THREE-DIMENSIONAL LINE TEXTURES FOR INTERACTIVE ARCHITECTURAL RENDERING

by

Kristin Carrie Potter

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

School of Computing

The University of Utah

May 2003

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Kristin Carrie Potter

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Richard F. Riesenfeld

Peter Shirley

Charles Hansen

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of Kristin Carrie Potter in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date	Richard F. Riesenfeld Chair: Supervisory Committee

Approved for the Major Department

Thomas C. Henderson
Chair/Director

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

Architects and other design professionals create presentation graphics that intentionally avoid full realism. Successfully automating this style of imagery favorably affects the speed and cost of producing such illustrations. Automatically creating line textures allows a user interactively to modify and navigate a three-dimensional (3D) architectural scene while still maintaining the aesthetic appeal inherent of hand-drawn illustrations. Carefully choosing the set of lines to be drawn allows the direct use of 3D line primitives on commodity graphics hardware. The resulting system produces interactive drawings of high visual quality that are free of animation artifacts such as blurring or popping. This system also allows the level of abstraction of the rendered scene to change dynamically.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

Thanks to Amy Gooch, Bruce Gooch and Peter Willemsen for their contributions to this work.

CHAPTER 1

INTRODUCTION

Architects and other design professionals create presentation graphics utilizing a stylized convention devoid of photorealism that communicates the diverse aspects of a design. Intentionally avoiding full realism in different aspects of an image allows the implicit communication of distinct ideas throughout a scene. Successfully automating this style of imagery will favorably affect the speed and cost of producing such illustrations and allow interactive modification and navigation through three-dimensional (3D) scenes while maintaining an adequate level of aesthetic appeal implicit of hand drawn illustrations. A nontraditional approach using automatically placed 3D lines rather than texture mapping can allow the user interactively to manipulate the image while maintaining the speed needed for interactive walkthroughs and aesthetic appeal of hand drawn illustrations.

1.1 Motivation

There exists a need for images within a vast spectrum between highly realistic and completely nonphotorealistic depiction. A photorealistic image can lead a viewer to inappropriate assumptions about the validity, accuracy, and degree of certainty of the objects in the scene because the image tries to be as close to a photograph as possible, which suggests that a camera could have actually taken the picture. Many computer-generated images are created with the specific goal of photorealism. This goal, however, may not be appropriate in certain contexts. Often the output images of a CAD system are used to communicate the development stages of the design process and other design concepts. CAD plots and shaded images lead to a clearer understanding of the scene and objects in the image including spatial relationships because of their photograph like imagery. However, these realistic renderings are often misleading, since many of models used to create such images are not accurate or complete, whereas the depiction style suggests that the scene is accurate, complete and permanent. For this reason, many visualizations of architecture or archaeological reconstructions are sketches or line drawings with little detail. Hand drawn illustrations compensate for a lack of information by adding only enough detail to express the amount of information known, and using expressive techniques to convey the inaccuracies or incomplete aspects visible in the scene.

Presentation graphics are stylized images that incorporate realistic technical elements with aesthetic appeal. These images are used to visually express ideas that are difficult to communicate effectively otherwise. If presented with a highly realistic image, a client is often hesitant to discuss the overall design or its changes because the image seems to represent a complete idea that cannot be changed. A representation that resembles a sketch is an effective way to communicate the mutability of the design and encourage the client to interact in the design process. Effectively expressing the ideas can enhance the relationship between client and designer. Design professionals are aware that a viewer can be directed to an understanding of an image by using different illustration techniques in distinct ways to direct focus, separate parts from a whole, or express completeness of a design. Combining drawing styles is an effective way of communicating differences in an idea, while maintaining the unity of the work. The primary purpose of images used by these professionals is to express design concepts to a client. In addition to sparking viewer interest and appeal, information must be portrayed in a specific manner to avoid misleading the viewer, while allowing for a high degree of understanding. Specifications of a design, level of completion, and desired input must be explicitly conveyed. The style used to present the images has a major influence in how the client will interpret them. On the one hand, a fully realistic image will convey the sense of a completed design. On the other hand, a rough sketch suggests a design in the early stages. The professional designer must express sophisticated judgment in selecting a style for depicting ideas to convey as precisely as possible concept and intent of the design as well as the degree of completeness of the ideas. In addition, the images must interest the client, by offering aesthetically pleasing images. Thus, the effect of the rendering style on the user must be taken into account from the earliest time [22].

Illustration style has been found to have an affect on the interpretation of an image [24]. Highly realistic rendering styles can lead a viewer to conclusions about an image that may not be justified. For example, a photorealistic rendering may lead to interpretations of being complete or inviolate in a design process [25, 23] whereas a line illustration can convey mutability. The illustration style of an image can also have an effect on the attention and focus of the user [22].

There are numerous stages of the design process, each of which should be expressed using different rendering techniques. Integrating all levels of the design into one image is effective in understanding how the parts fit together, and how changes to those parts affect the whole design. Sketch drawings, which have a hand-made feel, likely spark more curiosity and interest in the viewer because of the irregular form of the lines, and invite the viewer to invest more cognitive effort in understanding the image. This can result in a greater willingness of the viewer to interact and discuss the image, which is desirable during the beginning stages of a design. On the other hand, realistic images are preferable during the latter stages of the design process when the design is becoming more solidified. All of these techniques share the common goal of creating an image that is both pleasing and easy to understand.

Using computer graphics to create presentation graphics allows one to explore architectural sites or archaeological reconstructions in a manner that is not possible with illustrations on paper. However, many of the details such as doors and windows are not completed or are long since lost, and although they are needed to visualize the site, including them is a matter of extrapolation. Four categories of data can be defined: i) findings that have actually been excavated; ii) deductions that can be directly derived from the site; iii) analogies that, though not found during excavation of this site, are found at similar sites; and iv) assumptions that are details having no empirical basis in the excavation. The style of visualization is important in conveying the category of the data. Nonphotorealistic renderings are preferable to researchers exploring hidden structures in data, while visitors in a museum prefer realistic visualizations that present a picture process of how the site once looked [13]. Nonphotorealistic rendering can move the creating of such images onto the computer, allowing an accurate representation and maintaining the viability of the model data. Emulating line illustrations can maintain the aesthetic appeal inherent in hand illustrations, and allow the expression of information over and above the geometric model through the use of varied line styles [23]. In addition, computer graphics allows the user to create a 3D world that is much less costly than a real scale model, and one that can be refined at will.

Figure 1.1 is an example of an interactive walkthrough of an archaeological reconstruction with the sketchiness of the line primitives varied according to the level of confidence in the data. The image depicts a Mayan temple that exists today in ruins. Archaeological sketches have been made depicting how the site probably once looked. The rendering in the presented figure encodes the information about the site's current state with the reconstruction information from the archaeologists as well as their expert speculation of the final look of the site. The lowest level of the model has a stone texture that is rendered using a more technical form, indicating this level of the temple is still in existence. The second and third levels of the temple are known to have existed because their rubble lies atop the lower level. These levels are depicted with a sketchy style, suggesting that they are not in the same state as the lowest level. The top level is a hut that has been suggested to have existed by some archaeologists; however this is a matter of conjecture and it is also possible that the top level was some sort of alter, or the like, instead. Thus, the rendering style of this top level is a sketchy outline without any hint of texture. The resulting final image not only conveys a sense of how the temple once may have looked, but also explicitly lets the viewer know what the designer put into the image as assumptions and guesses. This image can then be used as a tool to allow an archaeologist to express visually his or her ideas of a site and evoke discussion.

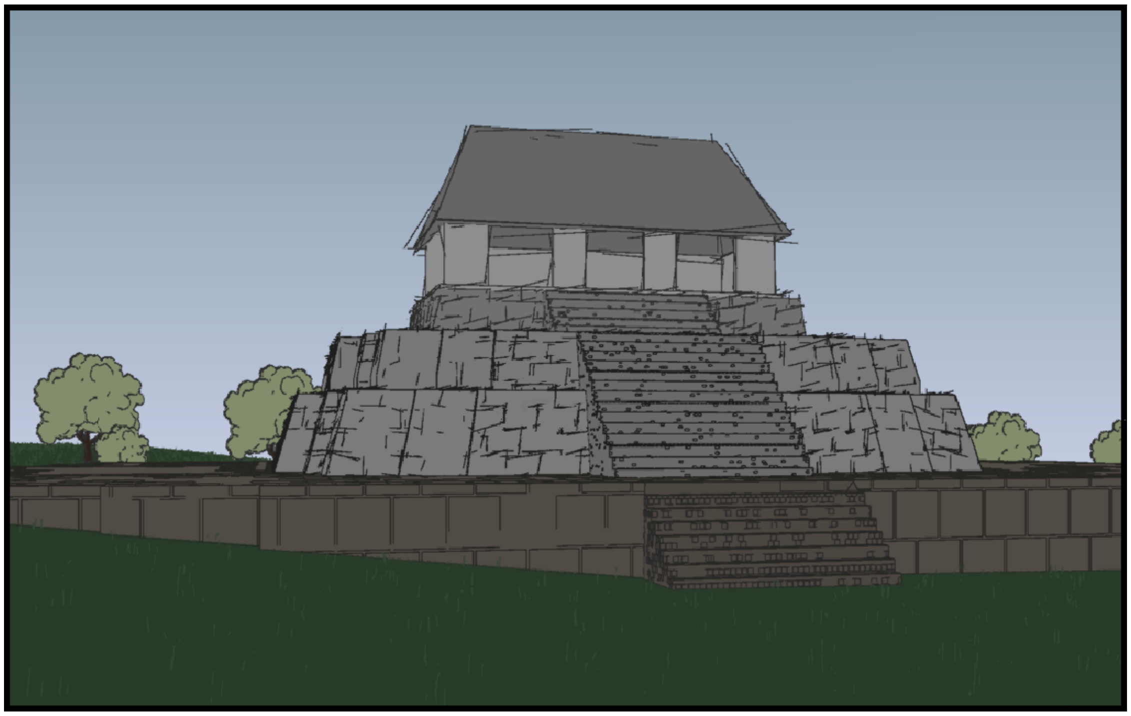

Figure 1.1. An example image from an interactive session with the system.

It is often difficult to verbally communicate ideas concerning the look of a site. Three-dimensional mockups are time consuming, expensive, and hard to change. Fully realistic renderings of speculative information may disturb a colleague who misinterprets the meaning of the image. Representations of information should reflect the status and type of information. Thus a rendering technique that can embody various ideas into one image is beneficial in a variety of ways.

1.2 Problem Statement

The work presented focuses on the generation of architectural scenes in a manner that conveys multiple levels of abstraction in a single image and uses line primitives rather than texture mapping to emulate the hand drawn strokes that would be placed by an artist. The combination of these ideas is used to create a prototypical system that allows interactive manipulation and virtual reality walkthroughs of a scene.

Colored line drawings are often used to depict scenes containing buildings. In these images lines are used to depict both edges and material properties. The sketchiness of the lines is often varied to indicate the degree of completeness in an architectural design or the level of confidence in an archaeological reconstruction. An example archaeological reconstruction with varying levels of sketchiness is shown in Figure 1.1. A method is described here for generating interactive renderings.

