributes, and operations. Based upon the generated codes, some more codes are added to complete the system.

This system is implemented in C++, OpenGL, and FLTK (*Fast Light Tool Kit*) on UNIX. FLTK is a C++ graphical user interface toolkit for X windows. In this system FLTK is used to build the interactive user interface, while OpenGL builds the graphics routines. Figure 5.1 shows the architecture of the system.

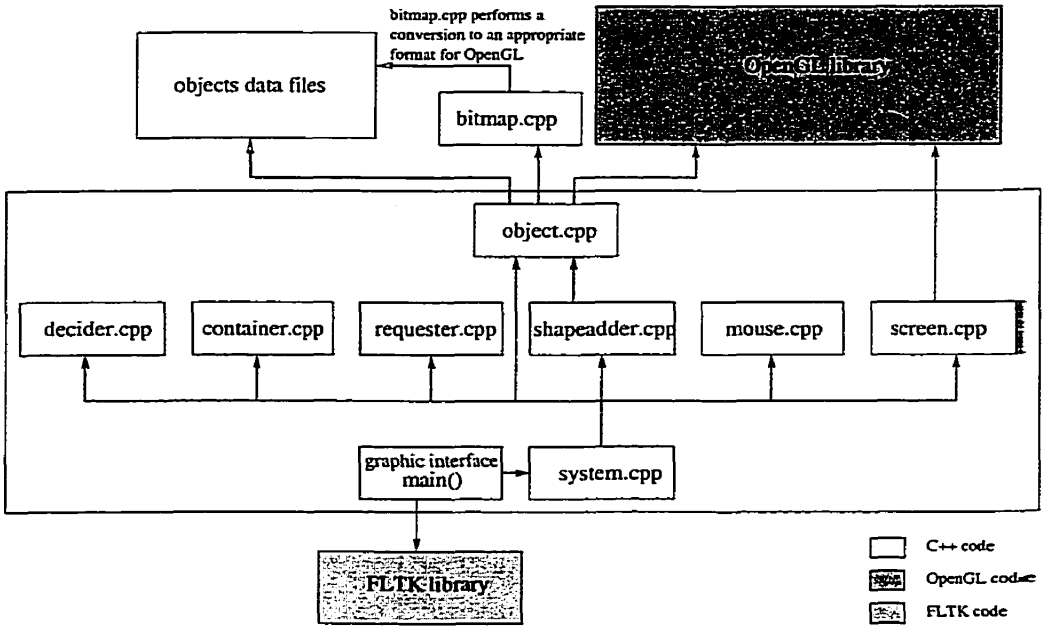


Figure 5.1: **An Architecture of the System**

5.2 Experiments

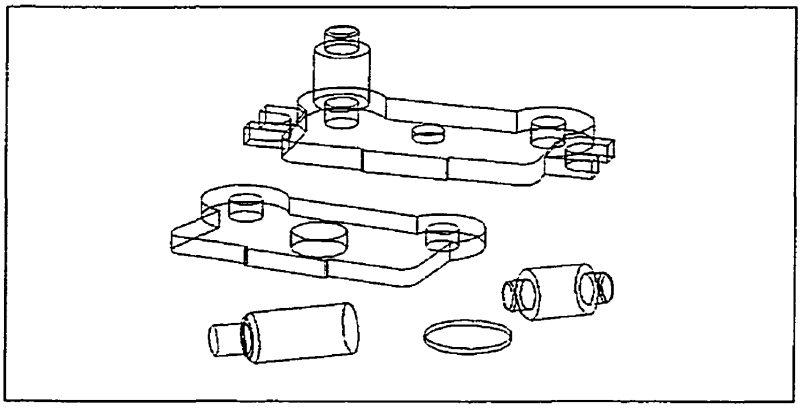
This section presents the experiments conducted for the M-Buffer algorithm and the analysis on the experimental results.

5.2.1 Experiments

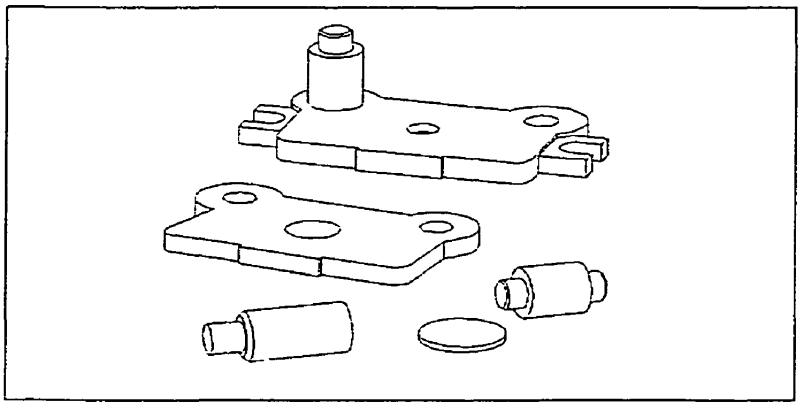
To test the M-Buffer algorithm, complex objects, such as machinery parts, are used. Some machinery parts are polyhedrons that consist of a set of smoothly joined polygons, and some are curved objects.

