Scene Depth Reconstruction on the GPU: A Post Processing Technique for Layered Fog

Tianshu Zhou, Jim X. Chen, and Peter Smith

George Mason University George Mason University Member IEEE Computer Society
tzhou@gmu.edu, jchen@cs.gmu.edu, peter_smith@computer.org

Abstract. Realism is a key goal of most VR applications. As graphics computing power increases, new techniques are being developed to simulate important aspects of the human visual system, increasing the sense of 'immersion' of a participant in a virtual environment. One aspect of the human visual system, depth cueing, requires accurate scene depth information in order to simulate. Yet many of the techniques for producing these effects require a trade-off between accuracy and performance, often resulting in specialized implementations that do not consider the need to integrate with other techniques or existing visualization systems. Our objective is to develop a new technique for generating depth based effects in real time as a post processing step performed on the GPU, and to provide a generic solution for integrating multiple depth dependent effects to enhance realism of the synthesized scenes. Using layered fog as an example, our new technique performs per pixel scene depth reconstruction accurately for the evaluation of fog integrals along line-of-sight. Requiring only the depth buffer from the rendering processing as input, our technique makes it easy to integrate into existing applications and uses the full power of the GPU to achieve real time frame rates.

1 Introduction

For the past three decades, dramatic advances in computer graphics hardware and research have made it possible for computers and rendering systems to closely approximate the physical behavior of the real world. During this time, the graphics subsystem of the standard personal computer has risen in status from a simple peripheral device capable of nothing more than monochrome text displays, to a major computational component with a dedicated communication pathway to the CPU, capable of rendering hundreds of millions of lit, textured polygons per second. The current generation of graphics hardware, consisting of multiple processing units and containing more transistors than the motherboard CPU, is now capable of enormous computational power with a high level of parallelism. The addition of programmable logic to the graphics pipeline has endowed these devices with almost cinematic quality capabilities.

At the same time, it has been recognized that improved accuracy in physical simulations and light transport modeling has not yielded the same level of improvements in human perception of computer generated images [5]. Recently, increasing awareness of the human visual system has lead to an improved understanding of perception.

R. Shumaker (Ed.): Virtual Reality, HCII 2007, LNCS 4563, pp. 165 – 174, 2007.

This in turn started to influence the way to generate 3D graphics. We need to consider the human visual system in generating visually realistic images in order to improve immersion of the human participants in virtual environments (VE).

The effectiveness of sensory-immersion in virtual environments largely relies on the ability of synthesized scenes to match the user's sensory channel. In virtual environments, including augmented reality, simulators and games, perception enhancement through graphics generation process is a major area. Therefore, generating special effects to improve realism is an active research area.

Although we live in a 3 dimensional world, the scenes captured by human eye are 2 dimensional; similarly, everyday visual perception is based on interacting with a 3 dimensional world, but computer generated scenes are typically showed on a 2 dimensional display. Humans have learned to use additional information (depth cues) about the 3 dimensional world to process the 2 dimensional retinal images to perceive space and distance, through individual experiences. Accurate perception of distance from computer graphics is particularly important in immersive interfaces that aim to give a person the sensation of actually being in a virtual world that in fact only exists as a computer model. Adding depth cues into computer generated scenes helps depth perception in computer graphics match the capabilities of human visual system in understanding their real environment.

Effects such as depth of field and fog rely on accurate per pixel scene depth for quality results, yet existing scan-line techniques devote little effort towards generating accurate scene depth information, resulting in artifacts in the generated scenes. Alternatives, such as using special render targets, impose limitations and make system integration difficult.

In developing new techniques for generate depth based effects using rasterization, we developed a new post-processing based scene depth reconstruction technique that can be implemented entirely on the GPU. This technique decouples scene rendering from effects generation in the post-processing step, and allows easy integration of the technique into existing applications. In this paper, we present this new hardware oriented method and use layered fog as an example to demonstrate how this technique achieves high quality results in real time.

This paper is organized as follows: section 2 describes our new technique to reconstruct scene depth and 3D fragment position; section 3 presents how our new technique is used for generating layered fog, as well as the implementation details and integration of the technique to the existing rendering application; Section 4 shows the results; Section 5 concludes with future work.

2 Techniques

2.1 Scene Depth Reconstruction

There are currently two main approaches to generating scene depth information for depth based special effects, both of which lay down linear depth information to an alternate render target. The first is to use customized rendering shaders that can output linear scene depth values at the same time as rasterizing the scene in the frame buffer.

