Shared-Frustum Stereo Rendering

by

Michael Vincent Capps

B.S., Mathematical Sciences, University of North Carolina (1994)
M.S., Computer Science, University of North Carolina (1996)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2000

© Michael Vincent Capps, MM. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute
publicly paper and electronic copies of this thesis document in whole or in
part.

Author ......................

Department of Electrical Engineering and Computer Science
January 14, 2000

Certified by ......................

Nathaniel I. Durlach
Senior Research Scientist
Thesis Supervisor

Accepted by .............

Arthur C. Smith
Chairman, Department Committee on Graduate Students
Shared-Frustum Stereo Rendering
by
Michael Vincent Capps

Submitted to the Department of Electrical Engineering and Computer Science
on January 14, 2000, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

This thesis involves two approaches for accelerating stereo image generation: Joint Frustum Culling (JFC) and Lazy Shared Rendering (LSR). Both methods aim to reduce the amount of work required to redraw an image from an additional viewpoint by exploiting spatial coherence between the two viewpoints.

In Joint Frustum Culling, the objects in a scene are culled to a viewing volume that is the union of the left and right eye viewing volumes. As the two volumes potentially have significant overlap, this can substantially reduce the amount of culling needed to render each eyepoint.

For Lazy Shared Rendering, a portion of the scene is purposefully rendered without stereo. Image generation time is reduced by only drawing this portion once; this portion is shared by both eyepoints. Faraway objects do not change their appearance or position appreciably between nearby viewpoints, such as two closely placed eyes. So, by partitioning the viewing volume into near and far sections, objects in the far section need only be rendered once. The near section is rendered separately for each eye, and overwrites the far section in the frame-buffer where occlusion occurs.

This thesis includes an evaluation of the performance gains from these techniques. The application of LSR assumes that these computational savings outweigh the cost of reduced stereo fidelity. A series of informal user experiments were performed, which appear consistent with that hypothesis. A formal user study, which would explore this quality-speed trade-off space in depth, is described for follow-on work.

Thesis Supervisor: Nathaniel I. Durlach
Title: Senior Research Scientist
Acknowledgments

Nat Durlach, my thesis advisor, went beyond the call of duty by offering to step in and help with my thesis. His careful consideration and kindness have been a model for me in advising my own students. Rudy Darken at NPS was a tremendous asset to this process, and goes unrecognized as an advisor only because of the inconvenience of his affiliation.

Thanks also to Arthur Smith and Marilyn Pierce for their advice and attention during a difficult time. Justin Legakis was patient and helpful in getting a final version from my remote location into the right hands. Many of the concepts brought to fruition in this work originated in a team project with Douglas De Couto of MIT.

Previous thesis research, on Awareness Management in MERL’s Java OpenCommunity project, remains unpublished—but those who aided that effort should go unacknowledged. Many thanks to Barry Perlman and David Anderson for a stimulating research environment. Thanks also to Dick Waters and James Foley for their support, as well as the whole SPLINE/Schmoozer team.

My Ph.D. advisor and mentor, Michael Zyda, will receive his due in a much larger document to follow. However, it is appropriate to mention his exceptional support of this work in the face of many other pressing projects.

Finally, my most grateful appreciation to my fiancee, and now wife, Laura. She encouraged me to complete this work upon departing MIT, and her energy and devotion seems undiminished by the knowledge of years more research to come...
Contents

1 Overview 9
  1.1 Issues in Stereo Rendering 9
  1.2 Contribution of this Thesis 10
  1.3 Organization 10

2 Stereo Rendering 12
  2.1 Monoscopic Rendering 12
    2.1.1 The Viewing Volume 12
    2.1.2 Perspective Projection 14
  2.2 Stereopsis 14
    2.2.1 Stereo Viewing Frusta 15
    2.2.2 Generating Stereo Pairs 17
    2.2.3 Stereo Pair Presentation 17

3 Shared Culling 20
  3.1 Frustum Culling 20
    3.1.1 Shared Frustum Culling 21
    3.1.2 Improvements 22
  3.2 Back-Face Culling 22
    3.2.1 Stereo Back-Face Culling 24
  3.3 Cohen-Sutherland Clipping 24
  3.4 Liang-Barsky Line Clipping 25
4 Shared Rendering

4.1 Lazy Shared Rendering (LSR) .............................................. 27
  4.1.1 Generating Front and Rear Frusta .................................. 28
  4.1.2 Evaluation of LSR .................................................. 29
  4.1.3 Known Shortcomings ................................................. 29

4.2 Previous Work ............................................................... 30
  4.2.1 Pixel Shifting ......................................................... 30
  4.2.2 Hi-Lo Stereo Fusion .................................................. 31
  4.2.3 Active Edge List Sorting ............................................ 32
  4.2.4 Accelerated Ray Tracing .............................................. 32

5 Implementation ............................................................... 33

5.1 Software ................................................................. 33
  5.1.1 Graphics Libraries ................................................... 33
  5.1.2 Software Features .................................................... 34

5.2 Display Platform ......................................................... 35
  5.2.1 Time-Multiplexed Stereo: StereoEyes Glasses .................. 35
  5.2.2 Time-Parallel Stereo: Virtual I/O Glasses ....................... 36

5.3 Implementation ........................................................... 37
  5.3.1 Software Triangle Culling ........................................... 37
  5.3.2 Joint Frustum Culling ............................................... 38
  5.3.3 LSR Shared Rendering ............................................... 38

6 Performance Comparisons .................................................. 41

6.1 Test Models and Paths .................................................. 41

6.2 Timing Results ............................................................ 42

7 User Experiments ............................................................ 45

7.1 $\beta$ Investigation ....................................................... 45

7.2 Time-Parallel vs. Time-Multiplex ...................................... 46

7.3 Motion Effects ............................................................. 47
8 Conclusions and Future Work

8.1 Suggestions for Future Work
List of Figures

2-1 Monoscopic viewing frustum ........................................... 13
2-2 Perspective Projection .................................................. 14
2-3 Stereoscopic View Frusta: Convergent View ......................... 15
2-4 Stereoscopic View Frusta: Parallel View .............................. 16

3-1 Overlapping frusta facilitate shared computation. ................. 22
3-2 Polyhedron partially consisting of back-facing polygons. ....... 23
3-3 Cohen-Sutherland outcode culling. .................................... 25

4-1 Shared rendering areas. .................................................... 28
4-2 Foreshortened eye frustums. .......................................... 28
4-3 Pixel shifting in a single scanline. .................................. 30

5-1 Virtual I/O interleaved scan-lines. .................................... 36
5-2 Shared culling implementation. ....................................... 39
5-3 Culling for shared rendering. ......................................... 39

6-1 Average rendering time vs. $\beta$. .................................. 43
6-2 Average rendering times, with $\beta = .4$ for combined rendering. 44
List of Tables

6.1 Average frame rendering times (milliseconds) 43
Chapter 1

Overview

1.1 Issues in Stereo Rendering

The pathway to the human brain with the highest information bandwidth is the eye. Computer graphics has experienced such general success because it offers the opportunity to present information visually, thereby tapping that high-capacity connection. Yet in nearly all cases, computer displays present information in only two dimensions—essentially requiring only a single eye to view.

