Efficient Simulation of Lens Distortion in OpenGL

Yang Lu

A Thesis

in

The Department of Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements for the Degree of Master of Computer Science at Concordia University

Montreal, Quebec, Canada

April 2008

Your file Votre référence
ISBN: 978-0-494-40947-3
Our file Notre référence
ISBN: 978-0-494-40947-3

Abstract

Efficient Simulation of Lens Distortion in OpenGL

Yang Lu

In computer graphics, 3D objects are projected onto the viewing plane in either parallel or perspective mode. As far as perspective projection is concerned, traditional rendering engines are designed to carry out only linear planar projections, as in OpenGL. But what if we expect to get certain distorted views? For instance, we see distorted scenes when we look through a rain drop on the windshield. When we play video games, we also might want to have a pair of "magic" lenses that let us see a deformed virtual world. Currently the nonlinear projections and implementations still remain largely unexplored in 3D graphic environments.

In the first part of this thesis, several nonlinear projection models are investigated and an algorithm is presented to apply these models to perform nonlinear perspective projections and 3D view deformations in real time.

Wide-angle lenses are often used to take landscape photos and they are also useful for pictures of large groups of people, but usually they introduce "barrel" distortions into photos. Researches have been done to remove these distortions from the images. However, for some artistic reasons people may want to keep them. So can we simulate this effect in the computer graphics visual world?

I developed a polynomial approximation model and applied the real-time algorithm introduced in the first part of the thesis to achieve the barrel distortion effect of a wide-angle lens.

An experimental system based on the above algorithm and models has been developed to navigate 3D world with various deformed views.

Acknowledgement

It is a great pleasure to take this opportunity to express my gratitude to my supervisor, Professor Peter Grogono, whose help, stimulating suggestions and valuable hints helped me in all the time of research and writing of this thesis.

I would also like to give grateful thanks to my grandmother, my parents, my brother and sister-in-law, for their constant support and endless encouragement.

Contents

List of Figures

Chapter 1 Introduction

Computer graphics has many applications. For some applications, it is important to present an image that is accurate and precise. For many applications, however, the goal is to create an illusion that is convincing to the eye and the brain. The most obvious example is the use of a sequence of static frames to convey the illusion of motion. The first experiments in 3D graphics used simple colour models and gave the impression that everything was made of plastic. Today's graphics use shading, blurring, fog, and other techniques, to obtain the illusion of realism. For applications where rapid motion is required, such as animation and games, these effects must be achieved without much computation in order to maintain a high frame rate.

In all kinds of 3D computer graphics, 3D models must be mapped onto a 2D screen. The mapping is usually achieved by a parallel or a perspective projection. As far as perspective projection is concerned, traditional rendering engines such as OpenGL and Direct3X are only designed to carry out linear perspective projections. But nonlinear 3D deformation has a wide range of uses. For example, "magic" lenses in video games let you see deformed virtual scenes, and magnifying lenses let you examine the details of distant 3D objects. In addition, nonlinear deformation allows you to correct or simulate camera lens distortion and generate movie morph and distortion effects. So presenting an efficient 3D nonlinear projection algorithm is the main issue that my thesis addresses to.

Researchers have applied nonlinear projections in computer-generated 2D images and 3D virtual environments for a variety of purposes, as described in "Section2.1 related work", however, nonlinear projections and implementations remain largely unexplored in real-time navigations of 3D environments. In my thesis, I introduced a nonlinear perspective projection method that deforms 3D virtual world in real time with less performance degradation.

1.1 Nonlinear Perspective Projection

Although linear projection is used in almost all graphics applications, it may cause undesirable effects. For example, it gives the outer object a wider image than the central object on the projection plane, which contradicts what the eyes see. Eyes use crystalline lenses to present spherical deformation and curved surface image formation phenomena [HECH 01]. In contrast, the perspective projection in traditional rendering engines uses a simple pinhole camera model.

Nonlinear projection in a 3D view has a wide range of applications. For example, "magic" lenses let you see deformed virtual scenes, and magnifying lenses allow you to study the details of distant objects in a 3D scenario. In addition, nonlinear deformation makes it possible to correct or simulate camera lens distortion, and to generate movie morphing and distortion effects. It has many application platforms in conceptual design, computer games and scientific visualization.

Researchers have applied nonlinear projections in computer-generated 2D images and 3D virtual environments for a variety of purposes. However, nonlinear projections and implementations still remain largely unexplored in real-time navigations of 3D environments.

1.2 Motivation for Simulating Wide-Angle Lens Distortion

Wide-angle lenses are useful for packing more into a picture and for capturing images with a slightly unusual perspective. They are frequently used to take landscape photos and they are also suitable for pictures of large groups of people. But wide-angle lenses have a downside: they generally introduce distortions into photos.

The central region of a wide-angle lens is similar to that of a standard lens; however the periphery of a wide-angle lens is shaped to allow a larger field of view. The end result is that the image from a wide-angle lens is distorted, especially in the periphery [POLI 93]. This type of distortion is referred to as "barrel" distortion. It is clear that distortions are small near the center of the image and became progressively greater further away from the center. And the wider the lens, the more pronounced the distortion effect will usually be, as shown in Figure1.1.

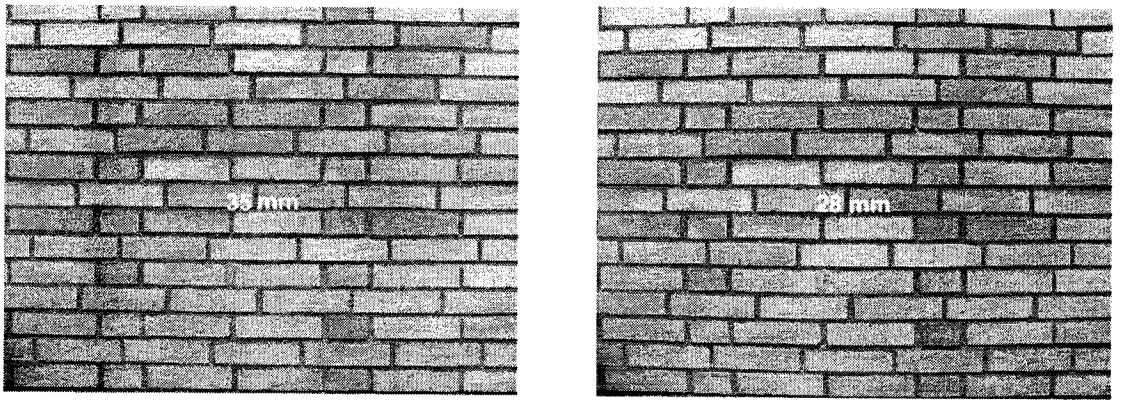


Figure 1.1 an example of barrel distortion of two wide-angle lenses with focal length 35mm and 28mm [KARB 05].

However, for some artistic reasons or otherwise, some photographers like to keep certain amount of these "barrel" distortion effects. But a linear perspective projection will not give "barrel" distortion because it simulates a pinhole camera. Instead of trying to eliminate or correct the effects of wide-angle lens distortion, some graphic artists might like to create it. I developed a polynomial approximation model to simulate "barrel" distortion using a real-time nonlinear perspective projection algorithm.

1.3 Thesis Organization

The remainder of this thesis is structured as follows. In Chapter 2, a real time algorithm is addressed to perform nonlinear perspective projections and 3D view deformations. Several nonlinear projection models are investigated and applied to generate various deformed virtual views. Chapter 3 describes a polynomial approximation model to achieve the barrel distortion effect of a wide-angle lens. Chapter 4 provides the implementation details of an experimental system which allows navigating 3D world

with various deformed views in real time. Finally, Chapter 5 presents conclusion and future work.

Chapter 2 A Real-Time Nonlinear Perspective Projection Algorithm

In this chapter, a real-time nonlinear perspective projection algorithm is presented and several nonlinear models that generate deformed 3D visual effects are analyzed. The approach and models differ from the existing work primarily in that it extends traditional 2D image deformation techniques to 3D space and performs the deformation only on the 2D frames generated by the 3D rendering pipeline. This requires no changes on the traditional 3D graphics pipeline, which makes it a good solution for run-time rendering with less performance impact. An advantage of the approach is that the distortion is more realistic than that of the 2D image distortion. In addition, some of the nonlinear models allow partially nonlinear deformation or linear magnification of 3D views.

2. 1 Related Work

Two-dimensional image deformation and warping are popular methods for manipulating 2D digital images, which implement the interpolation of the in-between images to provide natural morphs between two specified images. Magic lenses and toolglasses use 2D see-through widgets as tools for view filtering and visualization [BIER 93]. This approach presents a 2D magic lens to users as a translucent sheet floating above application windows. Carpendale has carried out research in the area of nonlinear magnification and has applied the results to information visualization [CARP 97].

Three-dimensional deformations extend 2D image deformation and warping and are common in 3D geometry manipulation. In some applications, modifying camera transformation and projection is preferable to changing the rendered object's 3D shape. Bayarri proposed a non-planar perspective projection method based on the computation of new absolute coordinates for transformation through a projection matrix [BAYA 95]. In this method, the projection system first projects each pixel of a 3D object in 3D space to a 3D spherical surface, then to the final 2D plane. However, the extra computation cost by this approach is too high for real-time navigation of complicated 3D virtual scenarios. [YONG 05]

Other than this, some other techniques and methods related to 3D view deformations have been conducted by researchers in recent years. Barr used nonlinear ray tracing to render deformed objects. In his research, a collection of new methods for ray tracing differentiable surfaces is developed. The methods are general, and extend the set of "ray-traceable" surfaces suitable for use in geometric modeling. A smooth surface is treated as a deformation of a flat sheet; the intersection problem is converted to a new coordinate system in which the surfaces are flat, and the rays are bent. These methods are suitable for shading calculations and for mapping textures onto the surface; they can also produce the local coordinate frame values, suitable for anisotropic lighting models [BARR 86].

When constructing 3D geometry for use in cel animation, the reference drawings of the object or character often contain various view-specific distortions, which cannot be captured with conventional 3D models. Rademacher presented a technique called view-dependent geometry, wherein a 3D model changes shape based on the direction it is viewed from. In his work, a view-dependent model consists of a base model, a set of key

deformations (deformed versions of the base model), and a set of corresponding key viewpoints (which relate each 2D reference drawing to the 3D base model). Given an arbitrary viewpoint, his method interpolates the key deformations to generate a 3D model that is specific to the new viewpoint, thereby capturing the view-dependent distortions of the reference drawings [RADE 99].

Besides, some efforts also have been made to correct or reduce the lens distortion effects. For example, Devernay explored nonlinear approaches to correct 3D deformations caused by optical lenses. Most algorithms in 3D computer vision rely on the pinhole camera model because of its simplicity, whereas video optics, especially low-cost wide-angle or fish-eye lenses, generate a lot of non-linear distortion which can be critical. To find the distortion parameters of a camera, Devernay used the following fundamental property: a camera follows the pinhole model if and only if the projection of every line in space onto the camera is a line. Consequently, if we find the transformation on the video image so that every line in space is viewed in the transformed image as a line, then we know how to remove the distortion from the image. The algorithm consists of first doing edge extraction on a possibly distorted video sequence, then doing polygonal approximation with a large tolerance on these edges to extract possible lines from the sequence, and then finding the parameters of the distortion model that best transform these edges to segments [DEVE 01].

