School of Computer Science and Software Engineering, Monash University

Bachelor of Digital Systems Honours (1200),
Clayton Campus

THESIS

Panoramic Rendering for Concave Surfaces of Revolution

Dominik Deak, ID : 12268666
Supervisor : Jon McCormack

November 5, 2001

Contents

1 Introduction **2**

2 Related Work **3**
2.1 Methods for Creating Panoramic Photographs 3
2.2 Image-Based Virtual Environment Navigation 4
2.3 Head-Mounted Displays 5
2.4 "Fish Tank" Virtual Reality 6
2.5 Multiple-Projector Panoramic Displays 6
2.6 Single-Projector Panoramic Displays 9

3 Panoramic Rendering **11**
3.1 Representing Projection Surfaces with Profile Curves 11
3.1.1 The Profile Curve 11
3.1.2 The Screen Space and the Projection Surface 13
3.2 Using Profile Curves for Ray-tracing 16
3.3 Techniques for Realtime Rendering 18
3.3.1 Projecting Points onto a Surface of Revolution 18
3.3.2 Implementing Realtime Transformations 21
3.3.3 Polygon Subdivision 23
3.4 Compensating for the Projector 25

4 Results **27**
4.1 The Ray-tracer Implementation 27
4.2 The Realtime Implementation 28
4.3 Projector Viewpoint Compensation 30

5 Discussion **31**
5.1 The Ray-tracer Implementation 31
5.2 The Realtime Implementation 32

6 Conclusion **35**
6.1 Review 35
6.2 Future Work 36

A Illustrations **38**
A.1 Images for the Ray-tracer 38
A.2 Images and Data for the Realtime Implementation 40

Abstract

Panoramas, or pictures with a wide field-of-view are not an entirely new concept, such visualization existed since the last decades of the 18th century. In this thesis, a number of techniques and methods for generating/displaying panoramic images were investigated. Most computer generated panoramas usually undergo a two stage transformation process. Initially the 3D objects are transformed into another coordinate system, such as cylindrical or spherical coordinates. These objects are further transformed into a 2D picture coordinate system, giving the final panoramic picture. The thesis will detail two mayor methods for panoramic rendering specific for concave surfaces of revolution. The techniques include a ray-tracer and a realtime implementation of the panoramic rendering system. Issues relating to point transformations, polygon subdivisions, projector viewpoint compensation and the panoramic distortion effects will be discussed.

1 Introduction

Panoramas or panoramic images try to overcome the restricted field-of-view (FOV) of traditional displays, such as televisions and computer monitors. This is achieved by capturing the scene in a greater field of view. Panoramas, also known as omnidirectional images [42] usually come in the form of cylindrical, spherical or cubic [18] images. A cylindrical panorama would be a single image "strip" that displays a 360^o view of the scene. Spherical panoramas often show a distorted view of the scene, similar to the image seen through a "fish-eye" lens [37]. Cubic panoramas can be considered as a construction of six picture faces, showing the scene from six different viewing directions.

Today, a number of movie theaters can display films in the standard 2.35:1 aspect ratio format. In this thesis, any display system that is capable of displaying images with an aspect ratio greater than 2.35:1 will be considered as a panoramic display. Panoramic display systems employ various techniques for creating a visually immersive environment. Many use projectors to display an image on a surface, such as a wall, a dome, or even irregular objects. The systems discussed in Section 2 can be classified into different categories, based on the following features:

- Display shape and type, eg.: Head mounted, curved, cubic, flat, etc.
- Renderer type: Realtime, or image-based.
- The steps involved in visualizing the panorama.
- The number of projectors/displays used.
- Interaction capabilities with the user.

Traditional perspective view rendering systems display objects by projecting 3D geometric shapes onto a view plane, which are eventually shaded in screen space. Ray-tracers in the other hand fire incident ray vectors trough the view plane, and perform ray-object intersection testing. The color at the object's intersection point is

stored at the corresponding screen location where the ray intersected the view plane. Similar rendering techniques can be used with panoramic rendering. However, in this case the viewing surface is no longer a plane, it is a curved surface that possibly provides a greater field of view.

Most computer generated panoramas usually undergo a two stage transformation process. In the first stage, the 3D objects are transformed into another coordinate system, such as cylindrical or spherical coordinates, depending on the type of panorama desired. In the second stage, these objects are further transformed into a 2D picture coordinate system, which gives us the final panoramic picture.

The thesis will detail techniques for panoramic rendering specific for concave *surfaces of revolution*. The entire project consisted of two main parts: The ray-tracer implementation and the realtime implementation. The two panoramic rendering techniques will be discussed separately in Section 3.

2 Related Work

2.1 Methods for Creating Panoramic Photographs

Early pioneers in photography devised a number of methods for constructing a 'single' panoramic photograph. Barnard's mosaic method is one example, where the panorama consisted of a series of tiled photos, which often showed discontinuities [13]. With the aid of computers, mosaic-based panoramas can be easily created with a standard hand-held camera [49]. The panoramic mosaics are generated by capturing a series of photographs during a horizontal panning sequence [54], and then by mapping them onto 2D cylindrical or spherical screen coordinates. The panning angles and translation values between two mosaics are recovered [54], and the final panorama is constructed by blending the overlapping mosaics. The disadvantage of this method is that imperfect image blending will cause obvious smears and "creases" in the panorama. Color discontinuities are also a problem, because each mosaic may not have the same color temperature or exposure.

To reduce the problem of smears and color discontinuities, Peleg and Herman [41] developed a scanning technique, called the Manifold Projection, where a panning camera captures the scene in a video stream. Each video frame is recorded at a close panning proximity from the others, and treated as a 2D mosaic image that contains a 1D image strip in the center. Two or more mosaic frames are used to estimate the direction of the camera's motion, which gives the correct alignment of the 1D center strips. The final panoramic image is constructed from a series of aligned center strips.

Mosaic related problems can be further minimized by using special purpose panoramic cameras, such as the early Cirkut and swivel-lens cameras, or cameras that are mounted with hyperbolic or parabolic mirrors (see [53], [52], [55], [38] and [39] for mirror related topics). Omnidirectional video cameras such as the FullView 360 [2], use five mirrors mounted at 45^o that form a five-sided pyramid. Each mirror has a video camera looking down the 45^o mirror surface, capturing $\frac{1}{5}$th of the panoramic view. The five images are stitched together to form a cylindrical panoramic image. Since each mirror/camera pair is fixed at one position, there is no need to estimate

the panning and translation values between each image segment. This minimizes blending errors and color discontinuities in the final panorama.

2.2 Image-Based Virtual Environment Navigation

Rendering software that implements virtual environment navigation aims to provide an interactive system, allowing the viewer to explore a panoramic scene. Image-based rendering systems such as QuickTime VR [18] and OmniVideo [42] use images and photographs (discussed in Section 2.1) to construct the virtual panoramic environment. The video stream or still images are treated like an environment map [18], and are projected onto a geometrical shape such as a cylinder, a cube or a sphere.

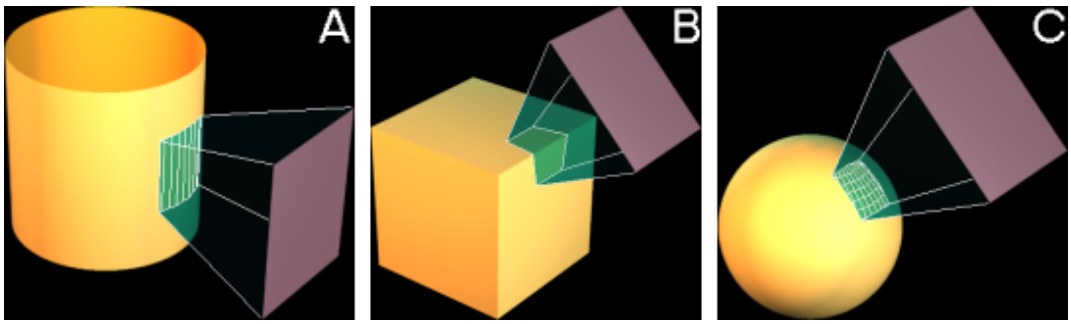

Figure 1: Displaying Portions of a Panorama. *(A)* Cylindrical, *(B)* Cubic and *(C)* Spherical Environment Map.

In Figure 1, the shaded regions on the geometrical primitives represent the projected environment map. Only a small, perspective correct region of the environment map is displayed at a time, because it is rendered on a conventional 2D display. The environment map region enclosed by the view volume is projected onto the view plane, giving the final image. The viewpoint or the camera is located in the center of the primitive.

The QuickTime VR system is restricted to render only a single perspective view from a static omnidirectional image. The OmniVideo [42] system overcomes this limitation by allowing multiple perspective and panoramic views from a single omnidirectional video input (Figure 2). Initially, the 360^o view of the scene is orthographically reflected onto the camera's CCD by a parabolic mirror [53] [52] [38]. The system obtains a 2D perspective view of the scene by re-projecting, or warping a small section of the omnidirectional picture onto a view plane (also called the *virtual perspective camera*). The pixel coordinates in the omnidirectional image for a given *virtual camera* pixel coordinate are computed using transformation equations specifically for parabolic mirrors [42]. Since the re-projections must be performed in realtime, these transformations are pre-computed and stored in a look-up table, known as a *geometric map*. This technique is also used for transforming the parabolic panorama into a 360^o cylindrical panorama.

A typical interactive environment that allows movement in the scene can be implemented by simulating a camera with six degrees of freedom [18]. This includes 3D camera translation and 3D rotation at a fixed point. Most image-based render-

Figure 2: The OmniVideo system. A panoramic view and multiple perspective views (right) are created from a single omnidirectional video frame (left).

ing systems have a fixed viewpoint, which usually reduces the camera's degrees of freedom to rotation and zooming only. Camera translation or *navigation* is possible by implementing "hot spots" [18], which act as a predefined path (or a link) between two panoramic scenes. The only limitation with such navigation is that the renderer will result in "jumping" from one scene to an other. Kang [34] suggested a method that allows navigation without jumping, by applying a transition algorithm between two or more scenes. As the viewer passes through a hot spot, image sections are taken from each nearby panoramic scene, which are perspective transformed and then stitched together to create a new virtual panorama around the viewer.

2.3 Head-Mounted Displays

Instead of implementing image-based rendering on a conventional 2D display, head mounted displays (HMD) can be used to render a perspective view of the virtual scene. A head tracking system can register the orientation of the head and update the camera orientation accordingly.

Some of the earliest HMD systems by Philco Corporation in 1961 [33], and later by Ivan Sutherland in 1968 [51], used a mechanical arm to implement head tracking. The system rendered a wire frame model of the scene on two CRT displays. Due to high development costs, most HMD systems today still employ low resolution displays, which may be inadequate for some practical applications. However, the latest in liquid crystal display (LCD) technology by Kraiser Electro-Optics [9] allow HMDs to render graphics at 1024×768 resolution. See-through HMD technology can be used to create *augmented virtual reality* (see Feiner [26]), where computer graphics are superimposed over visible objects.

Most HMD systems experience problems with the accuracy of the head tracking hardware. Prolonged use of the HMD can lead to the accumulation of tracking errors, which causes the measured head position to be out of alignment with the true position [33]. Occasional calibration of the head tracking is necessary to correct

the measurements. In some systems, there are also latency problems [40] between the user's motion and the display update. Latency is an important issue, because the user may develop symptoms of disorientation and seasickness.

2.4 "Fish Tank" Virtual Reality

Since HMD systems are usually expensive, an alternate method for creating an interesting VR environment is described by Deering [23] [22]. The system uses a head mounted shuttered glasses with a head motion tracker. The display itself is high resolution, stereo CRT desktop monitor. The system registers the location of each eye and computes the two corresponding view volumes. With the origin being the eye position, each view volume passes through the display screen, and extends "behind" the monitor. The system renders the left and right views of the scene, which are alternated on the screen and synchronized with the shutter glasses.

Deering found that the resulting stereo image had significant positional errors on the screen, mainly due to the physical properties of most monitors. To minimize such errors, it was necessary to compensate for the curvature and the refraction properties of the faceplate glass.

2.5 Multiple-Projector Panoramic Displays

The limitation of image-based rendering is that the user is restricted to view the panorama on a conventional 2D display only. While HMD systems can overcome such limitations, wearing the display itself can be inconvenient and can suffer from latency issues. However, panoramic scenery can be displayed on a large screen, sometimes large enough to require several projectors.

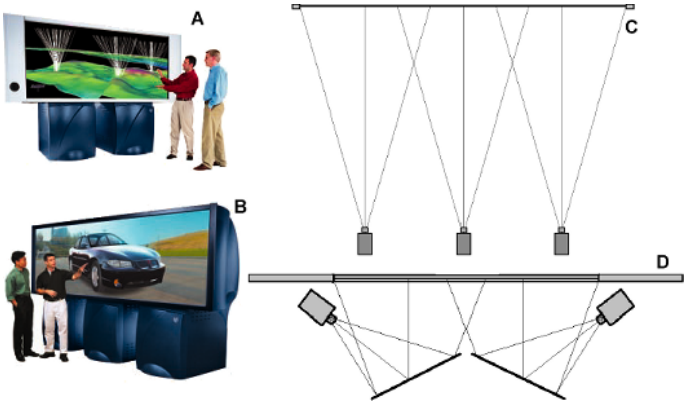

Figure 3: SGI Reality Center 3300W: *(A)* two and *(B)* three projector system. Panoram Technologies' *(C)* FFP Series (up to 30m wide) and *(D)* PanoWall2 KS.

An example of a three-projector theater display is the FFP Series [1], which gives an aspect ratio of 3.33:1 (Figure 3c). The PanoWall2 KS [4] rear-projection system is smaller and uses two projectors (see Figure 3d). In both systems, edge blending is applied on the image sections to form the complete picture. To prevent discontinuities in the final image, the projectors must be accurately aligned. The SGI Reality

Center 3300W [6] system tries to overcome such alignment problems by housing the projectors inside the unit (Figure 3a and 3b).

Theater-based screens that use multiple projectors don't offer a complete immersion, because the scenes are displayed in a forward view only. (However, they can be configured to act as an image-based VR system, as detailed in Section 2.2). An ideal system would be configured to surround the viewer with a spherical display. A cubic approximation of the spherical display was first demonstrated by the CAVE system [20] and later, the HUTCAVE [32] system. Most CAVE-like systems implement three or four display walls (the fourth being the floor); however the PDC VR-CUBE [12] uses six walls to completely surround the viewer (Figure 4b).

Inside the cube structure, the user's left and right viewpoints are constantly tracked. The shuttered glasses worn by the user are synchronized with the alternating left and right views projected on the display. Each display wall is represented as a separate view plane by the renderer (Figure 4a). During the rendering process, the objects are clipped and projected onto the view planes using off-axis perspective projection. Using the front view plane as an example, the projection of a 3D point is given by:

$$\begin{aligned} x_i &= x_p + \frac{(d - z_p)(x_o - x_p)}{z_o - z_p} \\ y_i &= y_p + \frac{(d - z_p)(y_o - y_p)}{z_o - z_p}. \end{aligned} \tag{1}$$

A typical view vector runs from the view origin $\mathbf{O} = [x_o, y_o, z_o]$ to a 3D point $\mathbf{P} = [x_p, y_p, z_p]$. The projection of $\mathbf{P}$ occurs when the view vector intersects the view plane at $\mathbf{I} = [x_i, y_i]$. The value for d in Equation 1 is the distance between the front wall and the center of the CAVE system.