Figure 5.2(a) is a wireframe picture of a number of machine pieces. Though

(a)



(b)



(c)

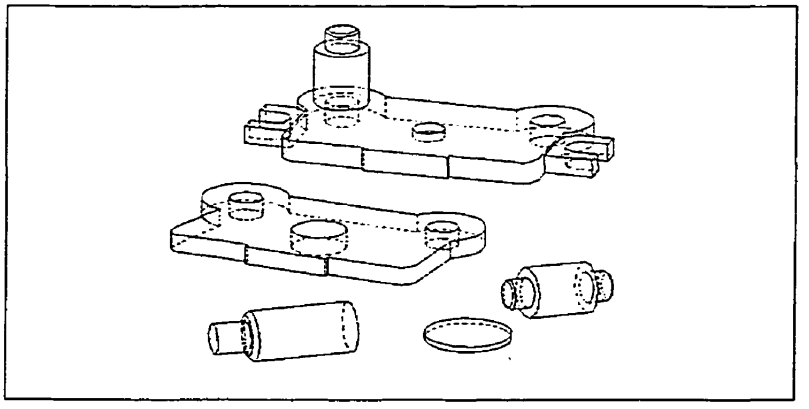


Figure 5.2: **Wireframe, Z-Buffer and D-Buffer Images of Machine Pieces**

this picture contains all the information of objects, the viewer may still be confused. The reason is that there is no difference between visible boundary lines and invisible boundary lines. As a result, it is hard to distinguish front from behind even for a simple object, such as the plate, in the figure.

In comparison, Figure 5.2(b) shows objects only with their surfaces or lines close to the viewer. It eliminates the visual perception ambiguity by removing invisible lines, but the objects' inner structures are not available. For instance, from Figure 5.2 (b), it is not clear that if there is a cylinder hole on the top-most object in the picture. This hole is necessary to stand up one of the two poles.

As an improvement to Figure 5.2(a) and (b), Figure 5.2(c) is picture rendered with the D-Buffer algorithm. It contains all the information about object boundaries and shapes, and it tells invisible lines apart from visible lines by displaying them in a different style. In this way, the viewer can easily comprehend the structure of objects. However, since object manipulation in the virtual assembly system involves two objects at a certain point, it is not necessary to display all object boundary lines. Displaying of all boundary lines, especially all the invisible lines, is a waste of computation power. In addition, it does not help focus attention with so many lines.

Figure 5.3 is rendered by the M-Buffer algorithm. Hidden lines of the top-most two pieces in the image are dashed, because of their involvement in the current object