Researchers have also applied nonlinear projections with multi-perspective panoramas. Ryoo used an orthogonal cross-cylinder mapping method to implement environmental mapping. Orthogonal Cross Cylinder (OCC) is an object created by intersecting the cylinder of Y-axis and the cylinder of the Z-axis. OCC mapping is a new

method for effectively mapping the environment. This method eliminates the singularity effect caused in the environment maps and shows an almost even amount of area for the environment occupied by a single texel. The surrounding environment can also be stored more effectively through more accurate sampling. Improvement is also achieved in the rendering time of the OCC mapping method and octahedral mapping method based on OCC mapping. Therefore, the OCC mapping method is suitable to be applied on environment navigation systems due to its effective storage of the environment and faster sampling time [RYOO 02]. Furthermore, Svoboda explored panoramic cameras to generate panoramic views of a 3D scene. He reviewed the design and principles of existing panoramic cameras and classified the panoramic cameras w.r.t. their construction, field of view, and existence of a single projective center. Using this classification, he stated a utility of the central catadioptric panoramic cameras [SVOB 00].

The algorithm and deformation models I presented in this thesis differ from the above existing work primarily in that it extends the traditional 2D image deformation techniques to 3D space and performs the deformation only on the 2D frames generated by the 3D rendering pipeline. This method is still based on the normal pinhole camera model and it doesn't touch the basic logic and structure of the traditional 3D rendering pipeline, so it won't introduce much performance impact and is an efficient solution for real-time applications.

2. 2 Background

Regular graphics rendering engines use a viewing volume and project 3D points onto a viewing plane either as parallel rays (orthographic projection) or converging rays (perspective projection). The objects within the viewing volume are eventually projected on to the projection plane through viewing transformation and scan-conversion. This creates a 2D image in the frame buffer. In this thesis, we consider only perspective projection. The nonlinear mathematic models developed later can be easily applied to parallel (orthographic) projection with minor changes.

Assume that we have a perspective projection, associated with the focal length d, the distance between COP (center of projection) and the projection plane. It maps a 3D Point $A_0(x_0, y_0, z_0)$ in the camera-centered coordinate system to an undistorted 2D image point $A_1(x_1, y_1, z_1)$ on the projection plane (Figure 2.1), where

$$x_1 = \frac{x_0}{z_0} d,\ y_1 = \frac{y_0}{z_0} d,\ z_1 = d \qquad (1)$$

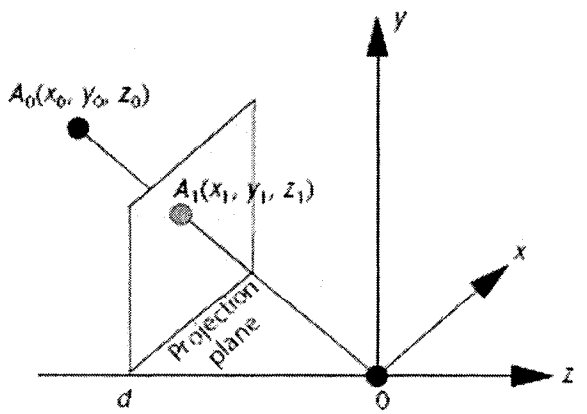
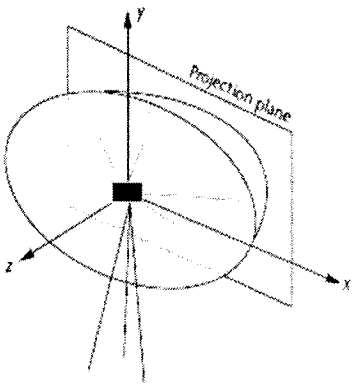
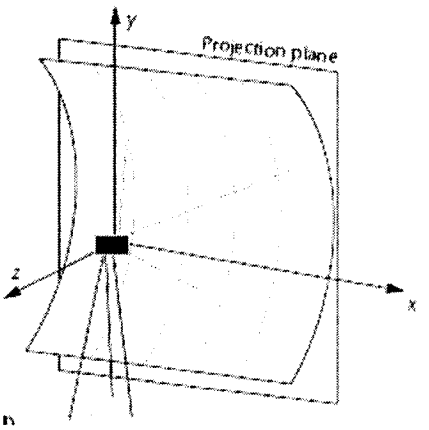


Figure 2.1 Normal linear perspective projection

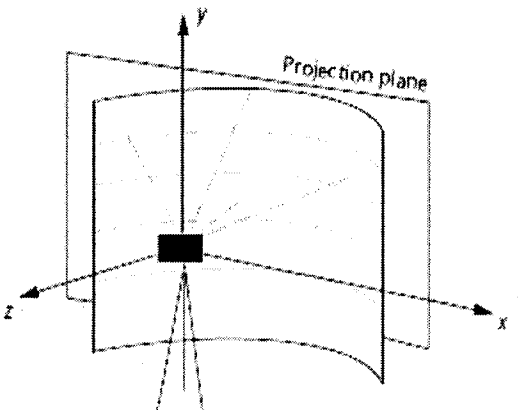
If we put a curved optical lens in front of the projection plane, the direction of the projection rays will be changed and we will see a distorted view. Different curved surfaces of the lenses create different deformation effects. Figure 2.2 shows three curved surfaces to alter projection rays.



(a) Spherical distortion



(b) Vertical distortion



(c) Horizontal distortion

Figure 2.2 Non-linear perspective projections [YONG 03]

In Figure 2.2(a), a half-spherical lens is replaced between camera COP and the projection plane. The projection rays emitted from the COP to the objects in 3D space are altered along all directions when they penetrate the half-spherical lens. The final image on the projection plane, which is made of the intersection (pixels) between the rays and the projection plane, suffers nonlinear distortion in all directions. The deformation level depends on many factors, including the sphere's radius, focus, and other optical features.

Figure 2.2(b) shows a lens with cylindrically-shaped surface that is placed behind the projection plane. The cylindrical surface has its long axis parallel to the x-axis. This curved surface alters projection rays only along the y-axis, and thus only deforms the view vertically.

In Figure 2.2(c), the cylindrical surface of the lens has its long axis parallel to the y-axis. Similarly, the view is horizontally deformed along the x-axis.

We can apply other curved shapes to deform the views. Different curved shapes (lenses) have their own features. In this thesis, I consider only spherical lenses and horizontally and vertically cylindrical curved lenses.

2. 3 Nonlinear Deformation Models

Let's imagine that a virtual curved projection surface is used to distort the view, as shown in Figure 2.3. A perspective projection ray emitted from a point A_0 on a 3D object to the center of projection (COP) passes the projection plane at point A_1, hits the curved surface at A_2, and is reflected back to intersect with projection plane at A_3. The reflecting direction of an incoming ray is determined by connecting the focus of the curved surface and the intersection between the incoming ray and the surface. We use A_3 as the final projection pixel of the A_0 on the projection plane. With the virtual curved projection surface, the projection of point A_0 is distorted from A_1 to A_3 on the projection plane. [YONG 05]

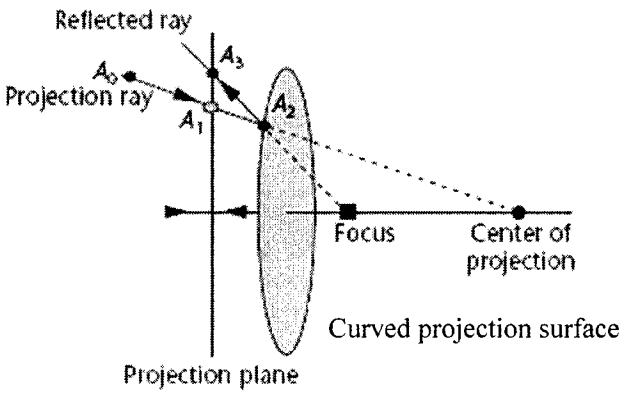


Figure 2.3 Projection surface and reflection

Clearly, the focus and radius of the surface, along with the distance between the projection plane and the surface together determine the view deformation. If the focus is

between the projection plane and COP, the view is nonlinearly magnified. If the focus is moved to COP, it is the regular linear perspective projection. If the focus moves right behind the COP, the view is nonlinearly reduced.

2.3.1 Spherical Deformation Model

The spherical deformation shown in Figure 2.2(a) can be simplified as in Figure 2.4 [YONG 05].

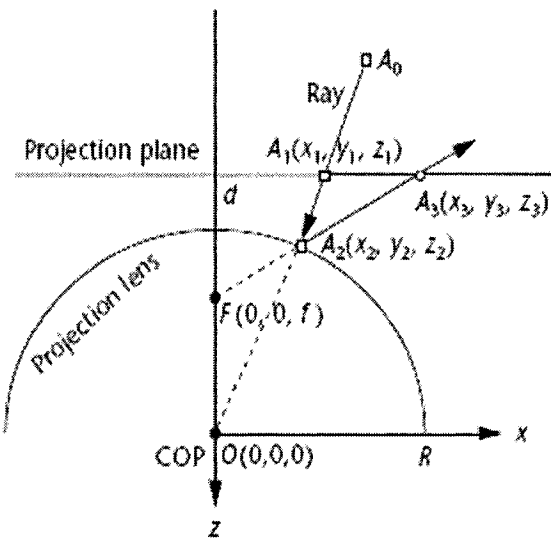


Figure 2.4 Spherical deformation

Let COP be the origin of the coordinate system, and ***R*** and ***F*** represent the radius and the focus of the curved projection surface, respectively. The distance between the

projection plane and COP is d. As discussed above, point A_3 on the projection plane is the nonlinear projection of point A_0 in 3D space.

To calculate the coordinates of A_3, we start with the coordinates of point A_2:

$$x_2 = \frac{R}{\sqrt{x_1^2 + y_1^2 + z_1^2}} x_1$$
$$y_2 = \frac{R}{\sqrt{x_1^2 + y_1^2 + z_1^2}} y_1$$
$$z_2 = -\sqrt{R^2 - x_2^2 - y_2^2}$$

The equation of line FA_3 is $x/x_2 = y/y_2 = (z-f)/(z_2-f)$. Thus, the coordinates of A_3 are:

$$x_3 = \frac{d-f}{z_2-f} x_2,\ y_3 = \frac{d-f}{z_2-f} y_2,\ z_3 = d \qquad (2)$$

Obviously, the bigger $|z_2|$ is, the smaller $|\ (d-f)\ /\ (z_2-f)\ |$ will be. This implies the edge portion of the view is more seriously distorted than the center part. Depending on where the focus is, we have the following three typical deformation effects.

(a) Lens focus locates at COP

If the focus locates at COP, $|f| = 0$. Equation (2) becomes:

$$x_3 = \frac{d-f}{z_2-f} x_2\Big|_{f=0} = \frac{x_2}{z_2} d = \frac{x_1}{d} d = x_1$$
$$y_3 = \frac{d-f}{z_2-f} y_2\Big|_{f=0} = y_1$$
$$z_3 = d \qquad (3)$$

Point A_3 overlaps A_1 on the projection plane. That is the normal linear perspective projection.

(b) Lens focus located between COP and projection plane

In this case, f is a negative value. As shown in Figure 2.4, A_1, the regular linear projection of point A_0, is moved outside to the nonlinear projection point A_3 on the projection plane, creating a nonlinearly magnified view. The edge portion of the half sphere suffers more serious deformation than the center portion. Figure 2.5 shows this deformation effect.