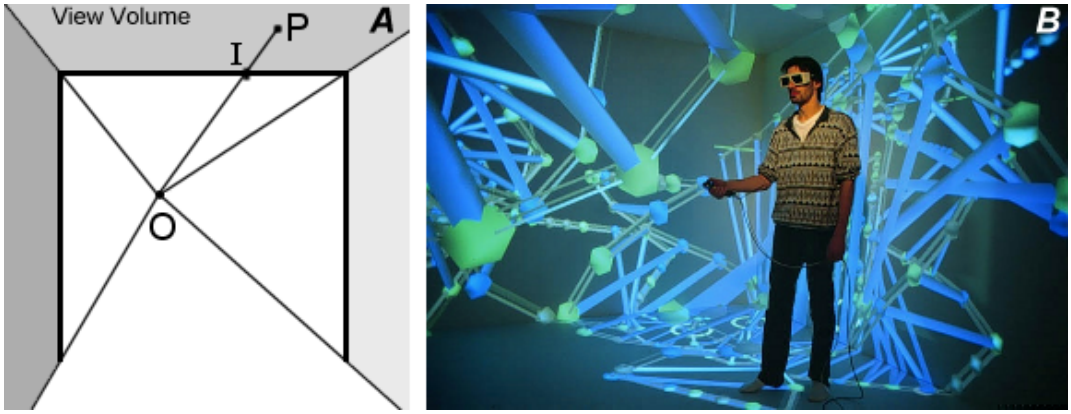

Figure 4: *(A)* Off-axis projection: Point **I** is the projection of the world coordinate **P** onto the view plane, with respect to the view origin **O**. The three separate grey regions are independent view volumes. *(B)* Inside the PDC VR-CUBE.

One benefit of using cubic approximation is that the renderer can be implemented with standard graphics libraries, such as OpenGL [11]. The difficulty with

most CAVE systems is the physical construction of the displays. Particular attention must be paid to edge matching and alignment of the display screens. Other common problems include geometrical distortions (caused by the projector) and the rendered frames not being synchronized for all displays [32]. Systems such as the RAVE [5] attempt to minimize edge matching and alignment related problems by using pre-built rear-projection televisions. Pre-built displays also offer the flexibility of constructing different viewing environments by rearranging the displays, rather than having a fixed cubic arrangement.

CAVE systems often show visible discontinuities in the panorama where the display edges are aligned. Such problems can be minimized by implementing a 360^o display in the shape of a cylindrical screen. These systems use three projectors to cover the entire display area. For each projector, the 3D objects are clipped and transformed onto 2D cylindrical coordinates using off-axis perspective projection. The resulting cylindrical segments are further transformed into a 2D picture coordinate system, which are later blended together during the rendering stage. An example is illustrated in Figure 5. However, due to the curved nature of the cylindrical screen, the projectors must be equipped with an infinite focus lens to minimize blurring in the image.

Kang's [34] image-based rendering system discussed in Section 2.2 was also implemented on cylindrical displays. In this case, the panoramic photographs were rendered on the cylindrical surface, and the user's viewpoint was fixed in the center of the cylinder.

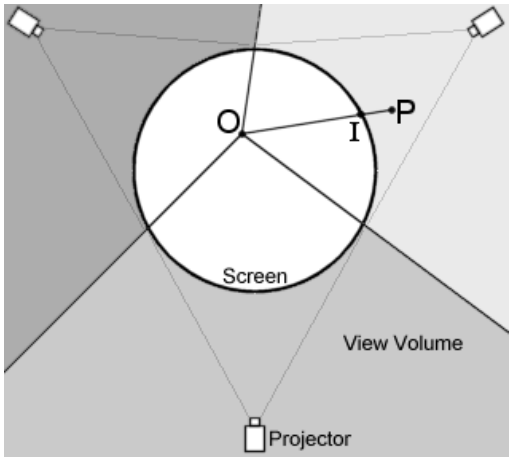

Figure 5: Projection onto a cylinder: **I** is the projection of the 3D point **P** onto the view cylinder, with respect to the eye position **O**. The three separate grey regions are independent view volumes.

Raskar [43] recognized some of the technical difficulties associated with the construction of multi-projector displays. Instead of using precise display configurations, Raskar's method allows the projectors to be causally aligned, and assumes that the display wall may be irregular in shape. With the aid of a stereo camera, the system first obtains stereo images of a test pattern for each projector. Each stereo image is

then used to evaluate the surface geometry using triangulation techniques, which are later combined into a common polygonal surface model. During the rendering stage, the scene is projected from the user's viewpoint onto the polygonal surface model. The texture mapped polygons are rendered from the projector's viewpoint and then displayed on the actual surface. Raskar extends his method [44] [45] by augmenting computer generated images onto the viewer's environment. This was achieved by using a head-tracking system, where a perspective correct view of the virtual objects are displayed on irregularly shaped real objects.

2.6 Single-Projector Panoramic Displays

In Section 2.5, some difficulties were identified with multiple projector systems. It is a challenging task to construct a multi-display system, because it requires constant maintenance to preserve the precise display configurations. A single-projector system is usually simpler to setup, because problems such as intensity matching, edge matching and alignment are no longer an issue.

Most single-projector systems consist of a dome screen, designed to cover the user's field of view, and a projector that displays a suitable panoramic picture. Since the panorama's field of view can reach 180^o, the projector is usually equipped with a "fisheye" or infinite focus lens. Before the panorama is rendered, the 3D objects in space are projected radially onto the dome to obtain the 2D polar coordinates [37]. These spherical coordinates are then re-projected onto 2D image coordinates, producing a suitable picture for the display. Since this transformation is non-linear, flat polygons must be subdivided to approximate the dome's curvature, otherwise the displayed objects will show geometrical distortions on the screen [56] [37]. The transformation also "compresses" the 2D image at the edge of the hemisphere, while the center is represented in a higher resolution [37]. Such distortions can cause problems for low-resolution projectors, because the final image will have poor texture detail at the edges.

In 1983, Max [37] proposed a 12m radius display system, based on a planetarium dome. Since the display was very large, unwanted distortions and aliasing effects present in some computer images would have been very obvious. To provide adequate image resolution for the display, the graphics were pre-rendered for the special Omnimax [3] film format.

The limitation of the Omnimax-based system is that the panorama is pre-rendered, making the display environment non-interactive. The VisionDome [56] and the VisionStation [7] system (Figure 6) overcomes such limitations by using graphics accelerators to implement a realtime rendering system. The OpenGL renderer, modified by ARC [8], allows polygon subdivision, and provides the 180^o field of view required for the display.

Dome-based displays limit the panorama's field of view to 180^o. The Panoscope 360 system [19] overcomes this limitation by using a different approach in displaying the panoramic image. In this case, the display surrounded the viewer with a 360^o panorama. The viewer's head was situated inside a large spherical mirror, through an opening. The top of the hemisphere was covered with an opaque canvas, where a 2D spherical panorama was projected from above (see Figure 7). The panorama was specifically distorted to approximate the curvature of the reflector. The spherical

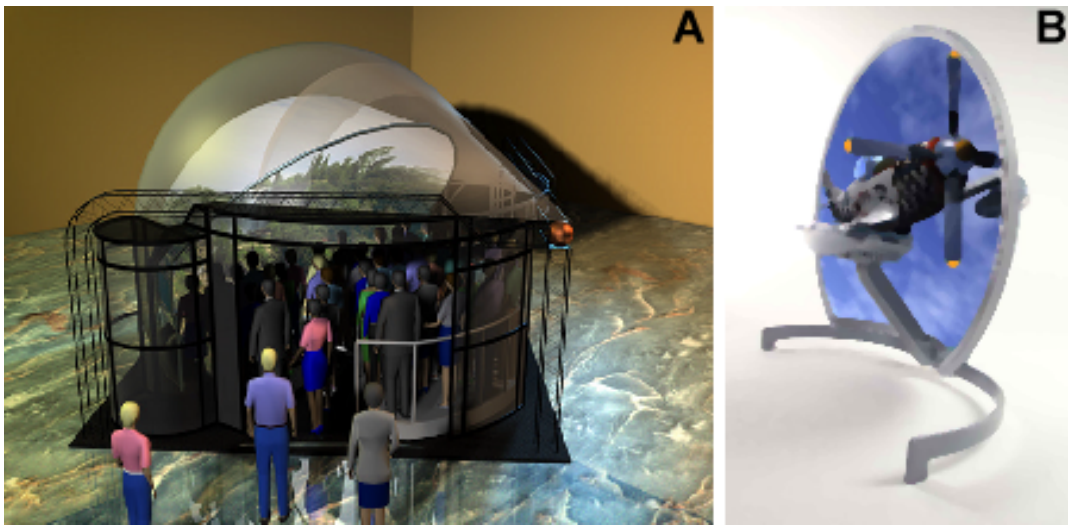

Figure 6: *(A)* The VisionDome. *(B)* The VisionStaion.

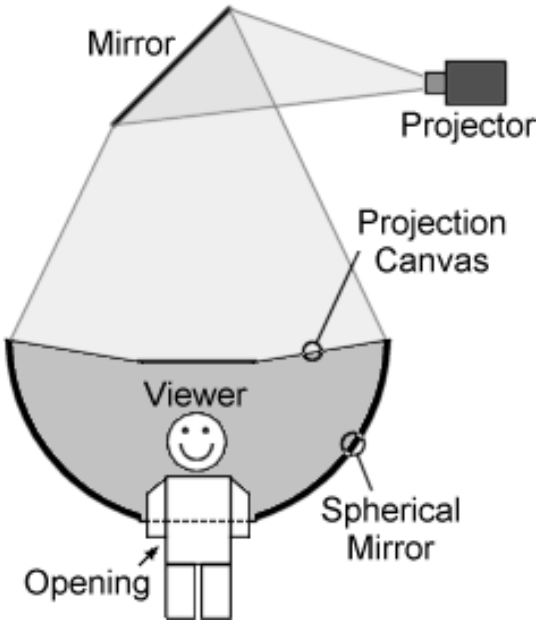

Figure 7: Panoscope 360^o (cross sectional view).

reflector corrects the distortion in the panorama, and the reflection seen by the viewer becomes perspective correct.

The greatest benefit with single-projector systems is the simplicity of the display construction. The drawbacks with such displays include the need for preprocessing of polygons (eg. subdivision) to minimize *geometrical distortions*. Also, projecting a 2D image onto the display causes additional problems, because pixels of fixed size are mapped onto a curved surface. Therefore, at the steepest regions of the screen, the projected pixels will be stretched over a greater display area, which results in low texture detail at the edges.

3 Panoramic Rendering

3.1 Representing Projection Surfaces with Profile Curves

This section will introduce the concept of panoramic rendering for concave surfaces of revolution. The relationship between 3D points and 3D surfaces of revolution, and the significance of the "profile curve" will be described. The discussions presented here are intended to be an introductory topic for the subsequent Sections 3.2 and 3.3, which deal with the ray-tracer and the realtime implementation of the panoramic renderer respectively.

3.1.1 The Profile Curve

A surface of revolution can be constructed by revolving a 2D curve around a line, the *principal axis*. The geometric shape of the symmetrical surface is governed by the 2D function, the *profile curve*. Since most curved displays are symmetrical about its principal axis, 2D profile curves provide a convenient way for modelling a display's shape. Therefore Sections 3.2 and 3.3 will only focus on surfaces of revolution (also referred as *projection surfaces*, or simply surfaces).

When projecting a point onto a surface or when determining the direction of an incident ray, the surface's symmetrical property becomes an advantage. It's symmetry allows the transformations to be done in the 2D *profile curve space*, rather than directly with the 3D surface itself. To illustrate this point, the projection surface in Figure 8a can be represented with an infinite number of 2D profile curve "slices" revolving around the Z axis. This means that any point **P** in 3D space will lie in the plane of a slice, the profile curve space. As illustrated in Figures 8b and 8c, the point's X and Y displacement in the 3D space is represented by the radius **R** in the profile curve space; and the point's Z displacement remains unchanged for both coordinate systems:

$$\begin{aligned} r &= \sqrt{x_p^2 + y_p^2}, \\ z &= z_p. \end{aligned} \tag{2}$$

The profile curve resides in the profile curve space and it can be defined as a function of r:

$$z = f(r) \quad \text{for } 0 \leq r \leq r_{max}. \tag{3}$$

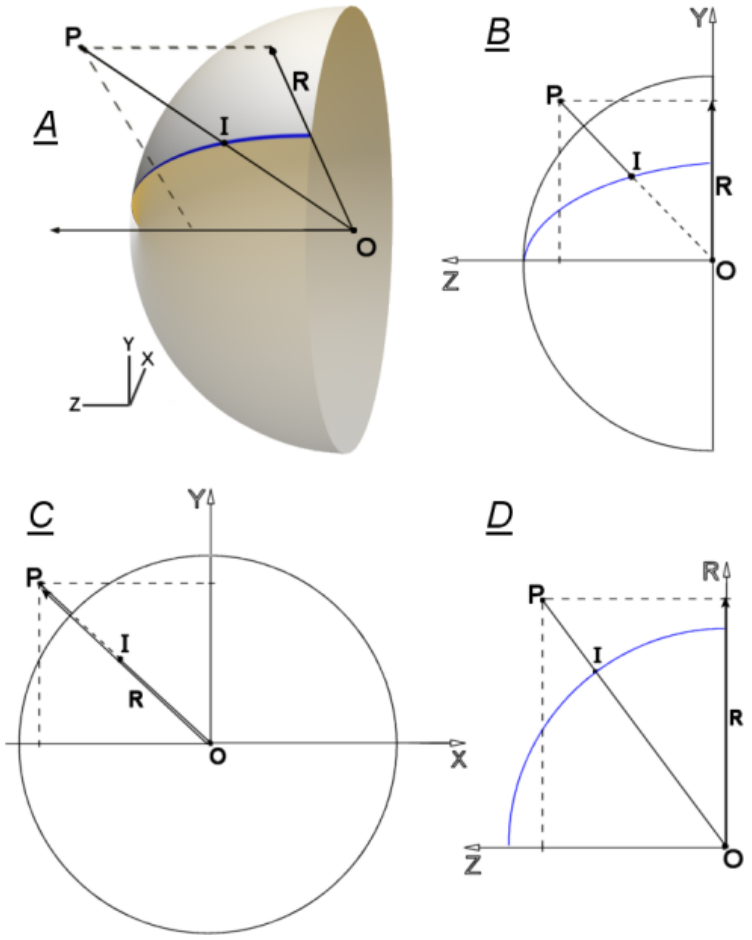

Figure 8: The relationship between the 2D profile curve and a 3D point **P**. *(A)* 3D view. *(B)* Orthographic view in the Y and Z plane, and *(C)* in the X and Y plane. *(D)* The point **P** in the 2D profile curve space.

r_{max} is the upper limit for the function and its value will depend on the following: When specifying the interval [0, r_{max}], it is assumed that the function $f(r)$ is *continuous*, that is $f(r)$ is defined for all r in that range [50]. However, defining r_{max} simply on the notion of continuity is not sufficient. The value for r_{max} should be selected carefully, because it also affects the panoramic *field of view* (FOV) of the rendering system. Details on specifying r_{max} will be explained in Section 3.1.2. From this point onwards, when $f(r)$ is defined or referred to, only the curve between the interval of [0, r_{max}] will be of interest.

$f(r)$ must also pass the *vertical line test*. The curve is classified as a function, only if no vertical line crosses the curve more than once. This is an important property, because when projecting the panorama from the surface onto a *2D image plane* (see Section 3.1.2), each pixel in the image plane cannot be assigned with more than one color.

It is assumed that the rendering system will deal with arbitrary profile curve functions. Therefore it is essential to represent $f(r)$ in a dynamic manner. The

software implementation accepts a function string using the infix notation from the user. The software converts this function string to the Reverse Polish notation. The resulting postfix representation will consist of a sequence of custom operation codes. Each code is used as an index for a custom jump table, where the CPU jumps and executes the corresponding arithmetic operation. This code sequence is executed every time $f(r)$ is called or referred to.