Stereoscopic displays exploit the full potential of the human visual system by presenting images with depth information. Hardware for the presentation of stereo images was first developed in 1968 [23]; commodity hardware is now available for corporate [1], academic [11] and even home [16, 13, 12] use. Stereoscopic display techniques have proven especially useful for scientific visualization, but have a place in computer gaming and location-based entertainment as well.

In a standard stereo rendering implementation, the left and right eyes’ viewing volumes are created and rendered independently. This method essentially doubles the amount of work for rendering each frame over that in a monocular view. In computer graphics, the graphical subsystem is usually the display bottleneck, so doubling the complexity of the rendering task halves the frame rate. This significantly reduces the set of models that can be viewed in stereo with useful fidelity. More about the process of generating stereoscopic images is found in chapter 2.
1.2 Contribution of this Thesis

This thesis involves two approaches for accelerating stereo image generation: Joint Frustum Culling (JFC) and Lazy Shared Rendering (LSR). Both methods aim to reduce the amount of work required to redraw an image from an additional viewpoint by exploiting spatial coherence between the two viewpoints.

In Joint Frustum Culling, the objects in a scene are culled to a viewing volume that is the union of the left and right eye viewing volumes. As the two volumes potentially have significant overlap, this can substantially reduce the amount of culling needed to render each eyepoint. It is therefore only necessary to cull a much smaller additional volume for each of the left and right views after culling to the combined view.

Several techniques exist for exploiting spatial coherence in stereo rendering, and these are discussed in detail in Chapter 4. This work takes a novel approach; rather than re-projecting a single image, a portion of the scene is purposefully rendered without stereo. Image generation time is reduced by only drawing this portion once; this portion is shared by both eyepoints. Faraway objects do not change their appearance or position appreciably between nearby viewpoints, such as two closely placed eyes. So, by partitioning the viewing volume into near and far sections, objects in the far section need only be rendered once—hence the term “Lazy Shared Rendering.” The near section is rendered separately for each eye, and overwrites the far section in the frame-buffer where occlusion occurs.

1.3 Organization

The remainder of this thesis is organized as follows: Chapter 2 discusses standard stereo rendering methodology. Chapter 3 discusses the new shared culling method, as well as previous work in this area. Chapter 4 gives a similar treatment for a new technique for accelerating stereo rendering. This is followed by the details of the software implementation and hardware configuration, in Chapter 5.

Chapter 5 gives actual performance improvements, and overhead calculations. The following chapter discusses a series of preliminary user experiments, in which were presented
with varying mixes of 2- and 3-dimensional imagery. The results of those experiments will hopefully lead to more formal studies, which can be used to explore the performance gains safely attributable to the LSR technique.

This work is a stand-alone contribution to the field of stereo rendering. As with many research projects, however, this investigation led to as many questions as answers. Chapter 8 contains an analysis of the final state of this work, as well as a number of promising avenues for follow-on research.
Chapter 2

Stereo Rendering

The generation of stereoscopic images involves rendering an identical scene from two slightly different viewpoints. This chapter reviews the perspective projection for a monoscopic image, as it is equally necessary for stereo images. This is followed by a general discussion of stereoscopy, and the various means by which stereo pairs are generated and displayed.

2.1 Monoscopic Rendering

This section offers a very brief review of rendering concepts, in order to establish terminology for later stereo rendering sections. Additional detail on these topics is available in computer graphics textbooks such as [6] and [8].

2.1.1 The Viewing Volume

Central to the concept of graphically rendering three-dimensional data-sets is the viewing volume. Figure 2-1 shows the viewing volume. This is a six-sided polyhedron, essentially a pyramid missing its peak, known as a frustum. The frustum can be defined in a variety of ways; in this thesis, a frustum consists of the following:

- **Center of Projection (COP)** defines the peak of the viewing volume, and is usually thought of as the eyepoint.
The near and far distances define the range in which objects are visible. Objects closer than the near plane, and farther than the far plane, are not drawn.

Field of View (FOV) is the angular width and height of the frustum. Figure 2-1 shows only the horizontal field of view (hFOV) angle. Vertical field of view (vFOV) is usually defined by a constant display aspect ratio, hFOV:vFOV.

Up-vector ($\mathbf{UPV}$) defines the vertical direction from the COP.

Viewing Vector ($\mathbf{V}$) is the direction of gaze from the COP; $\mathbf{V}$ connects the COP to the centers of the near and far plane.

The View Window is included for illustrative purposes, though it is not technically part of the viewing frustum. This represents the position of the viewing plane, such as a monitor screen, inside the virtual world.

Given the definition of the view frustum, the first step for rendering is to determine which objects intersect the frustum. Objects which are entirely outside the frustum are rejected to avoid unnecessary additional computation; this process is known as culling. Those objects which are neither entirely inside nor entirely outside the viewing frustum are reduced to only their visible portions; this is the clipping task.
2.1.2 Perspective Projection

To maintain three-dimensional realism when displaying with a two-dimensional device, it is vital to consider viewing differences stemming from perspective and distance. The perspective projection is used to project distant points onto a plane perpendicular to the viewing vector. Similarly to a real-world view, any set of lines parallel in object space appear to converge at a distance. Lines parallel in object space with each other and the projection plane remain parallel in the display.

Figure 2-2 shows the effect of a perspective projection on a single point. Point $P_0$ projects to $P_s$. To solve for $P_s$, we assume that the COP is at the origin, and the projection plane is at distance $h$ in the $V$ direction, where $U\bar{P}V = +Y$ and $V = +Z$. Then the equations to find $P_s$ are:

$$P_s = (x_s, y_s) \quad x_s = \frac{hx_0}{z_0} \quad y_s = \frac{hy_0}{y_0}$$

2.2 Stereopsis

A stereo image requires a different viewing frustum for each eye, to generate appropriate binocular disparity. The COP for each eye is slightly different, to correspond with the
morphology of the human head. The space between the eyes is known as the interocular distance $d$, and is generally about 2.5 inches [7]. Just as the eyes can determine depth by fusing images from two visual sensors, slightly-separated stereo frusta generate different images which give a stereoscopic effect.

Varying the parameters of the stereo frusta affects how well the viewer can blend together the two images. Generally, to fuse a stereo pair, the eyes accommodate to a scene by adjusting lens focal length. The eyes can also both focus on a single distant point, known as convergence. By altering the interocular distance $d$, the viewer is granted super-human visual ability to perceive minimal, or extreme, depth changes.

### 2.2.1 Stereo Viewing Frusta

Many methods are available for selecting the stereo frusta parameters [3, 18]. We examine two in particular, convergent view and parallel view, shown in Figures 2-3 and 2-4.

The converging stereo frusta have two viewpoints, which are $\frac{d}{2}$ to the left and right of the original COP, along the vector $\vec{V} \times U\vec{P}V$. The left and right view directions converge at
Figure 2-4: Stereoscopic View Frusta: Parallel View
a distant point $C$ on $\vec{V}$; they meet at an angle of size $\theta$. These view directions are calculated as the vector $\vec{V}$ rotated about $\vec{U} \vec{P} \vec{V}$ by $-\frac{\theta}{2}$ and $\frac{\theta}{2}$ respectively.