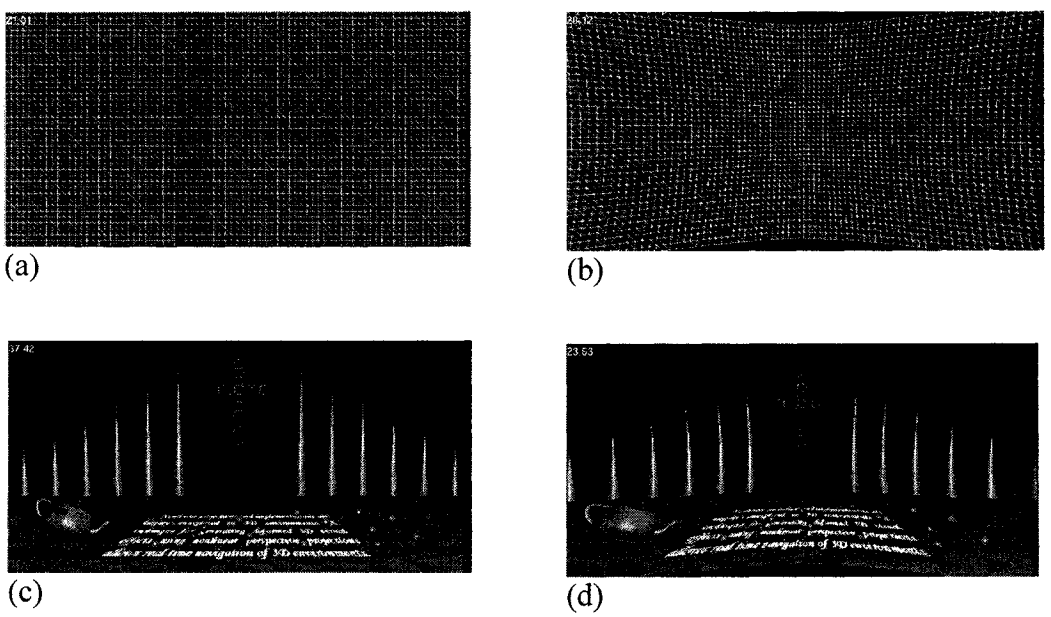


Figure 2.5 Spherical deformation (f = -1.5): (a) normal polygonal mesh, (b) distorted mesh, (c) normal 3D view, and (d) distorted 3D view.

(c) Focus located behind COP along z-axis

If the focus is behind the COP along the positive z-axis, f becomes a positive value. A_3 in Figure 2.4 will locate at the left side of Point A_1. Then the view is nonlinearly reduced. This deformation effect is presented in Figure 2.6.

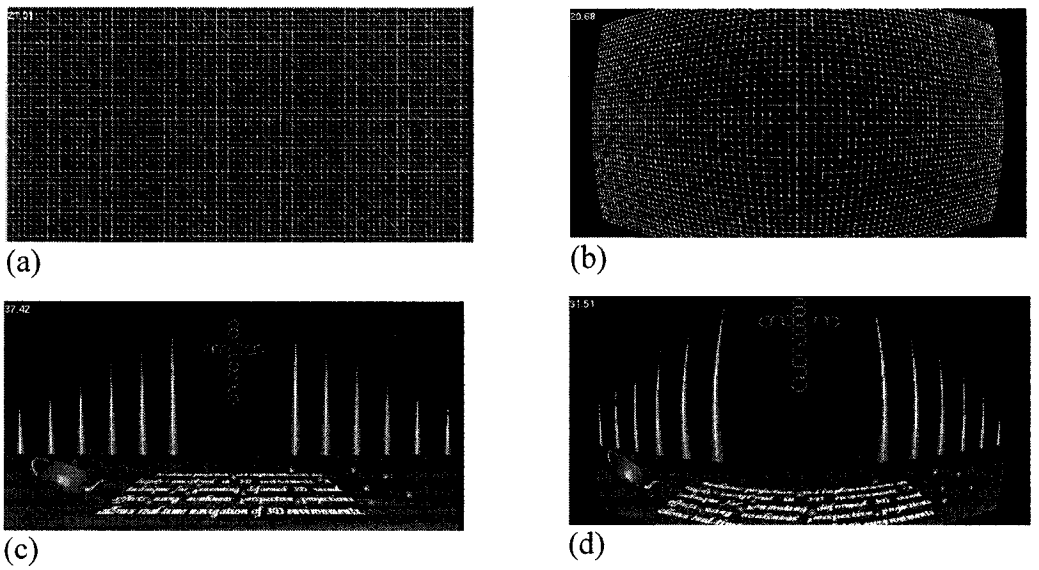


Figure 2.6 Spherical deformation (f = 3.0): (a) normal polygonal mesh, (b) distorted mesh, (c) normal 3D view, and (d) distorted 3D view.

2.3.2 Horizontally Cylindrical Deformation Model

In Figure 2.2 (c), the long axis of the cylindrical surface is parallel to y-axis. The nonlinear deformation only takes place along x-axis, while the projection along y-axis remains as the regular linear perspective projection. In this case, Equation (2) becomes:

$$x_3 = \frac{d-f}{z_2-f} x_2,\ y_3 = y_2,\ z_3 = d \qquad (4)$$

The deformation effect is shown in Figure 2.7.

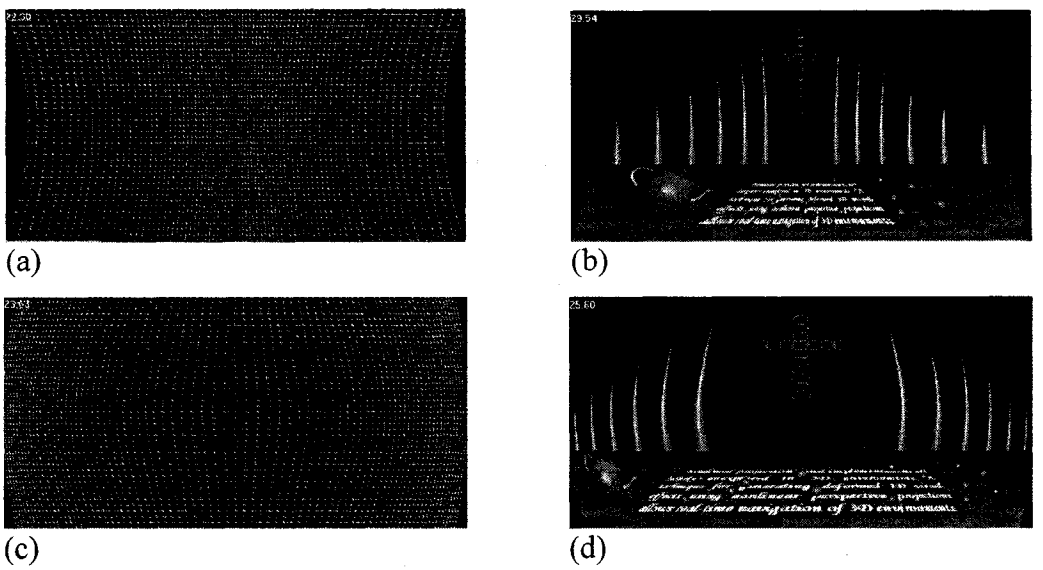


Figure 2.7 Horizontally cylindrical distortion: (a) distorted mesh with f = -2.0, (b) distorted 3D view with f = -2.0, (c) distorted mesh with f = 5.0, and (d) distorted 3D view with f = 5.0.

2.3.3 Vertically Cylindrical Deformation Model

Similarly, Figure 2.2 (b) shows the cylindrical surface with its long axis parallel to x-axis. The nonlinear deformation is carried out along the y-axis only while the projection along x-axis remains as the regular linear perspective. Equation (2) becomes:

$$x_3 = x_2,\ y_3 = \frac{d-f}{z_2-f} y_2,\ z_3 = d \qquad (5)$$

Figure 2.8 shows this deformation with different lens focus.

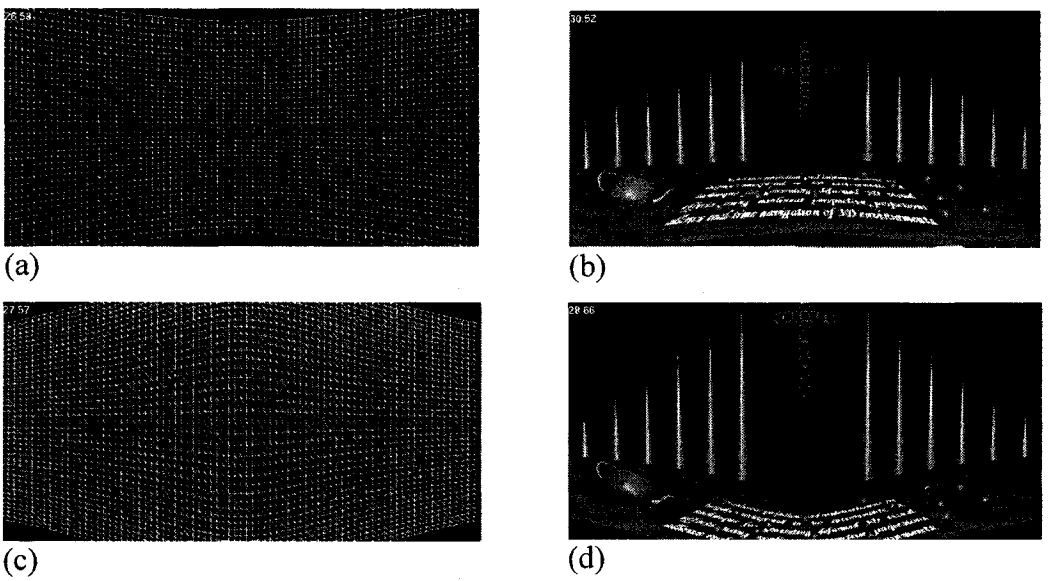


Figure 2.8 Vertically cylindrical distortion: (a) distorted mesh with f = -1.0, (b) distorted 3D view with f = -1.0, (c) distorted mesh with f = 5.0, and (d) distorted 3D view with f = 5.0.

2. 4 3D Distortion Lens Models

Section 2.2 presents three nonlinear projection models which apply deformations onto the entire 3D view space. However, under certain circumstances we might want to achieve deformation of a 3D scene partially, for example, looking through glass covered by dotted raindrops, observing a virtual world through a pair of telescopes, and looking at an object in 3D scene with a pair of "magic" lenses, and so on. In this section, two mathematical models are explored to accomplish this goal — partially deform a 3D scene.

2.4.1 Magic Lens

Imagine holding a "magic" lens and looking at a 3D scene through it. Only the portion of the view under the lens is distorted. The following mathematical model represents a magic lens. As Figure 2.9 shows, let R be the lens radius and let O be the lens center. The lens's optical effect causes point $A_0(x_0, y_0)$ to move to a new position, $A_1(x_1, y_1)$ [YONG 05]. To simplify the calculation, we assume that the lens center is located at the coordinate system's origin.

