Real-time stereo imaging of gaseous phenomena

Tyler M. Johnson*, David F. McAllister[Δ]
Dept. of Computer Science, NC State Univ., Campus Box 8206, Raleigh, NC USA 27695-8206

ABSTRACT

Gaseous phenomena such as clouds, fog, and mist have been difficult to render in realistic monoscopic imaging environments. Such phenomena are transparent, cast shadows, have dynamic behavior, and are of variable density. This paper describes a method based on splatting, billboarding, and alpha-blending that works well in a realistic real-time stereo imaging environment. Splatting is used to reconstruct a discretely sampled 3D volume to produce a 2D image with appropriate density. Efficient reconstruction is gained through the use of texture-mapped billboards, whose transparencies are determined by a Gaussian reconstruction kernel. To achieve the fastest rendering, it is possible to orient all billboards to face the viewplane rather than the viewpoint. The parallax error introduced by this approach is analyzed. The authors give examples to illustrate how the number, transparency, color, and size of the billboards in a scene can be used to create different effects. The discussion does not treat the problems of self-shadowing or dynamic behavior, although the approach could be used as the basis for simulating both.

Keywords: rendering, splatting, billboarding, stereo, fog, mist, clouds

1. INTRODUCTION

1.1 Background

Stereo imaging has been used for over a century to provide depth cues in both synthetic and photographic scenes. The use of stereo requires that two views of the same scene be delivered to the viewer; one to the viewer's left eye and another to the right eye. The *horizontal parallax* or *binocular disparity* of a point in a scene is the difference between the position of the point in the left eye view vs. the position of the corresponding point in the right eye view. Binocular disparity enables the human visual system to "fuse" the images and perceive a "true" three-dimensional or stereo image[11]. Any method for presenting the two eye views must prevent *ghosting* or *cross talk*, i.e., the left eye must not be able to see the right eye view and vice versa. The authors will use the term "stereo" to mean that the left and right eyes see different perspective views of a scene allowing the human visual system to perceive depth, as opposed to "monoscopic," which will mean that both eyes see the same perspective view of the scene. The term 3D will refer only to models from which correct parallax for left and right eye views can be computed.

The rendering of the left and right eye images does not necessarily require twice the work of rendering only a monoscopic view[11,12]. Often the many forms of coherence can be used to reduce the rendering time. It has been shown by the authors and others however, that methods that work well for monoscopic rendering do not necessarily extend naturally to stereoscopic rendering[6,7]. In addition, the extension of monoscopic methods based on texturing and compositing can suffer artifacts that produce ambiguous depth cues and ghosting. Stereo methods must be based on models that produce visually consistent horizontal parallax[11].

The rendering of natural gaseous phenomena such as clouds, fog, and mist has been a research topic in computer graphics for many years. Such phenomena are transparent, cast shadows, have dynamic behavior, and are of variable density. These phenomena are also difficult to render in realistic monoscopic imaging environments. Real-time monoscopic techniques have been described recently for gaming applications. Additionally, attempts using physical modeling based on fluid dynamics have been successful, but are compute-bound and not suited for real-time rendering environments[4].

* esox@bellsouth.net
Δ mcallist@ncsu.edu

The focus of this paper is a technique for rendering gaseous phenomena in a real-time stereo imaging environment. Issues such as illumination and dynamic simulation of these phenomena are not the topic of this paper, although the rendering system described allows for the incorporation of these properties.

1.2 Previous work

Work in the area of real-time cloud rendering in the gaming industry has recently been published. One technique presented represents clouds as hundreds of viewer-oriented billboards positioned in space by an artist to give the cloud shape. These billboards are texture mapped with various alpha-blended textures created by an artist, yielding a cloud-like appearance[4]. *Dynamically generated imposters* are used to speed cloud rendering. An imposter is a 2D image of an object rendered from a certain viewpoint. Since the projected image of an object in a scene remains relatively unchanged for small changes in the viewpoint, e.g. for successive frames rendered with a dynamic camera, an imposter rendered from a slightly different viewpoint can be displayed in the place of the real object with little discrepancy[5]. Rendering an imposter thus saves the cost of re-rendering the entire geometry of the object each frame.

Another method for real-time cloud rendering has also been presented by Mark Harris and Anselmo Lastra[3]. Here clouds are represented as particles which are rendered using a volume rendering technique called *splatting*. By representing clouds as discrete density samples in space, the clouds have a definite geometry which remains consistent while changing viewpoints, a necessary requirement for stereo viewing. This technique also incorporates the use of imposters to speed rendering.