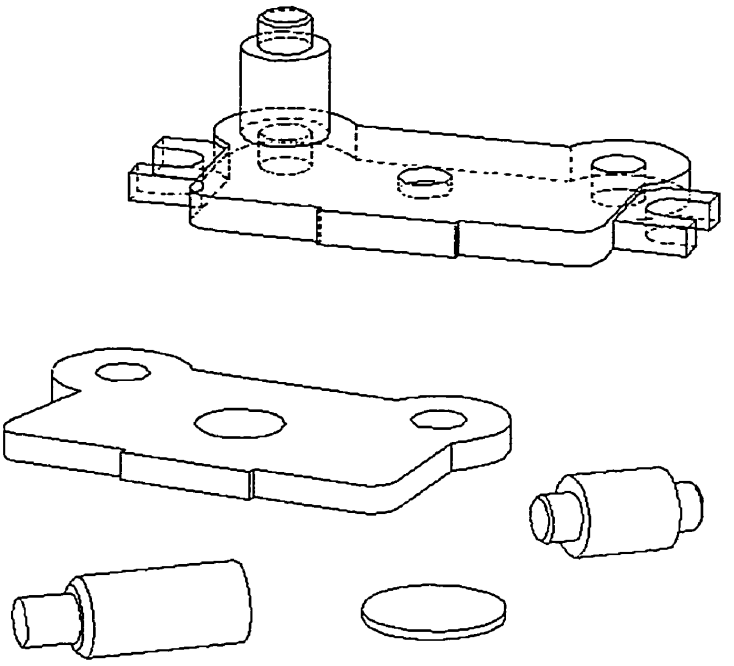


Figure 5.3: **Focus of Attention of Machine Pieces**

manipulation. For instance, when the top-most pole is moved and has overlap with the shoe-shaped machine piece, the system will display the hidden lines of this two pieces as dashed, so that the user may understand their geometric structure clearly. In the meantime, the hidden lines for the other objects are removed. The illustration with dashed hidden lines makes these two objects stand out from the others. Furthermore, their internal and rear structures become available for manipulation.

5.2.2 Measure of Visual Detail

To analyze the perceptual nature of pictures rendered by the M-Buffer algorithm, the following discussion introduces a measuring method, i.e., *spatial frequency.*

Spatial frequency [28] was initially introduced by Schade [31] in 1956. It is a quantitative measure for the space between a series of alternating light and dark vertical bars. It can be applied to measure the details presented to a visual system, e.g., a computer graphics system. Spatial frequency is defined in units of contrast cycles per degree of visual field (c/deg). When the view point is stationary, as in the virtual assembly system, the orientation of spatial frequencies is of little importance. In such cases, relevant spatial frequency becomes a measure of visual detail. By feature we mean a region enclosed by boundary lines for any two-dimensional visual segment. In a picture, its fundamental relevant spatial frequency is calculated in

units of cycles per pixel (c/pixel). In essence, it measures how many pixels a feature extends over each orientation.

The spatial frequency of a feature is inversely proportional to its size. The largest size of a feature is the longest contiguous line of pixels at a horizontal direction. Applying a scaling of 1/2 to the calculation then produces the bars in a contrast grating. Since a full cycle has a peak and a trough, the line of pixels is, in fact, half of a contrast cycle. Therefore, a general relationship can be developed as follows:

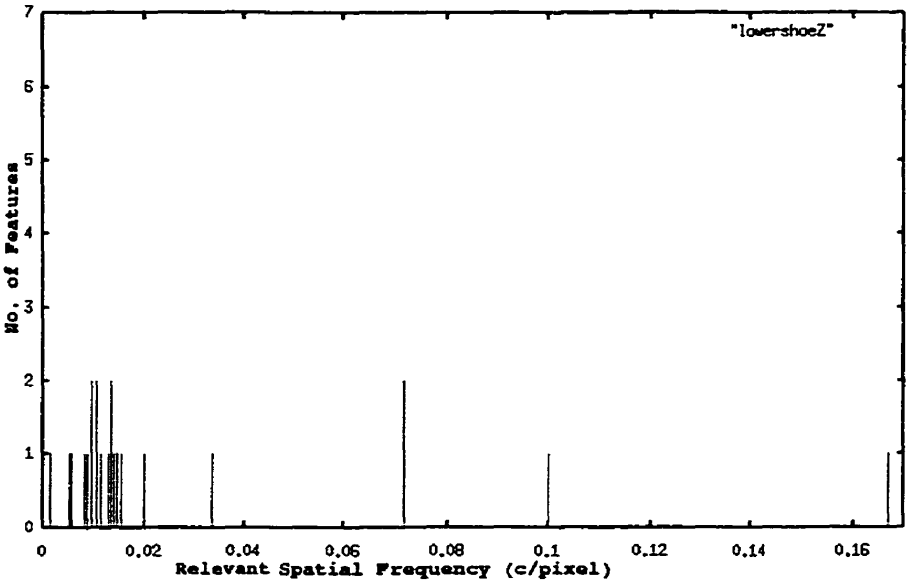
$$RSF = \frac{1}{2l} \tag{5.1}$$

where RSF represents the relevant spatial frequency, and l is the length of the longest contiguous line of pixels in a feature.