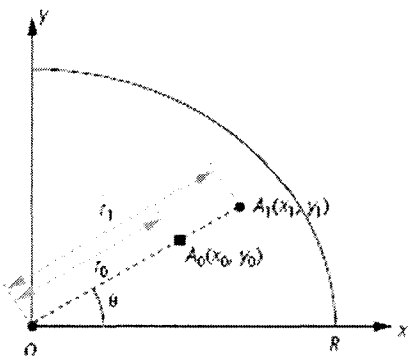


Figure 2.9 Magic Lens

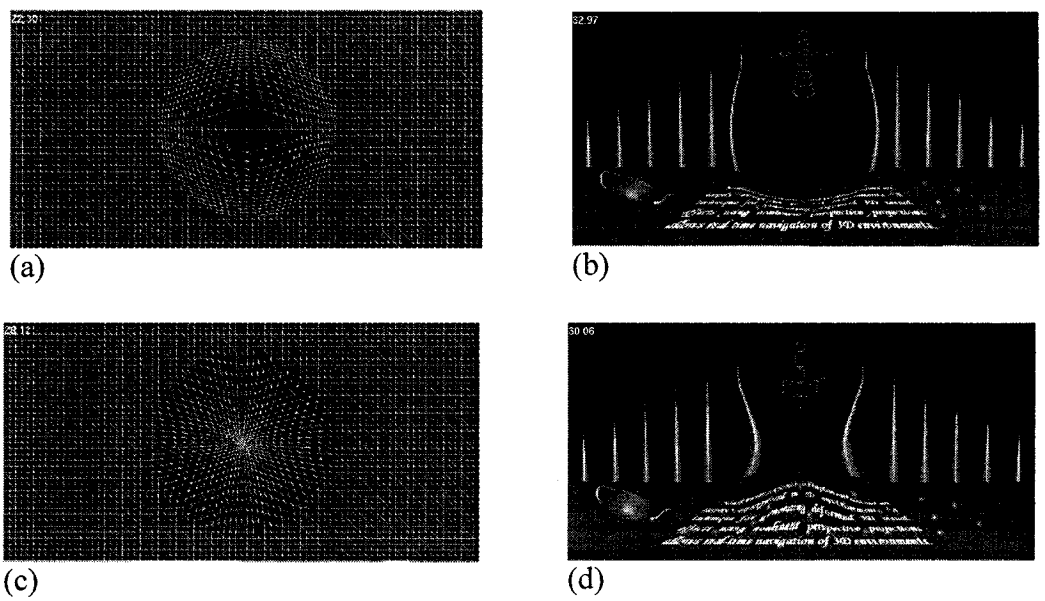
The distance of point A_0 form the origin O is:

$$r_0 = \sqrt{x_0^2 + y_0^2}$$

The distance from the origin O to point A_1, the position of A_0 after distortion, can then be calculated by the function:

$$r_1 = r_0(r_0/R)^{f-1}, \quad f \text{ is a constant distortion coefficient.}$$

Therefore, the coordinates of A_1 can be computed as $x_1 = r_1\cos\theta$, $y_1 = r_1\sin\theta$. The distortion coefficient f determines the lens's optical feature: if $f < 1.0$, it is a convex lens in which the center area under the lens appears pulled out, as in Figures 2.10(a) and 2.10(b); if $f > 1.0$, then it is a concave lens, and its center portion looks as though it's being pushed away from the lens center, as in Figures 2.10(c) and 2.10(d). Obviously, a lens with $f = 1.0$ is a flat lens with no distortion ($r_1 = r_0$).



(a) (b)

(c) (d)

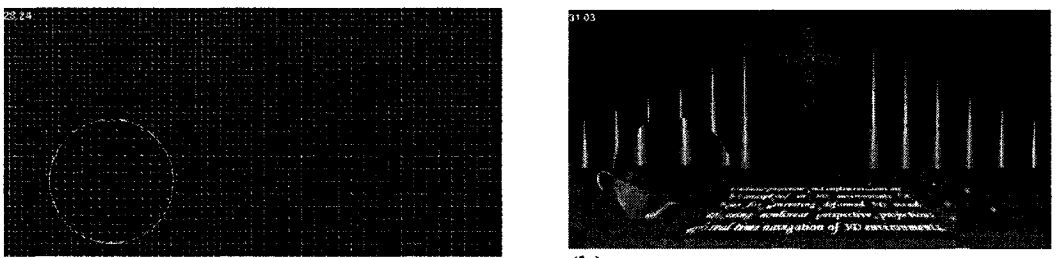
Figure 2.10 Magic lens distortion: (a) distorted mesh with $f = 0.5$, (b) distorted view with $f = 0.5$, (c) distorted mesh with $f = 2.0$, and (d) distorted 3D view with $f = 2.0$.

2.4.2 Magnification Lens

Another typical example of 3D distortion lens is a magnification lens, which simply magnifies everything under it. Here we also assume the lens's center is at the coordinate system's origin (see Figure 2.9). Let $A_0(x_0, y_0)$ be a point under the lens of radius $\boldsymbol{R}$. To magnify the portion under the lens $\boldsymbol{k}$ times, we simply move $A_0(x_0, y_0)$ to its new location, $A_1(x_1, y_1)$:

$$x_1 = \boldsymbol{k}x_0$$
$$y_1 = \boldsymbol{k}y_0$$

If $|OA_1| > \boldsymbol{R}$, we scale x_1 and y_1 down so $|OA_1| = \boldsymbol{R}$. Figure 2.11 shows the magnification effect.



(a) (b)

Figure 2.11 Magnification lens distortion: (a) distorted mesh with $k = 1.37$, (b) distorted 3D view with $k = 1.37$.

2.5 Real-time Nonlinear Perspective Projection Algorithm

In recent years, researches and efforts have been made to integrate nonlinear perspective projection models into computer graphics rendering engines. For example, as discussed in Section 2.1, Bayarri defines a nonlinear projection matrix to calculate the 2D projection coordinates of each pixel of a 3D object [BAYA 95]. But as discussed in "Section 2.1 related work", this method is not suitable for real-time applications.

Here we introduce an easier method — applying the nonlinear distortion models to the 2D RGB image in the frame buffer before it is displayed. This method doesn't touch the rendering pipeline's basic logic in 3D space and it can achieve the same distortion effect. The following steps demonstrate the implementation details [YONG 05]:

Step 1: Create a 2D texture with a void image.

Step 2: Feed the 3D scene to the regular rendering pipeline. After transformation, clipping, normal linear projection, and rasterization, the rendering pipeline generates the 2D image that is stored in the frame buffer.

Step 3: Replace the void image created in Step 1 with the image in the current frame buffer.

Step 4: Redefine the current viewing area as a 2D orthographic viewing region. A 2D polygonal mesh is constructed to cover the entire region, as in Figure 2.12.

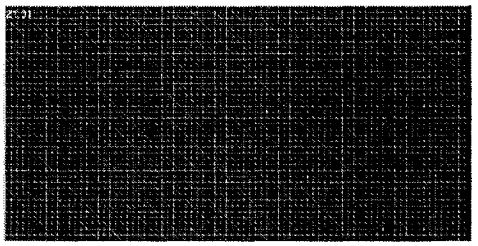


Figure 2.12 2D polygonal mesh.

And then the nonlinear distortion models are applied to alter the coordinates of the polygonal mesh's vertices.

Step 5: Map the 2D texture generated from Step 3 onto the polygonal mesh.

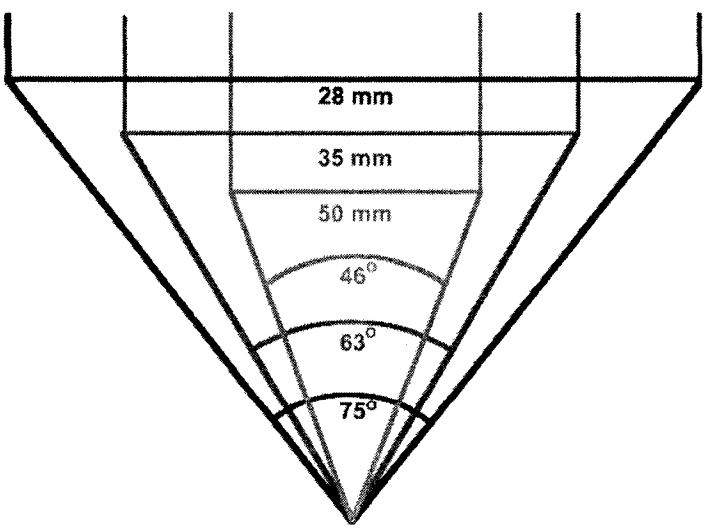
Step 6: Render the resulting distorted view.

In Step 4, we apply the nonlinear perspective projection models (developed in Section 2.2 and 2.3) to reproject the mesh vertices by assuming that the mesh is behind a virtual curved surface or distortion lens. The implementation's extra cost is the time spent computing the coordinates of the deformed mesh vertices (Step 4) and the texture mapping (Step 5), which doesn't significantly impact performance and makes this algorithm an ideal solution to perform nonlinear perspective projection of 3D view in real time.

Chapter 3 Simulation of Wide-Angle Lens Distortion

3.1 Camera Lens Introduction

Camera lenses can be classified in three categories: standard, wide-angle and telephoto. In standard measurements, which assumes a 35mm film with an image area of 36mm x 24mm, a standard lens has a focal length of 50mm. When the focal length is shorter than a standard lens, we have a wide-angle lens; on the other hand, a telephoto lens has a focal length larger than 50mm [KARB 05]. As the focal length decreases, the image gets smaller and the angle of view (FOV) becomes wider. A short-focal-length lens gives a wide-angle view. This is why short-focal-length lenses are also called wide-angle lenses [NAVY 05].



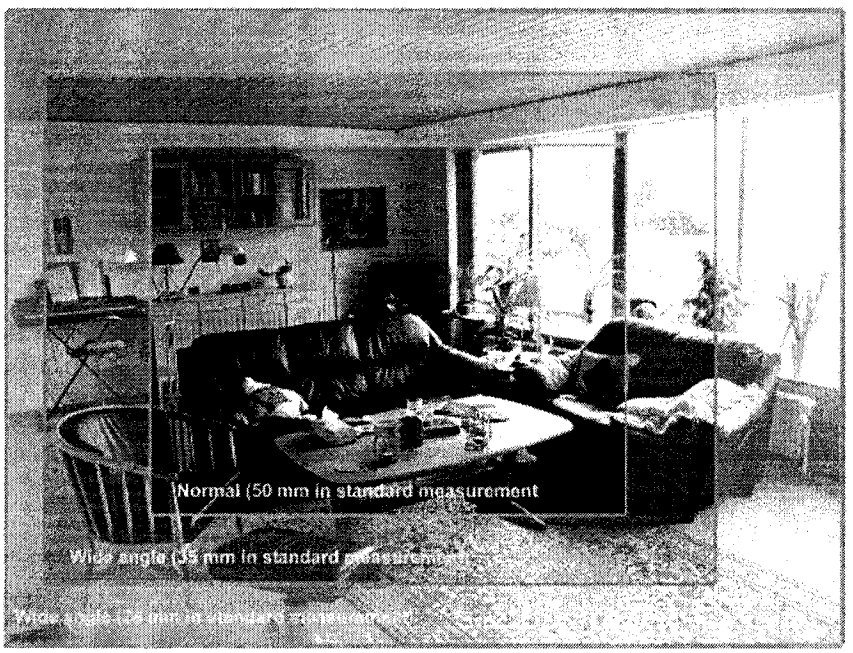


Figure 3.1 The shorter the focal length is, the bigger viewing angle is, the more we can get into a picture [KARB 05].

Most photographers use zoom lenses rather than fixed-focus lenses in most situations. A zoom lens provides adjustment of focal length. Zoom lenses have more complex optics and are more prone to distortion. A typical zoom lens has good performance at its mid-range, but is inferior to a good wide-angle lens at short focal lengths and inferior to a good telephoto lens at long focal lengths.

There are two types of obvious field curvature: ***barrel*** and ***pincushion***. Both types occur with zoom lenses built in to the camera. ***Barrel*** distortion means straight lines in the scene bow outward in images. The closer to the edges of the image, the worse is the barrel distortion. Barrel distortion usually occurs in wide-angle lenses. On the other hand, ***pincushion*** distortion means straight lines in the scene bow inward in images. Similar to

barrel distortion, the closer to the edges of the image, the worse is the pincushion distortion. Pincushion distortion usually occurs in telephoto lenses [SHEN 04].