The important characteristics of an interactive colored line drawing system are:

1. high visual quality of individual frames;
2. animation free dynamic artifacts, such as popping; and
3. high frame rate.

The first two items suggest using 3D line primitives, as they can be antialiased in screen space producing high visual quality. In addition line primitives do not need level-of-detail management to maintain constant width or brightness in screen space. However, it is natural to think that interior lines should be rendered using texture mapping for efficiency. Indeed, texture mapping has been used effectively to accomplish interior line rendering by others [5], who used careful generation of Mip mapped line textures to avoid dynamic artifacts. Unfortunately, this technique makes the line textures static, so line sketchiness cannot be varied at runtime.

The question remains of whether 3D line primitives can be used while maintaining an interactive frame rate. Although lines are not used in most graphics programs, they are highly optimized by hardware makers because of the CAD market. Using lines directly has several advantages over texture mapping:

- line primitives can be antialiased without a multipass algorithm
- line primitives can have their sketchiness varied at runtime by perturbing vertices in a hardware vertex program
- line primitives preserve their width in screen space even for extreme close-ups.

The last item could be viewed as an advantage or a disadvantage depending on one's priorities; having constant width lines in screen-space makes for a clean drawing reminiscent of the type drawn by human draftsmen, but eliminates line-width depth cues.

The main contribution is a prototypical system in which high frame rates can be achieved using line primitives in scenes of realistic complexity. An algorithm is provided to automatically place line textures on objects in order to perform material property "indication," i.e., a small number of texture elements indicates that the entire surface is textured. Finally, a program is presented using 3D lines that is relatively simple to design and build, making line primitives a practical alternative to texture mapping with respect to software engineering as well as efficiency issues.

CHAPTER 2

PREVIOUS WORK

Previous work related to this work can be divided into work in the area of nonphotorealistic rendering and architectural rendering. Nonphotorealistic rendering focuses on emulating the work of artists by simulating the medium, or by simulating the process that the artist goes through to create the image. Architectural rendering looks to move the architectural design process onto the computer, visualize architecture and archaeological reconstructions and models, and create walkthroughs and virtual environments. A synthesis of these two types of rendering can create systems with more expressive power than each individually.

2.1 Nonphotorealistic Rendering

The move from pen and pencil to the computer requires that we explicitly define characteristics that are inherent in traditional media. Strictly using lines and curves to represent the strokes of a drawing creates an image that is mechanical and cold. A typical stroke of a pen or pencil exhibits thickness and waviness, which vary throughout the stroke. Properties such as these can be simulated by adjusting the weight and direction of a series of computer drawn line segments. The direction or curve of each stroke also contributes to the comprehension of shape and can be drawn following contours of the model. Completely automating this process is difficult, so most systems need human interaction to define where the computer should place strokes [8].

An automatic 3D rendering system is available that generates computer illustration by employing the techniques of traditional pen-and-ink artistry [26]. Because a scene rendered with pen-and-ink contains only pen strokes, all information must be conveyed by combining the individual strokes in meaningful ways. The characteristic of single strokes as well as the proper grouping of multiple strokes will express the tone, texture, outline, lighting and material properties of an object. Converting hand pen-and-ink techniques presents an interesting change to the traditional graphics pipeline. Tone and texture are no longer separate since both are conveyed through lines. Clipping must be done so as not to look mechanical, and the outlining of models has to convey material as well as boundary information. Prioritized stroke textures are collections of pen strokes that contain both material and tone information. The set of strokes is drawn highest priority first, until the desired tone is achieved. Higher priority strokes contain more important information such as the defining texture details like brick outlines. Lower priority strokes contain tone information. The variability of the stroke textures allows for multiresolution output. Indication is another technique of artists that should be employed when creating computer generated illustrations. Drawing a consistent tone and texture level across an entire image will result in monotony and an overload of information. Instead, artists hint at a texture and tone in important areas of the image, and allow the imagination of the viewer fill in the missing information. This technique leads to more interesting images, as well as lending itself nicely to the economy of the illustration. However, indication is a difficult problem for artists to master, so an automatic method is extremely challenging. The solution is to allow input from the user such that the important areas of the model are marked by the user, and the stroke placement method places more texture around the user defined areas. Additional texture and model details that are not captured by the above techniques are defined by outlines. Specifically, outlines are used to differentiate between areas of similar tone but varying texture.

Another technique is presented in [11] that incorporates both cartoon shading and pencil sketch textures to create an artistic look. The cartoon shader called Painter simulates the artist painting a cell that has already been inked. Hard shading is achieved by encoding the material color and the shadow color in a 1D texture map and using as the texture coordinate the dot product of the surface normal at each vertex and the light position. If the resulting dot product is negative, 0 is used for the texture coordinate. The result is a hard line between the object in light and the object in shadow. Variations on the shading can be achieved by encoding the light information in higher dimensional texture maps or by using more than two colors in a single dimension texture. In the same manner, pencil strokes can be encoded into a texture map and rendered with differing density to create tone. The Inker module of the system enables the model to be rendered using stylistic techniques such that the model itself has a hand-drawn look. The silhouette, crease and border edges are the only edges rendered since these edges help define the space occupied by the model. These edges are drawn using strokes from a texture map that are curved or straight depending on the slope of the important edge. These techniques are easily scaled to animation and moving models. As much of the computation is done as a preprocess, the remaining calculations are quicker and can readily be moved to hardware. Motion is indicated with motion lines that are calculated by tracking the translation of an object frame to frame. The system works at interactive rates on a range of platforms.

Much work in NPR is done with respect to a single image using nondeterministic methods to achieve images resembling hand-drawn illustrations. Interactive NPR algorithms can suffer from a lack of frame to frame coherence. The stochastic nature of processes used to create single images can result in an unsmooth transition to three dimensions or animations due to the moving and popping of the strokes used to simulate the artist's hand. This is caused by NPR illustrations that simulate the single instance of an image, rather than a repetitive sequence needed for animation. Therefore, the paint or pen strokes from one frame must subtly find their way to the next frame to avoid the popping and flashing artifacts, effects that too often occur. An additional challenge is maintaining the tone and texture of an object's surface. One solution to maintaining frame coherence is achieved by separating the style of a line from the path of the line and reconstructing line segmentations that disturb the coherence of a drawing path. The parameterization of the way of drawing the line allows the line style, width and saturation to vary throughout the animation. The coherence restraint may also be lessened to create the look and feel of motion.

The system modifies the daLi! system to allow for intersecting models [14]. Other solutions have been found using texture mapping approaches. Hatch maps can be used to maintain the structure of a set of lines by using less lines in the distance to maintain tone; and ink maps can add artistic detail [5]. Art maps can be used in image based rendering to maintain coherence between paint strokes [10].

To create communicative illustrations, the important features of the scene must be identified. The shape of the objects comprising a computer modeled scene can be better understood if geometric features such as silhouettes, creases, boundary and contour lines are identified. The G-buffer is introduced as a way to separate geometric properties such as depth or surface normal from physical properties such as shading and texture mapping. This separation allows artificial enhancement processes to be applied to the geometric properties, which helps in understanding the shape features of the image. The enhancement of border and internal edges and contours through darkened or lightened lines or curved hatching give the viewer information about the image that would be much less apparent without the enhancements. However, determining which enhancement process to use, and to what extent, is a difficult task that can be accomplished only through user trial and error. Thus, the G-buffer stores intermediate values which allows fast recomputation, so interactively altering the image becomes possible [21]. The style in which these features are rendered can also aid in comprehension. The type of line, such as dashed, dotted, thick or thin, can express direction, distance and location. End point conditions can convey the relation of the line with respect to other lines and surfaces in the scene [3]. Line illustration conventions are apparent in hand-drawn work, and have been adopted by the computer graphics community.

2.2 Architectural Rendering

Current work in architectural rendering is composed of systems aimed at moving architectural design onto the computer, visualizing reconstructions of ancient architecture, and creating architectural walkthroughs. The differences among existing systems relate primarily to the interface with the user. Architectural CAD systems specialize in giving the user an interface that is similar to 2D paint programs, and giving the user tools to create images that resemble the hand-drawn work of artists. Reconstruction systems tend towards an accurate and objective display of archaeological information. This information can be used to speculate on the appearance of ancient architecture. These systems must be careful to convey the difference between known structures, that is those seen today, and approximations of the appearance of the structure in ancient times. The third type of architectural rendering focuses on walkthroughs. Most work in this area is done to reduce frame rates and improve performance. The work presented here will combine various elements from all three types of architectural renderings.

The Piranesi system [20] is an interactive system that allows the user to paint a 3D model using the same techniques used in a 2D paint program. The results from the system are images that look as though they were images created by architects. The interactive interface begets a working relationship among the user, client and image. Presentation graphics can be created and refined quickly, making the computer a medium rather than a tool, in essence, a digital sketchpad.

In the same sense, SKETCH [28] is a system that allows the user to model in a gestural manner. The advantage is that the user can doodle the model, and the system transforms the doodle into an approximate model. SKETCH is not designed specifically for architectural rendering, but the idea of freehand drawing is an integral part of the architect's job. Often it is also important to be able to use a sketchy model to inform the client of the state of a design. The deliberate impreciseness of the model is an important aspect of the early stages of design. A combination of a quick sketch and a visually pleasing rendering could possibly convey more information than either separately.

The combination of multiple rendering styles has been used in the reconstruction rendering of ancient architectural sites. It has been noted that using highly realistic rendering styles can lead a viewer to conclusions about an image that may not be empirically justified [23], since often the information being displayed is from archaeological sites or introduced by educated guessing. A convincing photorealistic image can be suggestive of reality, and may limit the discussion and revision of the current idea of the look of site [25]. To circumvent this problem, nonphotorealistic rendering is used [13]. Creating a line drawing of the architecture simulates the hand-drawn effect typically used by architects and archaeologists. The computer enables interactive revisions and walkthroughs [15]. Multiple styles of nonphotorealistic rendering can also be used to portray the different degrees of known information. A realistic image or photograph is used to display the existing foundation, whereas line drawing and sketchy rendering are used for the parts no longer in existence but known to have been present, and elements that are only guesses of what may have been, respectively. Also, changing the contrast of an image can direct focus or emphasis importance [15]. Overall, it is important to use different rendering styles to display various levels of information and control the interpretation of an image.

Architectural renderings and presentation graphics are essentially works of art [9, 4]. Although they may contain high levels of information, and be of a somewhat technical manner, their ultimate job is to appeal to the viewer. As such, the final look of any image is of utmost importance. The aesthetic appeal of architectural rendering has a major impact on the interest of the viewer. The images created must convey information in a manner that is not only clear and precise, but also pleasing. Illustration style has been found to affect the interpretation of the image [24]. A designer will take illustration style into account when creating an image, evaluating how the drawing style conveys specific information. Photorealism leads to interpretations of permanence or completion. Line illustration conveys a sense of incompleteness that allows for discussions between designer and client that may otherwise be difficult. Illustrations have numerous advantages in some circumstances over realistically rendered computer graphics [26].