As a final note, the graph in Figure 8d shows point **P** lying in the profile curve space. The view origin **O** is located at [0, 0, 0] in the viewing coordinate system, while the Z-axis specifies forward direction. The projection of **P** on the surface is indicated by **I**. This occurs when the view vector running from from **O** to **P** intersects the curve $f(r)$. The details on computing the intersection point **I** is discussed in Section 3.3.1.

3.1.2 The Screen Space and the Projection Surface

When rendering a panoramic image, the resulting picture is eventually stored in a 2D image space (or screen space). To be more precise, the panoramic transformation of a 3D object is considered as a two-step process: Initially the object is projected onto a surface of revolution, then it is re-projected onto a 2D image plane. Therefore it is necessary to discuss the relationship between the actual projection surface and the 2D image space. In addition, if the panorama is to be displayed on a physical curved display, the projector's viewpoint must be also taken into consideration (see Section 3.4). For the moment, the projector's viewpoint will be ignored and only the surface - image plane relationship will be discussed.

The 2D image plane is assumed to be coincident with the XY-plane of the viewing coordinate system. The view origin **O** and the 3D surface are centered on the image plane, where the principal axis runs parallel with the plane's normal. This arrangement is intended to create an *orthographic* projection of the surface on the plane. The image plane's area is defined by its width [x_{left}, x_{right}] and its *fixed* height [y_{top}, y_{bottom}] (the units also correspond with the viewing coordinate system):

$$\begin{aligned} y_{top} &= 1, & y_{bottom} &= -1; \\ x_{left} &= -X_{res}/Y_{res}, & x_{right} &= X_{res}/Y_{res}. \end{aligned} \tag{4}$$

The plane's width varies according to the aspect ratio of the screen's resolution (in pixels), X_{res} and Y_{res}. Therefore, in some cases the plane's area may be too small to accommodate the entire panoramic image, because the panorama is *clipped* by the image plane's edges during the orthographic projection. As a result the panoramic FOV is not only dependent on r_{max}, but also on the image plane's area. The following discussion assumes that the image plane has an aspect ratio of 4:3, because most video projectors use this screen format.

The clipping of a panorama will largely depend on the profile curve configuration. The examples in Figures 9a to 9c show three different projection surfaces. No clipping will occur in Figure 9a, because the function is discontinuous when $r > 1$. Therefore r_{max} is set to 1, which coincides with the screen's top and bottom edges, y_{top} and y_{bottom}.

However, $f(r)$ can define a curve that is continuous in [0, $r_{max} = \infty$], as shown in Figure 9b, creating a FOV that is greater than 180^o. Infinity is not a practical

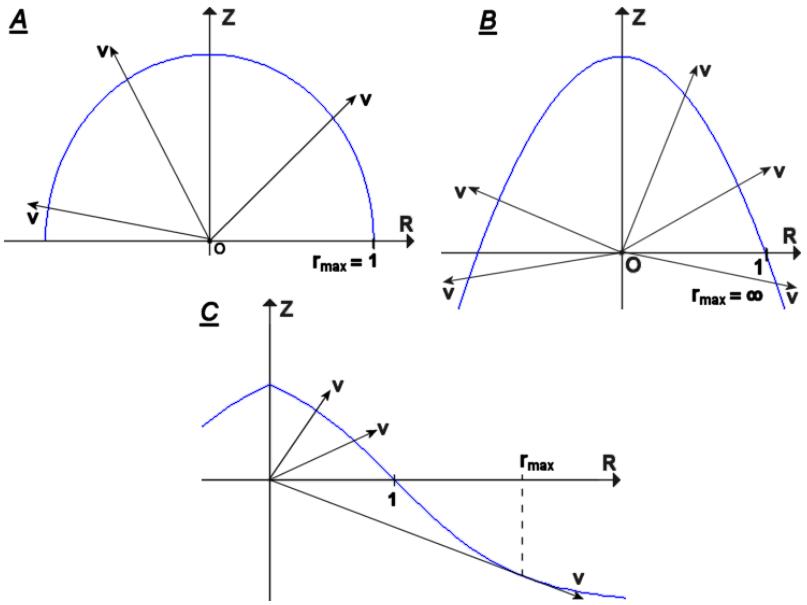

Figure 9: Cross-sectional view of a surface defined by *(A)* $f(r) = \sqrt{1 - r^2}$, *(B)* $f(r) = -r^2 + 1$ and *(C)* $f(r) = -tanh(r - 1)$. The possible directions for the view vector $\mathbf{V}$ is also illustrated.

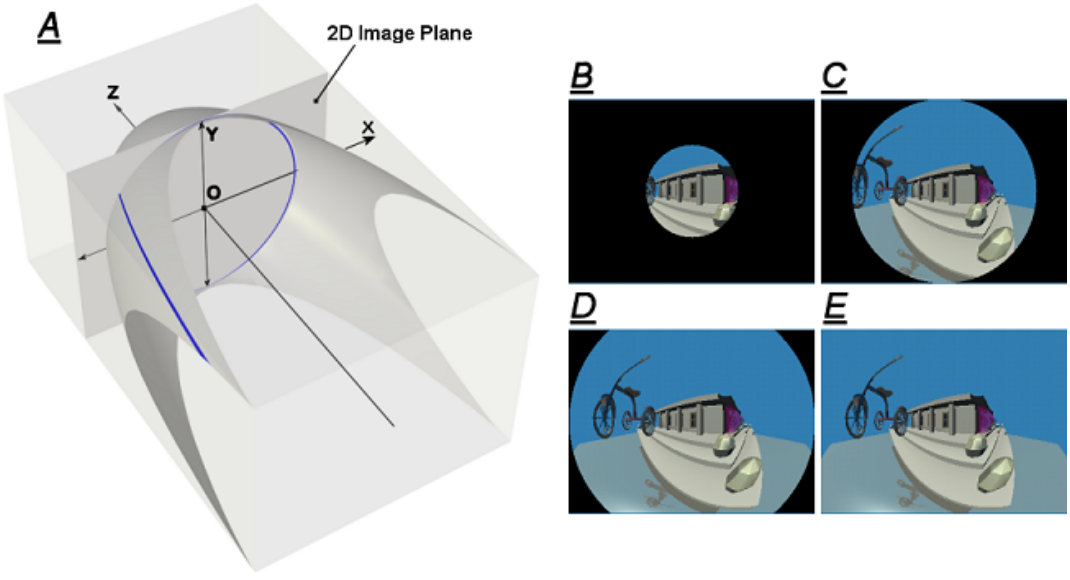

Figure 10: *(A)* An image plane with a 4:3 aspect ratio is clipping a parabolic projection surface. The four images *(B - E)* show a parabolic panorama (on a 4:3 image plane) with different values for r_{max}: *(B)* 0.5, *(C)* 1.0, *(D)* 1.333 and *(E)* 1.666 $= \sqrt{1.333^2 + 1^2}$.

value for r_{max}, because the screen's edges will clip the projection surface, and the panorama's FOV will be limited (see Figure 10a). Therefore it is more practical to set r_{max} to a value that coincides with the image plane's maximum radius, r_{screen}. With this approach, any object that is potentially projected outside r_{max} is marked as invisible and can be discarded. To fully utilize the screen's area (as in Figure 10e), r_{screen} can be initialized with the image plane's diagonal radius:

$$\begin{aligned} x_d &= \tfrac{1}{2}(x_{right} - x_{left}); \\ y_d &= \tfrac{1}{2}(y_{top} - y_{bottom}); \\ r_{screen} &= \sqrt{x_d^2 + y_d^2}. \end{aligned} \tag{5}$$

However, it is also sufficient to initialize r_{screen} with either x_d or y_d (see Figures 10c and 10d). As a final test, if $f(r)$ is continuous in $[0,\ r_{screen}]$, then $r_{max} = r_{screen}$, otherwise r_{max} is initialized with a value that marks the end of the function's continuity.

The function in Figure 9c shows a special case where some potential view vectors may not cross the profile curve, even though $f(r)$ is continuous in $[0,\ \infty]$. The bottom view vector $\mathbf{V}$ in Figure 9c just grazes the profile curve. Any potential vectors below it will completely miss the curve, and therefore no projection is possible. In this case r_{max} should be positioned where the curve's tangent is coincident with the view vector.

The final panoramic picture is represented as a *raster image* with a fixed resolution. The orthographic transformation described earlier, directly projects the 3D surface onto raster image plane. Unfortunately this process does not provide a uniform sampling of the curved surface. There may be regions on the surface where gradient is high relative to the image plane. These regions will be represented with less detail in the raster image, as shown in Figure 11a.

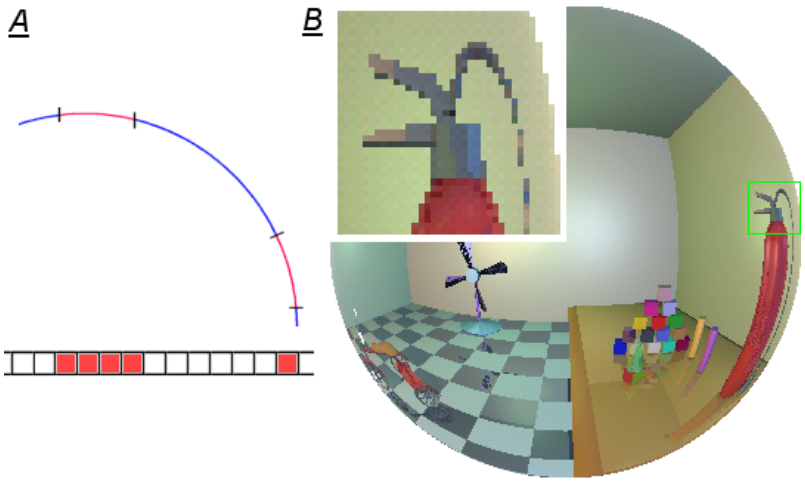

Figure 11: *(A)* Two segments of the profile curve in equal length (red) are projected onto the image plane below. *(B)* The fire extinguisher's hose is under sampled and suffers from aliasing.

The two curve segments in the illustration (shown red), are mapped on the image plane below, which has a fixed pixel resolution. Both segments are equal in length. However, one segment is represented with four pixels on the image plane, while the other is under sampled. In the latter case, more information is compressed into a small raster image area.

As shown in Figure 11b, aliasing effects are also more prevalent in the under sampled regions of the panorama. Objects with fine detail are not sampled adequately and they partially disappear (see the fire extinguisher's hose). A method for minimize such effects, is detailed in Section 3.2.

3.2 Using Profile Curves for Ray-tracing

Section 3.1 introduced some of the concepts behind the profile curve, the projection surface and the image plane. In this section, the theory will be used to describe a panoramic renderer using the ray-tracer [30] implementation. Issues such as how to fire the initial rays into the scene, and how to relate between the *raster* coordinate system and the image plane coordinate system will be described. Additionally, a method for minimizing aliasing in the final image is also detailed.

The raster coordinate system simply defines how the individual pixels are accessed in the image. The origin in the image is located at the top left corner. The relationship between a raster coordinate $[X_s, Y_s]$ and an image plane coordinate $[x_v, y_v]$ is given by:

$$\begin{aligned} x_v &= (X_{sc} - X_s)/X_{sc}, \\ y_v &= (Y_s - Y_{sc})/Y_{sc}, \end{aligned} \tag{6}$$

where $[X_{sc}, Y_{sc}]$ correspond to the image center in raster coordinates. The coordinate given by $[x_v, y_v]$ is a normalized offset from the origin $\mathbf{O}$, within the area specified by Equation 4. It also represents the direction of the initial ray vector (or the view vector) in the XY-plane of the viewing coordinate system. The ray's third direction, z_v, will be evaluated by using x_v, y_v and the profile curve $f(r)$.

Point $\mathbf{I}$ corresponds to the intersection point of the view vector $\widehat{\mathbf{V}}$ with the profile curve. As mentioned earlier, the surface to image plane projection is orthographic. Therefore $[x_v, y_v]$ is a direct projection of the point $\mathbf{I}$. The radius r_v of the offset $[x_v, y_v]$ is used to evaluate the ray's Z-direction, z_v:

$$\begin{aligned} r_v &= \sqrt{x_v^2 + y_v^2}, \\ z_v &= f(r_v), \qquad \text{for } r_v \leq r_{max}. \end{aligned} \tag{7}$$

The view vector $\widehat{\mathbf{V}}$ is the unit vector of $[x_v, y_v, z_v]$. It is important to note that the ray-tracer discards any incident rays whose radius r_v lies outside r_{max}. In such situations $f(r_v)$ and $\widehat{\mathbf{V}}$ are not evaluated and the corresponding pixel at $[X_s, Y_s]$ is set to the default background color.

Section 3.1.2 introduced the problems relating to the non-linear sampling of the 3D surface. The remaining paragraphs in this Section will discuss an anti-aliasing

feature for the ray-tracer. Anti-aliasing does not solve the nonlinear sampling problem nor the panoramic distortion problem. The intention is to minimize defects such as the staircasing effects along the edges and the partially disappearing objects (see Figure 11b).

Anti-aliasing is a widely researched area, and several techniques have been developed, see details in [27] [35] [14] [57]. The panoramic renderer uses the *stochastic sampling* technique, where irregular sampling is applied for each pixel. The idea behind irregular sampling is to represent high frequency image details with featureless noise, which is less perceptible than the defects caused by aliasing [24].

The ray-tracer uses an adaptive anti-aliasing feature that only enables the algorithm when a contrasting feature in the panorama was detected. A contrasting feature may include object boundaries, or a sudden transition in brightness. Defects such as the staircasing effect are more obvious at the object boundaries, because the human visual system is most sensitive to the luminance (or the brightness) component in the image. Therefore, multi-sampling should be applied where the contrasting features occur. The luminance is derived from the weighted average of a `RGB` color. The weighting factors were adopted from the PAL and NTSC standards [17] [36].

$$\mathtt{Y} = 0.2989\mathtt{R} + 0.5866\mathtt{G} + 0.1144\mathtt{B}. \tag{8}$$

The ray-tracer keeps track of the luminance computed at the previous pixel location $\mathtt{Y}_{i-1}$ and at the pixel location immediately above (in the previous scan line), $\mathtt{Y}_{i-s}$. The luminance for the current pixel $\mathtt{Y}_i$ is also computed. If the absolute difference between $\mathtt{Y}_i$ and $\mathtt{Y}_{i-1}$, or $\mathtt{Y}_i$ and $\mathtt{Y}_{i-s}$ is above a tolerance value, the anti-aliasing is enabled. This method allows the ray-tracer to selectively multi-sample contrasting features, and dedicate less computing time for the smooth features.

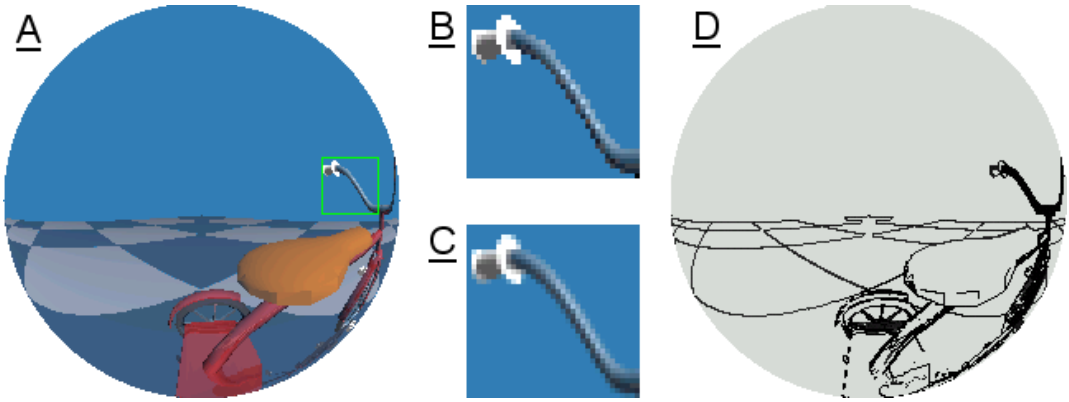

Figure 12: Anti-aliasing: *(A)* Original spherical panorama. *(B)* Original enlarged image area. *(C)* Image area with anti-aliasing. *(D)* Hot-spot image, highlighting the luminance boundaries.