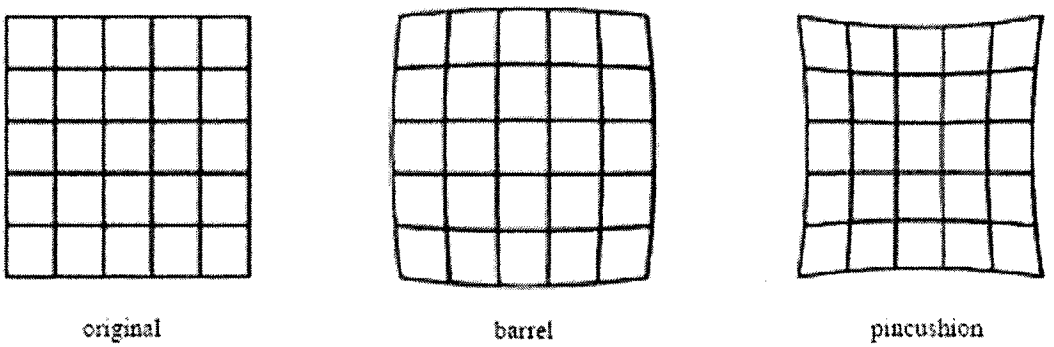


Figure 3.2 Example of barrel and pincushion radial lens distortion.

3.2 Motivation

Wide-angle lenses are great for getting more of the scene into the picture and for capturing images with a slightly unusual perspective. They are frequently used to take landscape photos, and they also come in handy for pictures of large groups of people. However, comparing to standard and telephoto lenses, wide-angle lenses are notorious for distorting what we see. The central region of a wide-angle lens is similar to that of a standard lens; but the periphery of a wide-angle lens is shaped to allow a larger field of view. The end result is that the image from a wide-angle lens is distorted, especially in the periphery [POLI 93]. Some of these distortions are considered unwanted, such as a curved horizon, the stretching of elements located in the corners of the image, or trees and buildings that are tilted up or down.

However, to some extent, photographers desire this distortion. After all, this is why we use wide angle lenses: to see more than what our eyes see, which in turns implies distorting reality as we see it with your eyes. Furthermore, some photographers like to keep some of these "effects" for artistic purposes [BRIO 06].

"*To some extent, wide-angle distortion doesn't bother me, except at times when the photograph takes on the look of only being about being shot with a wide angle lens. But actually, wide angle distortion is part of photography, since photos are visually seen through a glass lens. Do you agree? Some people tend to not bother with distortion. Lens distortion needs not to be considered seriously, for instance, in portrait photography where straight lines are practically non existent and the background covers the area along the edges. Distortion must be considered seriously when investing in a lens for architectural, product, industrial or any type of scientific photography. It is, however, not enough to mean lot in practice.*"

"*Do this and you'll discover a completely different perspective in your shots as short focal lengths have a unique way of slightly distorting what you're photographing - sometimes creating quite a dynamic effect.*"

"*While the photos you get in shooting this way might not give you an accurate likeness of your subject it can certainly give you shots with a distinct and unique look and feel (and some of them will give you reason to chuckle for quite some time).*"---- quotations from on-line photography forums.

Based on the above, instead of trying to eliminate the effects of lens distortion, I developed a nonlinear model to simulate the barrel distortion of a wide-angle lens.

3.3 Barrel Distortion Model

Polynomial approximation is the preferred method of modeling lens radial distortion. Most of the radial distortion-focused research is based on a polynomial approximation model. [MACH 03]

Polynomial models of lens radial distortion can be expressed by the following equation. In this equation, r_u is the undistorted radius, r_d is the distorted radius, and k_i (i = 1, 2, ...) are the distortion coefficients [TSAI 87].

$$r_d = f(r_u) = r_u + k_1 r_u^3 + k_2 r_u^5 + \ldots.$$

For the polynomial model above, the distorted radius of a point r_d is a polynomial in terms of the undistorted radius r_u. Theory predicts that the distortion is especially dominated by the first two terms (r_u , $k_1 r_u^3$) and it has also been found that more elaborate modeling with higher order not only would not help, but also would cause numerical instability [TSAI 87], [WEIM 94].

In this thesis, I apply a polynomial-based radial distortion model with the linear and the cubic term. This has the advantage of being the most robust modeling since it can achieve a significant effect of simulating the wide-angle lens "barrel" distortion, as proved in my experimental system below in Figure 3.3. Also, from inspection it is clear that a negative value of k_1 means the lens is causing barrel distortion while positive value means pincushion distortion. Hence the problem of modeling the distortion is reduced to that of finding the appropriate k_1 that simulates the distorted image. The modeling gives us a formula for finding r_d as a function of r_u:

$$r_d = f(r_u) = r_u + k_1 r_u^3$$

From the following derivation,

$$r_d^2 = (r_u + k_1 r_u^3)^2$$

$$= [r_u (1 + k_1 r_u^2)]^2$$

$$= r_u^2 (1 + k_1 r_u^2)^2$$

$$= (x_u^2 + y_u^2)(1 + k_1 r_u^2)^2$$

$$= x_u^2 (1 + k_1 r_u^2)^2 + y_u^2 (1 + k_1 r_u^2)^2$$

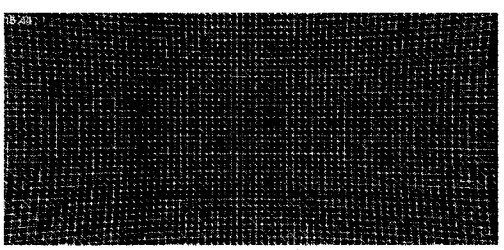
$$= x_d^2 + y_d^2$$

we get the equation:

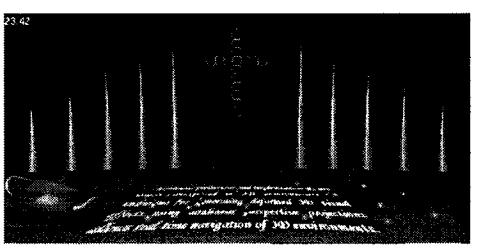
$$x_d = x_u (1 + k_1 r_u^2)$$

$$y_d = y_u (1 + k_1 r_u^2) \quad \text{where, } r_u^2 = x_u^2 + y_u^2$$

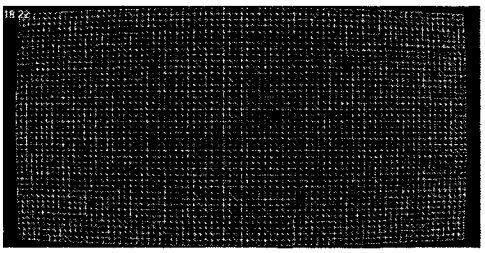
Consistent with theory, the distortion coefficients k_1 start out negative (barrel distortion) for wide angle imaging and then progressively turn positive (pincushion) at telephoto setting.



(a)



(b)



(c) (d)

Figure 3.3 Wide-angle lens barrel distortion: (a) distorted mesh with *k1* = 0.0001, (b) distorted view with *k1* = 0.0001, (c) distorted mesh with *k1* = -0.000035, and (d) distorted 3D view with *k1* = -0.000035.

Chapter 4 Implementation of Nonlinear Projection Navigation system

4.1 Coding

In order to navigate 3D scenario with various deformed views, I developed an experimental nonlinear navigation system by integrating the nonlinear projection models and the real-time algorithm with the OpenGL rendering engine. This makes it possible to examine various nonlinear perspective projection models in real time. Figure 4.1 is a snapshot of the system.

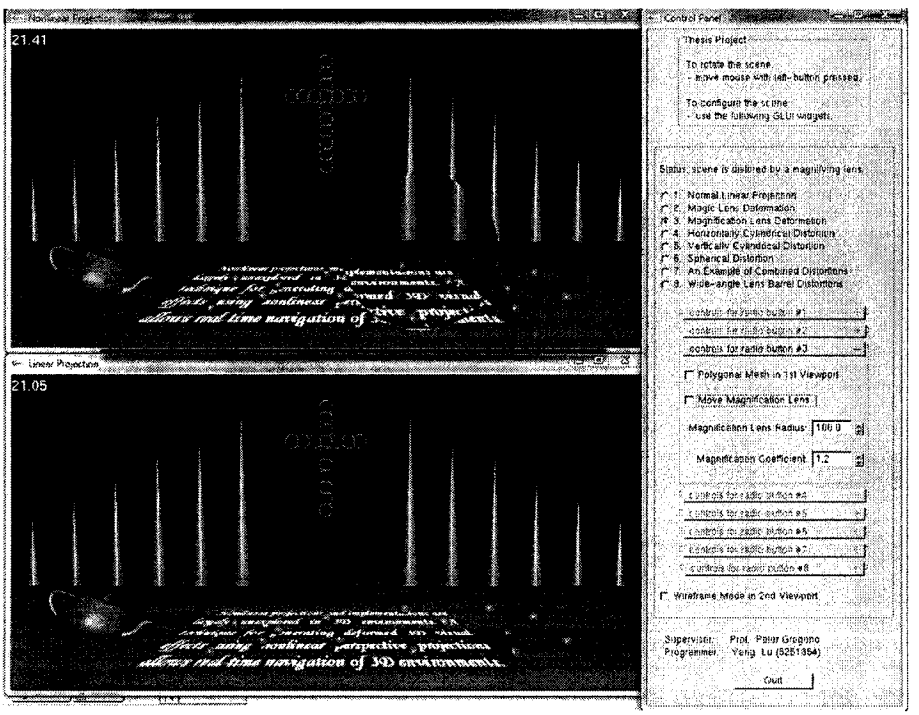


Figure 4.1 An experimental 3D navigation system.

There are two major parts of the code. The first part is the OpenGL implementation of the real-time nonlinear perspective projection algorithm. The second part is the coding of the nonlinear projection models.

4.1.1 Coding for Real-time Nonlinear Projection Algorithms

In Section 2.5, a six-step algorithm is presented to perform nonlinear perspective projections and 3D view deformations in real time. In this experimental system, I implemented these steps of the algorithm by applying the following OpenGL function calls.

Step 1: Create a 2D texture with a void image.

```
//Define a 2D texture
GLuint  sceneTexture;

//Ceate a 2D texture structure
glGenTextures(1, &sceneTexture);
glBindTexture(GL_TEXTURE_2D, sceneTexture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, WIDTH, HEIGHTH, 0,
             GL_RGB, GL_UNSIGNED_BYTE, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
```

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

Step 2: Feed the 3D scene to the regular rendering pipeline. After transformation, clipping, normal linear projection, and rasterization, the rendering pipeline generates the 2D image that is stored in the frame buffer. The function *drawScene()* actually draws the scene, which may be arbitrarily complex.

```
// Draw Scene
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eyeP.x, eyeP.y, eyeP.z, atP.x, atP.y, atP.z, axis_c_Y.x, axis_c_Y.y,
        axis_c_Y.z);
drawScene();
```

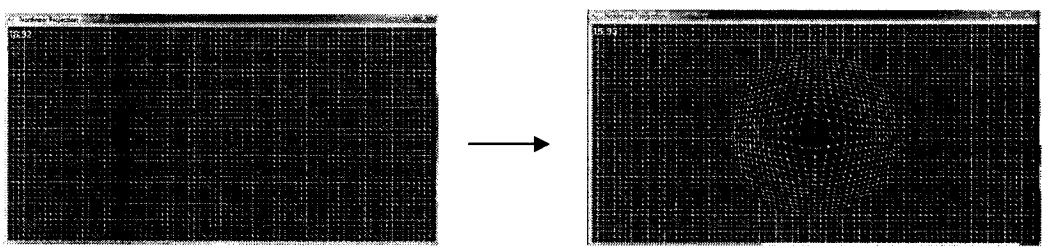
Step 3: Call *glCopyTexSubImage2D()* function to replace the void image created in Step 1 with the image in the current frame buffer.

```
// Copy current frame buffer to the 2D texture map
glBindTexture(GL_TEXTURE_2D, sceneTexture);
glCopyTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 0, 0, WIDTH, HEIGHT);
```

Step 4: Call *gluOrtho2D()* function to redefine the current viewing area as a 2D orthographic viewing region. A 2D polygonal mesh is constructed to cover the entire region (such as Figure 2.5a) and then the nonlinear distortion models are applied to alter the coordinates of the polygonal mesh's vertices.