Architectural renderings often contain information about building and landscape materials. These materials inform the viewer of the environmental setting of a building site, the materials used in the project, as well as an overall feeling of the completed project. Inclusion of material detail also aides in comprehension of scale. The rendering style of these materials must be considered in order to maintain coherence in the image. Nonphotorealistic trees [2] that are created by extracting important features of models work nicely in these types of images since their line styles may be altered. Stone, wood, and plant materials can be generated using random disturbances in procedurally generated textures [27], or through perturbing corners of a rectangular grid [16].

The final group of work done in architectural rendering focuses on using an architectural model for an interactive walk-through. The aim is to aid the client in understanding a model by allowing the client to wander around the model. Most work done in this field has been done to speed up the rendering rates. Changing between texture maps and real geometry has been presented [1, 18], as well as limiting the amount of geometry rendered per frame [7, 6]. This work is novel in that we use 3D lines instead of texture and Mip mapping. Lines in 3D do not suffer from the artifacts that occur with texture mapping because the lines remain entities in screen space. Upon initialization, all lines are placed in a display list. No new lines are generated or removed during runtime, therefore no visual artifacts occur. In addition, the endpoints of the lines can be perturbed using a vertex program, allowing the texture to change during runtime, a feature that is difficult to achieve using texture mapping.

CHAPTER 3

LINES VS TEXTURE

Using 3D line primitives rather than texture mapping is a viable alternative when visual quality is of utmost importance. The rasterization of line primitives can produce lines that maintain width and brightness in screen space no matter how close the view plane, output cleanly on printers, and can change interactively without having to use any special techniques. Figure 3.1 shows various views of a cube textured with 3D lines.

3.1 Line Scan Conversion

Line rasterization consists of scan conversion and clipping. The line itself is an infinitely thin line segment represented by two 3D endpoints. To scan convert the line, the pixels that the line touches must be computed. Since there are a finite number of pixels having a predefined size, the line must be approximated by the pixels often leading to aliasing problems.

The most common algorithm for scan converting lines is the midpoint line algorithm. This algorithm uses only integer or floating point arithmetic as opposed to fractional computations, and can be done incrementally such that previous calculations can be used to improve efficiency. The algorithm minimizes the distance from the pixel to the true line, thus giving the best-fit approximation to the line.

To determine the pixel approximation of a line, the midpoint line algorithm determines the midpoint between the two next candidate pixels and chooses the next pixel based on which side of the line the midpoint falls. The starting pixel is the endpoint of the line, or if the endpoint is a floating point, the pixel closest to the endpoint of the line. An important need of a scan conversion algorithm is to create a continuous line without missing pixels. This forces the choice of the next

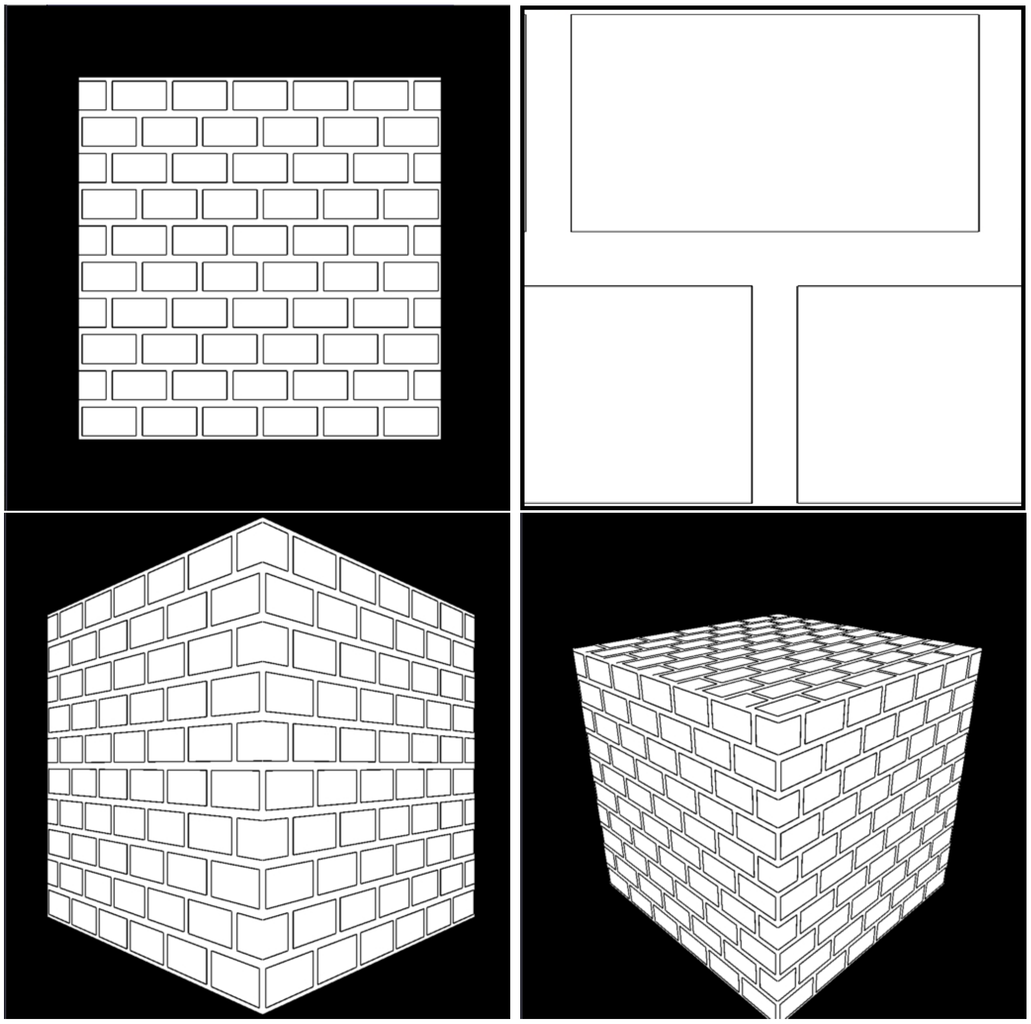

Figure 3.1. Antialiased 3D lines.

two pixels to be between the pixels that touch the current pixel. By forcing the slope of the line to be between 0 and 1, the choice of the next pixel becomes even more restricted. It has to be the pixel 1 increment in the x direction and the pixel 1 increment in the x direction and one increment in the y direction. All other slopes can be generalized by a rotation of the zero to one slope line about the principal axes. Now that the next two possible pixels are determined, the midpoint between the two pixels is found with respect to the function representing the line. The midpoint, which for all pixels will be 1 plus the previous x value and 1/2 plus the previous y value, is plugged into the implicit line equation. A positive return value indicates that the midpoint lies above the line, and thus the lower pixel should be used. A return value less than 0 tells the algorithm to choose the upper pixel, and a 0 return value shows that the line passes directly through the midpoint, thus either pixel can be chosen, and for consistency, the lower of the two pixels is selected. Additionally, the midpoint for the next grid increment can be determined by adding 1 in the x direction if the lower pixel is chosen as the next pixel, or by adding 1 in the x direction and 1 in the y direction if the upper pixel is chosen. Thus the value of the function of the line does not have to calculated with the next midpoint, the calculations from the previous iteration can be simply added onto.

3.2 Antialiasing

Once the line is scan converted, it must be antialiased because of the discrete nature of the pixels on an output device. A pixel is turned either on or off when it is chosen by the scan conversion algorithm and this results in a line that is jagged, having a staircase appearance. An example of this effect can be seen in Figure 3.2. This effect is reduced when the resolution is increased; however this does not solve the problem, it simply reduces its effect when viewed from similar viewpoints as the lower resolution image. Zooming into the higher resolution image will still display undesirable characteristics. The solution to this problem can be to antialias the line primitive, the entire screen, or a combination of both.

Line antialiasing is done by calculating the area of each pixel that the line covers. This is different from simple scan conversion because both of the candidate next pixels may be colored. The color of the pixels surrounding the line is determined by how much of each pixel contains the line and how close to the center of the pixel the line passes. Thus, for a black line, a pixel with more area covered by the line will be darker than a pixel with less coverage area; however pixels with the same coverage area may be colored differently depending on how close to the center of the pixels the line lies. Full screen antialiasing does not take into consideration the

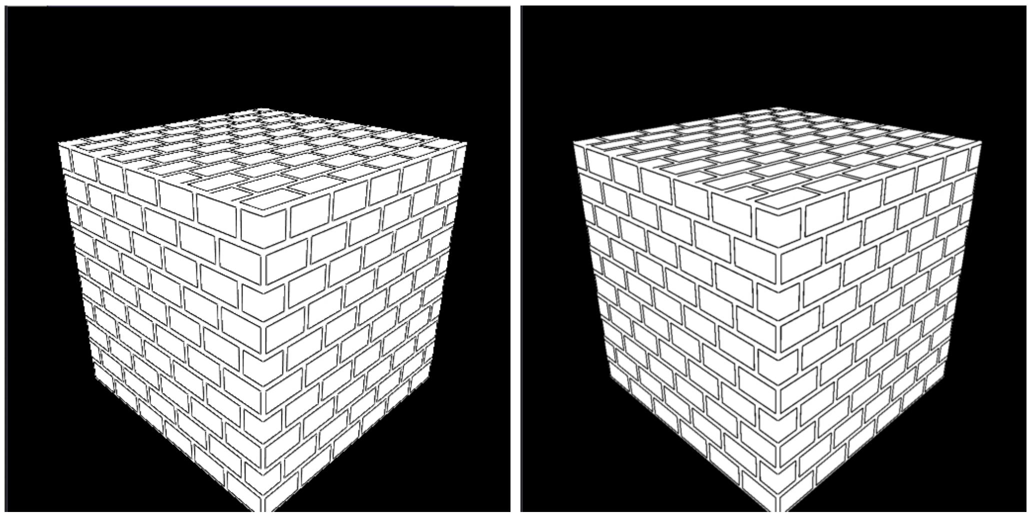

Figure 3.2. Aliased (left) and antialiased (right) 3D lines.

primitives drawn into the screen, but uses various methods of filtering to reduce artifacts.

3.3 Texture Mapping

A common method for placing texture onto a model is texture mapping. This technique takes a predefined image and maps it onto the image, similar to the way a label is pasted onto a soup can. There are many techniques to deal with distortions and other problems that would make this method less desirable. MIP maps are used to allow for the texture to be visually pleasing at multiple distances. This leads to the question of what the texture should look like up close and far away. Ideally, the texture map should reduce in size proportionate to the textured object when the viewpoint moves away. To do this without introducing disturbing visual artifacts, the texture must be filtered to an appropriate size.

The creation of a Mip map is done by specifying the size of the texture map, and then creating images in sizes decreasing by powers of two until an image with size one by one exists. The images are typically filtered versions of the original map that have been averaged such that four corresponding pixels in a large image reduce to one averaged pixel in an image with a size that is a power of two smaller. The effect of all of these different images is dependent on the filtering method used to combine them.