The second approach is a separate rendering pass to generate the depth information. The problem with the first approach is that the custom shaders can be difficult to integrate into existing rendering pipelines, while the second approach requires significantly more vertex and fragment processing power. In our new method, we reconstruct scene depth values directly from the depth buffer - a by-product of the normal rendering process that is usually discarded. While this solves many of the problems of other approaches, it presents some new ones. The contents of the depth buffer are non-linear, having been transformed by the modelview and projection matrices. In order to reconstruct the original scene depth information, we must transform these values back to linear values. A naive approach to this is to use an inverted modelview/projection matrix, along with normalized x and y values, to compute the full 3D fragment position relative to the camera. The inverse camera modelview matrix is then used to map this into a real world position. However, this is too computationally expensive, so we developed a more efficient method that allows us to reconstruct the fragment scene depth.

Given a point p in world coordinates, the mapping to canonical viewing volume coordinates p_0, is done by the projection matrix P.

$$p_0 = P \cdot p \tag{1}$$

Perspective projection is characterized by the camera properties as follows:

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{-2f \cdot n}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{2}$$

Where l; r; t and b are the left, right, top and bottom edges of the view volume, and n and f are the near and far clipping planes respectively.

With Equation (1) and (2), we can deduce

$$z = \frac{-P_{34}}{\frac{z'}{w'} + P_{33}} \tag{3}$$

where $\frac{z'}{w'}$ is the depth value in the depth buffer, and

$$P_{33} = \frac{f+n}{n-f} \tag{4}$$

$$P_{34} = \frac{-2f \cdot n}{f - n} \tag{5}$$

In our new method, we use the normal rendering pipeline of the application to construct the scene in the frame buffer. We then capture the depth buffer before rendering a single, screen aligned polygon using vertex and fragment shaders, with the depth buffer as a texture. Our optimized algorithms in the shaders are used to reconstruct the scene depth values on a per-fragment basis from the nonlinear values in the depth texture. By taking full advantage of the GPU capabilities, we have reduced the computation to one addition and one division per pixel. As an additional benefit, by taking advantage of the vector capabilities of the GPU, this technique allows the scene depth values of up to four pixels to be computed in parallel if needed. Some special effects, such as depth-of-field can be implemented at this point since they require no further information [10]. However, for layered fog, we need the full 3D spatial position of the fragment.

D Fragment Position Reconstruction. With the scene depth of each fragment available, we can recover the fragment position easily by setting up a normalized eye-to-fragment vector. Given the four corner points of the screen aligned polygon and the camera position as input, the GPU hardware can be used to automatically interpolate the eye-to-fragment vector for use in the fragment shader. The fragment shader can then compute the fragment position relative to the camera given the eye-to-fragment vector and the depth value.

In the next section, we use layered fog as an example to demonstrate our new scene depth and fragment reconstruction technique in generating high quality depth based effects in real time with easy integration.

3 Layered Fog

Fog is formed by a suspension of water droplets in the atmosphere. It causes scattering of light amongst the water droplets, and therefore reduces the contrast of the scene. In computer graphics, the simplest fog model is homogeneous fog, which has a uniform density in all three dimensions. Layered fog, or height dependent fog, introduces a variation into the fog density dependent upon height.

3.1 Problems

In OpenGL, homogeneous fog functions are provided that allow the blending of the fragment color and a fog color based on the distance between the view point and the fragment. Unfortunately, these functions use the fragment depth as an approximation of distance, rather than the true Euclidean distance. This causes a problem when the viewpoint rotates, since the Z depth of an object can change while the Euclidean distance does not. The result of this is that objects can appear out of the fog, or disappear

into the fog, simply as a result of rotating the viewpoint. Fig. 1. shows how these artifacts occur in OpenGL models. The grey gradient box represents the fog distribution. Although the Euclidean distance between viewpoint O and object point P does not change after the viewing plane rotates (viewing direction changes), the depth to P, however, is changed from z to z_α. As a result, the intensity of fogged fragment at point P will change.