An adaptive anti-aliasing example is shown in Figure 12. Picture 12a displays a spherical panorama, where a small square region is magnified and illustrated in Figure 12b. After applying the adaptive anti-aliasing algorithm to the entire panorama, the magnified square region in Figure 12c shows less signs of rough edges. The hot-spot image in Figure 12d highlights the areas where the ray-tracer has spent the

most time in rendering. The darker regions indicate more processing time, which is typical at the luminance boundaries and at the object edges.

3.3 Techniques for Realtime Rendering

The realtime implementation of the panoramic renderer uses OpenGL [11] for displaying the transformed primitives. This Section will cover topics related to panoramic transformation techniques only, and will not discuss OpenGL specific setup or interface routines. It is assumed that the OpenGL rendering subsystem was initialized with an orthographic viewport, with Z-buffering enabled.

Rendering panoramic images in realtime is more complicated than in the ray-tracer implementation. Unlike in the ray-tracer, the object transformation process is reversed. In this case the 3D geometry is projected onto the surface and then onto the image plane. The renderer faces the problem of how to execute such projections in realtime, given a set of arbitrary points and an arbitrary profile curve. This section will describe a method for such transformations in two parts: The first section will detail the projection of points in general, and the second will describe how such projections might be executed in realtime.

3.3.1 Projecting Points onto a Surface of Revolution

The problem of projecting a point onto the surface of revolution (see Figure 8), was stated at the end of Section 3.1.1: Given a view vector $\mathbf{V}$, running from the origin $\mathbf{O}$ to an arbitrary point $\mathbf{P}$ in space, how is the vector's intersection point $\mathbf{I}$ determined with an arbitrary profile curve? By examining Figure 8d in more detail, the view vector in the profile curve space can be simply expressed as a linear function of r:

$$\begin{aligned} z &= m \cdot r, \quad \text{whose gradient } m \text{ is given by} \\ m &= \frac{z_v}{r_v}, \end{aligned} \tag{9}$$

where z_v, and r_v is the Z and radial displacement of $\mathbf{P}$ from $\mathbf{O}$, (see Equation 2). In general, the intersection $\mathbf{I}$ occurs when the profile curve $f(r)$ meets the line $m \cdot r$:

$$\begin{aligned} m \cdot r &= f(r), \\ 0 &= f(r) - m \cdot r. \end{aligned} \tag{10}$$

The intersection coordinate $[z_i,\ r_i]$ is computed by solving for the root in Equation 10.

Solving for Equation 10 is difficult, because the renderer deals with arbitrary profile curves, and $f(r)$ has no explicit representation within the program. Therefore, it cannot use analytical methods for calculating the root directly. Besides, functions with higher order are difficult to solve analytically, and there is no known method for solving transcendental functions [50]. However, there are numerical methods for finding roots [21], such as the Direct Search Method, the Newton-Raphson Method, the Secant Method and the Bisection Method [10].

In this project the Bisection Method was adopted, because of its simplicity, accuracy and low computational cost (it only calls $f(r)$ once per iteration). The Bisection

Method iteratively searches for the root r_i between the search range r_1 and r_2. When applying this algorithm on profile curves, the initial search range becomes $r_1 = 0$ and $r_2 = r_{max}$. In each iteration, the Bisection Method computes the mid value r_{mid}, which is substituted into Equation 10 to obtain z_{mid}. The search range narrows, depending on the sign of z_{mid}:

$$
\begin{aligned}
& r_{mid} = \tfrac{1}{2}(r_1 + r_2); \\
& z_{mid} = f(r_{mid}) - m \cdot r_{mid}; \\
& \\
& \texttt{if } (z_{mid} > 0)\texttt{, then } r_1 = r_{mid}\texttt{; else } r_2 = r_{mid}\texttt{;} \\
& \texttt{if } (|r_1 - r_2| < \epsilon)\texttt{, then } r_i = r_{mid}\texttt{, exit iteration loop.}
\end{aligned}
\tag{11}
$$

After several iterations, the range $[r_1, r_2]$ will eventually converge to r_i, and the loop is terminated when the absolute difference between r_1 and r_2 falls below a chosen tolerance ϵ. However, there is a limitation with the Bisection Method. The algorithm assumes that there is no more than one root between r_1 and r_2. Therefore when defining $f(r)$, it is important to make sure that no view vectors cross $f(r)$ more than once in the $[0, r_{max}]$ interval.

Once the root r_i is found, the Z component of the intersection point $\mathbf{I}$ is calculated by solving for $f(r_i)$. r_i is equally useful for finding the 3D intersection point with the 3D surface:

$$
\begin{aligned}
& t = \frac{r_i}{r_v}, && \text{the scaling value } t, \\
& x_i = x_v \cdot t, \\
& y_i = y_v \cdot t, \\
& z_i = f(r_i), \\
& \text{therefore } \mathbf{I} = [x_i,\, y_i,\, z_i].
\end{aligned}
\tag{12}
$$

Equation 12 essentially scales the X and Y components of the view vector (or more precisely the X and Y components of $\mathbf{P}$) with the ratio of r_i and r_v. Note that the surface to image plane projection is orthographic, and there is no need for finding z_i. The value for z_i is only useful when the $[x_i, y_i]$ components need compensation for the projector's viewpoint (see details in Section 3.4). Otherwise, x_i and y_i are expressed as $\mathbf{I}'$, the direct orthographic projection of $\mathbf{I}$ on the image plane:

$$
\begin{aligned}
& x'_i = x_i, \\
& y'_i = y_i, \\
& z'_i = \sqrt{x_v^2 + y_v^2 + z_v^2} && \text{(see below for details)}, \\
& \text{therefore } \mathbf{I}' = [x'_i,\, y'_i,\, z'_i].
\end{aligned}
\tag{13}
$$

As mentioned earlier, the OpenGL rendering system was used for displaying all the transformed objects. OpenGL implements a Z-buffer for removing hidden surfaces, and therefore it is necessary to keep the transformed primitives "sorted" on the display. When passing $\mathbf{I}'$ to the rendering subsystem, it is essential to specify a *depth value* as well. This depth value ensures that the transformed objects are

occluded in the right order. It would be incorrect to use the value z_i from Equation 12 for Z-buffering, because all the projected points $\mathbf{I}$ lie *on* the same curved surface, and therefore z_i does not distinguish which object is situated further away from the viewer. However, the magnitude of the vector $\mathbf{V}$ (running from $\mathbf{O}$ to $\mathbf{P}$) can be treated as the depth value (see Equation 13). Therefore, the depth parameter in $\mathbf{I}'$ will be given a larger magnitude for points that are further away from the viewer.

Apart from point transformations, the realtime renderer faces an additional problem of dealing with partially visible objects. For example, if some points in the polygon lie outside the profile curve's clipping region (that is outside r_{max}), how should the renderer deal with such points? This is a major problem, because the curve outside r_{max} is possibly undefined, and therefore no projection is possible beyond r_{max}.

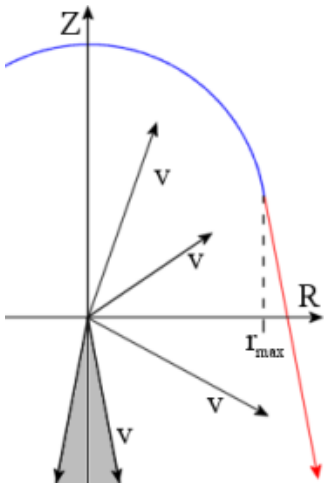

Figure 13: Using a tangent line (red) to extend the profile curve (blue). The potential view vectors $\mathbf{V}$ are also illustrated.

A simple method that gets around this problem uses a tangent line to extend the profile curve beyond r_{max}. Figure 13 shows how the tangent line (in red) begins from r_{max} and runs to infinity. This arrangement allows the transformation of partially visible objects onto the tangent line. Note that if all the points in the object fall on the tangent line (that is outside r_{max}), the object is discarded and not passed to the rendering subsystem. The grey area in Figure 13 indicates the *blind spot*, a region where no projection is possible. This blind spot is present because some potential view vectors $\mathbf{V}$ either run parallel or do not cross the tangent line. In this case, any point that falls into the grey region is assigned with an extreme value, which places it outside r_{max} (therefore making the point invisible).

The tangent line is determined just before $f(r)$ terminates at r_{max}. Its gradient m_T is given by:

$$m_T = \frac{f(r_{max} - \delta) - f(r_{max})}{-\delta}, \tag{14}$$

where δ is a small R-displacement. The tangent line is expressed as

$$z = m_T(r - r_{max}) + z_{max}, \tag{15}$$

where $z_{max} = f(r_{max})$. By expressing a view vector in the form of Equation 9, its R-intercept r_i with tangent line is obtained as follows:

$$\begin{aligned} z &= m \cdot r, && \text{the view vector from Equation 9,} \\ m \cdot r &= m_T(r - r_{max}) + z_{max}, && \text{which simplifies to...} \\ r = r_i &= \frac{m_T \cdot r_{max} - z_{max}}{m_T - m}. \end{aligned} \tag{16}$$

The view vector is in the blind spot when $(m_T - m) \geq 0$, which in that case r_i is given an extreme value, placing it well outside r_{max}. The 3D intersection point $\mathbf{I}$ for the view vector $\mathbf{V}$ and the tangent plane is found by substituting r_i into Equation 12. After applying Equation 13 to the point $\mathbf{I}$, the partially visible objects are simply passed to the rendering pipeline, and possibly clipped by the 2D image plane during the orthographic transformation.

3.3.2 Implementing Realtime Transformations

One drawback with the transformation methods detailed in Section 3.3.1 is that they are computationally expensive. The main problem lies with finding the R-intercept r_i for the profile curve and a view vector. The Bisection Method iteratively determines the root for each intersection. When processing thousands of points, the time required for processing the entire database can become very significant. Therefore, this Section will introduce a simple speedup technique based on lookup tables.

The main priority is to eliminate all the function calls to the Bisection algorithm and to the profile curve execution unit during point transformations. Such calls are removed by evaluating a number of view vectors $\mathbf{V}$, and storing the corresponding intersection points in a lookup table. This is where the projection surface's symmetrical property becomes a real advantage: The lookup table needs be implemented in the 2D profile curve space only.

As shown in Figure 14, the view vector is rotated 180^o in small increments, $\Delta\Theta$. The angle Θ at each increment (measured relative to the R-axis) is converted and used as a lookup table index (see Equation 17). The profile curve or the tangent intersection point $\mathbf{I}$ is calculated for every view vector (as in Section 3.3.1). Only the r_i and z_i components of $\mathbf{I}$ is stored in the table.

$$\texttt{TabIdx} = round\left(\tfrac{90^o - \Theta}{\Delta\Theta}\right). \tag{17}$$

When transforming the actual 3D point $\mathbf{P}$, the vector $\mathbf{V}$ running from $\mathbf{O}$ to $\mathbf{P}$ is determined. This vector is then expressed in the form of Equation 9. By using right angle trigonometry, the gradient m (given by Equation 9) is used to compute the angle of $\mathbf{V}$, relative to the R-axis:

$$\Theta = tan^{-1}(m). \tag{18}$$

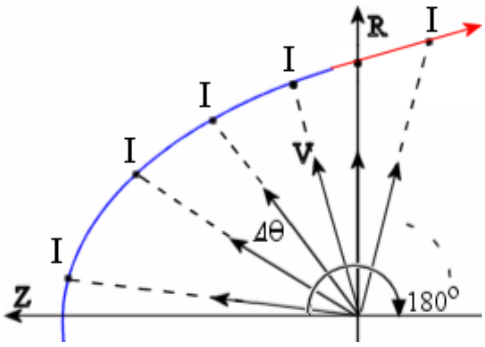

Figure 14: The view vector **V** is rotated by small angle increments $\Delta\Theta$, and the corresponding intersection points **I** with $f(r)$ (blue) or the tangent line (red) are evaluated.

The angle is then converted to the corresponding lookup table index, `TabIdx` (see Equation 17). Equations 12 and 13 will use the r_i and z_i entries from the table to find the intersection point **I**, and its orthographic projection $\mathbf{I}'$ respectively.

Equations 18 and 17 will only index the nearest r_i and z_i values for a given view vector gradient. Therefore, the size of the angle increment $\Delta\Theta$ will have a dominant role in the accuracy of the point transformations. If the resolution is too low, the 3D points will be snapped to the nearest 3D projection point **I**. This effect is very noticeable in animated scenes, because moving objects pop or jitter from one place to an other. Also, small geometric objects will not be represented adequately and they will suffer from geometrical distortions.

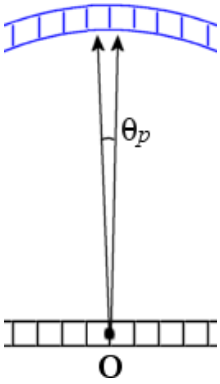

Figure 15: Θ_p, the smallest angle between two orthographically projected pixels on the profile curve. The image plane is illustrated underneath.

The smallest possible movement for an animated object is usually governed by the pixel resolution of the raster image. In order to minimize the popping or jittering of moving objects, the ideal angle increment $\Delta\Theta$ should correspond to Θ_p, the smallest angle between two orthographically projected pixels on the curved surface (see Figure 15). However, the value for Θ_p may be very small for high resolution raster images, and would result with a very large lookup table. A number of subjective trials reveal

that it is sufficient to use a fixed value for $\Delta\Theta$. Typically, this value should be adjusted for a given raster resolution, until the jittering effect of moving objects is no longer visible. For example, one-twentieth of a degree was found to be sufficient for a raster resolution of 1024×768 pixels.

A number of trials with different scene complexities reveal that the lookup table method gives up to 15 fold speed improvement over the direct transformation methods. Details on these trials are discussed in Sections 5.2 and 4.2.

3.3.3 Polygon Subdivision

When projecting straight-edged geometrical primitives onto a curved surface, a number of problems can arise. Polygon edges can't be mapped directly onto the surface without seeing *geometrical distortions* in the panorama, because the panoramic transformations are not affine [46]. This section will detail a polygon subdivision technique that attempts to minimize such geometrical distortions.

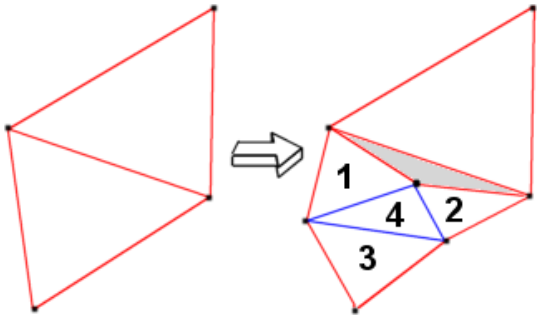

Figure 16: One of the two base polygons (left) undergoes isotropic subdivision (right). Note the cracking formed at the shared edge (right).

Polygon subdivision introduces intermediate points along the edges or within the primitive itself. These points are also projected onto the surface, thus giving a better approximation for the entire polygon. The extra points will increase the polygon count passed to the rendering pipeline, which directly affects the rendering performance. Therefore it is necessary to select and subdivide only those polygons that create the greatest geometrical distortion.