5.2.3 Result Analysis

With the measure standard set, it is now ready to analyze the pictures created by the M-Buffer algorithm. Figure 5.4(a) and (b) are diagrammatic results of relevant spatial frequencies for *lower_shoe* that is located at the top of Figure 5.2(b) and (c) respectively. In the figures, the x axis denotes increasing spatial frequency (c/pixel) while the y axis represents the number of features of *lower_shoe* at a particular spatial

(a)



(b)

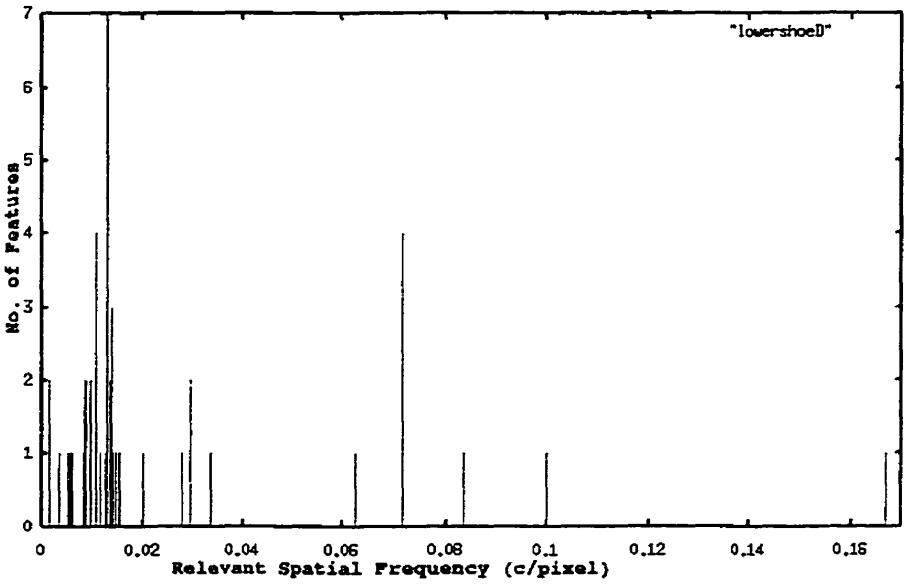


Figure 5.4: **Relevant Spatial Frequency Analysis for *lower_shoe*'s (a) Z-Buffer and (b) D-Buffer Generated Image**

frequency.

Figure 5.4(a) shows the relevant spatial frequencies of the *lower_shoe* in Figure 5.2(b) as created by the Z-Buffer algorithm. The total number of features is 23, each of which represents one of the visible facets of the *lower_shoe*. As can be seen from Figure 5.4(b), in addition to the twenty-three spatial frequencies, there are some more spatial frequencies that represent the features not displayed in the picture generated by the Z-Buffer algorithm.

From the figures, as well as spatial frequencies analysis, a conclusion can be drawn as follows: pictures generated by the D-Buffer algorithm contain more information about objects as it reveals the invisible surface and boundary lines by creating dashed boundary lines. They present object structures to the viewer. In contrast, an image rendered by the Z-Buffer algorithm has less information. As a combination of the two algorithms, the M-Buffer algorithm produces pictures with adjustable contents according to the viewer's interest in objects.

5.3 Discussion

Z-Buffer, D-Buffer and M-Buffer are image-space algorithms. The Z-Buffer algorithm calculates the depth value of every pixel of every object's projection, and compares

this value to that stored in the depth buffer to determine the visibility. The D-Buffer algorithm employs a boundary buffer and performs one more operation than the Z-Buffer algorithm. This additional operation performs simpler processing on fewer pixels. The D-Buffer algorithm has the same time complexity as the Z-Buffer algorithm does. Therefore, the computational complexity of both algorithms are $O(n)$, where n is the number of objects.

Moreover, the M-Buffer algorithm uses a status buffer to check the status information of pixels on the boundary lines, and it employs the operations of the D-Buffer algorithm and the Z-Buffer algorithm. Therefore, the M-Buffer algorithm is in the same computational complexity level as the Z-Buffer algorithm and the D-Buffer algorithm.

Chapter 6

Conclusion and Future Research

After a brief survey of the methods for time-critical rendering, three image-based hidden-line algorithms are reviewed. They are the Z-Buffer, P-Buffer and D-Buffer algorithms. All of them can display complicated objects quickly and informatively with line-drawing pictures. The advantages of efficiency, simplicity, and unlimited range of processable shapes have made the Z-Buffer algorithm a good choice for time-critical rendering. However, the Z-Buffer algorithm cannot display the invisible lines because it removes the hidden lines. The P-Buffer algorithm may render hidden lines as dashed lines, but its effectiveness depends on the static filter pattern that may not be available to evenly dash hidden lines. In comparison, The D-Buffer algorithm is able to generate dashed (with adjustable length of dashes and spaces) hidden-line

segments of any three-dimensional shapes by employing one more buffer: a boundary buffer. But the computing time of the D-Buffer algorithm is slightly slower than the Z-Buffer algorithm.