Filtering of the Mip maps depends on the size of the texture images in relation to the polygon being texture mapped. If the size of the polygon is larger than the largest Mip map image, then magnification occurs and thus the largest, or base level Mip map image is used. If the polygon is smaller, the correct Mip map level must be chosen. This is called *minification*. With minification, there exists an option to choose the Mip map level that is closest, or interpolate two Mip map levels. Figure 3.3 shows the results of different Mip map filtering as well as a comparison to 3D lines. In addition, anisotropic texture filtering can further improve artifacts.

3.4 Comparison

In many instances, 3D lines render to the screen with higher visual quality. As shown in Figure 3.3, close up 3D lines maintain their screen space width and antialias well. Texture mapping causes the texture lines to blur when the viewpoint is extremely close, or where the lines are diagonal. An interesting artifact of texture mapping can be seen in the close-ups in center and rightmost columns of Figure 3.3. The white of the brick turns grey when texture mapped due to filtering. For these extreme viewpoints, texture mapping can easily be optimized using different filter methods to produce acceptable visual quality; however, this must be done with the desired viewpoint in mind. 3D lines, on the other hand, seem to give the desired visual quality at all viewpoints, and although maintaining screen space width may not always be desirable, it is a quality that is appropriate for this type of image creation. In addition, the implementation of texturing with 3D lines is straightforward, similar results using Mip mapping techniques have complicated implementations. Overall, 3D lines are a viable alternative to texture mapping, and although they may not be appropriate for all contexts, their place in the graphics community should be recognized.

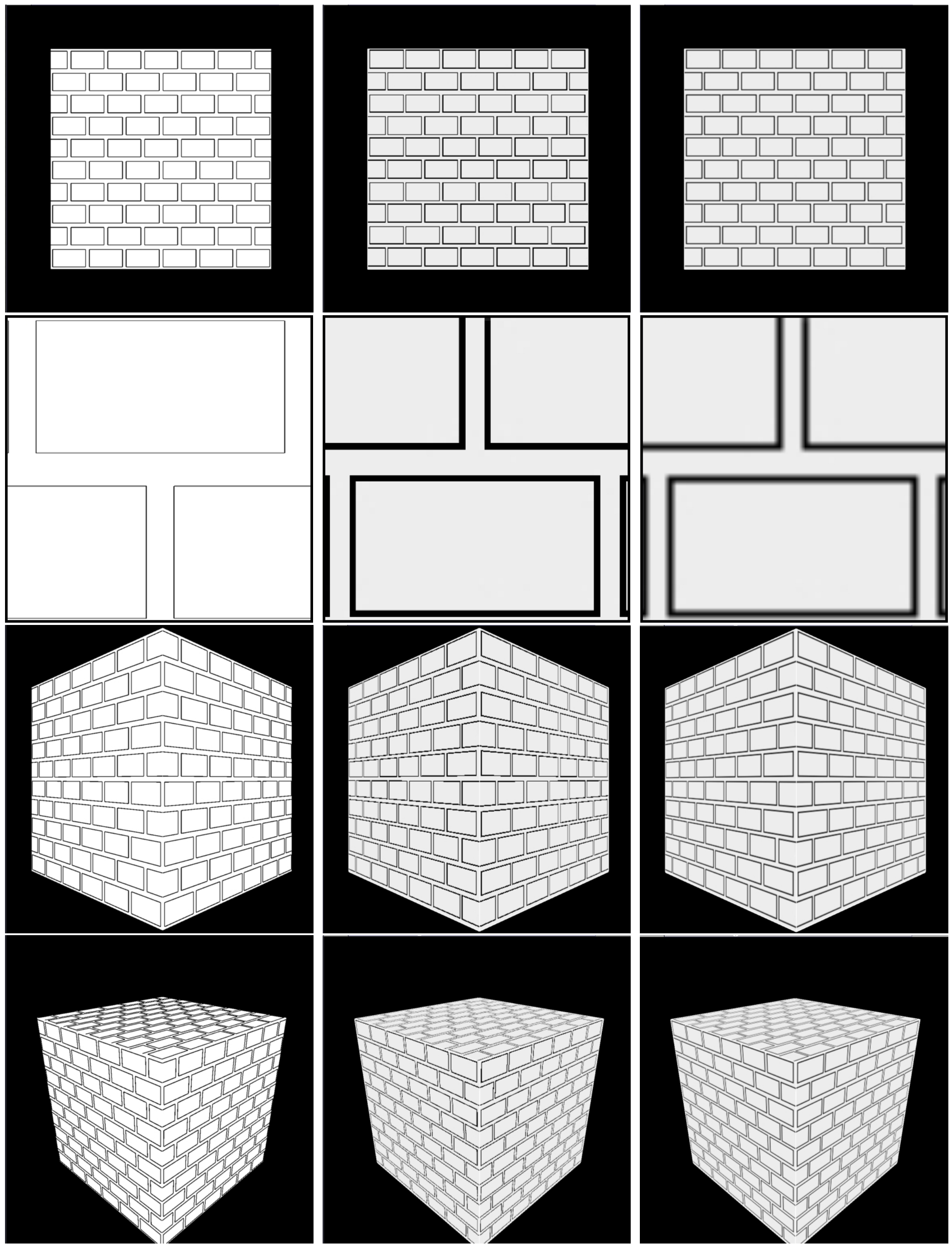

Figure 3.3. 3D lines (leftmost column), nearest filtered texture mapping (center column) and linear filtered texture mapping (rightmost column).

CHAPTER 4

ALGORITHM

The algorithm presented here consists of preprocessing the model to find important edges, determining the placement of and laying the texture, and finally clipping the texture to the model. New techniques include using 3D lines as an alternative to texture mapping, automating the indication of texture such that texture is minimized and placed sparsely across the model as well as along important edges, and creating sketchy lines through vertex programs. The system uses simple algorithms and currently supports a variety of textures such as bricks, stone work, shingles, thatch, stucco, and siding. Figure 4.1 shows examples of some of the most commonly used textures.

4.1 Feature Edges

An important clue to understanding an architectural model is to be able to easily see the features of the building. These features are distinguished by the corners and edges that separate architectural elements and define the overall shape of the building. However, the computer models used to create the scenes are polygonal meshes, made up of hundreds of triangles, rather than a single polygon per face. Rendering the outline of every triangle in the mesh results in too much irrelevant information being displayed, leading to an unintelligible image. Instead, only the outline of the building should be rendered. In addition, augmenting the important edges of the model enhances these visual cues and leads to a better understanding of the model's shape [21].

To find the important edges of a model, the model is first divided into material groups (e.g., bricks or grass) and then processed to identify the creases (e.g., the corner of a building) and boundary edges (e.g., a window). Texture is then placed

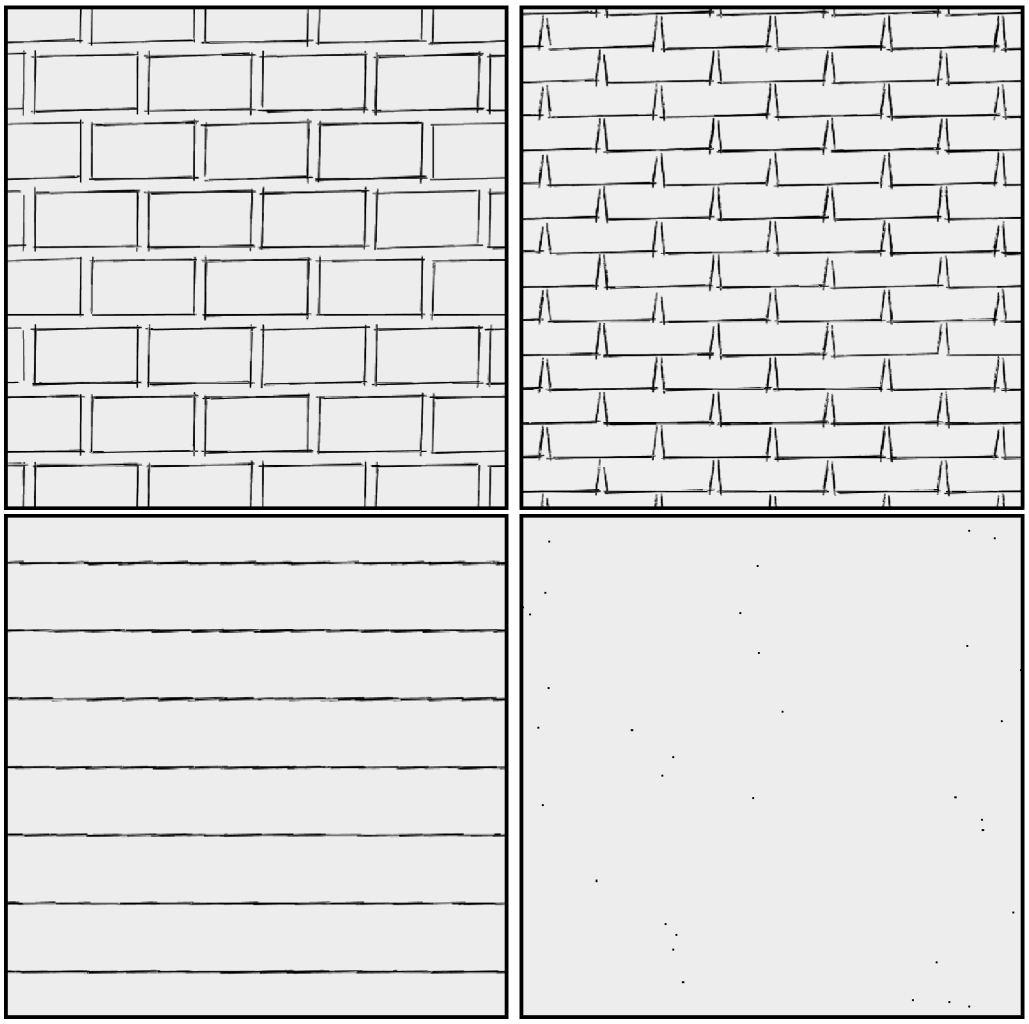

Figure 4.1. Examples of (clockwise from upper left) brick, shingle, siding, and stucco textures.

along these edges leading to a higher texture density along material boundaries and feature edges. Figure 4.2 illustrates the enhancement of feature edges by a higher texture density.