The renderer implements a recursive *isotropic* subdivision algorithm, where the triangular base-polygon's edges are bisected and form four sub-polygons [31] [16] (see Figure 16). The sub-polygons are numbered 1 to 4 to aid the description of the algorithm below:

```
//---- The subdivision routine ----
DivideRecurse(BasePoints[3], LOD)
    begin
        if (LOD = 0) then           //Check if the current recursion depth is 0
            begin
                //If DivideRecurse() was called for the first time, then apply
                // the panoramic transformation for all vertices in BasePoints[ ].
            end
```

```
    else begin
            //Apply the panoramic transformation for those vertices in
            // BasePoints[ ] that weren't transformed in the previous recursion.
        end;

    if (all vertices in BasePoints[ ] are invisible) then return;

    //-- Initiate subdivision if the magnitude for the longest edge is above the threshold --
    if (Magnitude_LongestEdge(BasePoints) > THRESHOLD) and (LOD < Max_LOD) then
        begin
            //Bisect each edge in the triangle and save the midpoints
            MidPoints[3] := BisectEdges(BasePoints);

            NewPoints[3] := Form sub-triangle 1 from MidPoints[ ] and BasePoints[ ];
            DivideRecurse(NewPoints, LOD+1);

            NewPoints[3] := Form sub-triangle 2 from MidPoints[ ] and BasePoints[ ];
            DivideRecurse(NewPoints, LOD+1);

            NewPoints[3] := Form sub-triangle 3 from MidPoints[ ] and BasePoints[ ];
            DivideRecurse(NewPoints, LOD+1);

            NewPoints[3] := Form sub-triangle 4 from MidPoints[ ];
            DivideRecurse(NewPoints, LOD+1);
        end

    //-- Render the polygon --
    else begin
            RenderPolygon(BasePoints);
        end;
end;
```

The recursive function `DivideRecurse()` accepts an array of points that form the base triangle. If `DivideRecurse()` was called for the first time, all the points in the array will undergo the panoramic transformation. Otherwise, only the new vertices generated in the previous recursion will be transformed. If all the transformed points in the array are invisible (outside the profile curve's clipping radius r_{max}), then `DivideRecurse()` will exit.

The `Magnitude_LongestEdge()` function will return the magnitude for the longest edge in the triangle. If this magnitude is above the `THRESHOLD` value, the subdivision is enabled. `THRESHOLD` represents the length of longest straight edge allowed in screen space. This parameter is supplied by the user, and it can be adjusted to allow finer or coarser edge divisions.

If the division is activated, the midpoint for each edge is calculated, and the four sub polygons will be formed, as shown in Figure 16. Each sub polygon is called with the `DivideRecurse()` function. The recursive cycle repeats until the magnitude of the subdivision edges fall below the `THRESHOLD`, or until the algorithm's recursive depth reaches a maximum *level of detail*, `Max_LOD`. This prevents the rendering pipeline from being overloaded with sub-polygons. When the recursion stops, the triangle is drawn.

In the worst case scenario, this algorithm will generate all the possible sub polygons for all recursion levels (up to `Max_LOD`). The worst case complexity for a given recursion level L is given by

$$\begin{aligned} O_t(L) &= 4^L; \\ O_p(L) &= \frac{(\sqrt{4^L}+1)(\sqrt{4^L}+2)}{2}, \quad \text{for } 0 \le L < \texttt{Max\_LOD}, \end{aligned} \tag{19}$$

where $O_t(L)$ is the maximum number of triangles generated, and $O_p(L)$ is the maximum number of points transformed. This complexity only occurs when the magnitude for all the subdivision edges (in all recursion levels) are above the `THRESHOLD` value. If a scene consists of N polygons, the maximum number of points transformed (in the worst case) is given by:

$$O_p(L) = N\frac{(\sqrt{4^L}+1)(\sqrt{4^L}+2)}{2}, \quad \text{for } 0 \leq L < \texttt{Max\_LOD}. \tag{20}$$

When no subdivision occurs ($L = 0$), the best case is given by:

$$O_p(L) = 3N. \tag{21}$$

Unfortunately, subdivision introduces the *cracking* problem as shown in Figure 16. This occurs when one polygon receives a greater level of detail than its neighbouring polygon. The intermediate points drift away from the shared edge, because they provide a better approximation of the projection surface. Unfortunately, due to lack of time, the cracking problem was not addressed properly (details in Section 5.2), and therefore this topic is left open for possible future work.

3.4 Compensating for the Projector

So far only the orthographic surface/image plane relationship was discussed in detail (Section 3.1.2), and projector's viewpoint was largely ignored. The orthographic transformation is adequate for viewing a panorama on a conventional 2D monitor. However, if the panorama is displayed on a curved display, the direct orthographic transformation method is insufficient. In this case, the projector's viewpoint must be taken into consideration.

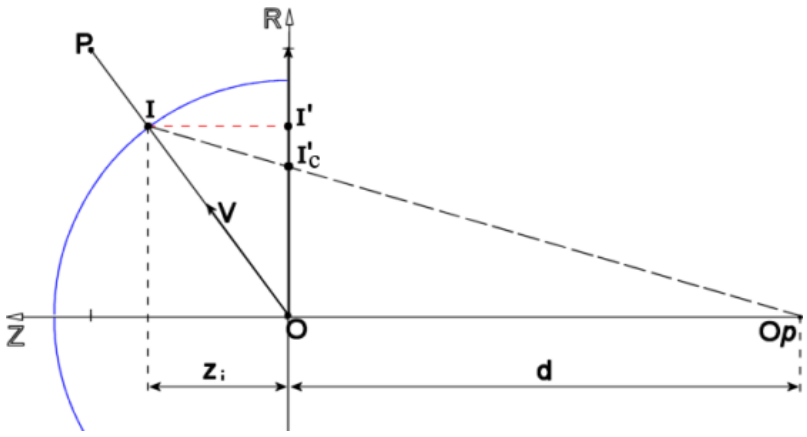

Figure 17: Projector viewpoint compensation for point transformations.

Figure 17 shows the projector's position $\mathbf{O_p}$, located at some distance d away from origin $\mathbf{O}$. The assumptions are the following: The system uses a conventional video projector. The projector is always behind the view origin; $\mathbf{O} > \mathbf{O_p}$. The projector must be positioned along the principal Z-axis. The distance d is adjusted

so that the image height displayed by the projector corresponds to the height of the 2D image plane in the viewing coordinate system.

Let's assume that the value for the intersection point $\mathbf{I}$ was already computed by Equation 12, and its orthographic transformation $\mathbf{I}'$ was evaluated by Equation 13. By observing the projector's view vector (long dashed line), it is easy to see that projector's view of the surface is perspective, and therefore the correct position of $\mathbf{I}'$ would be at $\mathbf{I}'_\mathbf{c}$. This does not suggest that the orthographic transformation method should be abandoned. Instead, a projector viewpoint compensation can be applied on the point $\mathbf{I}'$. This essentially adjusts $\mathbf{I}'$ to the correct position $\mathbf{I}'_\mathbf{c}$, which involves the scaling of the x'_i and y'_i components in $\mathbf{I}'$. The scaling value is proportional to the projector's Z-displacement from the origin $\mathbf{O}$ and the point $\mathbf{I}$:

$$\begin{aligned} t &= \left| \frac{d}{(z_i - d)} \right|, \quad \text{the scaling value } t, \\ x'_{ic} &= x'_i \cdot t, \\ y'_{ic} &= y'_i \cdot t, \end{aligned} \tag{22}$$

where $[x'_{ic},\ y'_{ic}]$ is the image plane coordinate at $\mathbf{I}'_\mathbf{c}$. This compensation applies for point transformations only, and therefore it is only useful for the realtime rendering algorithms discussed in Section 3.3.1.

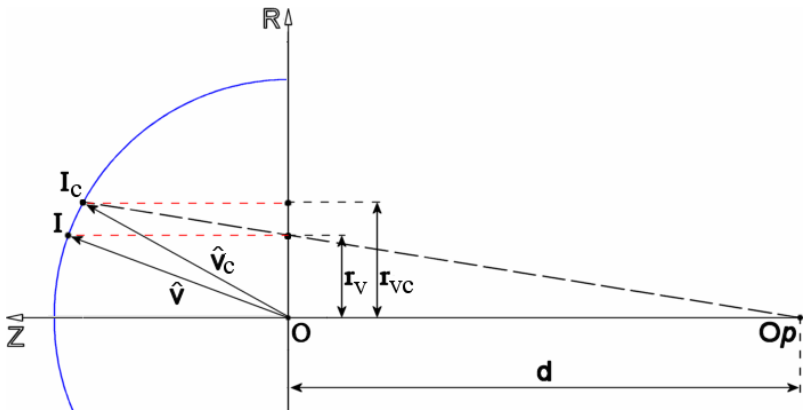

Figure 18: Projector viewpoint compensation for ray-tracing.

The ray-tracer rendering system requires a different approach for projector viewpoint compensation. In this case the initial view vectors must be adjusted. As illustrated in Figure 18, the original view vector $\widehat{\mathbf{V}}$ is fired into the scene for a given radius r_v, and passes though the intersection point $\mathbf{I}$. With the projector viewpoint compensation enabled, the correct view vector would be $\widehat{\mathbf{V}}_\mathbf{c}$, passing trough $\mathbf{I}_\mathbf{c}$. This would give a new radius, r_{vc}. The compensated intersection point $\mathbf{I}_\mathbf{c}$ lies on the projector's view vector (long dashed line), which passes trough the given radius r_v.

The correct view vector $\widehat{\mathbf{V}}_\mathbf{c}$ is determined by evaluating point $\mathbf{I}_\mathbf{c}$ and r_{vc} first. The Bisection Method detailed in Section 3.3.1 (Equation 11) can be adopted for solving the root r_{vc}. This requires the projector's view vector to be expressed as a

linear function:

$$\begin{aligned} z &= m(r - r_v), \quad \text{whose gradient } m \text{ is given by} \\ m &= \left| \frac{d}{r_v} \right|, \end{aligned} \tag{23}$$

where d is the projector's distance. Just like in Equation 10, the intersection point occurs when the profile curve $f(r)$ meets the projector's view vector. Therefore,

$$\begin{aligned} m(r - r_v) &= f(r), \\ 0 &= f(r) - m(r - r_v). \end{aligned} \tag{24}$$

The root r_{vc} is solved by applying Equation 24 in the Bisection algorithm (Equation 11). Finally, the new view vector $\widehat{\mathbf{V}}_\mathbf{c}$ is calculated by scaling the x_v and y_v components of $\widehat{\mathbf{V}}$ and by computing its Z-direction, z_{vc}:

$$\begin{aligned} t &= \frac{r_v}{r_{vc}}, \qquad \text{the scaling value } t, \\ x_{vc} &= x_v \cdot t, \\ y_{vc} &= y_v \cdot t, \\ z_{vc} &= f(r_{vc}), \quad \text{for } r_{vc} \leq r_{max}, \end{aligned} \tag{25}$$

where $\widehat{\mathbf{V}}_\mathbf{c}$ is the unit vector of $[x_{vc}, y_{vc}, z_{vc}]$.

4 Results

This section will summarize how the testing and the results were obtained for the ray-tracer and for the realtime implementation. The all the relevant pictures are placed in Appendix A.1 and A.2, which correspond to the ray-tracer and the realtime results respectively. The discussions and comments relating to all the results will be detailed in Section 5.

4.1 The Ray-tracer Implementation

The testing procedure for the ray-tracer was straightforward. Initially, a test scene was modelled, called "Toys". The scene contained a number of objects with varying complexity and detail. A back-and-white tiled floor was also used to highlight the panoramic distortion effects in the final image. A number of profile curve configurations were tried to observe the panoramic effects. Figures 21a to 21d are four examples of panoramic pictures. Their profile curve configurations are illustrated in Figures 21e to 21h respectively.

A panoramic simulation was also performed, because no curved display was available for testing. During the simulation, a hyperbolic panorama was perspective projected (texture mapped) onto a virtual display (shown in Figure 22). The texture mapping was applied as if a real projector would project the panorama onto the physical display. The virtual projector's displacement was -5.0 units from the origin. A perspective camera was positioned in the center of the display, which emulated the user's viewpoint. The user's perspective view of the display is illustrated in Figure 22c.

The anti-aliasing feature of the ray-tracer was also tested. Figure 23 shows two images of the same scene with anti-aliasing disabled and enabled (Figures 23a and 23b respectively). As detailed in Section 3.2, the adaptive anti-aliasing algorithm was only activated if the change in the luminance value for each pixel was above a tolerance value. In Figure 23b, the tolerance was set to 3% and number of samples per anti-aliased pixel was set to 32. A small section of the panorama was enlarged to highlight the differences.

4.2 The Realtime Implementation

The evaluation for the realtime rendering system involved a series of test runs, with different scene complexities and profile curve configurations. The testing procedure was divided into two main parts; one used the direct transformation method and the other used the lookup table method for transforming points. The direct method made a number of function calls to the Bisection routines (Equation 11) and to the profile curve. The lookup table method implemented the algorithms discussed in Section 3.3.2. The five test scenes used in the trials are illustrated in Figure 24. The scenes in Figure 24a to 24e have increasing number of polygons, and their properties are also listed in Table 1.

Scene	**Number of Objects**	**Polygon Count**
Cubes	8	336
Pool	13	2020
Hardware	20	4848
Toys	86	6934
Biology	15	17242

Table 1: Test scene properties.

When the trials were executed, the polygon subdivision routines were disabled (the maximum level of detail (LOD) was set to 0). This was intended to prevent the subdivision algorithm from introducing intermediate points within the polygons. The position of the viewpoint and the movement of objects may also trigger polygon subdivision. As a result, the total number of points transformed would dynamically change. Initially, the interest lies in comparing scenes with a fixed (or known) number of points.

The direct transformation method and the lookup table method executed four trials in each scene. This was intended to observe the performance of the transformation routines with different profile curve configurations (see Equations 26). The first equation was a simple linear function, the second was a cubic and the last one was a hyperbolic. Note that the equations are represented exactly as they were fed to the equation compiler. The fourth profile curve configuration was "NULL", where the panorama transformations were disabled and only the standard perspective trans-

formations were used.

$$\begin{aligned} &\text{Curve 1: } f(r) = -r + 1; \\ &\text{Curve 2: } f(r) = -r \cdot r \cdot r + r \cdot r + r + 1; \\ &\text{Curve 3: } f(r) = -\sqrt{4 + 6r^2} + 3. \end{aligned} \tag{26}$$

The testing procedures described above were conducted on a mid-range Pentium III processor (450MHz), using a 3dFx Voodoo3 2000 graphics card. Figure 26 shows the performance (in frames per second) for the direct transformation and for the lookup table method. The horizontal axis in Figure 26 illustrate the number of vertices transformed per frame in each scene. These results are summarized in Table 2. The table also includes the *performance ratio* between the two methods. Each ratio was calculated by dividing the lookup table's frame rate with the corresponding direct transformation's frame rate.

Scene	**FPS (Direct Method)**			**FPS (Table Method)**			**Performance Ratio**		
	C.1	*C.2*	*C.3*	*C.1*	*C.2*	*C.3*	*C.1*	*C.2*	*C.3*
Cubes	60.80	51.80	32.00	60.30	60.40	60.40	0.99	1.17	1.89
Pool	18.00	8.70	5.40	60.30	60.30	60.30	3.35	6.93	11.17
Hardware	7.50	3.60	2.20	33.40	33.20	33.60	4.45	9.22	15.09
Toys	5.20	2.50	1.60	22.40	22.40	22.40	4.31	8.96	14.00
Biology	2.10	1.00	0.60	9.30	9.30	9.30	4.43	9.30	15.50

Table 2: Performance summary with the polygon subdivision disabled. Note: C.1, C.2 and C.3 represent Curve 1, Curve 2 and Curve 3 respectively.