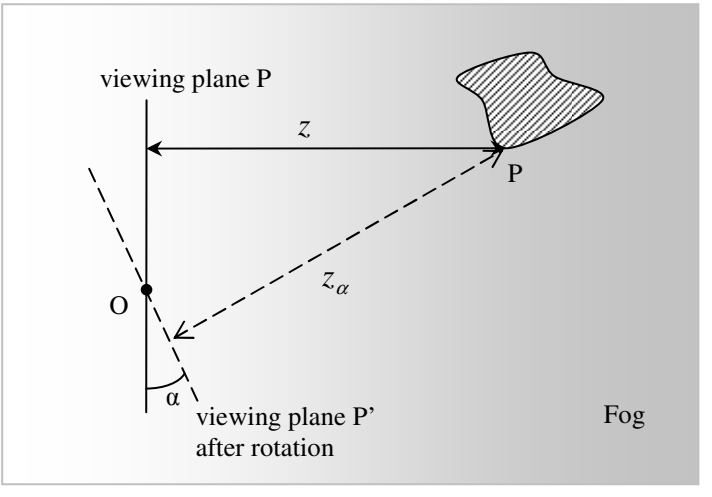

Fig. 1. Undesirable Artifacts of the Standard Fog Model

OpenGL provides a fog extension that allows per vertex specification of depth values for fog computation, enabling generation of effects such as height dependent fog. While it can be used to produce reasonable results, it also presents its own set of problems. Firstly, it can affect performance, since the additional per-vertex data required in this fog extension can potentially become a system bottleneck, in particular, demanding more CPU cycles and bus bandwidth. Secondly, since the depth value is provided once per vertex, it must be interpolated across the polygon. Large polygons can therefore make it very difficult to vary the fog density over small scales. The only way to improve this situation is to reduce the size of the polygons, which can cause further performance issues.

A great deal of effort has been devoted to generating atmospheric effects including fog using global illumination models[2]. While many can produce high quality results, they generally come with a price of high computational cost, and are not yet real time frame rates, particularly when integrated with generic visualization systems. An alternative is to generate visually pleasing fog effect without global illumination consideration. Perlin[6] documented a simple way of producing layered mist by integrating a vertical ray and then enlarging it along line-of-sight. A few techniques have

been implemented using this idea, including texture table lookup [4], or pixel texture [3][1]. However, they are not fast enough.

In this paper, we accurately recover 3D fragment position based on scene depth reconstruction, and apply it when evaluating fog integral along line-of-sight to produce realistic layered fog. Our new method performs fog computation entirely on the GPU at a post-processing step, achieving real time rendering and permitting easy integration into existing visualization applications.

3.2 Computation

We adopt the similar computation proposed by Perlin[6]. It is known[3] that the attenuation of the fog medium is exponentially distributed, suppose

$$f = e^{-F} \tag{6}$$

where F is the integral along the camera-fragment vector for a given fog density function, then

$$F = \int_{camera}^{fragment} \delta(t)dt \tag{7}$$

where δ is the fog density function, t is the space point. For layered fog, δ is only dependent on y, then the above equation can be simplified as

$$F = \frac{l}{\left| y_{fragment} - y_{camera} \right|} \int_{y_{camera}}^{y_{fragment}} \delta(y)dy \tag{8}$$

where l is the Euclidean distance between the camera and the fragment. The blending result of the fog color and the fragment is computed by

$$C = f \cdot C_{frag} + (1 - f) \cdot C_{fog} \tag{9}$$

where f is defined in Equation (6), C_{frag} is the fragment color, C_{fog} is the fog color.

Once the fragment position is computed based on scene depth reconstruction (details in section 2.1), the fog is computed by evaluating the fog density function along the line of sight, from the camera to the fragment (Equation (8)). This is done explicitly in the case of analytical functions (as in our example), or may use pre-computed integrals stored as a texture and provided to the fragment shader. Once the fog density has been evaluated for the fragment, the shader outputs the fog color and alpha value for blending with the scene fragment in the frame buffer.

3.3 Implementation

Our implementation is based upon a custom hierarchical scene management and OpenGL based rendering engine developed by the authors, although any OpenGL based rendering engine should be suitable. Written in C++, our application provides

'hooks' into various parts of the rendering pipeline, allowing us to easily insert our post processing algorithms. The rendering engine itself required no modification.

The first step is to render the scene into the frame buffer, which is done by the application as before. Once the scene rendering is complete, the scene post processing is triggered just prior to the frame buffer swap. At this point, the depth buffer is captured using the OpenGL copy-to-texture functionality. Next, the depth conversion parameters are computed based on the camera properties. The post processing is then initiated by drawing a screen-aligned polygon using the fog color, textured with the depth texture, and using custom vertex and fragment shaders written in the OpenGL Shading Language. The vertex shader is used to set up the eye-to-fragment unit vector, defined in real world coordinates. This is interpolated by the hardware pipeline, and delivered to the fragment shader for use in computing the fragment position. The depth conversion parameters needed for converting the depth buffer values are passed to the fragment shader as uniform variables. The fog computation is performed in the fragment shader based on Equation (8).