The parallel view frusta have similarly placed viewpoints. They differ from the convergent frusta in that they share the original viewing vector $\vec{V}$, essentially converging to a point at infinite distance. The two methods of deriving stereo viewing volumes are equally effective, and the computations are similar.

In the remainder of this paper, the parallel view frusta are used exclusively, for three reasons. Foremost, this method yields left and right viewing volumes with a shared far-plane, which makes for a more efficient union volume. This improves the results from the shared-culling techniques explained later in chapter 3. Second, as demonstrated in [9], the convergent frusta can cause vertical parallax errors. Briefly, due to the $\theta$ rotation, a polygon perpendicular to $\vec{V}$ can have different nearest edges in each view. With the perspective transformation, these different nearer edges appear larger, and the eyes are unable to fuse the two views. And finally, the point of convergence is rarely varied in real-time graphical applications, since it is generally impossible to know the user's instantaneous depth of focus. Instead, most applications choose a fix point of convergence, requiring the user to adapt their focus accordingly.

### 2.2.2 Generating Stereo Pairs

The standard method for rendering a pair of stereo images is to create the left and right images separately. That is, similar to monoscopic rendering, the frustum is set and the model is sent through the graphics pipeline. This is performed once for each eye, so each object is rendered twice, and exactly twice the work is done. Later chapters describe methods for exploiting the spatial coherence between the stereo viewpoints.

### 2.2.3 Stereo Pair Presentation

The last stage in the graphics pipeline is display. Once the images have been generated, we must present them to the user such that each eye is given the appropriate information. A primary decision is that between time-multiplexed versus time-parallel display.
A time-multiplexed display alternates between the left and right image, and includes appropriate hardware to ensure the signals reach only the correct eye. The moving-slit method, for instance, employs a viewing gap which is moved in conjunction with image display. The Stereoptiplexer [5] and Parallactiscope [24] are examples of moving-slit technology. Oscillating mirrors can be used, timed with image display to direct one view while blocking another [25]. The most commercially-successful method, at least in recent years, has been liquid-crystal shutter systems [16]. A signal, either by wire or infrared transmission, is sent to a set of glasses with each image change. That signal causes the crystal in one lens of the glasses to become opaque and the other transparent. Crystal shutter glasses are lightweight, and most importantly, can work with any standard display device.

Time-multiplexed displays generally have one serious drawback: they halve the frame rate of the display device. Time-parallel displays instead draw both right and left eye images simultaneously. The obvious method for this is presenting the images separately, such as via dual parallel displays. Head-mounted displays take this approach [23]; by delivering small images close to the eye, the illusion of a large field of view is given. Additionally, there is no possibility of interocular crosstalk (also called ghosting) which plagues many time-parallel display solutions.

Delivering stereo by presenting a single display to both eyes is possible, though more complex. Chromostereopsis, the tendency of red objects to appear nearer than blue objects, can occur due to the refraction of varying light wavelengths in the vitreous fluid of the eye. The effect is weak, but can give the impression of depth with no additional hardware [10]. A more traditional use of color has been seen in so called 3-D movies [15]. The viewer is given glasses with red and blue lenses, and the left and right eye images are presented simultaneously in the opposite color scheme. The red lens blocks all red light, and similarly with the blue lens, so in theory only the desired images pass through. Interocular crosstalk can be significant, though, since the images use more hues than red and blue. A more successful single-display time-parallel method uses the polarization property of light [11]. The left and right image are presented simultaneously, polarized at a right angle, and polarized glasses are worn to transmit only the proper signal. Ghosting is still an issue, though much less so, and there is no color limitation.
Chapter 5 describes in greater detail the primary display hardware used for this thesis. Each eye is presented its image via a dedicated CRT, a time-parallel no-interocular-crosstalk method.
Chapter 3

Shared Culling

In this chapter, we describe a number of approaches to geometric culling which reduce culling computations for stereo rendering. The first, shared frustum culling, is a novel algorithm introduced in this chapter. Though the technique is straightforward, it merits discussion; implementation choices made for this algorithm affected design of other system components. Implementation details are discussed in chapter 5.

The remaining sections of this chapter discuss previous work in shared culling. Each technique is explained both in its traditional monoscopic domain and from the stereo perspective.

3.1 Frustum Culling

Geometric culling is a procedure for partitioning geometric primitives. Here, we refer to its standard use—to distinguish whether geometric primitives are inside or outside of a viewing or picking frustum.

A standard implementation checks each geometric primitive in the scene against the viewing frustum. Any unculled polygons are then sent down the graphics pipeline to be drawn. In the case of stereo viewing, this procedure is repeated once for each eye—essentially testing each polygon twice before drawing.

Culling to the viewing frustum requires performing an intersection between an irregular six-sided polygon and the geometric primitive, and can be expensive. The culling procedure
is often approximated; for example, visibility is often tested against bounding boxes rather than exact primitives. Culling is performed only to reduce the complexity of later steps in the graphics pipeline. Therefore, any conservative approximation which guarantees to not cull visible polygons is acceptable.

3.1.1 Shared Frustum Culling

The naïve approach to culling for stereo applications is to repeat the frustum culling process independently for each eye. We aim to accelerate stereo rendering by exploiting the coherence between the left- and right- eye frusta. Figure 3-1 shows that there can be considerable overlap between the spaces encompassed by the two view frusta. Let frustum $C$ be a combination of the left and right frusta $L$ and $R$, namely the smallest frustum that encompasses the same space as $L$ and $R$ that has view direction $\vec{v}$ and up vector $\vec{u}$.

This yields four distinct areas we can describe:

1. where $L$ and $R$ overlap
2. where neither $L$ nor $R$ occupy space
3. in $L$ but not $R$
4. in $R$ but not in $L$

In a precomputation step, the full scene is culled to the volume $C$. Then, only the polygons in $C$ must be considered when determining the sets $L$ and $R$ for rendering the two eye views.

When we cull to $C$, we have all the polygons in areas 1—4. Then, to cull to $L$, we need only remove the polygons in areas 2 and 4; generating $R$ is similar. Depending on the values of the interpupillary distance $d$ and focal angle $\theta$, areas 2, 3, and 4 can be extremely small in relation to the sizes of $L$ and $R$.

To evaluate this technique, we assume that polygons are distributed uniformly throughout the scene. Let $N$ be the number of polygons in the scene, and $N_c$ the number polygons encompassed by $C$, where $N \gg N_c$, and $N_c \approx N_l \approx N_r$ (i.e. the stereo geometry is favorable, like that in Figure 2). Then, without shared culling, we have to cull $2(N - N_c) \approx 2N$
polygons. With shared culling, we only have to cull \( N + 2N_c \approx N \) polygons. Depending on how culling is implemented, this can halve the amount of work needed to cull for each stereo frame.

3.1.2 Improvements

Of course, if we have a sublinear algorithm for culling to a frustum, the savings in work becomes less striking. For example, if a spatial data structure is used, it could be possible to traverse the structure in such a way that only the polygons contained in the desired frustum are encountered. This would be the ideal case, as we would then only have to process \( 2N_c \) polygons in our culling. Our geometric primitives, as described in chapter 5, are loaded as unorganized “triangle soup” which forces a linear culling approach.