1.3 Splatting

Splatting is a volume rendering algorithm, or a method for rendering volumes represented as discrete density samples in space, and was first described by Lee Westover[1]. An example of a discretely sampled volume would be a collection of points in a cloud where the amount of condensed water vapor was measured. Using this discrete density data, splatting allows the rendering of a reconstructed image of the entire cloud through the use of digital signal reconstruction methods, or convolution. This representation characterizes data such as electron density maps, light wavelength data for galaxies, and medical imaging scans[1].

Convolution can be thought of either as the process of recreating the contribution of all input samples at each output sample or as the process of accumulating each sample's contribution to all output samples. The former is called a feed-backward method and the latter a feed-forward. In the sense of volume rendering, a feed-backward method reconstructs the density of the volume at any point, e.g. screen pixels, using the combined contribution, or density, of all input samples at that point. A feed-forward method however, accumulates the density contribution of each individual input sample to the density at all points in the entire volume. Feed-backward and feed-forward reconstruction will produce identical results.

Prior to splatting, feed-backward algorithms were in use[8]. These methods map the viewplane to the volume data by casting a ray into the volume through each pixel onscreen, determining which samples affect that pixel and to what extent. Splatting is a feed-forward rendering algorithm and reconstructs the sampled volume by recreating the 3D contribution of each density sample to the entire volume. This is done by mapping the contribution of each sample onto any affected pixels on the viewplane. Splatting thus avoids the compute-intensive task of ray-casting and can be executed in parallel. As a tradeoff however, splatting results in significant overdraw; a result of many samples' contributions overlapping onscreen.

Convolution, or the reconstruction of the volume through the contributions of the individual samples, is performed using a 3D *reconstruction kernel*. Such a function defines the contribution of a sample at points other than its own. In splatting, a reconstruction kernel is centered at each sample in R^3, allowing the volume to be reconstructed at any point.

To reconstruct the contribution of a sample at a point (x, y) on the viewplane, the reconstruction kernel is integrated along a ray through (x, y) perpendicular to the viewplane[2]. The contribution of a sample D at an offset (i, j) from the sample's projected point onscreen is

$$contribution(i, j) = \rho \int h(i - D_x, j - D_y, w)dw, \quad (1)$$

where h denotes the volume reconstruction kernel and ρ the density of sample D. Thus, to determine the contribution of a sample at any pixel onscreen, a 1D integration of the reconstruction kernel is required. The contribution of the sample at each pixel is then combined with the sample's density to yield the final density contribution for the sample at each pixel. Using this density contribution as the basis for transparency, the contribution is alpha-blended into the final image. The *screen-space extent* of a sample is defined as the set of pixels for which a given sample has a non-zero contribution and corresponds to the extent of the projection onscreen of a sample's reconstruction kernel in world-space.

To avoid integrating the reconstruction kernel for each pixel in the screen-space extent of each sample, the integration step is precomputed for a generic kernel centered at the origin. A function *footprint*(x, y) is defined which gives the result of integrating the reconstruction kernel along the z-axis at discrete (x, y) positions. The footprint function is evaluated discretely in preprocess, and the results are stored in a table called the *footprint table*. We can then find the contribution of a sample onscreen by mapping all pixels in the sample's screen-space extent to corresponding entries in the footprint table, incorporating some kind of interpolation scheme. Since a mapping from the screen-space extent to the footprint table may not be 1-1, especially under a perspective projection, a mapping must be determined. For a square screen-space extent and footprint table, such a mapping is

$$pixel(i, j) = table\left[\frac{i(S-1)}{(T-1)}, \frac{j(S-1)}{(T-1)}\right], \tag{2}$$

where S is the width of the footprint table, T is and width of the screen-space extent, and (i, j) in the range [*0, T-1*], is the offset in pixels from the top left corner of the screen-space extent.

2. REAL-TIME STEREO IMAGING OF GASEOUS PHENOMENA

Since splatting allows volumes to be represented as a set of density samples in space, volumes have a geometry which remains consistent while changing the viewpoint. This allows stereo views of the volume to be rendered simply by splatting the volume from both eye points. Our method is based on the monoscopic approach described by Harris[3].

2.1 Implementing splatting

We use a generic Gaussian kernel for signal reconstruction. The h in equation (1), then becomes

$$h(x, y, z) = e^{\frac{-(x^2+y^2+z^2)}{2}}. \tag{3}$$

A Gaussian kernel has two very desirable qualities. Firstly, it tapers off gradually as distance from its center increases. This grants our volume the "wispy" appearance characteristic of gaseous phenomena such as clouds and mist, which lack sharp boundaries. Secondly, the Gaussian reconstruction kernel is rotationally symmetric. These two properties are vital to performance enhancements described later.