By adding one more buffer, the status buffer, the M-Buffer algorithm combines the advantages the of Z-Buffer and D-Buffer algorithms, meanwhile overcoming the disadvantages of these two algorithms. Through the M-Buffer algorithm, pictures not only may be rendered quickly, but also may display hidden lines in different styles adaptively according to a user's interest in the objects. However, there are still several improvements that will enhance the performance of the M-Buffer algorithm.

- At present, the algorithm is applied to binary pictures. It is desirable to introduce greylevel line-drawings into the algorithm, so that pictures of more diversity can be produced.

- Currently, the algorithm works only with wireframe pictures. More rendering styles, such as shading, can be added to the algorithm. Even different shading methods, such as phone shading, gouraud shading or raytracing, can be used. In this case, not only can the pictures be color ones, but also objects can be displayed as different styles: wireframe or shading.

- The current M-Buffer algorithm works only with stationary view points. It may

be improved by allowing users to observe the object from different view points or by rotating the object.

- This algorithm may be extended to the multimedia rendering. A medium player, such as audio, can be used to describe special events. For example, when conflict happens between two objects, sound may be produced to indicate the user that the moving object conflicts with another object. Moreover, objects made from different materials may have different conflict sounds.

Bibliography

[1] Airey, J. M., J. H. Rohlf, P. Frederick, and J. Brooks, Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments, In *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24(2), pp. 41-50, 1990.

[2] Booch, G., I. Jacobson, and J. Rumbaugh, *The UML Specification Documents*, Rational Software Corp., website: http://www.rational.com, 1997.

[3] Booch, G., I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley Publishing Co., 1998.

[4] Bouknight, W. J. and K. C. Kelly, An algorithm for producing halp-tone computer graphics presentations with shadows and movable light sources, In *Proceedings of the Spring Joint Computer Conference*, pp. 1-10, 1970.

[5] Bouknight, W. J., A procedure for generation of three-dimensional half-toned computer graphics presentations, In *Communications of the ACM*, 13(9), pp. 527-536, 1970.

[6] Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Splines*, PhD thesis, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.

[7] Clark, J. H., Hierarchical Geometric Models for Visible Surface Algorithms In *Communications of the ACM*, 19(10), pp. 547-554, 1976.

[8] Collins, T. Robert, Multi-Image Focus of Attention for Rapid Site Model Construction, In *http://www.cs.cmu.edu/ rcollins/*.

[9] Constantine, L. L., and E. Yourdon, *Structured Design*, Englewood Cliffs NJ: Prentice-Hall.

[10] Cutts, G., Structured Systems Analysis and Design Methodology, In *Information Technology for Organisational Systems*, Amsterdam: Elsevier, pp. 363-370, 1988.

[11] Dong, Xiaoming, *D-Buffer: A New Hidden-Line Algorithm in Image-Space*, Master thesis, Department of Computer Science Department, Memorial University of Newfoundland, St. John's, NFLD, 1999.

[12] Foley, J. D., A. van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Co., 1996.

[13] Freeman, H., Computer processing of line-drawing images, In *ACM Computing Surveys*, 6(1), pp. 57-97, 1974.

[14] Funkhouser, T. A. and C. H. Sequin, Adaptive Display Algorithm for Interactive Fram Rates During Visualization of Complex Virtual Environments, In *Computer Graphics (SIGGRAPH '93 Proceedings)*, Vol. 27, pp. 247-254, 1993.

[15] Goldberg, A. and D. Robson, *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, Reading, Massachusetts, 1983.

[16] Grimson, W., G. Klanderman, P. O'Donnell and L. Ratan, An Active Visual Attention System to 'Play Where's Waldo', In *Arpa Image Understanding Workshop*, Monerey, CA, pp. 1054-1065, Nov 1994.

[17] Hitchner, L. E. and M. W. McGreevy, Methods for User-Based Reduction of Model Complexity for Virtual Planetary Exploration, In *Proceeding of the SPIE-The International Society for Optical Engineering*, Vol 1913, pp. 622-636, 1993.

[18] Holloway, R. L., *Viper: a Quasi-Real-Time Virtural-Worlds Application*, UNC Technical Report No. TR-92-004, Department of Computer Science, University of North Carolina, Chapel Hill, NC, 1991.

[19] Interrante, V., H. Fuchs and S. M. Pizer, Conveying the 3D Shape of Smoothly Curving Transparent Surfaces via Texture In *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 2, pp. 98-117, April-June, 1997.