A feature edge is an important edge of a model that can be categorized as a crease, border or silhouette. Crease edges adjoin two polygons whose surface normals have a dihedral angle greater than θ, for some threshold value of θ (see Figure 4.3). A border edge is one that is only contained in a single polygon, or which separates materials (see Figure 4.4). A silhouette is an edge that adjoins a

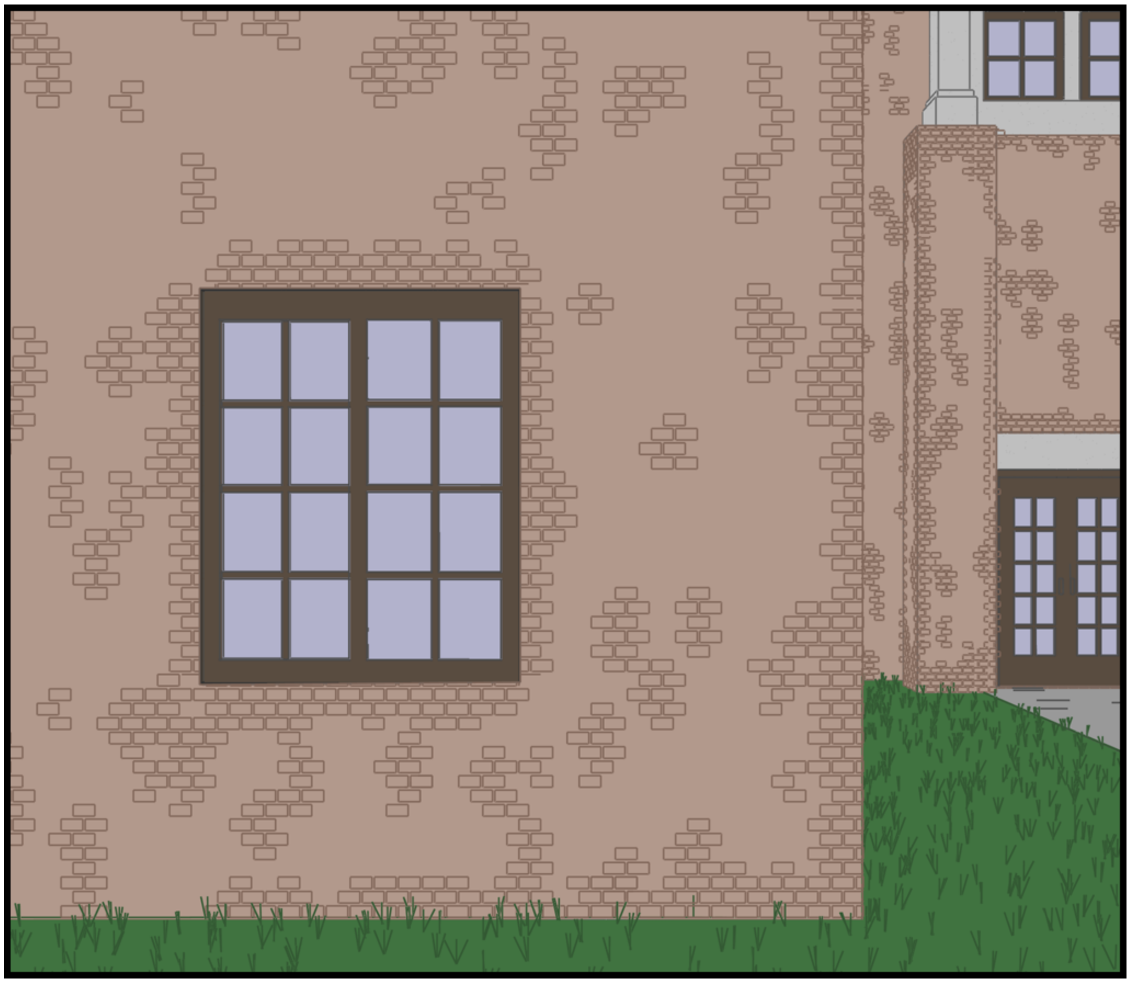

Figure 4.2. Texture is increased around crease and boundary edges to enhance the features of the model.

facing and a back facing polygon. Silhouette edges are view dependent, since the determination of front or back facing must be done with respect to the viewpoint (or viewing direction). Alternatively, creases and boundary edges are view-independent because the angle between two edges will not change unless specified. Because the demarcation of crease and border edges remains static these edges can be found before runtime. This reduces computation time and contributes to the efficiency of the system. Silhouette edges cannot be found at runtime due to their dynamic nature, and thus must be recomputed each time the viewpoint or model position change. The nature of the applications of the system tend toward models with

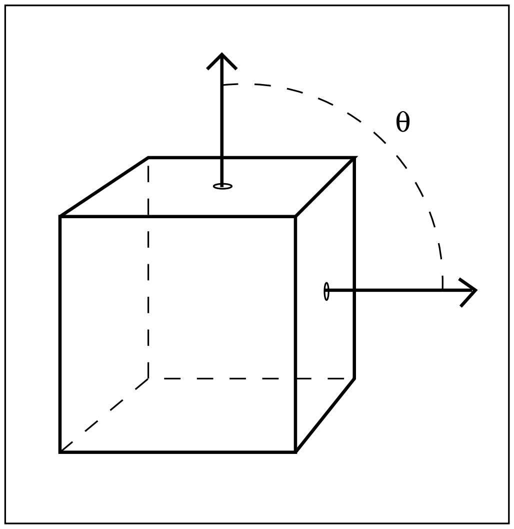

Figure 4.3. Example of a crease edge, the dihedral angle is 90 degrees.

the characteristic that crease edges coincide with edges that would most likely be determined as silhouette edges (reverse: silhouette edges coincide with edges already classified as creases). That is, the edge that is between a front facing and a back facing polygon often also possesses the characteristic of having the dihedral angle between the two polygons greater than the specified theta. Under this assumption, the run-time silhouette edge finding algorithm is most likely to find edges that have already been found as creases, and thus is a repetitive step. Therefore, only crease and boundary edges are found, thus relying on the assumption that these edges will give enough information to the viewer. This assumption may eliminate the use of some models with this system, like a castle with a cylindrical tower; however fast silhouette methods exist and could be applied to remedy this problem [19].

The method for finding crease and border edges is a simple brute force processing of the polygonal model. First, the model is separated into material groups such as grass or brick. This can be done as a polygon tag, but in our implementation we separate the model into different files and process each material section separately. This also enables the single polygon border test to find material borders without modification of the algorithm. There are many approaches to finding crease and

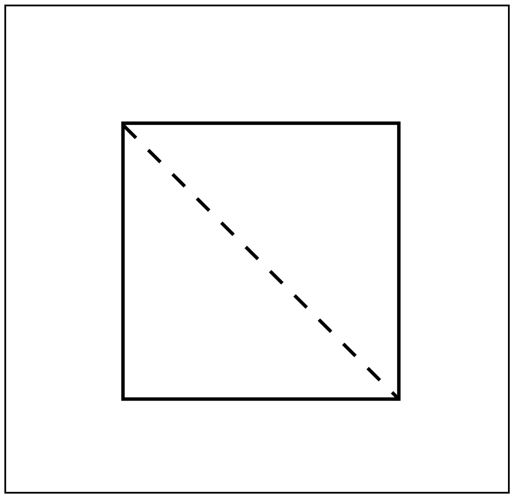

Figure 4.4. An example of a border edge.

boundary edges fast, however; brute force is an appropriate approach here because speed does not matter for the pre-process, and accuracy is of utmost importance.

The crease and boundary edges are placed into an array, then rendered as lines. The entire model is rendered as colored triangles without outlines and the crease and border edges are rendered with a small offset in the direction of the surface normals. Additionally, the list of feature edges is used during the texture placement and clipping steps.

4.2 Texture Placement

Using partial texture follows the idea of indication in which the texture is hinted at rather than fully illustrated. This is intriguing for the viewer because large amounts of line textures would be distracting, and the imagination of the viewer is engaged to fill in the texture where it is omitted. In addition, the number of lines used to suggest texture is reduced, which helps maintain performance. Implementing indication automatically is a difficult problem because it is hard for artists to describe the process of deciding where to place texture. In previous implementations, systems have relied on the artist to input areas of the model that should be enhanced by texture [26]. Looking at the images created by these systems, it seems that the feature edges are common areas to receive more texture. Although feature edges may not be the only such areas to receive texture indication, this method enhances feature edges because they are so often enhanced by artists and also because it has been shown that the enhancement of these edges aids in the understanding of the model. Additionally, material boundaries are enhanced by more texture indication, another phenomenon captured by this implementation. The remaining areas of the model receive sparse texture to reduce clutter, increase efficiency and maintain a clean look [12].

Texturing the interior of the model is done according to a heuristic that thresholds Perlin solid noise [17] to place clusters of texture. An atomic texture element (e.g., a single brick or blade of grass) is placed on the triangle if the function:

$$\frac{1 + 3 \times \ N(kx, ky, kz)}{2},$$

(where N is the Perlin function), is above a threshold. Figure 4.5 shows an example of the Perlin function and the threshold placed on the Perlin function. This function gives a uniformly random distribution of texture. The threshold can be changed to allow more or less texture on any portion of a model. This heuristic gives the texture an irregular distribution without excessive accumulations or concentrations. Figure 4.6a shows the area of a model that is likely to generate texture clusters. Notice that the enhancement of feature edges is paired with the noise function. The resulting placement of atomic texture elements is illustrated in Figure 4.6b, as well as the populating of texture about the atomic texutre elements (Figure 4.6c) and the resulting texture after clipping (Figure 4.6d).

The texture lines are defined in texture space, mapped into the texture space of the model, and then transformed to model space using barycentric coordinates. The texture is defined between 0 and 1 for ease. Each polygon in the model then looks up the threshold noise function, and if the function returns a value corresponding to an area that should be textured, the corresponding texture lines are found, and transformed into model space.

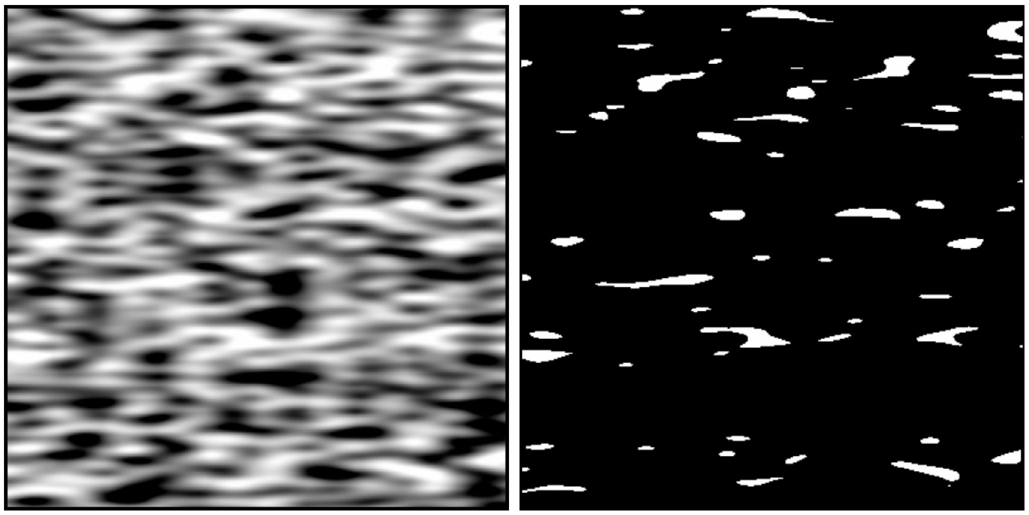

Figure 4.5. The Perlin noise function (left). Threshold placed on the Perlin noise function (right).