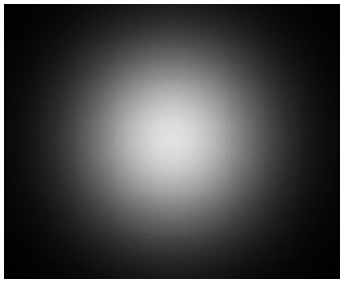

Figure 1: A splatted Gaussian reconstruction kernel.

In the splatting algorithm proposed by Westover, samples lie on a regular grid. However, since splatting maps the contribution of every sample onto the screen independent of other samples, we can allow samples anywhere and think of the reconstruction kernel around the sample as a particle; the concepts of volume reconstruction still apply[3]. Gaseous phenomena in our method are thus represented as a finite set of particles of a certain radius, each consisting of a density, an (*x, y, z*) center, and a color. The density of each particle lies in the range [*0, 1*] and acts as the opacity of the particle at its center. For clouds, mist etc., the density represents the relative amount of water vapor present at the center of the particle. Via splatting, this opacity is ultimately modulated at each pixel in the particle's onscreen projection by the pixel's corresponding entry in the footprint table, allowing the opacity of the particle to vary throughout its extent as determined by the reconstruction kernel. This gives the volume the "wispy" appearance characteristic of clouds and mist. The radius of the particle represents the extent of the reconstruction kernel in world-space and ultimately determines the screen-space extent of the particle. The color of each particle remains uniform throughout its extent, but appears to fade gradually due to the exponential decline in opacity characteristic of the Gaussian reconstruction kernel.

For splatting, many 1D integrations of the reconstruction kernel are required. Integration is performed numerically along the z-axis at sample (*x, y*) values for a generic reconstruction kernel centered at the origin. These integrations represent the relative density observed at (*x, y*) in the projection of the particle. In our case, the reconstruction kernel is the generic Gaussian from equation (3). We performed the integration at uniformly spaced (*x, y*) values over the interval [*-3, 3*], also clamping the interval of integration along the z-axis to this range. We found the range [*-3, 3*] sufficient to capture the nature of the Gaussian while reducing computation time. Outside this range, the contribution of the kernel is minimal and not visible on display hardware. Integration was performed using a composite numerical quadrature scheme of 100 equally-sized intervals.

In splatting a particle, the footprint table is centered around the onscreen projection of a particle. The table values act as relative opacities for the pixels in the screen-space extent of a particle and prevent us from having to integrate the kernel at run-time. Since the reconstruction kernel was integrated for uniformly spaced (*x, y*) values, the footprint table will form an *n x n* array. In mapping integrations to the footprint table, for each table entry (*i, j*) in the range [*0, n - 1*],

$$entry(i, j) = \int h(x_{\min} + i\Delta x, y_{\min} + j\Delta y, w)dw \tag{4}$$

with $\Delta x = \frac{(x_{\max} - x_{\min})}{(n-1)}$ and $\Delta y = \frac{(y_{\max} - y_{\min})}{(n-1)}$. For the clamped Gaussian reconstruction kernel, $x_{max} = y_{max} = 3$, and $x_{min} = y_{min} = -3$. This results in the reconstruction kernel being centered in the footprint table. In addition, after computing the integrations, we scale each entry by the maximal entry, resulting in all table entries lying in the range [*0, 1*]. This range is desirable since the entries in the footprint table reflect the relative contribution of the reconstruction kernel at that point and serve to modulate the opacity of the particle from maximum opacity at its center to lesser opacity as distance from the center increases.

2.2 Rendering

In splatting a particle, three steps must be performed. First, the center of the particle must be projected onto the viewport. Secondly, the screen-space extent of the projection of the particle must be calculated. Finally, the particle must be drawn on the screen with the aid of the footprint table.

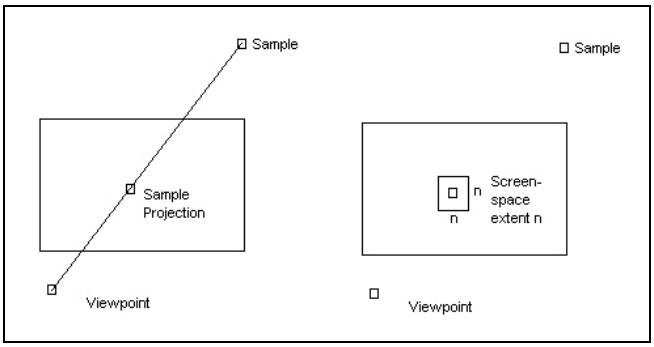

Figure 2: Splatting particles.