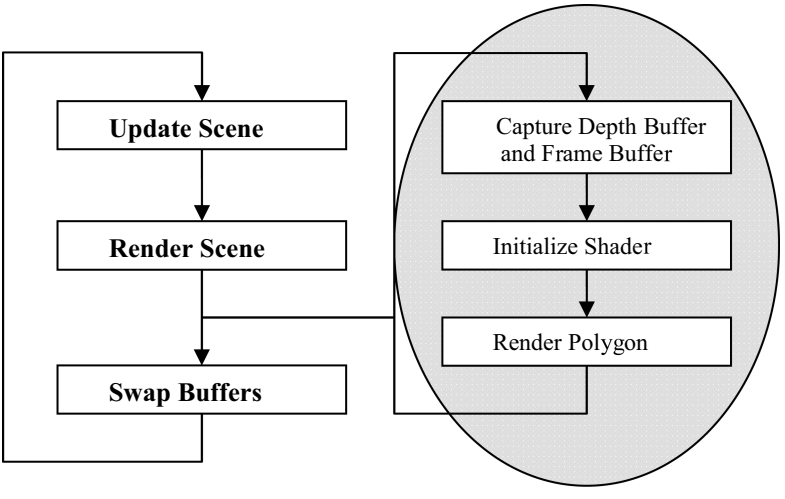

Fig. 2. Application Integration of The Post-processing

To blend the fog color with the fragment color, we take advantage of the standard OpenGL blending functionality and simply compute an alpha value in the fragment shader. The output from the fragment shader is a fragment color consisting of the RGB fog color, and the computed alpha (fog) value. The rendering pipeline takes care of the rest of the work by blending the fog color with the existing fragment in the framebuffer.

Fig. 1 shows the steps in our post processing implementation, and how it integrates into the rendering application.

4 Results

The algorithms in this paper have been implemented on a 2.8GHz Pentium IV platform, with 1GB RAM and an nVIDIA 6800 graphics card, running Linux.

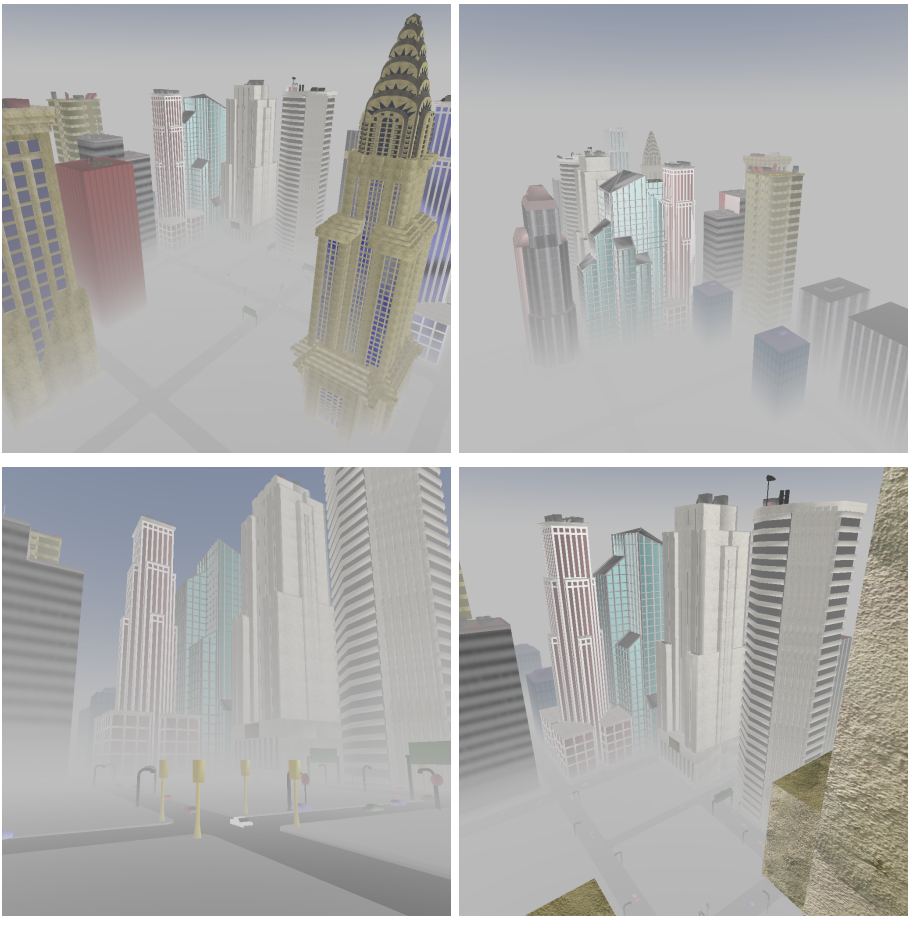

Fig. 3. Layered fog in a city scene

A number of test scenes were chosen using different fog functions at differing screen resolutions, from 512 x 512 up to 1024 x 1024. Figures 3, 4, and 5 shows a selection of images generated using the technique. Figure 3 uses a simple exponential fog function, decreasing with increasing height, to simulate a typical city fog. Figure 4 uses a sinusoidal fog function to create a band of fog just above the surface of the water. Figure 5 uses a simple ramp function to simulate a dry-ice type fog in an indoor scene.