3.2 Back-Face Culling

Back-Face detection is the simplest method to avoid drawing polygons that cannot be seen, and is usually performed early in the graphics pipeline. Generally, half of the polygons in
Given a normal for each face, back-face culling is the process of removing any polygons that face away from the viewing direction. Most graphics systems consider polygons to be one-sided; this simplification is generally correct given opaque convex polyhedra. Figure 3-2 illustrates this process; the dotted faces of the polygon are culled.

Given a viewing direction $V$ and plane normal $N$, it is only necessary to determine if these vectors are pointing in the same direction. This can be computed with a vector dot product: visible faces solve the equation $V \cdot N > 0$. If the dot product is equal to zero, the polygon is dropped since it could only appear as an infinitely thin line to the viewer. If the product is less than zero, the polygon is facing away and is culled.

Alternatively, we can determine the equation of the polygon in the form $Ax + By + Cz + D = 0$ such that $N = (A, B, C)$. Then back-faces are identified whenever $A \cdot COP_x + B \cdot COP_y + C \cdot COP_z + D \leq 0$, where $(COP_x, COP_y, COP_z)$ is the center of projection.
3.2.1 Stereo Back-Face Culling

The back-facing property for any polygon will only be different between the eyes if the polygon's orientation is nearly perpendicular to the viewing vector. Assuming the variation between eye position is solely in the $x$ direction, it is possible to exploit spatial coherence to reduce computation [4].

If the normal of a face points generally in the positive $x$ direction ($A > 0$), then we first compute whether the polygon is back-facing for the left eye with $A \cdot (COP_x - \frac{d}{2}) + B \cdot COP_y + C \cdot COP_z + D < 0$. If so, the polygon must be back-facing for both eyes. If not, we compute $A \cdot (COP_x + \frac{d}{2}) + B \cdot COP_y + C \cdot COP_z + D < 0$, the back-face equation for the right eye. Assuming that we store the value of $B \cdot COP_y + C \cdot COP_z + D$, we can avoid wasted re-computation. This process is symmetric for normals facing in the negative $x$ direction.

The operation count for duplicate back-face operations is 6 multiplications, 6 additions, and 2 comparisons. When exploiting spatial coherence, the worst case is 4 multiplications, 4 additions, and 3 comparisons; the best case is 3 multiplications, 3 additions, and two comparisons.

3.3 Cohen-Sutherland Clipping

Cohen-Sutherland outcode clipping is used primarily in partitioning applications or for trivial rejection for visibility computation. This algorithm does not have any notable adaptation to stereo rendering, but is discussed here to provide terminology for the chapter on implementation.

For each primitive, we assign a single bit to each culling boundary. In a simple two-dimensional example, there would usually be four culling boundaries, requiring four bits. Figure 3-3 shows a sample set of culling codes. Each vertex defining the geometric primitive is tested. If the point's value is outside a boundary, it gets a 1 for that boundary's digit; otherwise, it receives a 0. This results in a 4-bit code for the vertex as shown in the figure.

The culling algorithm proceeds as follows:
If all vertices have the 0000 code, the primitive is trivially accepted. This is the case for the line labeled “accept” in the figure.

The codes for each vertex of the primitive are combined with a logical-and operation.

If the result is not all zeroes, the primitive is trivially rejected. This is the case for the line labeled “reject”: 0011 \& 0001 = 0001.

A result of all zeroes is inconclusive, as in the line labeled “maybe” in the figure. This step is usually followed by an algorithm such as Liang-Barsky, below.

### 3.4 Liang-Barsky Line Clipping

Clipping is the process of reducing a geometric primitive such that it does not exceed a certain boundary. This is in contrast to culling, which leaves all or none of the primitive intact. Clipping is usually applied to ensure that primitives fit within screen window boundaries. For example, the Liang-Barsky line clipping algorithm truncates line segments to fit within a rectangular space. Each line is mapped to separate parametric representations for x and y; then the parameter is varied between 0 and 1 (the segment’s endpoints) to determine any intersection of the line segment and each of the window boundaries in turn.

Since the y value of a point does not vary between the left and right stereo views, it is possible to compute the y-boundary intersections for a segment only once. Adelson et al.
implemented this algorithm, and confirmed the expected 25% reduction in computational complexity [4].
Chapter 4

Shared Rendering

For most systems, the rendering task requires significantly more computation than the culling task. Concomitantly, the opportunity for acceleration by sharing rendering is greater than that from sharing culling volumes. This chapter explores a number of techniques for reducing computation for stereo view generation. As in the previous chapter, the first section describes a novel algorithm, and later sections explain previous work in this area.

4.1 Lazy Shared Rendering (LSR)

Our technique takes advantage of the similarity of faraway objects between nearby viewpoints such as stereo pairs. Essentially, we render faraway objects just once, and use that same rendering of those objects for both eyes. Nearby objects in each eye frustum are rendered independently, so there is still a perceivable stereo effect.

We can describe this lazy shared rendering process with a parameter $\beta$ (see Figure 4-1), which determines the fraction of the objects in the combined viewing frustum which are rendered separately for each eye. For a $\beta$ of one, every object in the combined frustum is rendered twice, once for each eye. This describes normal stereo rendering. A $\beta$ of zero indicates all objects are rendered only once, and that rendering is shared for each eye. This is the equivalent of a monocular view. This 3D/2D decision is made solely upon distance from the viewpoint; therefore, $\beta$ describes the fraction of the distance between $C$'s near and far clipping planes ($C_{near}$ and $C_{far}$). Only objects nearer than the plane defined by $\beta$
4.1.1 Generating Front and Rear Frusta

If we look at our combined frustum $C$, the objects whose rendering is shared are those in the "back" part of $C$. To render this view, we create a new "back" frustum $B$, which encompasses just those objects, so that $B_{far} = C_{far}$ and $B_{near} = C_{near} + \beta(C_{far} - C_{near})$ (see Figure 4-1). The viewpoint of $B$ is the same as the viewpoint for $C$, and $B$ and $C$ share the same aspect ratio. Then, all the objects contained by $B$ are projected onto this new near plane, and rendered to the framebuffer for both eyes. Once all the objects are rendered, the Z-buffer for each eye is reset to ensure that closer objects will always be drawn on top of the objects whose rendering is shared.

To render the objects in front, we use a shortened eye frustum, $S$, where $C$ is the eye frustum, $S_{near} = C_{near}$, and $S_{far} = \beta(C_{far} - C_{near})$ (Figure 4-2). Then we render the left and right views separately from the regular left and right eye positions, using $S$ as
the current frustum for each. The near object renderings are superimposed on the shared renderings in the left and right views, using standard depth buffering hardware.

### 4.1.2 Evaluation of LSR

Assuming that the objects are distributed uniformly throughout the scene, it is possible to analyze performance gain. Let \( \beta \) be such that half the polygons contained in \( C \) are also contained in \( B \); with even distribution, that implies that the volume of \( B \) is half the volume of \( C \). Since the frustum grows away from the center of projection, \( \beta \) will be some value greater than one-half.