[20] Kemeny, A., A Cooperative Driving Simulator, In *Proceedings of the international Training Equipment Conference (ITEC)*, London, UK, pp. 67-71, 1993.

[21] Larman, C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice-Hall, 1998.

[22] Lindstrom, P., D. Koller, L. F. Hodges, W. Ribarsky, N. Faust, *Level-of-Detail Management for Real-Time Rendering of Phototextured Terrain*, Technical Report No. TR96-06, Graphics, Visualization and Usability Centre, Geogia Institute of Technology, Atlanta, GA, 1995.

[23] Metelli, Fabio, The Perception of Transparency *Scientific American*, v. 230, No. 4, pp. 91-98, April, 1974.

[24] Nakayama, K., S. Shimojo and V. Ramachandran, Transparency: relation to depth subjective contours, luminance, and neon color spreading, *Perception*, Vol. 19, pp. 497-513, 1990.

[25] Norman, D. A., *Turn Signals are the Facial Expressions of Automobiles*, Reading, MA: Addison-Wesley, 1992.

[26] Ohshima, T., H. Yamamoto and H. Tamura, Gaze-Directed Adaptive Rendering for Interacting with Virtual Space, *Proceeding of the IEEE Virtual Reality Annual International Symposium (VRAIS)*, Santa Clara, CA, pp. 103-110, 1996.

[27] Parnas, D., On the criteria to be used in decomposing systems into modules, *Communication ACM*, 15(2), pp. 1053-1058, 1972.

[28] Reddy, Martin, *A Measure for Perceived Detail in Computer-Generated Images* Technical Report ECS-CSG-19-96, Department of Computer Science, University of Edinburgh, 1996.

[29] Rumbaugh, J., I. Jacobson and G. Booch, *Unified Modeling Language Reference Manual*, Addison-Wesley Publishing Co., 1999.

[30] Schachter, B. J., Computer Image Generation for Flight Simulation, *IEEE Computer Graphics and Application*, Vol. 1, No. 4, pp.29-68, 1981.

[31] Schade, O. H., Optical and Photoelectric Analog of the Eye. *The Optical Society of Americas*, Vol. 46, No. 9, pp.721-739, 1956.

[32] Seligmann, D. D. and Steven Feiner, Automated Generation of Intent-Based 3D Illustrations In *Computer Graphics*, Vol. 25, No. 4, pp. 123-132, July 1993.

[33] Sutherland, I., *Sketchpad, A Man-Machine Graphical Communication System* PhD thesis, Massachusetts Institute of Technoloty, Jan. 1963.

[34] Warnock, J. E., *A hidden line algorithm for halftone picture representation*, Technical Report TR 4-5, NTIS AD 761 995, Computer Science Department, University of Utah, Salt Lake City, UT, May 1968.

[35] Warnock, J. E., *A hidden-surface algorithm for computer generated half-tone pictures*, Technical Report TR 4-15, NTIS AD 753 671, Computer Science Department, University of Utah, Salt Lake City, UT, June 1969.

[36] Watkins, G. S., *A Real Time Visible Surface Algorithm*, PhD thesis, Computer Science Department, University of Utah, Salt Lake City, UT, 1970.

[37] Watson, B., N. Walker, and L. F. Hodges, A User Study Evaluating Level of Detail Degradation in the Periphery of Head-Mounted Displays, In *Proceedings of the FIVE '95 Conference, QMW*, University of London, UK, pp.203-212, 1995.

[38] Weaver, P., *Practical SSADM, Version 4*, London: Pitman, 1993.

[39] Wirth, N., Program development by stepwise refinement, In *Communication ACM*, 14(4), pp. 221-227.

[40] Wirth, N., *Systematic Programming: An Introduction*, Englewood Cliffs NJ: Prentice-Hall, 1976.

[41] Wloka, M. M., *Incorporating Update Rates into Today's Graphics Systems*, Technical Report CS-93-56, Department of Computer Science, Brown University, Providence, RJ, 1993.

[42] Wylie, C., G. W. Romney, D. C. Evans and A. C. Erdahl, Halftone Perspective Drawings by Computer, In *Proceedings of the Fall Joint Computer Conference*, pp.49-58, 1967.

[43] Yan, J. K., Advances in Computer Generated Imagery for Flight Simulation, In *IEEE Computer Graphics and Applications*, 5(8), pp.37-51, 1985.

[44] Yuan, Xiaobu and H. Sun, P-Buffer: A Hidden-Line Algorithm in Image-Space, In *Computer & Graphics*, 3(21), pp. 359-366, 1997.