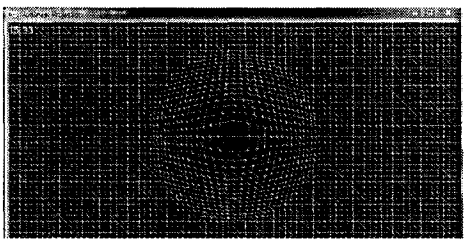
```
//Switch to orthographic viewing method
glMatrixMode(GL_PROJECTION);
```

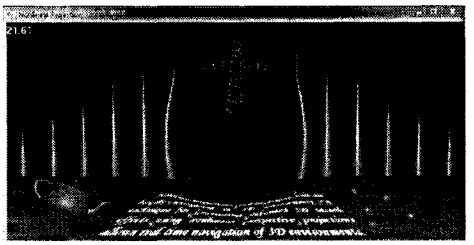
```
glPushMatrix();
glLoadIdentity();
gluOrtho2D(0, glutGet(GLUT_WINDOW_WIDTH), 0,
                glutGet(GLUT_WINDOW_HEIGHT));
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();
glClear(GL_COLOR_BUFFER_BIT);
//Construct 2D polygonal mess and apply nonlinear models to deform it
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glBegin(GL_TRIANGLES);
for (int x = 0; x < meshWidth; x++)
        for (int y = 0; y < meshHeight; y++)
         {
          //Here, we use DeformFunc() to represent one of the nonlinear
          //projection model functions,which apply the corresponding deformation
          //on undeformed mesh vertex (x, y) and create deformed (newx, newy)
          Deform_func(x, y); glVertex2f(newx, newy);
          Deform_func(x+1, y); glVertex2f(newx, newy);
          Deform_func(x, y+1); glVertex2f(newx, newy);

          Deform_func(x, y+1); glVertex2f(newx, newy);
          Deform_func(x+1, y); glVertex2f(newx, newy);
          Deform_func(x+1, y+1); glVertex2f(newx, newy);
         }
```



Step 5: Map the 2D texture generated from Step 3 onto the polygonal mesh.

```
//Texture mapping
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, sceneTexture);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
```





Step 6: Render the resulting distorted view.

4.1.2 Coding for Nonlinear Projection Models

Previously in Section 2.2, 2.3, and 3.3, we described six nonlinear projection models: spherical distortion, horizontally cylindrical distortion, vertically cylindrical distortion, magic lens distortion, magnification lens distortion, and wide-angle lens barrel distortion. In our experimental system, for each of these nonlinear models, an individual function is built to carry out the mapping from the coordinates of the 2D polygonal mesh points to the coordinates of the nonlinearly deformed vertices. In the description of Step 4 in Section 4.1.1, we use a general name for them --- ***DeformFunc()***.

4.1.2.1 Spherical Deformation Model

- Mathematical Equations:

$$x_{temp} = \frac{lens\_radius}{\sqrt{x_{old}^2 + y_{old}^2 + z_{old}^2}} x_{old}$$

$$y_{temp} = \frac{len\_radius}{\sqrt{x_{old}^2 + y_{old}^2 + z_{old}^2}} y_{old}$$

$$z_{temp} = -\sqrt{lens\_radius^2 - x_{temp}^2 - y_{temp}^2}$$

$$x_{new} = \frac{z_{old} - lens\_focus}{z_{temp} - lens\_focus} x_{temp}$$

$$y_{new} = \frac{z_{old} - lens\_focus}{z_{temp} - lens\_focus} y_{temp}$$

```
/*==============================================
/* Calculate (newx, newy) for (oldx, oldy) based on spherical distortion
/*==============================================
void spherical_distortion(float oldx, float oldy, float oldz)
{
 float temp = 0.0, tempx = 0.0, tempy = 0.0, tempz = 0.0;
 temp = sqrt(oldx*oldx + oldy*oldy + oldz*oldz);
 tempx = sphere_radius / temp* oldx;
 tempy = sphere_radius / temp * oldy;
 tempz = -1.0 * sqrt(sphere_radius*sphere_radius - tempx*tempx - tempy*tempy);

 newx = (oldz – sphere_focus) / (tempz - sphere_focus) * tempx;
 newy = (oldz - sphere_focus) / (tempz - sphere_focus) * tempy;
}
```

4.1.2.2 Horizontally Cylindrical Deformation Model

- Mathematical Equations:

$$x_{temp} = \frac{lens\_radius}{\sqrt{x_{old}^2 + y_{old}^2 + z_{old}^2}} x_{old}$$

$$y_{temp} = \frac{len\_radius}{\sqrt{x_{old}^2 + y_{old}^2 + z_{old}^2}} y_{old}$$

$$z_{temp} = -\sqrt{lens\_radius^2 - x_{temp}^2 - y_{temp}^2}$$

$$x_{new} = \frac{z_{old} - lens\_focus}{z_{temp} - lens\_focus} x_{temp}$$

$$y_{new} = y_{temp}$$

```
/*=============================================
/* Calculate (newx, newy) for (oldx, oldy) based on horizontally cylindrical
/* deformation
/*=============================================
void HC_distortion(float oldx, float oldy, float oldz)
{
 float temp = 0.0, tempx = 0.0, tempy = 0.0, tempz = 0.0;
 temp = sqrt(oldx*oldx + oldy*oldy + oldz*oldz);
 tempx = HC_radius / temp * oldx;
 tempy = HC_radius / temp * oldy;
 tempz = -1.0 * sqrt(HC_radius* HC_radius - tempx*tempx - tempy*tempy);

 newx = (oldz – HC_focus) / (tempz - HC_focus) * tempx;
 newy = tempy;
}
```

4.1.2.3 Vertically Cylindrical Deformation Model

- Mathematical Equations:

$$x_{temp} = \frac{lens\_radius}{\sqrt{x_{old}^2 + y_{old}^2 + z_{old}^2}} x_{old}$$

$$y_{temp} = \frac{len\_radius}{\sqrt{x_{old}^2 + y_{old}^2 + z_{old}^2}} y_{old}$$

$$z_{temp} = -\sqrt{lens\_radius^2 - x_{temp}^2 - y_{temp}^2}$$

$$x_{new} = x_{temp}$$

$$y_{new} = \frac{z_{old} - lens\_focus}{z_{temp} - lens\_focus} y_{temp}$$

```
/*================================================
/* Calculate (newx, newy) for (oldx, oldy) based on vertically cylindrical
/* deformation
/*================================================
void VC_distortion(float oldx, float oldy, float oldz)
{
 float temp = 0.0, tempx = 0.0, tempy = 0.0, tempz = 0.0;
 temp = sqrt(oldx*oldx + oldy*oldy + oldz*oldz);
 tempx = VC_radius / temp * oldx;
 tempy = VC_radius / temp * oldy;
 tempz = -1.0 * sqrt(VC_radius* VC_radius - tempx*tempx - tempy*tempy);

 newx = tempx;
 newy = (oldz – VC_focus) / (tempz - VC_focus) * tempy;
 }
```

4.1.2.4 Magic Lens Distortion Model

- Mathematical Equations:

$$r_{old} = \sqrt{x_{old}^2 + y_{old}^2}$$

$$r_{new} = r_{old}\left(\frac{r_{old}}{lens\_radius}\right)^{lens\_focus-1}$$

$$\sin\theta = \frac{y_{old}}{r_{old}},\ \cos\theta = \frac{x_{old}}{r_{old}}$$

$$x_{new} = r_{new} \cdot \cos\theta,\ y_{new} = r_{new} \cdot \sin\theta$$

```
/*=============================================
/* Calculate (newx, newy) for (oldx, oldy) based on magic lens distortion
/*=============================================
void magic_lens(float oldx, float oldy)
{
 float oldr, newr, sin_theta, cos_theta;
 oldr = sqrt(oldx*oldx + oldy*oldy);
 if (oldr <= magic_lens_radius)
 {
  sin_theta = oldy / oldr;
  cos_theta = oldx / oldr;
  newr = oldr * pow(oldr/magic_lens_radius, magic_lens_focus - 1);
  newx = newr * cos_theta;
  newy = newr * sin_theta;
 }
}
```

4.1.2.5 Magnification Lens Distortion Model

- Mathematical Equations:

$$x_{new} = x_{old} \cdot magnification\_\text{coefficient}$$

$$y_{new} = y_{old} \cdot magnification\_\text{coefficient}$$

```
/*=============================================
/* Calculate (newx, newy) for (oldx, oldy) based on magnification distortion
/*=============================================
void magnification_lens(float oldx, float oldy)
{
 float oldr;
 oldr = sqrt(oldx*oldx + oldy*oldy);
```

```
 if (oldr < magnification_lens_radius)
 {
  newx = oldx * magnification_lens_k;
  newy = oldy * magnification_lens_k;
 }
}
```

4.1.2.6 Wide-angle Lens Barrel Distortion Model

- Mathematical Equations:

$$r_{old} = \sqrt{x_{old}^2 + y_{old}^2}$$

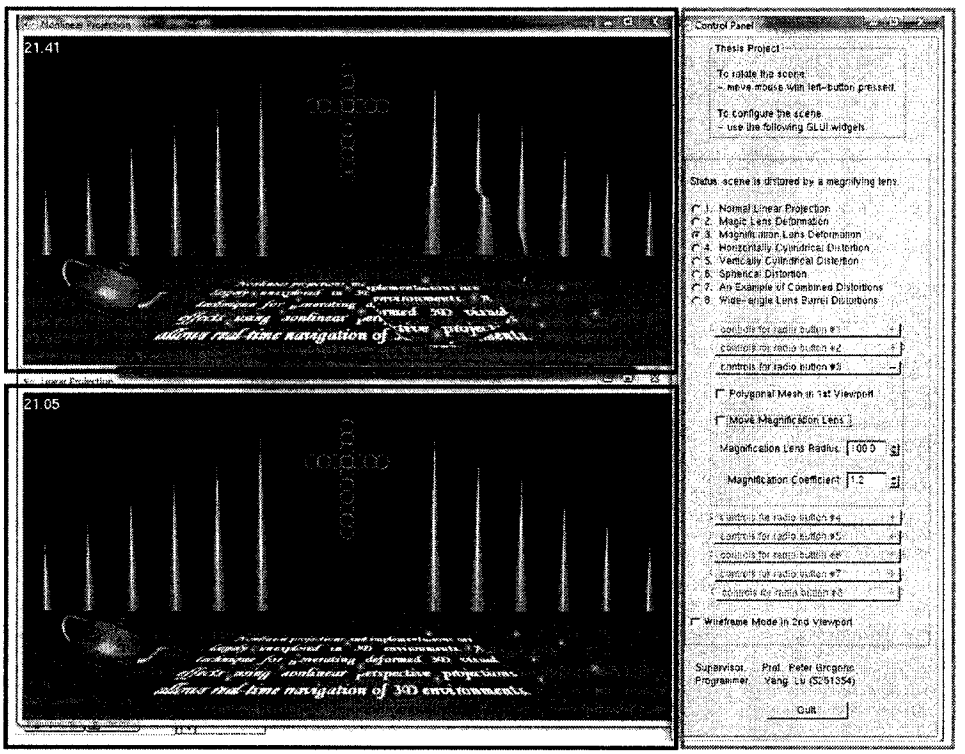
$$x_{new} = x_{old}\left(1 + distortion\_coefficient \cdot r_{old}^2\right)$$

$$y_{new} = y_{old}\left(1 + distortion\_coefficient \cdot r_{old}^2\right)$$

```
/*================================================
/* Calculate (newx, newy) for (oldx, oldy) based on wide-angle lens barrel
/* distortion
/*================================================
void barrel_distortion(float oldx, float oldy)
{
 float temp = oldx*oldx + oldy*oldy;
 newx = oldx * (1 + barrel_k1 * temp);
 newy = oldy * (1 + barrel_k1 *temp);
}
```

4.2 User Interface

As shown in Figure 4.2, the experimental system interface has three panels: a nonlinear panel, a linear panel, and a control panel.