The number of polygons in \( C \) is designated by \( N_C \). As in chapter 3, we assume that the two stereo volumes are essentially overlapping, so that \( N_C \approx N_L \approx N_R \). Then without shared rendering, we must render approximately \( 2N_C \) polygons (\( N_C \) for each eye). With shared rendering, we render \( \frac{3}{2}N_C \) polygons (\( \frac{1}{2}N_C \) polygons for frustum \( B \) and \( \frac{1}{2}N_C \) polygons for each of the left and right foreshortened frustums). This yields about a 25% improvement in overall rendering time, assuming the culling process takes equivalent time in either case.

We test this final assumption in chapter 6 to determine what overhead in computation, if any, to expect from the LSR technique. A series of user studies, described in chapter 7, indicate that there exist levels of \( \beta \) in which rendering rates are improved, including the overhead of the computations, with imperceptible or acceptable loss of image fidelity.

### 4.1.3 Known Shortcomings

The LSR method creates only an approximation of the real rendering; far away objects will not be rendered perfectly for each eye. Obviously the objects are drawn without stereo perspective. However, some objects which might not ordinarily be seen are included in the \( B \) volume. The frustum \( B \) is wider than either of the individual eye frustums; in fact, it encompasses them both. The background rendering phase then can include background objects in a small peripheral volume (on the right side of the left eye view, and vice versa) that would not normally be seen. As would be expected, the same issue occurs above and
below the background rendering, since $C$ has the same aspect ratio as $B$.

We assume that these additional volumes are acceptable, as they only occur in the distant background portion of the view. Also, stereo rendering systems are most likely used in highly dynamic environments, such as games and virtual environments, etc., where these errors are less likely to be noticeable.

## 4.2 Previous Work

### 4.2.1 Pixel Shifting

Love describes in his thesis [17] the method of pixel shifting to “back compute” the left and right stereo views from a single cyclopean image. This is essentially an image-warping technique.

The original image is generated facing directly down the Z axis, and split into a series of horizontal scanlines. By combining the depth at each pixel and the interocular distance, it is possible to derive the value of each pixel in the final left and right images. In figure 4-3, the gray object is seen from the center position. The appropriate value for the pixel representing point P on the object’s surface left and right views is generated by finding the pixel shifted the correct distance in the horizontal scanline. In the case of the figure, the depth of P and the resolution of the horizontal scanline suggest shifting about two pixel-widths.
This technique fails when the various viewpoints do not share the same set of visible objects, that is, when surfaces hidden in one view are exposed in another. The appearance of gaps is the most common error; this occurs when objects that should be visible were not visible in the original generated image.

Rather than generating the center image and pixel-shifting twice, it is possible to generate a single eye view and generate the other view with a single pixel-shift process. This saves one series of warp calculations, but generally introduces more error (as the distance between the views has been doubled). This technique is best used by alternating between the rendered view and the shifted view; the effect of the gap artifacts is reduced by the natural blending that occurs during human stereo fusion.

Papathomas et alia [19] were able to mathematically deduce the image for one eye from the other when rendering point sets. Obviously, point sets have none of the standard issues associated with rendering complex scenes, such as lighting effects dependent upon eye position, or visual occlusion.

4.2.2 Hi-Lo Stereo Fusion

First described at SIGGRAPH in 1996, hi-lo stereo fusion is related to the pixel shifting technique [20]. This method again makes use of the human eye's ability to fuse dissimilar images into a coherent stereo pair. Instead of combining an accurate and shifted image, however, hi-lo fusion refers to combining high- and low-resolution rendered images.

The process is simple: the image for the first eye is rendered in full detail. The image for the second eye is rendered independently, but at a lower resolution—and therefore a lower computation cost. These images are presented as a stereo pair, and when fused by the human eye the stereo pair appears to have the resolution of the higher-detail image. The eye receiving the high-resolution image is swapped with each frame to take advantage of retinal coherence. Informal user studies showed the application of the technique to be imperceptible. Computational savings in a polygon-bound (rather than pixel-fill-bound) can be as high as 50%.
4.2.3 Active Edge List Sorting

Adelson and Hodges have described a technique [4] which exploits the invariance of Y values between the eyes to share active edge lists. In a scanline-based rendering algorithm for convex polygons, the task of coloring each pixel is usually performed independently for each horizontal scanline. A list of the intersections between each polygon and the scanline is created; this is called the edge list. The list is ordered by x-intercept. As the algorithm works across the scanline, the list stores which edges the pixel is within; these are the active edges. For each pixel position, the depth of the active edges is compared to determine the color for the pixel.

Normally this process is repeated for each viewpoint. This technique shares the edge list for each scanline, since they are the same due to y-invariance between the eyes. The work along the scan lines still needs to be performed separately for each eye.

4.2.4 Accelerated Ray Tracing

Adelson and Hodges have demonstrated that spatial coherence can be exploited for rendering via ray tracing techniques as well [2]. Their method is essentially an adaptation of an earlier technique for speeding image generation in multi-frame animations. The ray tracing algorithm proceeds as usual, starting at one of the two eyepoints. When an object is hit by a ray traced from the eye, the algorithm attempts to trace backwards to hit the other eye. This reprojection process can have errors if more than one object projects to the same pixel in the second, generated view.
Chapter 5

Implementation

The techniques described in chapters 3 and 4 were implemented in a virtual flythrough program, both to test the efficacy of these algorithms and to offer a platform for user experimentation. The first sections of this chapter describe the software and hardware components upon which the flythrough was built, both to enable the repetition of these experiments and to justify their selection. The remainder of the chapter explores the flythrough software, with special attention to the lessons learned in various implementations of stereo image generation algorithms.

5.1 Software

5.1.1 Graphics Libraries

The choice of graphics library dictates many elements of any graphical application. A major factor in selection for this project was the ability to manipulate primitives at the triangle level; libraries with intrinsic scene graphs were specifically avoided. The OpenGL library was chosen for its low-level approach, as well as capabilities for stencil buffering and stereo display. Conveniently, the OpenGL library is freely distributed on a number of hardware platforms [26].

The choice of OpenGL left two additional concerns for this project: lack of graphical interface tools, and lack of model import operations. The GL Utility Toolkit (GLUT)
was chosen for creating windows and keyboard and mouse events. GLUT is also freely distributed for a variety of operating systems and hardware platforms [14]. GLUT does not employ any sort of scene graph or file format; to solve the modeling problem, the flythrough acquires model data via the OpenInventor library [22]. The OpenInventor file format is widely used, and a number of quality models and scenes are available without cost.

During the initialization phase, the flythrough program loads a user-specified OpenInventor model, creating a scene graph in memory. A special traversal of the graph is performed that identifies every individual triangle in the scene and stores them in a list. The scene graph is then removed from memory, and the triangle decomposition is rendered with OpenGL. The modification of that list of triangles is the key to the JFC and LSR implementations, and is described later in this chapter.

5.1.2 Software Features

The flythrough software includes a session recording feature, which tracks the viewing matrix of the user as she moves through the environment. These sessions are written directly to disk, and can be played back during current or future sessions. This function allowed user experiments to include tests with eyepoint motion, by presenting a real-time “animation” which was generated by session playback.