4.3 Hand Generated Texture Clusters

Once an atomic texture element is placed, a texture cluster is populated around it (see Figure 4.6c). A texture cluster is a pleasing group of texture elements, such as a group of bricks or a clump of grass. The aesthetic quality of these groupings is critical in getting a good image, since these clusters could easily look mechanical and not hand generated. Grouping texture is another way to indicate texture, rather than drawing texture across the entire model, groups are placed that suggest the texture to the viewer so as not to distract. The clusters must also be visually interesting since they convey most of the detail of the scene. Repetition of the same grouping across the model will result in a repetitive and ugly image, and not be interesting or appealing to a viewer. Randomly placing atomic texture elements across the model will result in a texture that is too sparse and noncoherent. Although automatic generation is possible, we have found that the criterion for what makes a good texture cluster is not obvious.

Human artists express texture and material properties without texturing the entire area, leaving out texture detail where it would be distracting and too cluttered. The method for deciding how to draw texture in such a manner is not well defined.

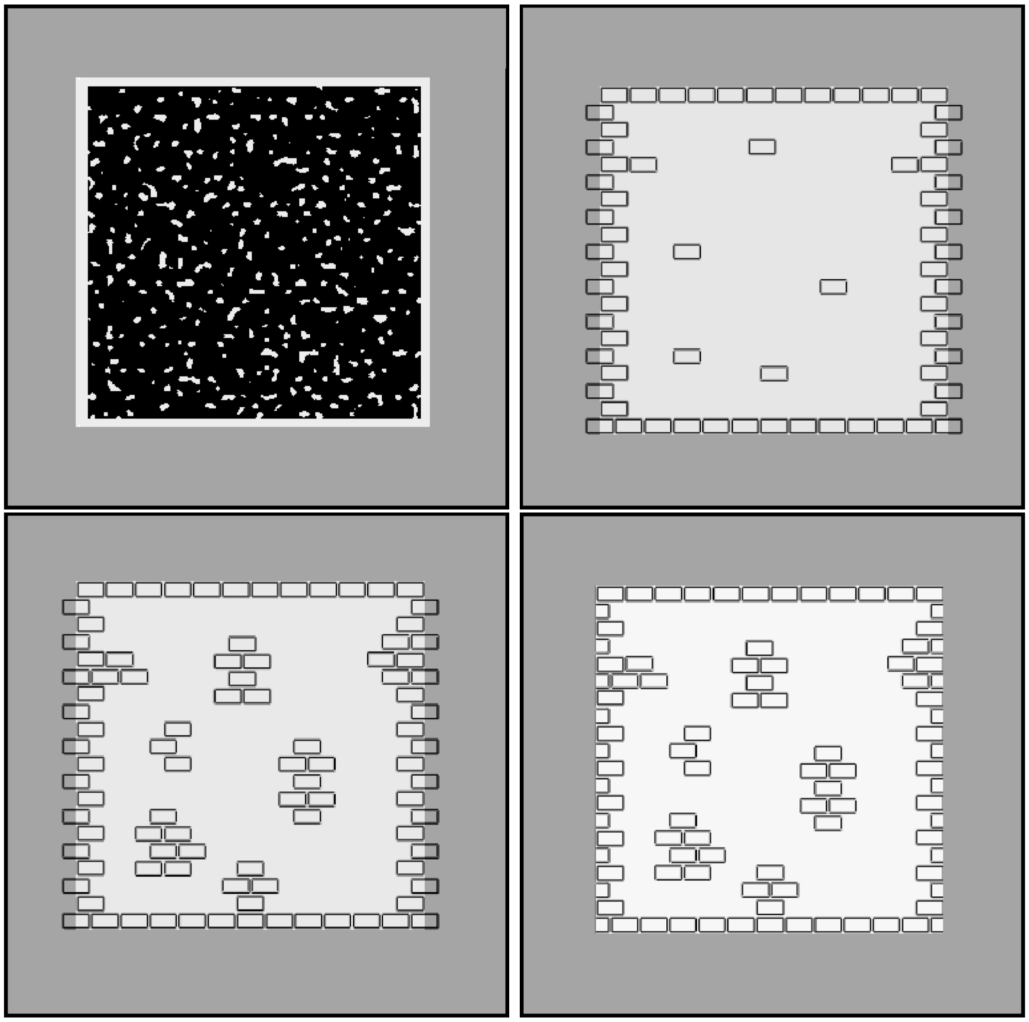

Figure 4.6. a) Threshold placed on Perlin noise. b) The placement of atomic texture elements. c)Texture clusters populated around atomic elements. d) Texture is clipped to crease and boundary edges.

For this reason grouping together single texture elements is done by a human user. This process could easily be automated; however, the result of a purely automatic grouping is too repetitive and mechanical. In addition, adding a touch of human processes is important in maintaining the overall visual appeal of the images. A main drive of this work is to automate much of the process of creating these images; however, removing the artist altogether is not desirable. Thus, artistic input is used in the creation of texture as well as in the final stage of altering the sketchiness of the image.

The texture library consists of a small number of groups of textures defined in texture space. These groups are defined such that the originating texture element is left out, since it was placed during a previous step after the Perlin function was queried. Thus, the library tells the system where to place more atomic texture elements, skipping the Perlin query. Determining which texture library element to use is a simple random choice. The size of the texture cluster does not matter in this choice, since the texture will be clipped after this step. However, the current texture clusters are fairly small, consisting of no more than six texture elements. The size of the texture cluster is directly related to the size of the area to be textured. If a large area needs to be textured, larger texture clusters may be desired; however this is simple to achieve by hand generating larger texture clusters. Alternatively, the threshold value on the Perlin function may be altered to trigger more texture generation. The texture is defined to be tileable, and the texture clusters may overlap, but they will not appear to intersect, which would cause visual artifacts. This approach, however, may cause the repetitive drawing of texture lines, thus reducing the efficiency of the system. Repetitive lines can be removed by processing the line files produced by the texturing stage of the system, checking for coincident endpoint vertices. Currently the texture library consists of bricks, grass, shingles and siding, but the library could be easily extended to incorporate additional materials [27, 16].

4.4 Clipping

Following the placement of the texture lines on the model, all lines are clipped against the creases and boundary edges of the model. Clipping against single triangles often breaks up the texture clusters leading to stray edges. These stray edges distract the eye and may give rise to a confusing interpretation of the type of texture. In addition the texture placement algorithm may not align texture across triangles, since each triangle of the model is processed separately. Properly clipping along all triangle edges will result in the triangulation of the model becoming very apparent. To resolve this problem, the texture is clipped only against feature edges. The feature edges contained in each triangle are flagged during the initial detection process as described in the first section of this chapter, and the texture is then tested against a triangle only if the triangle contains an important edge. This maintains the look of the texture across the entire model, as well as reduces the amount of clipping to be done. The final, clipped model can be seen in Figure 4.6d.

To clip along only the crease and border edges, barycentric coordinates are used. The first clipping check occurs if the generating triangle has an edge flagged as a feature edge. If the texture line intersects the feature edge of the originating triangle, it is immediately clipped. The clipping is done in texture space, and once clipped, the texture space coordinates are transformed to model space by first converting to barycentric coordinates, and then to model space. The next step is to determine if the texture line intersects any other feature edge of the model. This is also done in texture space, however the texture coordinates of the generating triangle, and thus the texture line coordinates may not correspond to the texture coordinates of the triangle to clip against. The solution is to convert the texture line coordinates to the texture space coordinates of the clipping triangle by converting to model space and then back to texture space to clip. The clipping process is kept in texture space because of the ease of clipping in two dimensions rather than three.

Another issue to deal with when clipping texture is when the texture element is clipped and thus becomes disjoint. This is often the case for bricks. The check for intersections may result in two edges of the brick getting clipped, but a non-intersection edge may not get clipped away. So all lines must be qualified to be inside the model, that is, all texture lines must be contained in a triangle that is textured with that material. Also, all lines that make up an atomic texture element must have distinct vertices, so that clipping does not change the shape of the element.

4.5 Sketchiness

Once all of the lines for the partial texture have been placed and clipped, it is possible to adjust the sketchiness of the lines. To achieve “sketchiness” the endpoints of the texture and feature edge lines are randomly perturbed, the extent of which can be modified by the user. A sketchy quality of the lines adds to the hand-drawn look of the imagery, and can be modified independently in different areas of the model, allowing each area to have a unique sketchy quality and maintain the unity of the scene. It is hard to determine the amount of sketchiness desired for the model, so allowing the user to modify the sketch quality parameter is desirable. Examples of varying levels of sketchiness are shown in Figure 4.7.

Using a hardware vertex program, the modification of the original line texture to sketchy lines is done interactively. The goal of our vertex program is to keep the basic structure and rough direction of the lines while adding a slight perturbation to the original vertices. For each vertex in the display list, the vertex program generates a perturbed vertex coordinate by adding a random perturbation vector $\vec{p}$ to the original vertex. The vector $\vec{p}$ is computed by the vertex program from two vectors: $\vec{v_d}$ and $\vec{r}$. The vector $\vec{v_d}$ contains the direction in which the vertex will be offset by while $\vec{r}$ contains the magnitudes of the offset in the three coordinates. These values are decoupled to provide user control over the sketchiness through the use of vertex constants that can be changed interactively, globally affecting the amount that the vertices in the scene are perturbed. Used to perturb the individual vertices, the random values stored in $\vec{r}$; and are generated while the lines are being added to the display list. These values are then stored on a per vertex basis in the vertex registers. The vertex program allows us to *twist* each line slightly out of its original plane while keeping the line’s overall direction the same. Three constants in the vertex program allow us to control the influence of the perturbations on the lines. Example code to perturb texture lines can be found in the appendix.