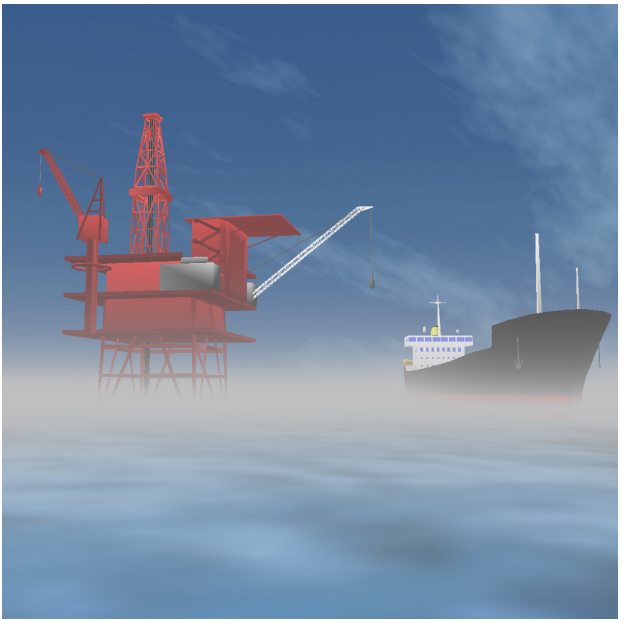

Fig. 4. Low level Sea Mist

Fig. 5. The Cloisters

Performance measurements showed that the technique takes from 0.4 up to a maximum of 2.1 milliseconds, depending upon the screen resolution and the type of fog implemented.

5 Conclusion and Future Work

The layered fog example demonstrates the effectiveness of our new GPU-based scene depth reconstruction technique. This technique has proved efficient and accurate when applied to generate other depth effects such as depth of field [10], and should prove effective for other depth cueing effects such as shadowing and motion blur.

Our GPU based integration framework presents easy integration of single effects into existing rendering applications, as well as combination of multiple effects through component level shader algorithms in the post-processing step. We are currently working on generating multiple depth effects using this integration framework in real time, with minimal impact of the existing rendering applications. The long term goal for this work is to improve realism and human immersion in the virtual environment, and in particular, to improve human subjects experience in perceiving distance and space in virtual environments.

Reference:

1. Biri, V., Michelin, S., Arques, D.: Real-time animation of realistic fog. In: Rendering Techniques 2002 (Proceedings of the Thirteenth Eurographics Workshop on Rendering) (June 2002)
2. Cerezo, E., Perez, F., Pueyo, X., Seron, F., Sillionn, F.X.: A survey on participating media rendering technique. The Visual Computer 21(5), 303–328 (2005)
3. Heidich, W., Westermann, R., Seidel, H., Ertl, T.: Applications of pixel textures in visualization and realistic image synthesis. In: Proc. ACM Sym. On Interactive 3D Grahpics, pp. 127–134 (April 1999)
4. Legakis, J.: Fast muilti-layer fog. In Siggraph'98 Conference Abstracts and Applications, p. 266 (1998)
5. McNamara, A.: Visual Perception in Realistic Image Synthesis. Computer Graphics Forum 20(4), 211–224 (2001)
6. Perlin, K.: Using gabor functions to make atmosphere in computer graphics. http://mrl.nyu.edu/perlin/experiments/garbor/ (year unknown)
7. Rost, R.J.: OpenGL Shading Language, 2nd edn. AddisonWesley, London, UK (2006)
8. Woo, M., Neider, J., Davis, T.: OpenGL Programming Guide, 5th edn. Addison Wesley, Reading (2005)
9. Zdrojewska, D.: Real time rendering of heterogenous fog based on the graphics hardware acceleration (2004) www.cescg.org/CESCG-2004/web/Zdrojewska-Dorota/
10. Zhou, T., Chen, J.X., Pullen, M.: Accurate Depth of Field Simulation in Real Time. To appear in Computer Graphics Forum