The flythrough was originally designed for manual manipulation of the viewing parameters; as a result, factors such as interocular distance, percentage of the world presented in 3D, and focal convergence could be adjusted to taste. While these functions were very useful for informal pilot testing, all such functions must be deactivated during stimuli presentation. Command-line arguments yield display of a given scene, with certain viewing parameters, for a presentation time within the accuracy of screen update (usually about 15 milliseconds). Experimentation sessions consist of repeated invocation of the stimuli viewer with varying arguments. A helper program was written that generates batch files that invoke the program, for all legal combinations of the given arguments, in a random order.
5.2 Display Platform

For each implementation of the flythrough, SGI hardware was used for computation and graphical rendering. IRIX, the SGI version of UNIX, was used exclusively; versions 6.2 through 6.5 were tested successfully. The primary hardware platform was an Octane SSE with dual 250 MHz R10000 processors and 512 MB of memory. An O₂ with 200 Mhz R5000 processor, and an Indigo² with Gallileo video board, were also used but only with time-parallel display hardware. These computational and graphics rendering hardware options are essentially homogeneous, but the actual stereo display hardware varies greatly. The project was realized upon two distinct stereo display platforms, using time-multiplexed and time-parallel hardware.

5.2.1 Time-Multiplexed Stereo: StereoEyes Glasses

The primary display hardware used for experimentation were StereoEyes shutter glasses¹. These glasses simply pass through the view from a standard computer monitor, which generally allows higher resolutions. The SGI Octane used for display was able to present at a resolution of 1024 by 768 pixels. Time-multiplex methods necessarily reduce the perceived display update rate by one-half; in this case, halving the monitor display rate of 96 Hz led to a perceived 48 Hz update rate in each eye.

OpenGL stereo is compatible with SGI video buffers and stereo switching, greatly simplifying the process of directing geometric primitives to left or right views. A simple command sets the destination buffer for drawn primitives; in the double-buffered case, the new options are back left, back right, front left, and front right. Objects to be drawn to both left and right views simultaneously are directed to the back buffer, just as this would normally be performed in a non-stereo mode. Timely switching between these drawing options was the key to the implementation of the JFC and LSR; this process is described later in this chapter.

¹Section 2.2.3 contains a discussion of stereo presentation techniques.
5.2.2 Time-Parallel Stereo: Virtual I/O Glasses

The flythrough also works with Virtual I/O Glasses, a low-end display system intended for the home entertainment market (now discontinued). These glasses use a small color LCD display for each eye, at a resolution of 300 pixels per line by 200 lines. The glasses accept a standard NTSC video signal for input at a resolution of 640 by 480. The even and odd scanlines in the NTSC signal are separated in hardware by the glasses; all the odd lines go to one eye display while all the even lines go to the other eye display (Figure 5-1). The line sets are directed to a particular eye by an external switch.

The 480 scanlines are divided into two sets of 240, and it is not clear how these sets are scaled into two sets of 200 lines. Most likely the glasses overscan, that is, they discard lines on the top and bottom of each eye screen. That implies only the middle 400 lines of the NTSC input signal are actually used. Similarly, the glasses scale 640 pixels per line into 300 pixels per line. Presumably, the glasses perform some mix of interpolation, averaging, and picking to reduce the number of pixels per line.

No standard SGI graphics modes support interlaced stereo rendering. Therefore, it was necessary to perform this interlacing in software. A naïve method would be to generate
the left and right views as images, and then combine them line by line using buffer copy operations. This process requires additional buffer memory (three copies of the rendered image instead of only one), but more importantly is very slow. Each line must be copied separately, requiring a large number of completely independent hardware calls.

A better solution is possible, using the OpenGL stenciling features. Each OpenGL buffer can have an associated stencil buffer, just as each buffer can have an associated depth buffer. When rendering, a "stencil" is used to allow a pixel to be drawn if and only if its associated stencil buffer value satisfies some criteria. Sample criteria are to draw only when the stencil value is equal to one, greater than zero, and so forth. In this implementation, a single 640 by 480 OpenGL buffer was created with associated depth and stencil buffers. Then, the stencil buffer was initialized so that all the even lines were set to 1, and all the odd lines were set to 0. To get different eye views on different lines, it was only necessary to change the stencil criteria before rendering each eye’s view. For example, before drawing the left eye’s view, the stencil mode was set to draw only where the stencil was equal to 1 (the even lines). Next, the stencil criteria was set to draw wherever the stencil was not equal to 1, and the right eye view was rendered into the odd lines.

5.3 Implementation

This section explores the implementation of each of the various culling and rendering algorithms described in previous chapters.

5.3.1 Software Triangle Culling

The baseline implementation uses simple software-based linear culling. Our method culls only triangles that can be trivially rejected, that is, which have all three vertices on the wrong side of any of the six planes bounding the view frustum. All other triangles are still examined by the hardware pipeline, where many are culled or clipped to the viewing window. The result of the tessellation of the OpenInventor model is stored in a one-dimensional array. The draw process simply loops through the array; each triangle is examined and
either culled or sent immediately to the graphics hardware. The process is performed separately for each eye.

The application of this process actually slows down the display rate of the flythrough. Performing these dot-product tests in the hardware graphics subsystem is certainly more efficient than using the main processor. This implementation is an intermediate step for the JFC implementation, but has been separated to give a better understanding of the calculation overhead of various algorithms. Efficiency comparisons for this and all the techniques below can be found in chapter 6.

5.3.2 Joint Frustum Culling

The JFC technique uses an additional array of integers, the same size as the primary triangle array. At the beginning of each draw cycle, each value in the new array is initialized to negative one. Then, the culling proceeds as before, but instead using the combined frustum $C$ (see chapter 3 for terminology) for the culling boundaries. If a triangle is not culled, its array index is placed in the first empty slot in the integer array (see Figure 5-2). The final result is a list of indices for triangles within $C$, which is less than or equal in length to the triangle array.

Then, the left and right views are culled and drawn just as in section 5.3.1. Depending on the percentage of triangles in $C$, the reduction in work (by testing those triangles once for $C$ instead of twice for $A$ and $B$) can be significant.

5.3.3 LSR Shared Rendering

The shared rendering implementation is a straightforward extension. This requires two additional arrays of integers, again to contain triangle array indices. First, shared culling is performed using $C$, as described in the previous section. The second cull, for frustum $B$, loops over the array of triangles not culled from $C$ (see Figure 4-1 for terminology). If a triangle is not culled from $B$ it is added to the new “back” array; otherwise, it is added to the “front” array (see Figure 5-3).

Next, the triangles from the back array are rendered into the framebuffer using $C$ as
Scene triangle array

Pointer array

Figure 5-2: Shared culling implementation.

Figure 5-3: Culling for shared rendering.
the current view frustum. These triangles are drawn to both the left and right eyes. For the shutter glasses, the drawing mode is set to render to both left and right back buffers simultaneously, and the “back” array is drawn in its entirety. The depth buffer is cleared, and drawing is set to the back left. Then the “front” array is rendered, but the triangles are first culled to the left eye frustum. This process is repeated for the right eye.