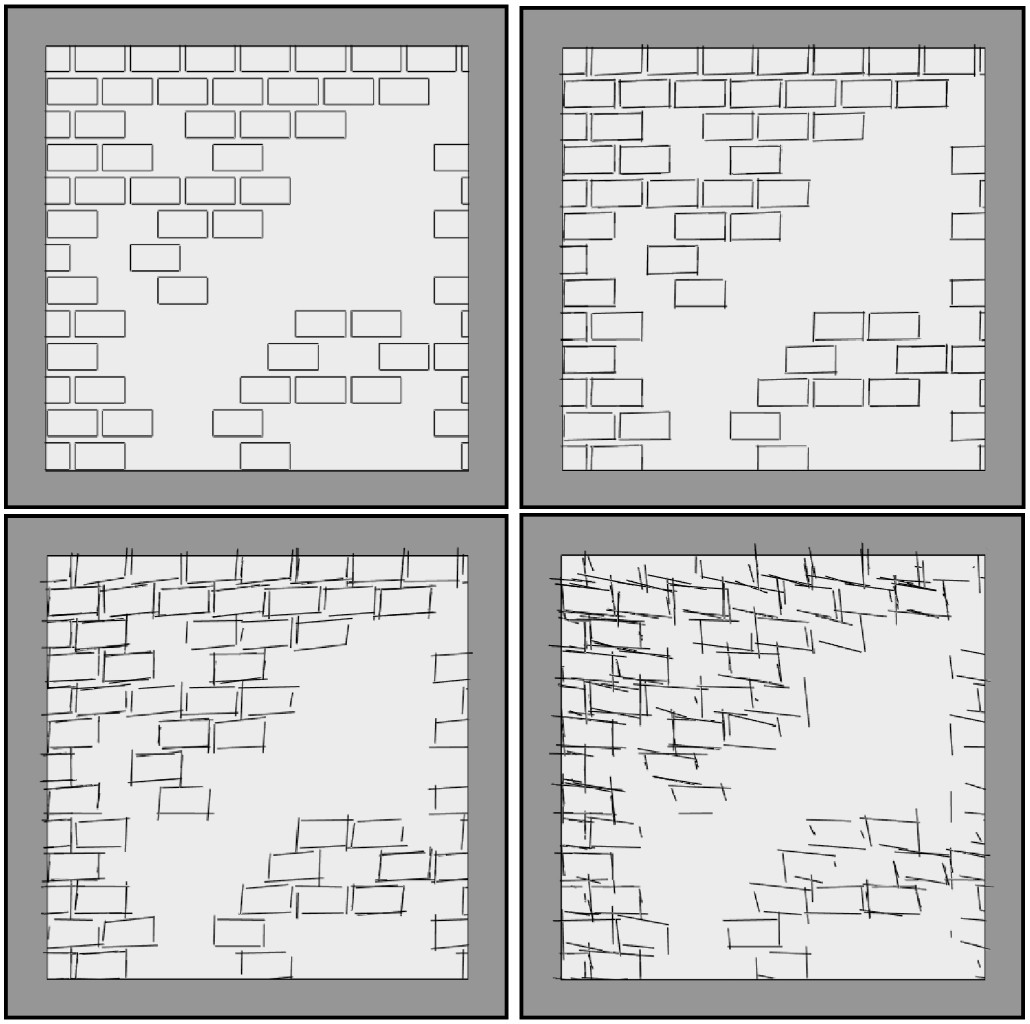

Figure 4.7. Four levels of sketchiness.

4.6 Argument Against Level of Detail

An interesting feature of our texture placement algorithm is maintaining the density of the texture with distance. This approach allows the tone of the image to vary with depth so objects farther away will have a higher texture density and thus a darker tone. The tone of the image can be thought of as the ratio of black ink to white paper. Allowing the tone to vary is a method often adopted by artists to create the illusion of depth in the image. Traditionally computer graphics techniques decimate texture density with distance to maintain tone across an image. Thus the density of texture on an object in the foreground would be at a level constant to the density of texture on a distant object. This is done to preserve the visual appeal of the image because the texture in the background can quickly become too dense, creating a very dark tone that is distracting and unappealing. However, removing texture can be perceptually confusing and lead to misperceived distance, results that conflict with the goals of this system. Thus, the implementation presented herein does not eliminate texture in the distance. A benefit of using partial textures to indicate the texture is that in the distance, the texture density is still lower than if the entire surface was textured. This keeps the tone at an appealing level throughout the scene while preserving the property that the tone is darker in the distance. A possible drawback to this approach is that the size of the environments used is limited, so the distant texture is not so far away as to become overly dense. The assumption that partial textures will maintain appeal at far distances may not hold when the model becomes very large and extend a long way into the distance. The lines used to convey the texture are colored lines which are not as distracting as black lines when grouped tightly, and may not be as unappealing in the background. Artists instead of removing texture in the distance will use a lighter line when drawing texture in the background. Thus, a possible solution for a texture density that is too high in the distance would be to fade with distance the lines that make up the texture.

CHAPTER 5

RESULTS, CONCLUSION AND FUTURE WORK

Line primitives are a reasonable alternative to texture mapping. Depending on the context of the application, 3D lines can give a higher visual quality, more aesthetic appeal, and allow for interactive modification of the image. The prototypical system presented here is designed as a proof of concept that 3D lines can be used as a viable alternative to texture mapping.

5.1 Results

The system runs on a 2 GHz Intel Pentium 4 with an Nvidia GeForce 3 graphics coprocessor. Requiring approximately 3000 lines of code written in C++ using OpenGL libraries, the implementation is fairly straightforward. The only drawing primitives used by the system are 3D constant-colored lines and 3D constant-colored polygons. Table 5.1 gives the polygon and line count as well as frame rates for a 1024x1024 pixel image. Figures 5.1, 5.2 and 5.3 are screen shots of an interactive session of the system using the Olympic Village model.

Table 5.1. Polygon count, line count, and frame rates for interactive walkthroughs using our system at an image resolution of 1024x1024.

Scene	Polygons	Line Count	frames/sec
Mayan Temple	1,259	57,859	48
Olympic Village	20,467	1,648,183	3

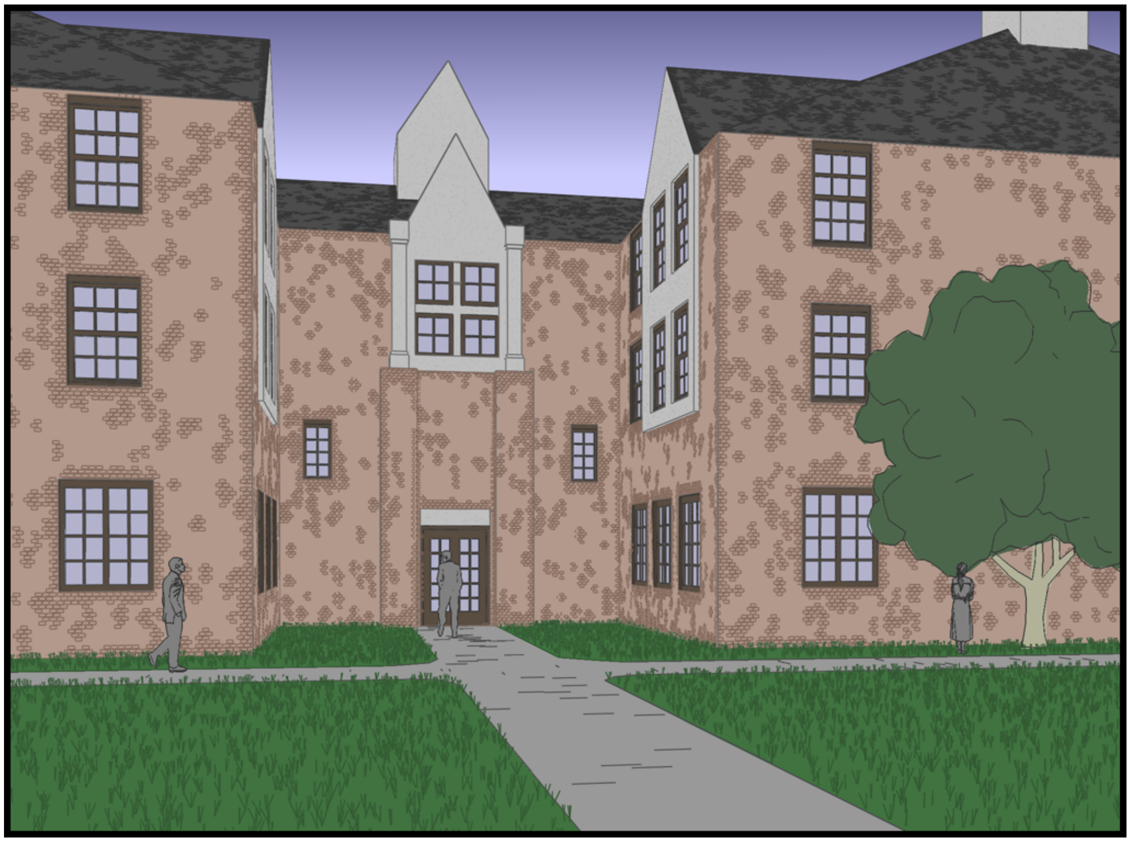

Figure 5.1. Screen shot of an interactive session using the Olympic Village model

5.2 Future Work

Future work for this system includes methods for achieving higher frame rates or using larger models, silhouette finding to allow the use of different types of models, and better entourage to improve the overall visual appeal of the images. The limiting factor on the speed of the system is the number of lines used to indicate texture. Reducing the amount of overall texture, that is, using fewer texture clusters or fewer lines to represent a texture material or removing texture in the distance where the individual texture lines can no longer be distinguished, will improve frame rates and allow larger models to be used. Also, fading texture lines in the distance will reduce any aliasing artifacts that may occur if the model goes into the distance such that the texture becomes distracting. Some applications may wish to use models that have elements with the characteristic of not having a crease

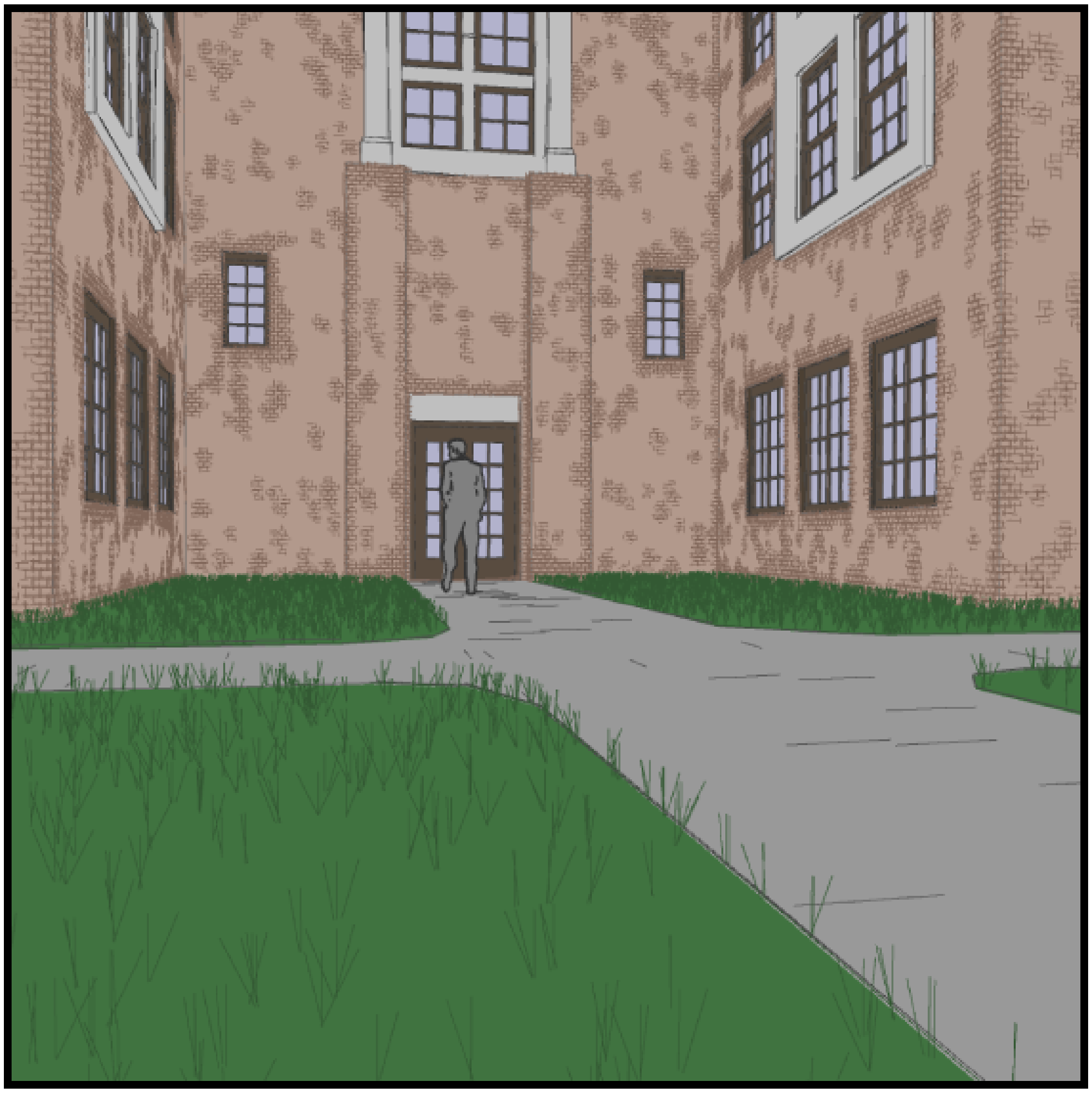

Figure 5.2. Screen shot of an interactive session using the Olympic Village model

edge, and a fast silhouette finding algorithm must be used, or an improved texture placement algorithm that places texture in such a way that reveals the structure of the model without needing to find the silhouettes. Finally, for walkthroughs, or for more visual appeal, an improvement to the entourage so that the people move in unobtrusive ways, or the entourage is in three dimensions would enhance the experience of the viewer.