To observe how the polygon subdivision affected the rendering performance, additional trials were conducted with the Toy scene. The Toy scene was chosen, because it features a variety of objects with differing sizes. The transformation routines used the lookup table method, with Curve 2 (from Equation 26) as the profile curve configuration. Each trial used a different maximum level of detail (LOD), ranging from 0 to 9. The division threshold value was fixed to 0.2. The trials used the same fixed viewpoint and viewing orientation. When the ten trials were completed, the whole testing procedure was repeated, but this time the division threshold value was changed to 0.1. The graph in Figure 25 illustrates how the polygon subdivision affected the rendering performance. Table 3 lists the ten LOD settings used with two division threshold values. Each row summarizes the average number of points transformed per frame.

Additional results for the realtime rendering system include Figures 27 and 28. Figures 27a and 27b demonstrate the polygon subdivision routines at work. These illustrations were prepared in wire-frame mode to highlight the polygon boundaries. The perspective views of the scenes are also included in the small sub-window. Figures 27c and 27d illustrate the captured screens for the realtime renderer.

Figure 28 demonstrates a simulation of the panoramic rendering system, which uses a similar arrangement as the ray-tracer's panoramic simulation in Figure 22. The panorama was rendered for a hyperbolic surface, which was perspective projected into the virtual display (Figure 28b). The virtual projector's displacement

LOD	Average Points Transformed	
	Threshold = 0.1	*Threshold = 0.2*
0	20802	20802
1	23034	21870
2	24714	22974
3	26550	23778
4	27534	24114
5	28278	24594
6	29238	25122
7	30198	25650
8	31158	26178
9	32118	26706

Table 3: Number of points transformed per frame in the Toy scene, when the polygon subdivision was enabled.

was -5.0 units from the origin. A perspective camera simulates the user's viewpoint and sees a perspective view of the panoramic scene (Figure 28c).

4.3 Projector Viewpoint Compensation

The final set of results are relevant to the projector viewpoint compensation techniques, discussed in Section 3.4. The Toy Scene was used again for the evaluation, because it contains a number of detailed objects. The profile curve was $f(r) = 2 - (r + 0.01)^r$, and the view orientation was downwards.

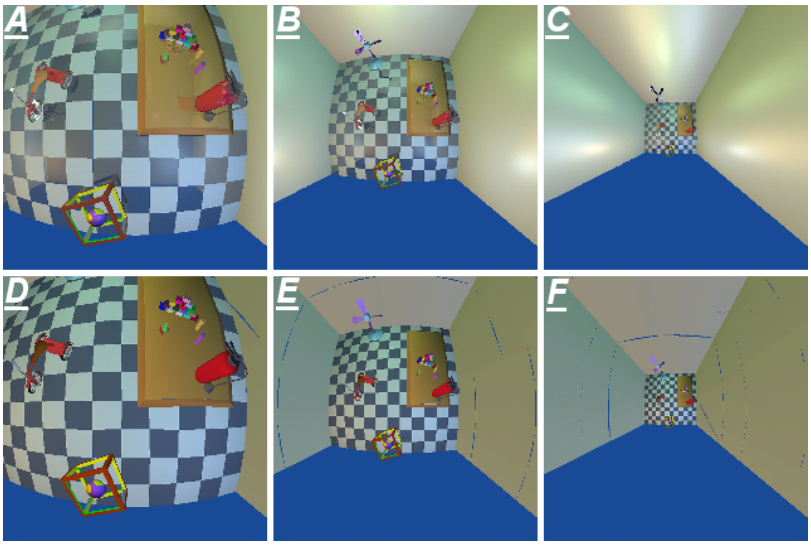

Figure 19: Perspective viewpoint compensation for the ray-tracer ($f(r) = 2 - (r + 0.01)^r$): *(A)* No viewpoint compensation, *(B)* $d = -1.0$, *(C)* $d = -0.25$. For the realtime renderer: *(D)* No viewpoint compensation, *(E)* $d = -1.0$, *(F)* $d = -0.25$.

Figures 19a to 19e reveal how the panorama was affected when the projector's viewpoint was changed for the ray-tracer and for the realtime renderer. The image group 19a to 19b are the results for the ray-tracer. The first picture is the original viewpoint, with no perspective viewpoint compensation enabled. The next two pictures illustrate the effect of viewpoint compensation, with the projector's origin successively moved closer to the viewer's origin. The image group 19d to 19e use the same configuration as the first three illustrations, but this time the realtime renderer was used to generate the images.

As the projector was moved closer to the user's viewpoint, the perspective compensation was increased. This effect was identical for the ray-tracer and the realtime renderer. If the projector was coincident with user's viewpoint, the compensation applied on the panorama would be extreme, and the resulting image would be unintelligible. However, in practical situations the projector is never coincident with the view origin. Most conventional projectors display images in the 4:3 aspect ratio, which provides a narrow field of view. In order to cover the entire curved display area, such projectors would be always situated at some distance away from the user's viewpoint.

5 Discussion

5.1 The Ray-tracer Implementation

The ray-tracer's biggest advantage is that the resulting panoramas did not show signs of geometric distortions - a problem that was specific for the realtime renderer (see Section 5.2). The ray-tracer rendered the panorama one pixel at a time. Therefore, all the objects were drawn to match the surface's contour as close as possible. (Note that the term *geometric distortion* should not be confused with *panoramic distortion*. Panoramic distortion is the actual warping effect caused by the wide FOV in the panorama.)

The images rendered by the ray-tracer are shown in Figure 21. Figure 21a illustrates the subtle panoramic distortions caused by the quadratic profile curve. In this case, the field of view was not very wide, because the panorama was clipped by the 2D image plane (see Figure 21e).

Some curves created extreme warping effects in the panorama, and sometimes allowed the view vectors to cross the profile curve twice. This effect is illustrated in Figure 21b, where the tricycle appears to be conjoined due to the double intersection of the view vectors. Such effects were only possible with the ray-tracer, because it did not use numerical methods for estimating the initial direction for the view vectors (assuming that the projector viewpoint compensation was disabled).

The spherical panorama in Figure 21c did not experience clipping by the 2D image plane, because the profile curve's upper limit r_{max} corresponded with the image plane's minimum radius. Figure 21d is an example of a parabolic panorama, that has an extremely wide field of view. Its corresponding cross-sectional surface plot (Figure 21h) reveals that the user's viewpoint was well inside projection surface. If this panorama was displayed on a physical surface, the viewer would be able to glance backwards.

The panoramic simulation in Figure 22 used a hyperbolic virtual display to match the panorama's distortion. The user's viewpoint in Figure 22b was emulated by a perspective camera, which created the result in Figure 22c. The contour of the virtual display corrected the distortion in the panorama, and made the scene appear perspective correct. This effect was highlighted by the fire extinguisher's edge. However, the panorama/virtual display relationship also had a side-effect; the objects in the perspective view showed signs of pixel stretching, which made the objects appear as jagged (Figure 22d).

The jagged edges and aliasing effects were partially caused by the nonuniform sampling of the curved surface, during the panoramic rendering process (see Section 3.1.2). These artifacts were predominant in the regions where the profile curve's tangent became very steep with respect to the image plane. These problems were very obvious in panoramas that had a very wide FOV (Figure 21d), or at the edges of the spherical panorama (Figure 21c).

The adaptive anti-aliasing technique introduced in Section 3.2 was designed to minimize the sharply notched edges at the object boundaries. Figure 23 shows two spherical panoramas of the same scene, with anti-aliasing disabled (Figure 23a) and enabled (Figure 23d). The smaller sub-windows magnify a portion of the panorama's edge. The anti-aliased image reveals that the staircasing artifacts at the edges and on the floor tiles were minimized.

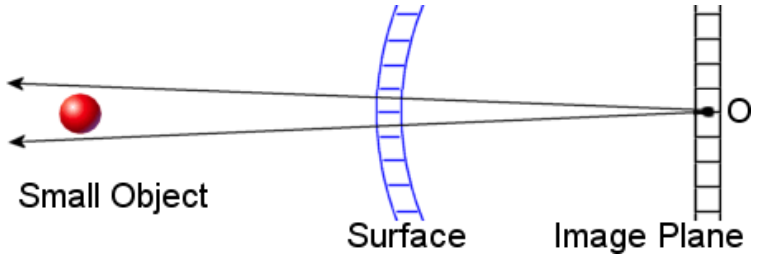

Figure 20: Two primary rays missing a small object.

As mentioned in Section 3.2, the ray-tracer only activated the anti-aliasing algorithm when a contrasting feature or a sudden transition in luminance was detected. This method for detecting edges did not work for certain situations. The adaptive anti-aliasing failed to detect luminance boundaries when an object in the scene was very far away, or when it was very small, typically one screen pixel in size. Figure 20 illustrates this problem. Two primary rays were fired at two neighbouring pixels. The primary rays completely missed the small object. As a result, the anti-aliasing was not activated, because the renderer assumed that there was no object present in the first place.

5.2 The Realtime Implementation

The primary goal for the realtime rendering system was to execute panoramic point transformations in realtime, typically around 30 frames per second. The results in Section 4.2 indicate that this was possible on a mid-range Pentium III class machine.

It is obvious that the direct transformation method suffered in performance when the number of vertices transformed per frame was increased. This was illustrated in the performance graph, Figure 26a. Additionally, the performance was not only dependent on the vertex count, the algorithm was also directly affected by the computation complexity of the profile curve equation (see Equations 26).

The first equation (Curve 1) performed the best, because it had the lowest computational complexity, about 2 arithmetic operations. As expected, the average frame rate for the cubic profile curve (Curve 2) was lower, because the penalty for executing $f(r)$ was 7 operations. The last equation was the worst performer, because it contained an expensive square root and a power operation.

As mentioned in Section 3.1.1, the equation was stored as a sequence of custom operation codes. As a result, there were additional overheads for executing the profile curve equation. One overhead was the time required for accessing the jump table, and the other was the delay for executing the jump instruction for each custom opcode. However, the main slowdown occurred in the Bisection algorithm (Equation 11), because the algorithm called the profile curve subroutine for several iterations.

The performance for the lookup table method was constant for all profile curve configurations in the same scene (Figure 26b). The dependency on the profile curve's complexity was eliminated, because intersection points were pre-calculated and stored in the table. The time required to calculate the table index (in Equation 17) and to access the lookup table was constant.

Table 2 illustrates that the frame rate improvement for the lookup table method was significant for the Hardware, Toy and the Biology scenes. The most significant performance gains were observed with the computationally expensive curves, such as Curve 2 and Curve 3. These gains are highlighted by the performance ratios, listed in Table 2. However, the lookup table method did not show very significant gains in the Cube scene. The scene's complexity was simple enough for the direct method to achieve up to 60.8 frames per second with Curve 1, which slightly outperformed the lookup table method. The performance results for the perspective renderer are also included in Figure 26. The graphs for the perspective renderer indicate that the maximum output for the OpenGL rendering pipeline was about 61 frames per second.

It has been shown that the lookup table method does provide some significant frame rate improvement over the direct transformation method. However, the test scenes in Figure 24 were relatively simple in complexity. Some practical applications may implement scenes that easily exceed the complexity of the test scenes used in this project, and would result in a significant performance drop for the panoramic renderer. Therefore, even the lookup table method may not be entirely practical for very complex scenes.

Figure 25 shows how the rendering system performed with different levels of subdivision detail (LOD). When the subdivision was disabled (LOD = 0), the system demonstrated the best case scenario, as specified by Equation 21. According to Equation 20, the worst case behaviour for the Toy scene (with 6934 polygons) would be over 900 million point transformations, given that LOD = 9. However, Table 3

illustrates that the actual number of points transformed was close to the best case scenario for both threshold settings.

The graphs in Figure 25 illustrate that the frame rate started to level out as the LOD was increased. It turned out that the number of subdivisions did not significantly change for the higher LOD settings, because the actual threshold level was reached by the majority of the subdivided triangles. This indicated that the system rarely reached the worst case complexity (Equation 20), and remained closer to the best case behaviour. The test also revealed that the performance overhead for transforming the division points was approximately 20% to 30%, with LOD = 9.

Figure 25 also illustrates that the overall performance worsened when the threshold value was decreased from 0.2 to 0.1. The worst case complexity was only possible when the subdivision threshold value was very close to 0. Therefore, the two main factors that directly affected the rendering performance were the maximum LOD setting and the subdivision threshold value.

However, the subdivision performance is also affected by other variables, such as the profile curve's shape, the viewing location and the size of various objects in the scene. For example, a scene consisting of very large triangles will make a perfect candidate for excessive subdivision. In contrast, small detailed objects may not trigger the division algorithms at all. In practical situations, the scene data may need some preprocessing to minimize the demand for subdivision in the rendering stage. For example, very large polygons can be broken down in the modelling stage. This may eliminate the need for changing the maximum LOD and subdivision threshold settings.

Figure 27 illustrates how the polygon subdivision algorithms attempted to approximate the curvature of the projection surface. Figures 27a and 27b highlight the actual polygon boundaries. Some surfaces (such as nearby walls) were greater in magnitude than the subdivision threshold, and therefore they were recursively decomposed into smaller triangles. The actual level of detail was dynamically varied when the viewpoint changed location. This caused popping effects in the geometric models, because the intermediate points were suddenly introduced (or removed) in the scene. Such effects were minimized by increasing the maximum level of detail and/or by decreasing the subdivision threshold. Of course, this increased the computational cost for rendering the entire scene.

Unfortunately, when a polygons's edge was divided, the algorithm had no way of determining which neighbouring polygons were immediately affected by this division. Therefore, when two (or more) neighbouring polygons received a different level of detail, the number of subdivision points along their edges did not match. This developed cracks within the model, see Figures 27c and 27d. The cracking effect was very obvious in the walls and ceilings, because they consisted of large polygons. This problem was worsened when the profile curve created extreme panoramic distortions in the scene.

Figure 28 illustrates a panoramic simulation for the realtime renderer. The panorama was rendered for a hyperbolic surface, and then it was re-projected onto a simulated display (Figure 28b). The perspective correct view of the scene (Figure 28c) high-

lights the importance of polygon subdivision. The fire extinguisher's edge suffered from slight geometrical distortions, because the model lacked subdivision points. These unwanted effects became worse when the level of detail was decreased. The ray-traced equivalent of this simulation did not suffer from geometric distortions, because no approximation techniques were used to render the objects (see Figure 22). As a result, the ray-tracer has the advantage of producing better quality panoramas than the realtime renderer.

6 Conclusion

Most perspective displays, such as CRT monitors, are limited by their narrow field of view. To overcome this limitation, a number of panoramic display systems were developed in the past. Unfortunately, Section 2 revealed that most commercial panoramic display systems are very expensive, and limited their use for specialized applications only. However, a cheaper alternative for panoramic visualization is possible. An effective panoramic display environment can be achieved by surrounding the viewer with a curved display. With this arrangement, a conventional projector can be used the displayed panorama on the curved surface. In order to make the scene look perspective correct, the rendering system must be capable of generating panoramas that match the curvature of the display.

6.1 Review

The project's aim was to develop a rendering software that allows conventional projectors to display panoramas on a symmetrical curved display. The entire project consisted of two main parts: The ray-tracer implementation and the realtime implementation. Since most curved displays are symmetrical about its principal axis, 2D profile curves provided a convenient way for modelling a display's shape. The rendering system accepted an arbitrary profile curve, that was used to construct a concave surface of revolution. This allowed the rendering system to generate panoramas for displays other than the standard dome configuration.