For the time-parallel glasses, the process is similar. The stencil function is turned off and the “back” array is drawn. The depth buffer is cleared and stenciling is enabled; each eye is drawn in turn from the “front” array after culling the triangles to the individual eye frusta.

A problem with this implementation is that the culling process is excessively sloppy. When the triangles in $C$ are split into the front and back groups, only trivial reject cases are put in the front array. Thus some triangles which should be culled from $B$ and placed into the front array are actually left in the back array. When the back array of triangles is sent to the hardware with $B$ as the current frustum, those triangles which should be drawn in the foreground are culled by the hardware, and not drawn. But since triangles are forced to be in only one of the back or front arrays, these triangles culled by the hardware are never drawn at all, even though they should appear in the shortened left and right eye frustums. This makes for some visible artifacts at the boundary between the front and back viewing volumes, but is hard to overcome without using a more exact (and more costly) culling mechanism.
Chapter 6

Performance Comparisons

We performed a set of tests, showing that our methods do save a constant fraction of work for suitable input distributions. We describe the test cases, methodology, and results below.

6.1 Test Models and Paths

We generated a set of seven simple models. The models were three-dimensional arrays of randomly-colored cubes, whose generation were governed by three binary variables:

- **Sparseness** The sparse models had cubes of dimension 1 and were separated by 10 units; the non-sparse were separated by 1 unit.

- **Size** The models were either $4 \times 4 \times 4$ or $7 \times 7 \times 7$ arrays.

- **Thickness** The cubes were made with either 12 or 120 triangles, to vary complexity.

So, models with names beginning with “sparse” are more separated, and models whose names begin with “busy” are tightly packed. We expect the sparser models to show more benefit from multiple culling passes. Models with “+” appended to the name are $7 \times 7 \times 7$ arrays, and models with “t” appended contain thick cubes ten times the number of triangles.

Our experiments used a frame timer which recorded the time between the start and finish of the drawing function. Our drawing function was invoked by `GlutIdleFunc()`, which runs the given function only when the program is idle. Therefore the perceived
frame rate is not deterministically predictable from our frame times, which actually gave us more accurate comparison data. We had pre-recorded “flight paths” which we used to “fly” through the models in a repeatable fashion, so that each test run encounters the same series of model views.

6.2 Timing Results

The results were generally consistent with our predictions of performance. Performing combined frustum culling before culling each eye, was always faster than simply culling the whole model to each eye. In some cases the speedup was less dramatic, specifically when the model was very dense and therefore generally completely contained within the combined frustum. Our flight paths through the model were chosen to have the models fully within the combined frustum a little more than half of the time, which explains the speedup.

The results of the shared rendering method generally followed our predictions as well. We knew this method would show improvements only in those places where the combined culling was effective. We also found that it was only useful when the section of the frustum designated as shared actually contained objects. We chose the 7 × 7 × 7 thick and sparse models as the most indicative of this type of model, and acquired extra data points for these models.

In the graph (Figure 8), $\beta$ is the percentage (distance-wise) of the combined frustum that is rendered individually for each eye point, as discussed above. The straight line at the top is the frame rate using combined culling only.

The shared rendering does have some overhead, as it requires an additional software culling pass through the combined frustum. Therefore, one can see that for high values of $\beta$, combined rendering is actually slower than combined culling alone.

Table 1 shows where texturing begins to pay for itself by providing more performance gain as the total rendering time increases. Note that there are many possible values of $\beta$ which we believe still give the three-dimensional experience but are faster than combined frustum culling alone. For all values of $\beta$, shared rendering is faster than the naive method.
Figure 6-1: Average rendering time vs. $\beta$.

<table>
<thead>
<tr>
<th>Model</th>
<th>Rendering method</th>
<th>$h/w$</th>
<th>$s/w$</th>
<th>JFC</th>
<th>LSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>sparse</td>
<td>primitive</td>
<td>92</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>sparse-t</td>
<td>primitive</td>
<td>762</td>
<td>170</td>
<td>133</td>
<td>128</td>
</tr>
<tr>
<td>busy</td>
<td>primitive</td>
<td>92</td>
<td>24</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>busy-t</td>
<td>primitive</td>
<td>787</td>
<td>359</td>
<td>320</td>
<td>337</td>
</tr>
<tr>
<td>sparse+</td>
<td>primitive</td>
<td>411</td>
<td>64</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>sparse+-t</td>
<td>primitive</td>
<td>4320</td>
<td>1302</td>
<td>942</td>
<td>754</td>
</tr>
<tr>
<td>busy+</td>
<td>primitive</td>
<td>431</td>
<td>161</td>
<td>131</td>
<td>161</td>
</tr>
</tbody>
</table>

Table 6.1: Average frame rendering times (milliseconds).
Figure 6-2: Average rendering times, with $\beta = .4$ for combined rendering.

In the table, h/w designates the naive stereo method with hardware culling, s/w is the same method but with software culling instead, JFC means joint frustum culling was used, and LSR means shared rendering was used along with shared software culling. Table 6.1 is also summarized in Figure 6-2.

We were very surprised to see how much more slowly the hardware ran in all cases (generally a factor of three), especially considering that we used a linear software culling algorithm. We can only speculate that this is a problem with graphics hardware bandwidth. We have been unable to track down anything in OpenGL that might be causing such overhead. However, it does seem odd that a machine with specialized graphics hardware would have such bad application throughput.
Chapter 7

User Experiments

We have examined the computational savings afforded by the use of LSR shared-frustum stereo rendering. (The Joint Frustum Culling technique does not degrade quality and therefore does not require user evaluation.) However, these savings come at a cost in the quality of the images produced. This chapter discusses user reactions to this reduced quality. Suggestions for more extensive and carefully controlled user evaluations can be found in chapter 8.

We conducted a series of informal pilot studies to better gauge the parameters for any following user study. Each of these studies is discussed below, with experimental results and the contribution their analysis made to our understanding of the LSR technique. Usually between three and five subjects were used, with very short test times on the order of five to fifteen minutes. The subject pool consisted primarily of fifteen students from the Naval Postgraduate School and Massachusetts Institute of Technology. All had average to extensive familiarity with virtual reality applications and technology. 80% of the subjects were male. Unless stated otherwise, the LCD shutter glasses were used with SGI Octane hardware.

7.1 \( \beta \) Investigation

Problem. To explore subject responses to changes in \( \beta \).
Method. The virtual flythrough was used for three scenes: a three-dimensional grid of cubes; a complex model of a biplane; and a cityscape with large virtual extent. The subject was presented these scenes in a series of still images, with $\beta$ levels in the set of $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. In each case, the subject was asked to respond to two questions: (1) whether the image gave the impression of three-dimensions via stereoscopy; and (2) whether $\beta$ had been set less than 1.

Results. Results varied significantly between models for the first task. For both the biplane and grid, the minimum value of $\beta$ that provided stereo quality was in the [0.2, 0.4] range. For the city model, responses were inconsistent across subjects for both directives. For the second task, subjects could correctly determine cases when $\beta$ was less than 1 almost 100% of the time for the grid and biplane models. For the city model, results were inconsistent.