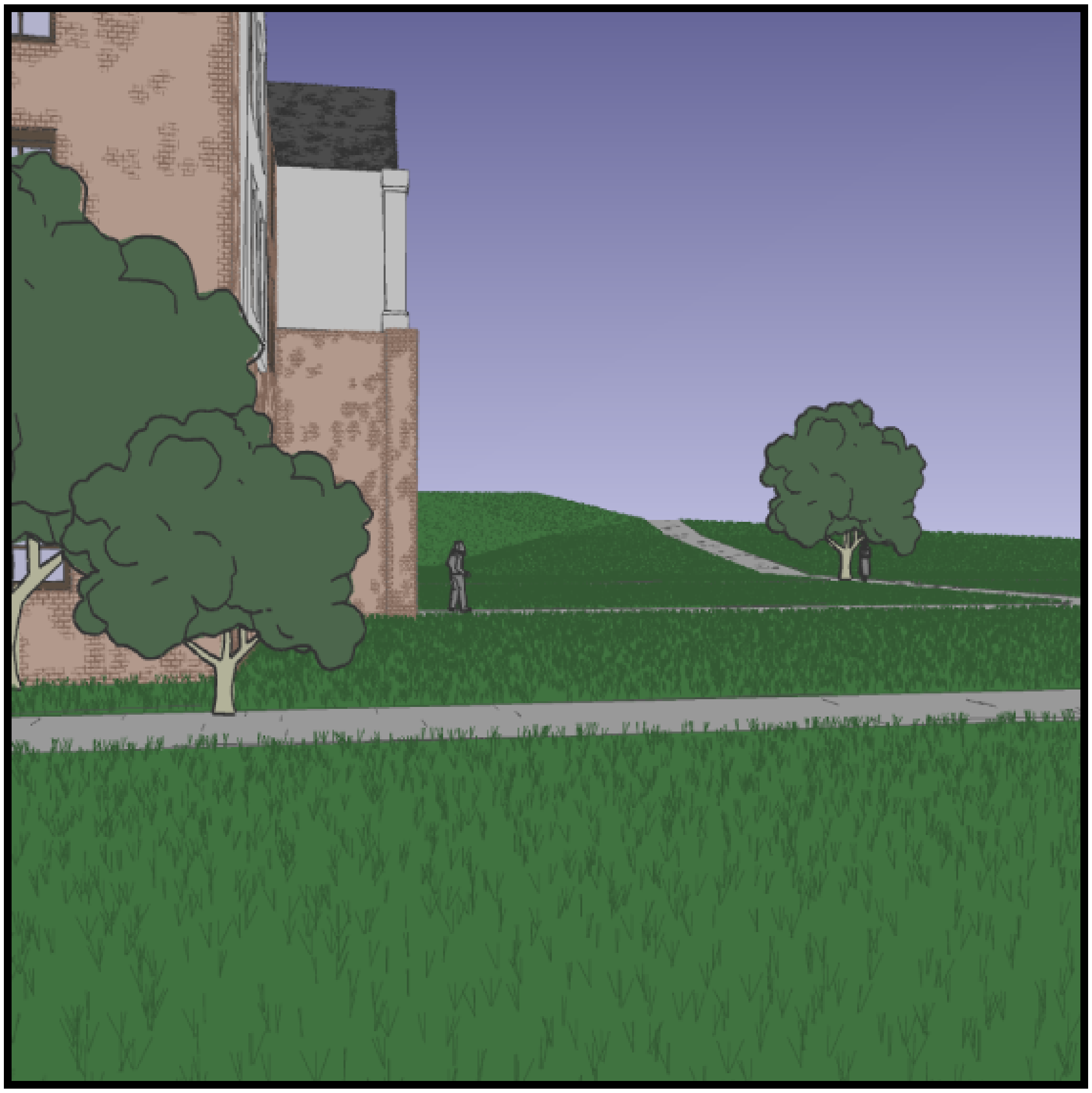

Figure 5.3. Screen shot of an interactive session using the Olympic Village model

5.3 Conclusion

Imagery style is an important issue to consider when rendering. The effect a rendering style has on the user can be advantageous when trying to express complex ideas or encourage specific reactions. Incorporating multiple rendering styles relates information in a natural way and assists in the communication between image creator and image viewer. Also, mimicking hand drawn illustrations keeps the effect of the human user in the renderings, aiding in the aesthetic appeal.

REFERENCES

[1] Aliaga, D. G., and Lastra, A. A. Architectural walkthroughs using portal textures. In *IEEE Visualization '97 (VIS '97)* (1997), IEEE, pp. 355–362.

[2] Deussen, O., and Strothotte, T. Computer-generated pen-and-ink illustration of trees. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Annual Conference Series, ACM Press, pp. 13–18.

[3] Dooley, D., and Cohen, M. F. Automatic illustration of 3d geometric models: Lines. In *Proceedings of the Symposium on Interactive 3D Graphics* (1990), ACM Press, pp. 77–82.

[4] Doyle, M. E. *Color Drawing.* Van Nostrand Reinhold Company, 1981.

[5] Freudenberg, B., Masuch, M., and Strothotte, T. Walk-through illustrations: Frame-coherent pen-and-ink style in a game engine. In *In Proceedings of Eurographics 2001* (2001), Computer Graphics Forum, pp. 184–191.

[6] Funkhouser, T. A. Database management for interactive display of large architectural models. In *Graphics Interface* (1996), pp. 1–8.

[7] Funkhouser, T. A., Sequin, C. H., and Teller, S. J. Management of large amounts of data in interactive building walkthroughs. In *Proceedings of the Symposium on Interactive 3D Graphics* (1992), ACM Press, pp. 11–20.

[8] Gooch, B., and Gooch, A. *Non-Photorealistic Rendering.* A K Peters, 2001.

[9] Guptill, A. L. *Rendering In Pen and Ink.* Waston-Guptill Publications, 1976.

[10] Klein, A. W., Li, W. W., Kazhdan, M. M., Correa, W. T., Finkelstein, A., and Funkhouser, T. A. Non-photorealistic virtual environments. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Annual Conference Series, ACM Press, pp. 527–534.

[11] Lake, A., Marshall, C., Harris, M., and Blackstein, M. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of the First International Symposium on Nonphotorealistic Animation and Rendering* (2000), ACM Press, pp. 13–20.

[12] Lin, M. W. *Drawing and Designing With Confidence : A Step-By-Step Guide.* John Wiley and Sons, 1997.

[13] Masuch, M., Freudenberg, B., Ludowici, B., Kreiker, S., and Strothotte, T. Virtual reconstruction of medieval architecture. In *In Proceedings of EUROGRAPHICS 1999, Short Papers* (1999), Computer Graphics Forum, pp. 87–90.

[14] Masuch, M., Schumann, L., and Schechtweg, S. Animating frame-to-frame coherent line drawings for illustrative purposes. In *Simulation und Animation '98* (1998), pp. 101–112.

[15] Masuch, M., and Strothotte, T. Visualizing ancient architecture using animated line drawings. In *Conference on Information Visualization (IV '98)* (1998), IEEE, pp. 261–266.

[16] Miyata, K. A method of generating stone wall patterns. In *Computer Graphics (SIGGRAPH '90 Proceedings)* (1990), vol. 24(4), pp. 387–394.

[17] Perlin, K. An image synthesizer. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (1985), vol. 19(3), pp. 287–296.

[18] Popescu, V., Lastra, A., Aliaga, D., and de Oliveira Neto, M. Efficient warping for architectural walkthroughs using layered depth images. In *IEEE Visualization '98 (VIS '98)* (1998), IEEE, pp. 211–216.

[19] Raskar, R. Hardware support for non-photorealistic rendering. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware (HWWS)* (2001), pp. 41–46.

[20] Richens, P. The piranesi system for interactive rendering. In *Proceedings of the Eighth International Conference on Computer Aided Architectural Design Futures* (1999), CAAD Futures, pp. 381–398.

[21] Saito, T., and Takahashi, T. Comprehensible rendering of 3-d shapes. In *Computer Graphics (SIGGRAPH '90 Proceedings)* (1990), vol. 24(4), pp. 197–206.

[22] Schumann, J., Strothotte, T., Raab, A., and Laser, S. Assessing the effect of non-photorealistic rendered images in cad. In *Proceedings of the Conference on Human Factors in Computing Systems : Common Ground* (1996), PAPERS: Empirical Studies of Graphics and Visual Design, ACM Press, pp. 35–41.

[23] Strothotte, T., Masuch, M., and Isenberg, T. Visualizing knowledge about virtual reconstructions of ancient architecture. In *Proceedings of the Conference on Computer Graphics International 1999* (1999), IEEE Computer Society, pp. 36–43.

[24] Strothotte, T., Preimand, B., Raab, A., Schumann, J., and Forsey, D. R. How to render frames and influence people. In *Computer Graphics Forum* (1994), vol. 13(3), pp. 455–466.

[25] Strothotte, T., Puhle, M., Masuch, M., Freudenberg, B., Kreiker, S., and Ludowici, B. Visualizing uncertainty in virtual reconstructions. In *In Proceedings of Electronic Imaging and the Visual Arts* (1999), p. 16.

[26] Winkenbach, G., and Salesin, D. H. Computer–generated pen–and–ink illustration. In *Siggraph 1994, Computer Graphics Proceedings* (1994), Annual Conference Series, ACM Press, pp. 91–100.

[27] Yessios, C. I. Computer drafting of stones, wood, plant and ground materials. In *Computer Graphics (SIGGRAPH '79 Proceedings)* (1979), vol. 13(3), pp. 190–198.

[28] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. Sketch: an interface for sketching 3d scenes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996), ACM Press, pp. 163–170.