The symmetrical property of the surface was an advantage for executing point transformations, or when determining the initial rays for the ray-tracer. These transformations were performed in the 2D profile curve space, rather than directly with the 3D surface itself. The rendering process involved two steps: Initially the object was projected onto a surface of revolution, then it was orthographically re-projected onto a 2D image plane. If the panorama was to be displayed on a physical surface, the final step had to compensate for the projector's perspective viewpoint.

The ray-tracer's task was to to determine the initial rays for each image plane coordinate. This coordinate also represented the ray's direction in the XY-plane. To solve for the Z-direction, the radius of the 2D coordinate was computed first. The radius was substituted into the profile curve equation, which returned the ray's Z-component.

This type of projection was orthographic, and it did not provide a uniform sampling of the 3D surface on the image plane. Regions where the surface gradient was

high relative to the image plane were under sampled. As a result, aliasing effects were prevalent in those regions of the panorama.

The adaptive anti-aliasing technique described in Section 3.2 did minimize the staircasing artifacts in object edges; however, there were cases where the anti-aliasing algorithm failed. The algorithm failed to detect luminance boundaries for very small objects, typically those that were one screen pixel in size. The primary rays that monitor the luminance values completely missed the small object, and as a result the anti-aliasing was not activated.

The realtime implementation faced several problems when it came to panoramic transformations. The projection of several points onto the 3D surface had to be executed in realtime for a given arbitrary profile curve configuration. Section 3.3.1 introduced a direct transformation technique, that used approximation methods for projecting points onto a surface of revolution. In order to deal with partially visible objects, the transformation algorithm used a tangent line to extend the profile curve beyond the limit r_{max}. This arrangement allowed the transformation of partially visible objects onto tangent line, and then passed to the rendering pipeline.

The performance tests demonstrated that the direct transformation method was inadequate for realtime rendering. Its performance was not only dependent on the number of points in the scene, it was also affected by the computational complexity of the profile curve equation.

The lookup table method (introduced in Section 3.3.2) eliminated the profile curve dependency problem, because the intersection points were pre-calculated and stored in the table. The access time for the lookup table was constant, even when computational expensive profile curve configurations were used. The test results illustrated that the lookup table method achieved up to 15-fold improvement in performance over the direct transformation method. Such improvements were particulary significant for complex scenes and computationally expensive profile curves.

Straight polygon edges suffered geometrical distortions in the panorama, because the transformations were not affine. To minimize this problem, a recursive isotropic subdivision algorithm was implemented in the rendering system. The algorithm dynamically varied the degree of subdivision, based on the size of each sub-triangle in 2D screen space. The performance tests revealed that the overhead for transforming the division points remained close to the best case behaviour (Equation 21), and it rarely reached the worst case complexity (Equation 20).

The subdivision algorithms also introduced the cracking problem, where gaps appeared between several triangles. The problem appeared when two (or more) neighbouring polygons received a different level of detail. As a result, the number of subdivision points along their edges to did not match. This problem was worse for large polygons and when the profile curve created extreme panoramic distortions in the scene.

6.2 Future Work

The lookup table method demonstrated that realtime panoramic rendering was possible. However, some practical applications may implement scenes that would easily exceed the complexity of the test scenes used in this project. In such situations,

the resulting rendering performance would be poor. The most expensive part in the lookup table algorithm was Equation 17, where the angle of the view vector was calculated. To further increase the performance, the arc tangent operation should be removed, perhaps replaced with a fast approximation algorithm or with an arc tangent lookup table.

The cracking problem in the polygon subdivision can be solved by adding new data management algorithms to handle *winged edge* representations of triangles [15] [47] [48]. Such structures use edge lists [28] to keep track of triangles that share a common edge. Each entry in the list separates two triangle faces and maintains a pointer for each. If a triangle's edge was bisected, its corresponding edge structure in the list will point to the neighbouring polygon affected by the subdivision. To eliminate the cracking problem, the neighbouring triangle's edge is also bisected.

The actual ray-tracing was a time consuming process, particulary when the anti-aliasing algorithms were used. The most expensive operation for the ray-tracer was the ray-object intersection tests. The renderer used a hierarchial bounding boxes [30] [25] for grouping objects together. The rendering efficiency was highly dependent on how well each bounding box fitted the enclosed object.

Ideally, the computation time for the intersection tests should be relatively independent of the scene's complexity. This property may be achieved by implementing a space partitioning data structure, such as *octrees* [29]. Octrees recursively subdivide the scene space into 8 sub-octants until the maximum partition criterion is met. More subdivision would be applied in the regions where the objects densely populate the scene. Each nonempty partition contains a list of objects that penetrate the partition volume. Rather than checking a ray against all set of bounding volumes, only those surfaces are tested that occupy the partition volume intersected by the ray.

A Illustrations

Appendix A.1 and A.2 contain all the illustrations, results for the ray-tracer and the realtime implementation respectively. The panoramic rendering software and the test scenes are available to download from the following web address:

`http://www-personal.monash.edu.au/~ddea3/project/index.shtml`

Please read the online documentation for more details.

A.1 Images for the Ray-tracer

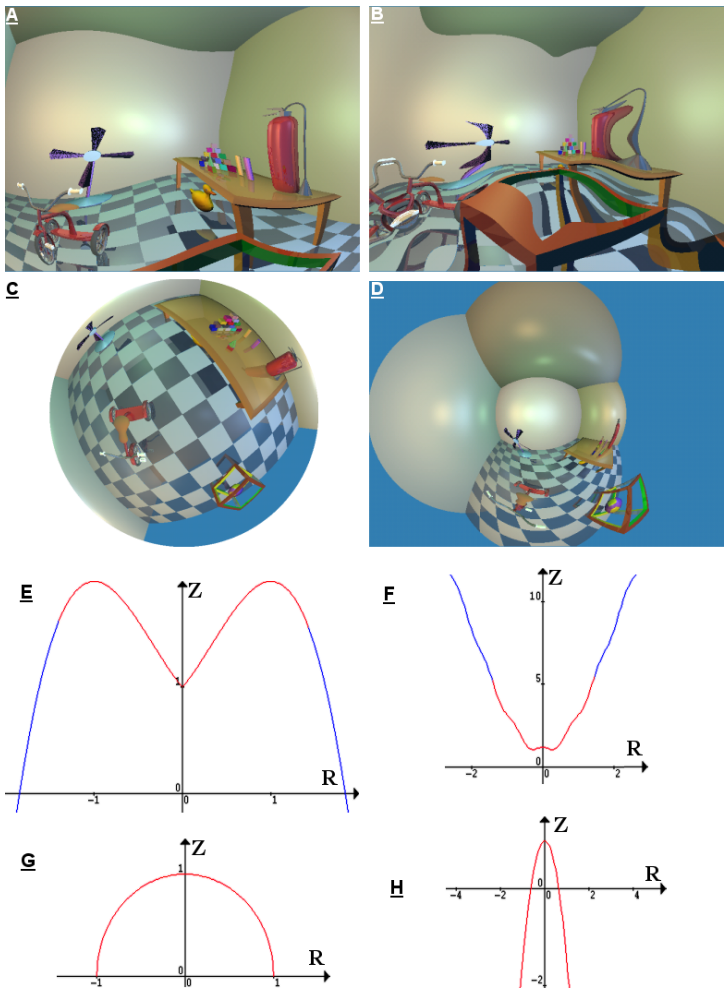

Figure 21: Ray-traced panoramas, with different profile curve configurations:
(A) $f(r) = -r^3 + r^2 + r + 1$. *(B)* $f(r) = -0.1r^4 + 2.25r^2 + 0.2cos(8r) + 1$.
(C) $f(r) = \sqrt{1 - r^2}$. *(D)* $f(r) = -2.5r^2 + 1$.
Slides *(E)* to *(H)* are the projection surfaces used to create the panoramas in *(A)* to *(D)*. The red regions highlight the visible portions of the surface on the 2D image plane.

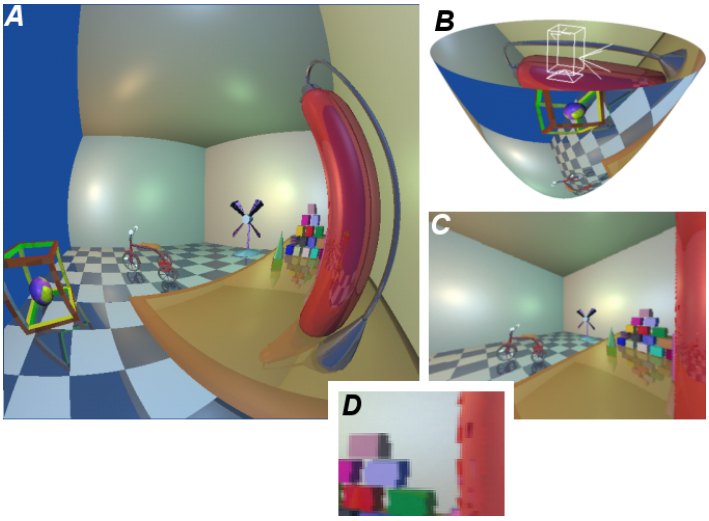

Figure 22: *(A)* The hyperbolic panorama, $f(r) = 3 - \sqrt{4 + 6r^2}$. *(B)* The panorama is projected into a virtual curved display, with a perspective camera in the center. *(C)* The perspective view obtained by the camera. *(D)* An enlarged portion of the perspective view, highlighting the pixel stretching effects.

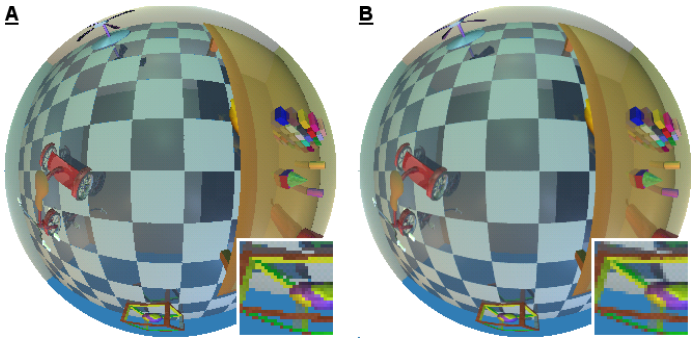

Figure 23: Anti-alias example ($f(r) = \sqrt{1 - r^2}$): *(A)* Original. *(B)* Anti-aliasing enabled.

A.2 Images and Data for the Realtime Implementation

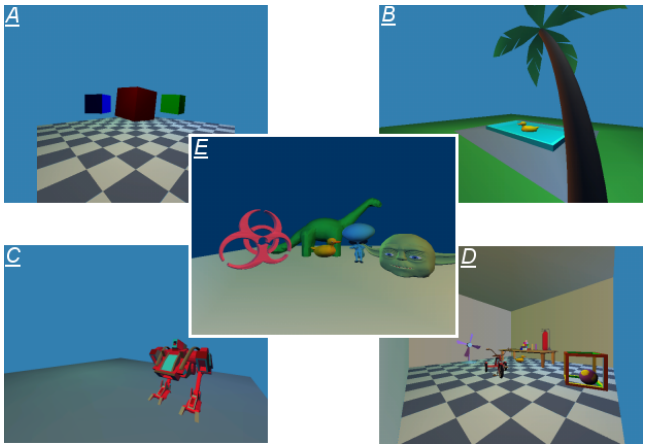

Figure 24: Perspective view of the test scenes:
(A) "Cubes", *(B)* "Pool", *(C)* "Hardware", *(D)* "Toys", *(E)* "Biology".

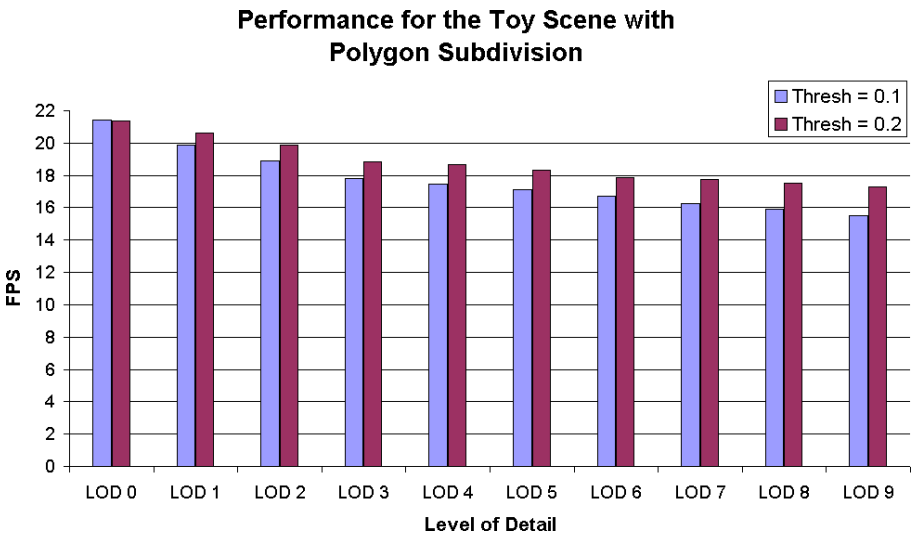

Figure 25: Frame rate for the Toy Scene, with the polygon subdivision enabled at different levels of detail. There were two tests, one with the subdivision threshold set to 0.1 and the other set to 0.2.

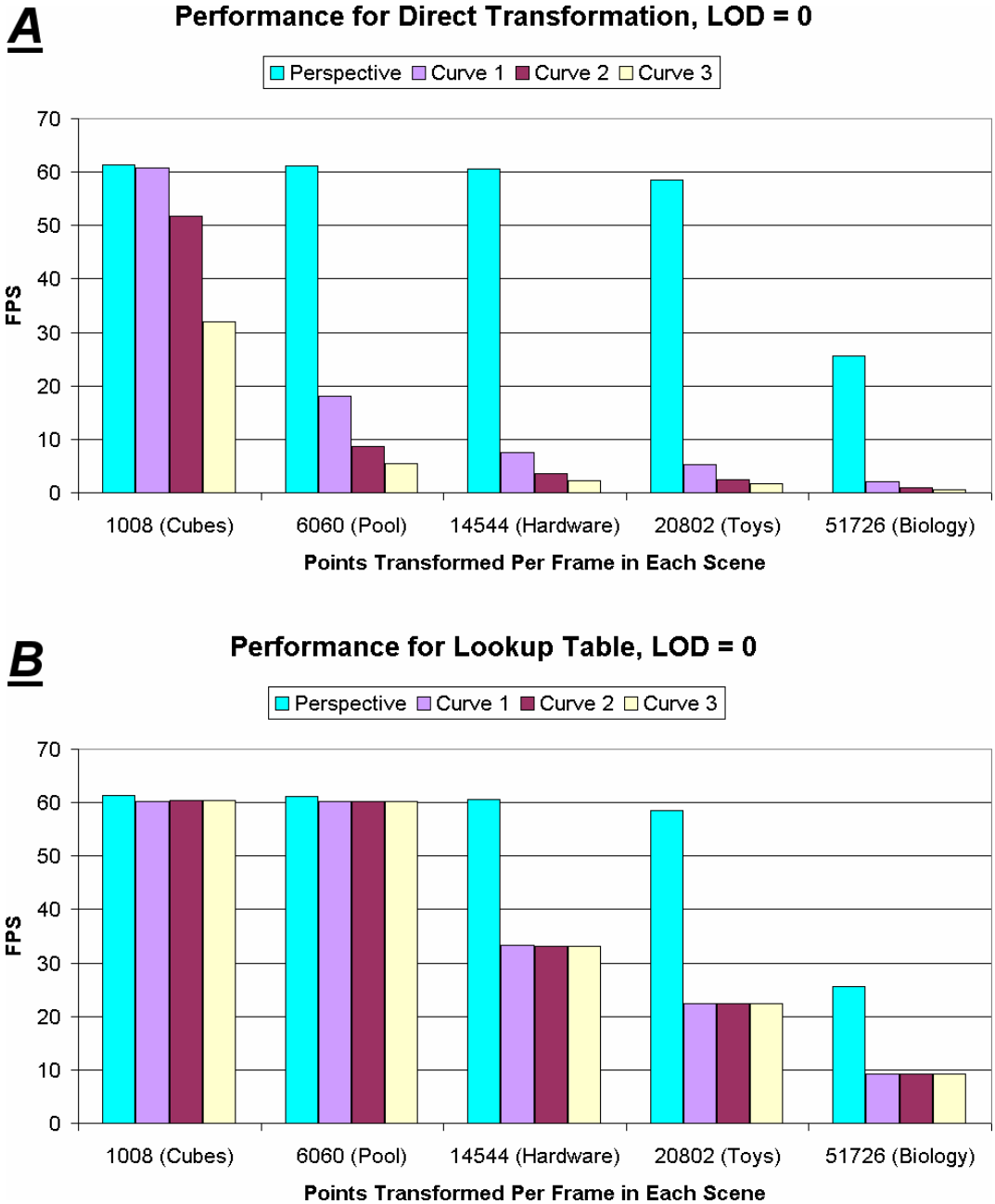

Figure 26: Performance results for *(A)* the direct transformation method and *(B)* the lookup table method, with three profile curve configurations. The performance for the standard perspective transformation is also included. The polygon subdivision was disabled.