● Nonlinear panel ● Linear panel ● Control panal

Figure 4.2 Three panels of the 3D navigation system.

The top left panel, nonlinear panel, renders the deformed view from nonlinear perspective projections. The bottom left panel, linear panel, displays the normal linear perspective view. Rendering the two views simultaneously makes it convenient to compare the nonlinear deformation effects.

The right panel, control panel, hosts the control widgets and buttons with which users can interactively choose nonlinear perspective projection models and modify their deformation parameters, such as the radius, focus, coefficients. Three sub-panels are organized in this area:

- Title sub-panel:

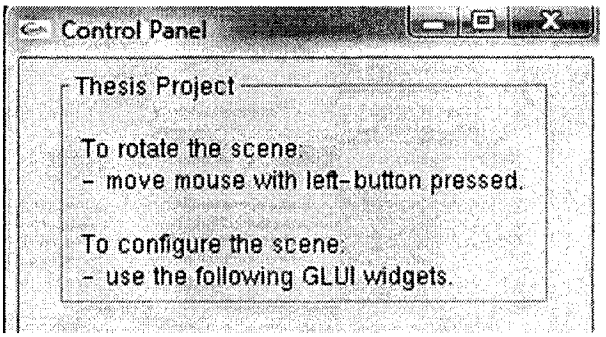


Figure 4.3 Title sub-panel in control panel

- Deformation manage sub-panel:

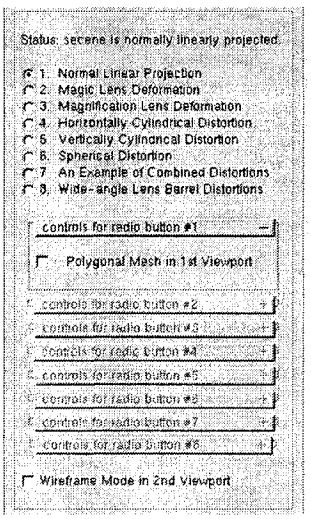
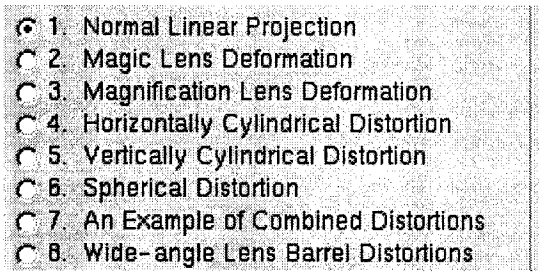
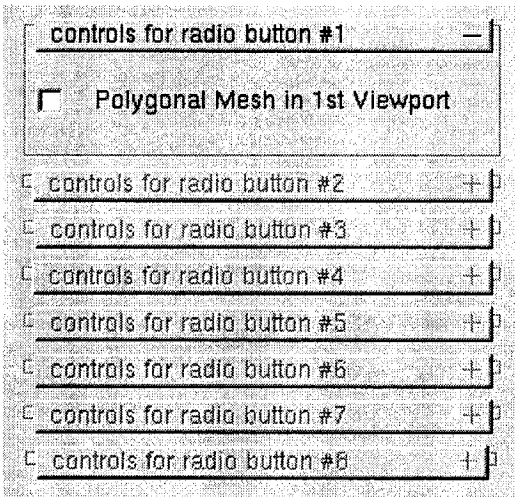


Figure 4.4 Deformation manage sub-panel in control panel

This area prints status information associated with the current deformation option.

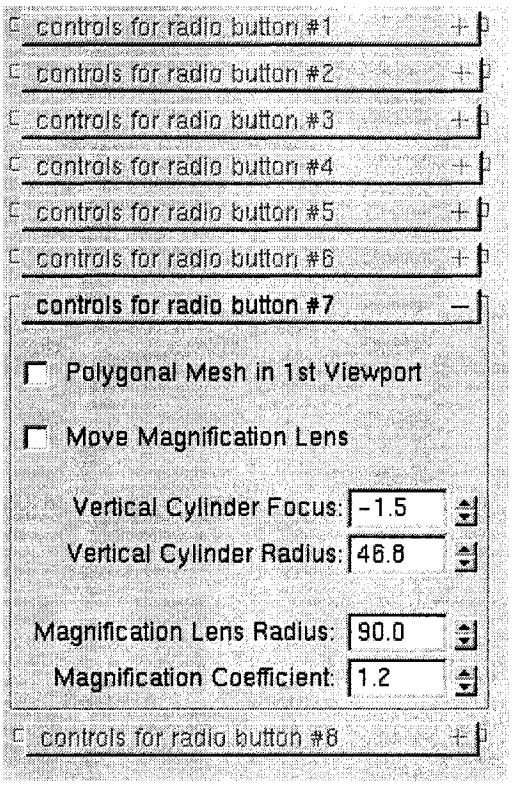


There are eight deformation options provided by the system altogether: six nonlinear deformation models, one normal linear projection option, and an example of combination of various deformations. Once one option is picked, the status information above will be updated.

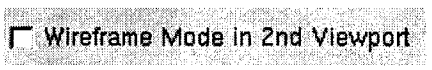


Since different nonlinear projection models have a different set of deformation parameters, I created eight GLUI rollout components corresponding to the eight deformation options. Each rollout component contains the control widgets to configure the parameters associated to the current deformation model. For example, if we choose

Then we will get



At the end of the sub-panel, a check-box is arranged to render the normal linear projection view in "wire-frame" mode, which will help us to examine the complexity of the 3D view.



- Quit button sub-panel:

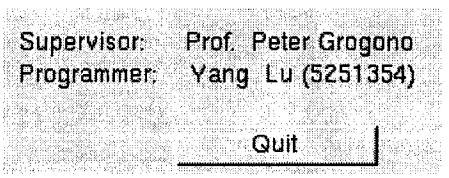


Figure 4.5 Quit button sub-panel in control panel

4.3 Performance Evaluation

I developed this experimental nonlinear navigation system with Microsoft Visual C++ 6.0 and OpenGL. In order to investigate the performance of the system, a series of evaluation tests were conducted on a HP Pavilion laptop running Windows Vista™ Home Premium(32-bit), with AMD Turion™ 64x2 1.9GHz CPU, 2.0GB DDR2 SDRAM, and a NVIDIA GeForce 7150M graphic card with 256 MB of video memory.

As described above, I integrated the real-time nonlinear perspective projection algorithm and the deformation models into the traditional 3D computer graphics rendering engine. The entire process of rendering one frame of a 3D scene can be divided into two major phrases:

- 1st phrase: feed the objects of a 3D scene into the traditional graphics pipeline;
- 2nd phrase: apply the real-time nonlinear perspective projection algorithm to deform the 2D image stored in the frame buffer.

Let's use T_1 to represent the time spent in the first phrase, 3D objects going through the entire graphics pipeline, and T_2 denoting the time consumed in the second phrase, nonlinearly deforming the 2D image. Then we get the frame rate as $\frac{1}{T_1 + T_2}$. While running on the same machine, T_1 depends largely on the scene complexity, whereas T_2 is primarily determined by the rendering region size and the 2D polygonal mesh resolution. Consequently, the system performance is mainly determined by three factors: **scene complexity**, **rendering region size** and **mesh resolution**.

4.3.1 Scene Complexity

Obviously, the performance will be affected by the complexity of 3D scenario. If the rendering area size and the 2D polygonal mesh resolution stay the same, the time slot required by the real-time nonlinear perspective projection algorithm (T_2) won't change as the 3D scene complexity varies, because the size of the image in the frame buffer doesn't change and the workload of deforming 2D mesh remains the same. However, the time for the traditional graphics pipeline to process the 3D scene (T_1) changes. As the 3D scene's complexity increases, the radio of T_2 to the total time (T_1+ T_2) decreases. So theoretically, the more complex the scene is, the less difference of the frame rate between normal linear projection and nonlinear projection will be.

In my experimental system, I provided only one 3D scene. This scene was designed to be visually interesting, and contains a number of different shapes, including a teapot, toruses, spheres, cylinders, and text. Since this scene is complex enough to demonstrate the interesting aspects of the non-linear projections that I developed, I have not included examples of other distorted scenes.

4.3.2 Rendering Region Size

The experimental system's performance is also a function of the rendering region size. Larger rendering regions have larger 2D RGB images stored in the frame buffer, and thus performing nonlinear deformation on these areas requires more calculations. In order to

evaluate the performance impacts introduced by various rendering region sizes, we render the same 3D scene and use a 2D 80x40-grid polygonal mesh covering the rendering area all the time.

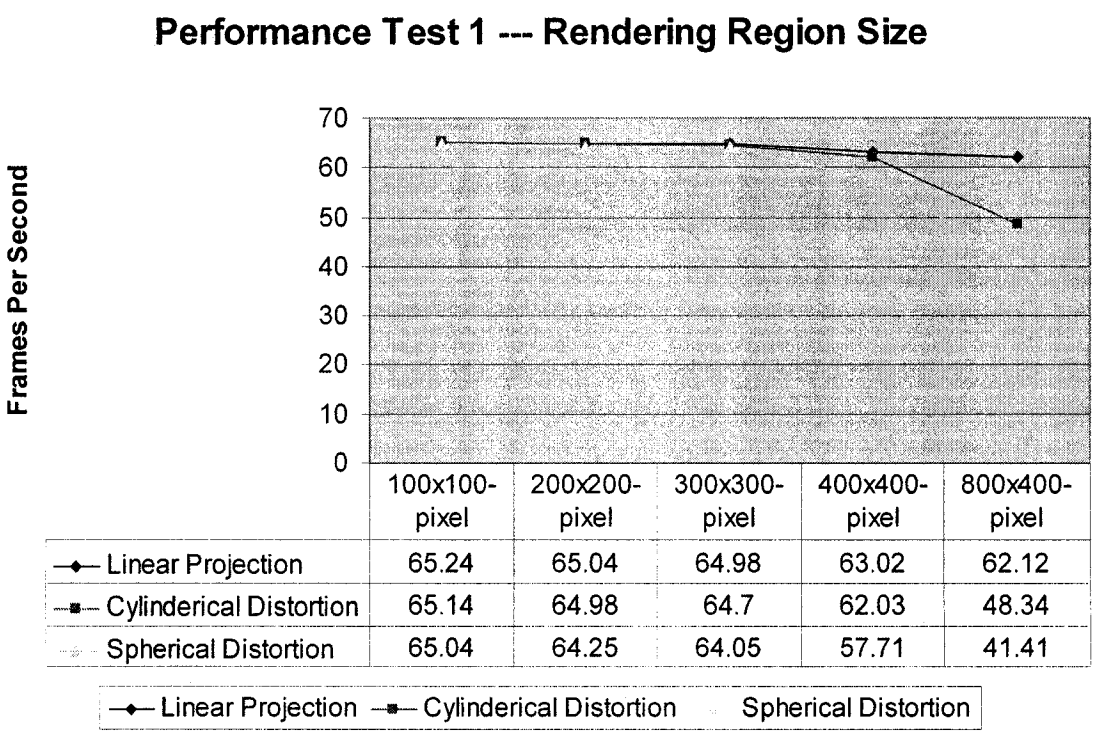


Figure 4.6 Performance of various render region sizes

As Figure 4.6 shows, frame rate decreases as rendering region size increases. The normal linear projection needs more time to prepare more pixels in the frame buffer, and the nonlinear algorithm requires more time to deform the larger frame. Larger rendering regions cause larger T_1 and T_2, which reduces the frame rate. Figure 4.6 also implies that the nonlinear projections perform worse than normal linear projection for larger rendering areas. In addition, as Figure 4.6 shows, the spherical perspective projection is slightly more costly than the cylindrical deformation. This is because the spherical

projection must calculate both the x and y coordinates for each vertex of the 2D mesh (Equation 2), whereas the cylindrical projection calculates only one: the x component (Equation 4) for horizontally cylindrical deformation or the y component (Equation 5) for vertically cylindrical deformation.