Discussion. Subjects felt that some stereo information could be omitted without significant loss of the immersion quality associated with stereo viewing.

The results imply that the Just Noticeable Difference for $\beta$ is quite low. This is as expected: a small change in $\beta$ can result in noticeable parts of a scene switching from flat to stereo presentation.

The extent of the cityscape scene is such that stereo disparity for most virtual objects is lost in the display resolution. Therefore, subjects could not possibly notice changes in $\beta$, nor notice any concomitant changes in quality.

7.2 Time-Parallel vs. Time-Multiplex

Problem. To determine the difference in stereo quality between the two available display devices, LCD shutter and dual-CRT glasses.

Method. Subjects were presented a series of stereo images in one of the stereo devices; the series was then presented (in the same order) in the other device. For parity, half of the subjects were given the LCD shutters first, and the others given the dual-CRT glasses first.
Subjects were then asked to choose the display they supposed to be the more generally effective.

**Results.** All subjects preferred the LCD shutter glasses.

**Discussion.** These results were as expected; the time-multiplex display offered an order of magnitude higher resolution, and higher frame rate. It is important to note that this was not a general selection of multiplex over parallel displays, but rather the selection of shutter glasses and a monitor over Virtual I/O I-glasses. For instance, shutter glasses can have interocular crosstalk (ghosted images of the wrong eye’s view) that dual-CRT displays do not, which might be crucial for some tasks. An evaluation of performance in a particular task, without the side-by-side comparison, would likely be a more appropriate test.

### 7.3 Motion Effects

**Problem.** To gauge the effect of motion on subject perception of stereo in the presence of varying $\beta$.

**Method.** A scene containing a three-dimensional grid of cubes was presented to the subject. They were given a slider to control the $\beta$ value and instructed to choose the minimal value for which they had the impression of three-dimensional stereoscopy. Subjects were then presented the same scene, with identical instructions, except in this case the viewpoint moved upon a pre-recorded flight path.

**Results.** Motion extended the range of $\beta$ for which subjects answered that they had the impression of stereoscopy. Subjects responded positively to values in the 55-100% range without motion, and in the 40-100% range for the same scene with motion.

**Discussion.** Motion parallax is a powerful cue for depth discrimination [21], and can sometimes be sufficient for immersive display of three-dimensional models without stereo. It is unsurprising, then, that subjects were willing to tolerate a lower value of $\beta$ in exchange
for motion. These results may be affected by improper subject instruction; subjects were asked to evaluate stereoscopy but may have mistakenly been evaluating three-dimensional immersion. It is important to note, however, that regardless of the subject’s perceived criterion, that criterion would likely be constant between the stationary and moving scenes. The experiment was designed to test the difference in lowest tolerable $\beta$, and so allowing the subject to define “lowest tolerable” was perfectly acceptable.

7.4 Presentation Time

**Problem.** To determine the effect, if any, of presentation time on the effects of a $\beta$ value less than 1.

**Method.** Subjects were presented with a series of ten scenes; all scenes’ virtual objects were within a range permitting stereo disparity. Each scene was presented with the same $\beta$ range as the initial experiment of section 7.1. Presentation was for a number of milliseconds chosen from the set \{50,100,200,400,750,1000\}. Subjects were asked to respond whether the image gave the impression of three-dimensions via stereoscopy. The responses were then compared with those of the initial experiment, in which presentation times were not limited.

**Results.** The responses for times of 50 and 100 milliseconds were inconclusive. Responses for a presentation of 200 milliseconds were slightly higher than the initial experiment which had unlimited presentation times. For all other time values, responses were the same as in the previous experiment.

**Discussion.** Given the two-alternative forced-choice, subjects simply chose responses randomly for the shortest time intervals. The quick display did not give an opportunity to fuse properly on a stereo image. Longer times (400ms and up) were essentially equivalent to infinite presentation time; in fact, subjects often would respond before the allotted time had elapsed.
From these results, it was determined that presentation time would not be a useful factor in a larger experiment. Times shorter than ideal quickly become too brief for proper accommodation, and any advantage afforded by increased presentation time appears to drop off quickly after the ideal value.

7.5 Chromostereopsis Factor

Problem. To determine the effect, if any, of color or contrast variations on stereo quality.

Method. A depth discrimination experiment, which depends closely upon stereo accuracy, was developed. The user was presented a series of scenes with two objects, placed at slightly varying or equal depth. They were then asked to state which object looked to be closer. The experiment was repeated with color variation (red and blue) and brightness variations (levels of grey) in the two objects.

Results. Accuracy in the depth discrimination test was inverse to the nearness in depth of the two objects. Color variation did not noticeably affect accuracy. Brightness variation did affect discrimination accuracy, with the brighter object occasionally perceived (incorrectly) as being closer to the viewer.

Discussion. Chromostereopsis effects can make red objects appear closer than blue objects. This effect is usually very minor and does not appear to affect results. Future experiments can likely incorporate extreme color variation in the virtual objects without concern. Due to liquid refraction properties of the eye, bright objects appear nearer than dark ones. In future experiments significant brightness variations should be carefully considered.
Chapter 8

Conclusions and Future Work

This work makes the following contributions:

- two novel algorithms for acceleration of stereo image generation, Joint Frustum Culling and Lazy Shared Rendering

- programmatic implementation of algorithms, with interface for experimentation with parametric variation

- evaluation of the algorithmic potential, and evaluation of actual performance

- early experimentation on human adaptation to mixed stereo/monocular presentation

- early evaluation of quality versus performance trade-off

- collateral software for automatic generation of virtual models, and for experiment protocol organization

8.1 Suggestions for Future Work

A primary lesson learned during this effort was the difficulty of the validation process for any quality trade-off. Few improvements in graphical rendering technology have to face this hurdle, but likely all should. To properly validate the effectiveness and utility of LSR, it is necessary to follow these pilot studies with more in-depth experimentation. The difficulty
is in planning a study that tests the technique properly. We recommend a follow-on study that varies frame rate, $\beta$, virtual scene form and complexity, and application task. The study would test three different criteria, all of which are important to this validation:

- **experience**, a qualitative expression of the enjoyment of the virtual experience, as affected by the stereo fidelity;

- **noticeability**, a qualitative expression of the degree to which the subject is aware of the application of the technique; and

- **performance**, a quantitative expression of the subject’s effectiveness at the application task.

Determining specific definitions for these factors, and proper instructions for subjects in such studies, would be a major component of this effort.

The evaluation phase unsurprisingly exposed limitations in the first implementation. For instance, the planar nature of the discontinuity between monocular and stereo rendering fields could cause visual artifacts. The mono/stereo decision was made at the level of the lowest primitive (the triangle) and so complex objects containing multiple triangles could be mixed between the two modes with unpleasant results. While this was technically correct, an object-centric implementation could allow for greater flexibility. The discontinuity would still be present for large objects, but small objects on the $\beta$ border could be placed entirely in the mono or stereo frustum. This could result in excellent quality for a model sparsely populated with small objects.

Finally, we are in the midst of constructing an immersive display that uses polarized light to deliver time-parallel stereo. We hope to find that these techniques are equally effective on this third viewing platform.
Bibliography