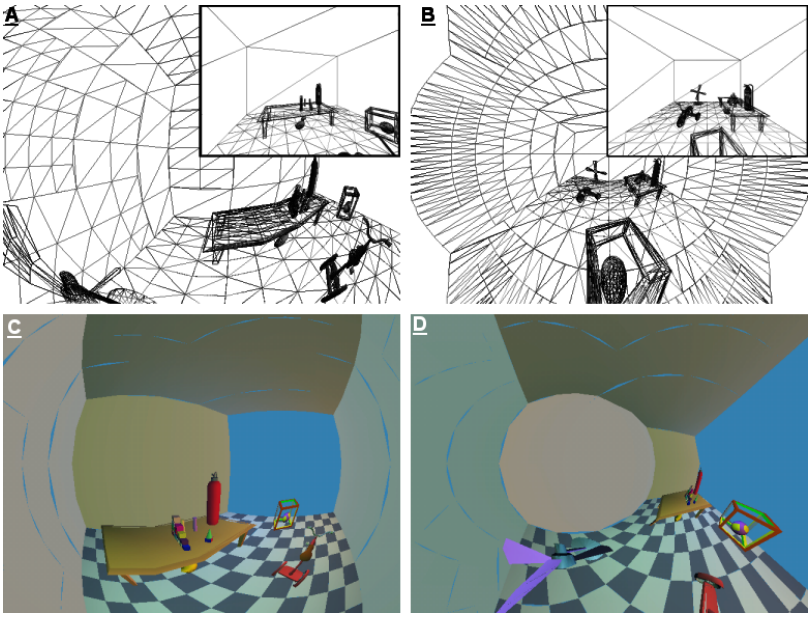

Figure 27: Subdivision example for *(A)* $f(r) = -r^2 + 1$ and *(B)* $f(r) = -log_e(r + 0.1)$. Realtime renderer for *(C)* $f(r) = -r^2 + 1$ and *(D)* $f(r) = -2r + 1$.

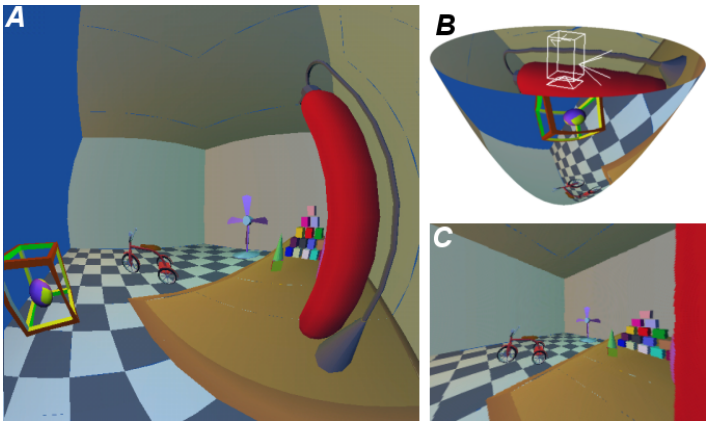

Figure 28: *(A)* The hyperbolic panorama, $f(r) = 3 - \sqrt{4 + 6r^2}$. *(B)* The panorama is projected into a virtual curved display, with a perspective camera in the center. *(C)* The perspective view obtained by the camera.

References

[1] *Ffp series facility displays*, (2000),
Panoram Technologies, Inc.
`http://www.panoramtech.com` [Accessed: April 2001].

[2] *Fullview 360*, (2000),
Panoram Technologies, Inc.
`http://www.panoramtech.com` [Accessed: April 2001].

[3] *Imax theatres*, (2000),
IMAX Systems Corporation
`http://www.imax.com` [Accessed: February 2001].

[4] *Panowall 2ks facility displays*, (2000),
Panoram Technologies, Inc.
`http://www.panoramtech.com` [Accessed: April 2001].

[5] *Rave: Fully reconfigurable display modules*, (2000),
FakeSpace Systems, Inc.
`http://www.fakespacesystems.com` [Accessed: April 2001].

[6] *Reality center 3300w*, (2000),
Silicon Graphics, Inc.
`http://www.sgi.com` [Accessed: June 2001].

[7] *The visionstation and visionstation3*, (2000),
Elumens Corporation, Inc.
`http://www.elumens.com` [Accessed: April 2001].

[8] *Custom visiondome designs*, (2001),
Alternate Realities Corporation (ARC)
`http://www.virtual-reality.com` [Accessed: February 2001].

[9] *Full immersion head mounted displays*, (2001),
Kaiser Electro-Optics, Inc.
`http://www.keo.com` [Accessed: March 2001].

[10] *Numerical root finding*, (2001),
`http://www.efunda.com/math/num_rootfinding/num_rootfinding.cfm`
[Accessed: June 2001].

[11] *Open graphics library*, (2001),
OpenGL Home Page
`http://www.opengl.org` [Accessed: June 2001].

[12] *The pdc vr-cube*, (October 1998),
Center for Parallel Computers, Kungliga Tekniska Hgskolan (KTH).
`http://www.pdc.kth.se/projects/vr-cube/` [Accessed: June 2001].

[13] *Taking the long view: Panoramic photos, 1851-1991*, (October 1998),
Prints and Photographs Division, Library of Congress
`http://www.loc.gov/` [Accessed: March 2001].

[14] J. Amanatides, *Ray tracing with cones*, Proceedings of SIGGRAPH 84 (July 1984), 129–135.

[15] Bruce G. Baumgart, *A polyhedron representation for computer vision*, National Computer Conference, AFIPS Conf. Proc. (May 1975), 589–596,
`http://64.2.61.71/winged-edge/winged-edge.html`
See also: `http://www.baumgart.org/` [Accessed: October 2001].

[16] Stephan Bischoff, Leif P. Kobbelt, and Hans-Peter Seidel, *Towards hardware implementation of loop subdivision*, (Agust 2000),
ACM, Inc. `http://www.acm.org/dl/` [Accessed: August 2001].

[17] Jim Blinn, *Jim blinn's corner: Dirty pixels*, Morgan Kaufman Publishers, Inc. (1998).

[18] Shenchang Eric Chen, *Quicktime vr, an image-based approach to virtual environment navigation*, Proceedings of SIGGRAPH 95 (August 1995), 29–38,
ACM, Inc.
`http://www.acm.org/dl/` [Accessed: June 2001].

[19] Luc Courchesne, *Panoscope 360*, (2001),
University de Montreal, Design Industriel
`http://www.panoscope360.com` [Accessed: May 2001].

[20] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti, *Surround-screen projection-based virtual reality: The design and implementation of the cave*, Proceedings of SIGGRAPH 93 (August 1993), 135–142.

[21] Germund Dahlquist and Åke Björck, *Numerical methods*, Prentice-Hall Series in Automatic Computation (1974).

[22] M. Deering, *The holosketch, the vr sketching system*, Communications of the ACM, Vol. 39, No. 5 (May 1996), 54–61,
`http://www.acm.org/dl/` [Accessed: June 2001].

[23] Michael Deering, *High resolution virtual reality*, Proceedings of SIGGRAPH 92 (July 1992), 195–202,
ACM, Inc. `http://www.acm.org/dl/` [Accessed: June 2001].

[24] Mark A. Z. Dippé, *Antialiasing through stochastic sampling*, Proceedings of SIGGRAPH 85 (July 1985), 69–78,
ACM, Inc. `http://www.acm.org/dl/` [Accessed: October 2001].

[25] Dave Eberly, *Intersection of orthogonal view frustum and oriented bounding box using separation axis testing*, Technical Report (2000),
Magic Software, Inc. `http://www.magic-software.com` [Accessed: June 2001].

[26] Steven Feiner, Blair MacIntyre, Marcus Haupt, and Eliot Solomon, *Windows on the world: 2d windows for 3d augmented reality*, UIST '93 (November 1993), 145–155,
ACM, Inc. `http://www.acm.org/dl/` [Accessed: June 2001].

[27] Jon Genetti and Dan Gordon, *Ray tracing with adaptive supersampling in object space*, Graphics Interface '93 (1993), 70–77.

[28] Andrew S. Glassner, *Maintaining winged-edge models*, Graphics Gems II, Academic Press, Inc. (1991), 191–201.

[29] Andrew S. Glassner, *Space subdivision for fast ray-tracing*, IEEE Computer Graphics and Applications 4 (1984).

[30] Eric A. Haines, *Introduction to ray tracing*, Proceedings of SIGGRAPH 87 (1987),
ACM, Inc. `http://www.acm.org/dl/` [Accessed: August 2001].

[31] Gerd Heber, Rupak Biswas, Parimala Thulasiraman, and Guang R. Gao, *Using multi-threading for the automatic load balancing of 2d adaptive finite element meshes*, CAPSL Technical Memo 20 (August 1998).

[32] Janne Jalkanen, *Building a spatially immersive display: Hutcave*, Licentiate Thesis, Department of Computer Science, Helinsky University of Technology (March 2000).

[33] Ros S. Kalawsky, *The science of virtual reality and virtual environments*, Addison-Wesley Publishing Company (1993).

[34] Sing Bing Kang and Pavan Kumar Desikan, *Virtual navigation of complex scenes using clusters of cylindrical panoramic images*, Graphics Interface '98 (June 1998), 223–232.

[35] Mark E. Lee, *Statistically optimized sampling for distributed ray tracing*, Proceedings of SIGGRAPH 85 (July 1985), 61–67,
ACM, Inc. `http://www.acm.org/dl/` [Accessed: October 2001].

[36] David Martindale and Alan W. Paeth, *Television color encoding and "hot" broadcast colors*, Graphics Gems II, Academic Press, Inc. (1991), 147–158.

[37] Nelson Max, *Siggraph '84 call for omnimax films*, Proceedings of SIGGRAPH 83 (January 1983), 73–76.

[38] Shree K. Nayar and Simon Baker, *Theory of catadioptric image formation*, Technical Report CUCS-015-97, Department of Computer Science, Columbia University (1997).

[39] Shree K. Nayar, Keith J. Thoresz, and Joshua Gluckman, *Real-time omnidirectional and panoramic stereo*, Technical Report, Department of Computer Science, Columbia University (1998).

[40] Marc Olano, Jon Cohen, Mark Mine, and Gary Bishop, *Combatting rendering latency*, Proceedings of SIGGRAPH 95 (1995),
ACM, Inc. `http://www.acm.org/dl/` [Accessed: June 2001].

[41] Shmuel Peleg and Joshua Herman, *Panoramic mosaics by manifold projection*, IEEE CVPR Proceedings (1997), 338–343.

[42] Venkata N. Peri and Shree K. Nayar, *Generation of perspective and panoramic video from omnidirectional video*, In Proceedings of the 1997 DARPA Image Understanding Workshop, New Orleans (May 1997).

[43] Ramesh Raskar, Michael S. Brown, Ruigang Yang, Wei-Chao Chen, Greg Welch, Herman Towles, Brent Seales, and Henry Fuchs, *Multi-projector displays using camera-based registration*, IEEE Visualization '99 (October 1999), 161–168.

[44] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry Fuchs, *The office of the future: A unified approach to image-based modelling and spatially immersive displays*, Proceedings of SIGGRAPH 98 (July 1998), ACM, Inc. `http://www.acm.org/dl/` [Accessed: June 2001].

[45] Ramesh Raskar, Greg Welch, and Henry Fuchs, *Spatially augmented reality*, First International Workshop on Augmented Reality (November 1998), ACM, Inc. `http://www.acm.org/dl/` [Accessed: June 2001].

[46] David Salomon, *Computer graphics and geometric modelling*, Springer-Verlag New York, Inc. (1999).

[47] Brian Sharp, *Subdivision surface theory*, Game Developer Magazine (January 2000),
Gamasutra, `http://www.gamasutra.com/features/20000411/sharp_01.htm` [Accessed: August 2001].

[48] Brian Sharp, *Implementing subdivision surface theory*, Game Developer Magazine (February 2000),
Gamasutra, `http://www.gamasutra.com/features/20000425/sharp.htm` [Accessed: August 2001].

[49] Heung-Yeung Shum and Richard Szeliski, *Panoramic image mosaics*, Microsoft Research Technical Report MSR-TR-97-23 (May 1997),
Microsoft Research
`http://www.research.microsoft.com` [Accessed: April 2001].

[50] James Stewart, *Calculus, third edition*, Brooks/Cole Publishing Company (1995).

[51] Ivan Sutherland, *A head-mounted three dimensional display*, Fall Joint Computer Conferences, AFIPS Conference Proceedings (1969), 757–764.

[52] Tomas Svoboda, Vaclav Hlavac, and Tomas Pajdla, *Epipolar geometry for panoramic cameras*, Fifth European Conference on Computer Vision (June 1998), 218–232.

[53] Tomas Svoboda, Tomas Pajdla, and Vaclav Hlavac, *Central panoramic cameras: Design and geometry*, Technical Report, Faculty of Electrical Engineering, Department of Control Engineering, Center for Machine Perception, Czech Technical University (February 1998).

[54] Richard Szeliski and Heung-Yeung Shum, *Creating full view panoramic image mosaics and environment maps*, Proceedings of SIGGRAPH 97 (1997), 251–258,
Association for Computing Machinery, Inc.
`http://www.acm.org/dl/` [Accessed: June 2001].

[55] Martin Urban, Tomas Svoboda, and Tomas Pajdla, *Transformation of panoramic images: From hyperbolic mirror with central projection to parabolic mirror with orthogonal projection*, Technical Report CTU-CMP-2000-09, Faculty of Electrical Engineering, Department of Cybernetics, Czech Technical University (August 2000).

[56] Graham Walker, Doug Traill, Mike Hinds, Alison Coe, and Matt Polaine, *Visiondome: A collaborative virtual environment*, Article, British Telecommunications Engineering, Vol.16 (October 1996),
British Telecommunications
`http://www.labs.bt.com/people/walkergr/IBTE_VisionDome/` [Accessed: April 2001].

[57] Alan Watt and Mark Watt, *Advanced animation and rendering techniques: Theory and practice*, Addison-Wesley (1992).