4.3.3 Mesh Resolution

To examine the performance differences of various 2D mesh resolutions, I conducted the test by rendering the same 3D scene and a fixed rendering region size of 800x400 pixels. The result is shown in Figure 4.7.

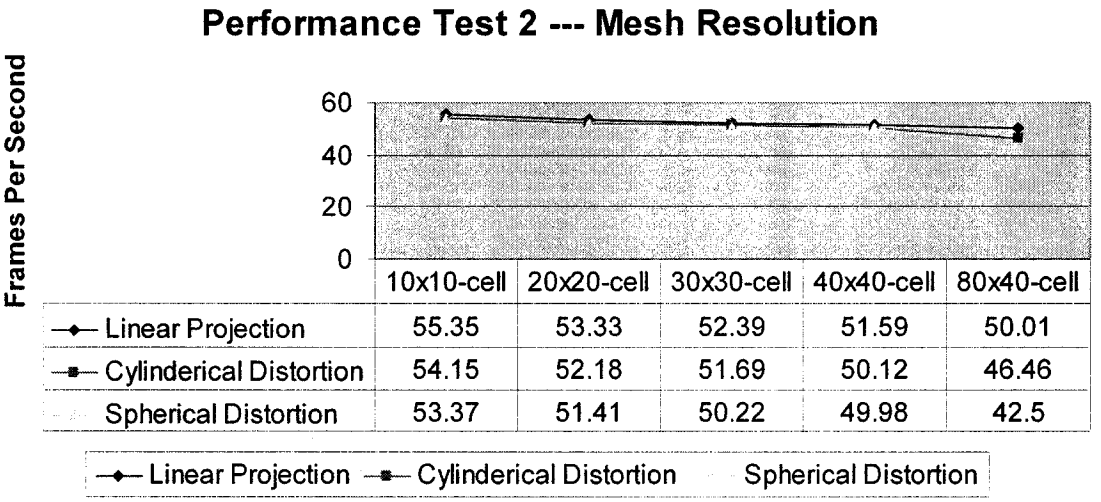


Figure 4.7 Performance of different mesh resolutions

Based on the above figure, a conclusion can be easily made: mesh resolution also affects the experimental system's performance. Higher mesh resolution yields better deformed image quality, but with a slower frame rate, because more vertices in the 2D

mesh cause more nonlinear perspective projection calculations to complete the deformation.

4.3.4 Performance of the Real-time Algorithm

From the above performance experiments, we found that the system performance is mainly determined by three factors: scene complexity, rendering region size and mesh resolution. Now let's do some evaluation about the performance of the real-time non-linear perspective projection algorithm itself.

Suppose that we render the same 3D scene with a fixed rendering region size of 800x400 pixels and a 2D 80x40-grid polygonal mesh covering the rendering area, my experimental system shows that the frame rate per second (FPS) of rendering the scene with normal linear projection is around 50, which is 20ms. After we apply spherical distortion on the scene, the FPS drops to 46 and thus the rendering time is about 21.7ms. This proves that the overhead of applying the real-time nonlinear perspective projection algorithm to deform the 2D image stored in the frame buffer, which is 1.7ms (21.7ms – 20ms), is much less than the normal rendering time (20ms). So we can confirm that the real-time nonlinear perspective projection algorithm is suitable for applications that require a high frame rate.

Chapter 5 Conclusion and Future Work

5.1 Summary

Computer graphics applies either parallel or perspective projective projection to generate 3D views. In fact, as far as perspective projection is concerned, traditional rendering engines such as OpenGL and Direct3X perform only linear projections based on a simple pin-hole camera model. I chose this topic for my research for the following reasons. First, linear projection of 3D objects onto a plane causes some undesirable effects. For example, objects near the side of the viewing area are wider than similar objects at the centre, which contradicts what the eyes see [HECH 01]. Second, nonlinear projection has more and more application platforms in conceptual design, computer games and scientific visualization. Third, little work has been done with nonlinear projection and its implementations still remain largely unexplored in real-time navigations of 3D environments.

In this thesis, a real-time nonlinear perspective projection algorithm is presented and several nonlinear models that generate deformed 3D visual effects are analyzed. The approach and models extend traditional 2D image deformation techniques to 3D space and perform the deformation only on the 2D frames generated by the 3D rendering pipeline.

As a special application case of nonlinear perspective projection, we have also examined the simulation of wide-angle lens distortion. Wide-angle lenses allow a larger part of the scene to be captured from a given distance and they are frequently used to take landscape photos. However, the central region of a wide-angle lens is similar to that of a standard lens; but the periphery of a wide-angle lens is shaped to allow a larger field of view. The end result is that the image from a wide-angle lens is distorted, especially in the periphery [POLI 93]. However, to some extent, photographers desire this distortion and keep it for artistic purposes. In the thesis, I have developed a polynomial-based radial distortion model with the linear and the cubic term to simulate the "barrel" distortion effect of a wide-angle lens.

In addition, in order to navigate 3D scenario with various deformed views, I developed an experimental nonlinear navigation system by integrating the nonlinear projection models and the real-time algorithm with the OpenGL rendering engine. This makes it possible to examine various nonlinear perspective projection models in real time.

5.2 Conclusion

Based on the research we have done so far on this topic, we have reached the following conclusions:

- The six-step nonlinear perspective projection algorithm has been proved to be a feasible solution for real-time rendering with less performance impact, because it requires no changes on the traditional 3D graphics pipeline, and comparing to the regular linear perspective projection, the only extra cost consumed by this algorithm is to compute the

coordinates of the deformed mesh vertices (Step 4) and the texture mapping (Step 5). Our experiments show that we can still achieve real-time navigation of 3d virtual worlds when the rendering region size is not too big.

- The polynomial-based radial distortion model with the linear and the cubic term is good enough to simulate the "barrel" distortion effect of wide-lenses. Any more elaborate modeling with higher order not only would not help, but also would cause numerical instability [TSAI 87], [WEIM 94].

5.3 Future Work

The nonlinear perspective projection can be used to simulate the deformed effects of various distortion mirrors, such as a car blindspot mirror, a carnival mirror or an illusion mirror. In this thesis, we have explored two mathematical models — magic lens and magnification lens — to perform partial 3D scene distortion. And there are some other applications possibilities in this direction. For example, we can simulate the partial nonlinear distortion caused by raindrops on a glass window or car windshield. These applications would be useful for developing more realistic real-time training systems, such as services for civil and military aviation industry.

Implementing the real-time algorithm presented in this thesis is fairly straightforward because it only performs the computation of nonlinear perspective projection models on each frame generated by the 3D rendering pipeline. A possible next step might be the optimization of the algorithm by improving its performance (using a hardware implementation and stencil buffer, for example) and animating other view distortions.

Also worthy of effort is the efficient integration of the nonlinear algorithms into 3D computer games.

As mentioned in Section 4.3.1, another work in the future is to build more 3D scenes into the experimental system I developed. In this way, we can investigate the performance evaluation with different scene complexities, which is not available now because only one 3D scene is provided in the experimental system.

References

[HECH 01] Eugene Hecht, *Optics*, 4th edition, Addison Wesley Professional, 2001.

[BIER 93] E.V. Bier et al., "Toolglass and Magic Lenses: The See-Through Interface", *Proc. ACM Siggraph*, ACM Press, 1993, pp. 73-80.

[CARP 97] M.S.T. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia, "Extending Distortion Viewing from 2D to 3D", *IEEE Computer Graphics & Applications*, vol. 17, no. 4, 1997, pp. 42-51.

[BAYA 95] S. Bayarri, "Computing Non-Planar Perspectives in Real Time", *Computers & Graphics*, vol. 19, no.3, 1995, pp. 431-440.

[YONG 05] Yonggao Yang, Jim X. Chen, Mohsen Beheshti, "Nonlinear Perspective Projections and Magic Lenses: 3D View Deformation", *IEEE Computer Graphics & Applications*, vol. 25, no.1, 2005, pp. 76-84.

[YONG 03] Yonggao Yang, Mohsen Beheshti, Jim X. Chen, "Apply nonlinear projections to deform 3D worlds", *Parallel and Distributed Computing*, Aug. 2003, pp. 809-813.

[YONGGAO 03] Yonggao Yang Chen, J.X. Woosung Kim Changjin Kee, "Nonlinear projection: using deformations in 3D viewing", *Computing in Science & Engineering*, vol. 5, no. 2, 2003, pp. 54-59.

[BRIO 06] Alain Briotinger, "The Eye and the Camera", <http://www.luminous-landscape.com/columns/eye-camera.shtml>,

November 2006, (January 27, 2008).

[NAVY 05] U. S. Navy, "Conjugate Foci", <http://64.78.42.182/sweethaven/MiscTech/Photog01/default.asp?unNum=1&lesNum=7>, *SweetHaven Publishing Service*, (Feb. 03, 2008).

[POLI 93] Jeffrey Poliner, Robert Wilmington, Glenn K. Klute, Angelo Micocci, "Evaluation of Lens Distortion Errors in Video-Based Motion Analysis", <http://www.arielnet.com/main/adw-10e.html>, *NASA Technical Paper 3266*, 1993, (Feb. 03, 2008).

[KARB 05] Michael Karbo, ELI Aps., "Karbo's Photo Book", Chapter 34, <http://www.karbosguide.com/books/photobook/chapter34.htm>, September 2005, (Feb. 04, 2008)

[TSAI 87] Roger Y. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses", *IEEE Journal of Robotics and Automation*, vol. 3, no.4, Aug 1987, pp. 323-344.

[WEIM 94] G. Wei, S. Ma, "Implicit and explicit camera calibration: theory and experiments", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 16, no.5, May 1994, pp. 469-480.

[MACH 03] Lili Ma, Yangquan Chen, Kevin L. Moore, "Rational Radial Distortion Models with Analytical Undistortion Formulae", *CoRR: Computer Vision and Pattern Recognition*, 2003.

[SHEN 04] Ching-Kuang Shene, "Lens Overview", < http://www.cs.mtu.edu

/~shene/DigiCam/User-Guide/990/ON-CAMERA-LENS/overview.html>, 2004, (Feb. 09, 2008).

[BARR 86] A. Barr, "Ray Tracing Deformed Surfaces", *Proc. ACM Siggraph*, ACM Press, 1986, pp. 287-296.

[RADE 99] P. Rademacher, "View-Dependent Geometry", *Proc. ACM Siggraph*, ACM Press, 1999, pp. 439-446.

[DEVE 01] F. Devernay and O. Faugeras, "Straight Lines Have to be Straight", *Machine Vision and Applications*, vol. 13, 2001, pp. 14-24.

[RYOO 02] S. T. Ryoo and K. H. Yoon, "Full-View Panoramic Navigation Using Orthogonal Cross Cylinder", *J. Winter School of Computer Graphics*, 2002, pp. 381-388.

[SVOB 00] T. Svoboda and T. Pajdla, "Panoramic Cameras for 3D Computation", *Proc. Czech Pattern Recognition Workshop*, Czech Soc. for Pattern Recognition, 2000, pp. 63-70.