Splatting can be implemented by calculating the projection of each particle's center and screen-space extent, and then manually blending the particle's contribution at each pixel directly into the framebuffer. This approach however, yields very sluggish results and is not viable for use in real-time since the rendering computations are performed mainly by the CPU. For this approach, the framebuffer must be updated manually, and the cost of this operation is on the order of 100 ms per frame without direct access to the framebuffer.

2.2.1 Billboarding

Due to the rotational symmetry of the Gaussian kernel, whose rendered image will be the same from any viewpoint except for a uniform scaling, each particle can be represented as a textured billboard for simpler and faster rendering[3]. We thus create a quadrilateral of the same size as the particle and texture map this polygon with an image of the splatted reconstruction kernel. This *kernel texture* is created in preprocess from the footprint table using the mapping in equation (2).

If we orient all billboards to face the viewpoint, rendering the billboard will yield identical results to splatting pixel by pixel to the framebuffer. Using this technique, the graphics API can handle all pixel filling, clipping, and projection operations directly on the graphics hardware, resulting in an enormous speedup. Also, since the billboard polygon is made to fit the size of the particle, no screen-space extents must be calculated.

Using a blended texture, it is possible to allow particle color to vary between particles while using only a single kernel texture. At each pixel in visible projection of a textured polygon, blended texturing combines the color and opacity of the texture (C_t, A_t) with the color and opacity of the texture-mapped polygon (C_p, A_p). For example, the texture function for a blended texture in OpenGL® is

$$C_f = C_p(1 - C_t) + C_c C_t \tag{5}$$

$$A_t = A_p A_t \tag{6}$$

where C_f is the final color and A_f the final opacity of the pixel which is ultimately blended into the framebuffer. An additional parameter, which is not needed for our purposes, is the texture environment color C_c. By setting all three color channels of the kernel texture to pure black (0), the texture function for color collapses to

$$C_f = C_p, \tag{7}$$

and the color of each particle will remain independent of the kernel texture. To change to color of a particle, only a change to the color of its polygon is needed.

By mapping the alpha channel of the kernel texture to the footprint table values, the texture can be used to modulate the opacity of the particle as determined by the reconstruction kernel. This can be accomplished by setting the alpha channel

of each particle's quadrilateral to 1 (completely opaque), causing the final opacity of the particle to be determined only by the kernel texture.

3. BILLBOARD ALIGNMENT

For a precise reconstruction under a perspective projection, billboarded particles must be aligned with the monocular viewpoint or center of projection (COP) on the z-axis in camera coordinates. This requirement however, results in each billboard having a unique alignment which must be computed every frame for a dynamic camera. Alternatively, aligning all billboards with the viewplane yields acceptable results, even stereoscopically, and is much more efficient. Using this approach, only one rotation must be computed per frame for all particles.

3.1 Projection error

We consider the *projection error* produced by orienting billboards parallel to the viewplane as opposed to their correct orientation; directed at the viewpoint. We define projection error to be the distance on the viewplane between a projected point on a viewpoint oriented billboard and the same point on the billboard's viewplane oriented counterpart. All references to a coordinate system refer to the camera coordinate system, which consists of a viewpoint V located at the origin and a viewplane parallel to the xy-plane a distance N from V along the negative z-axis of a right-handed coordinate system. In this system, a point P projects to

$$P^* = \left(\frac{NP_x}{-P_z}, \frac{NP_y}{-P_z} \right). \tag{8}$$

Since both viewplane and viewpoint oriented billboards share a common center point, there is no projection error at the center of the billboard. Projection error will occur however, for all points not at the center of the billboard, and this error increases as the distance from the center of the billboard increases. To see this, recognize that the center of the billboard is also the center of rotation in changing the alignment of a billboard from viewpoint-oriented to viewplane-oriented. Since the distance traveled under a rotation is proportional to the distance from the center of rotation, larger distance from the center of the billboard results in a greater disparity between corresponding points on viewpoint and viewplane oriented billboards. Depending on which the octant the billboard lies in, maximum projection error will occur for one of the corner points.

We choose to perform our analysis on the point E located at the midpoint of the right edge of a billboard as seen from the camera. Let C be the center of a billboard and r be the radius of the billboarded particle, then E lies a distance r from center of the billboard. We consider E in our analysis of the projection error because it provides insight into the nature of the error while allowing simplicity in the analysis. Figure 3 shows C, E, and r for a viewplane oriented billboard.

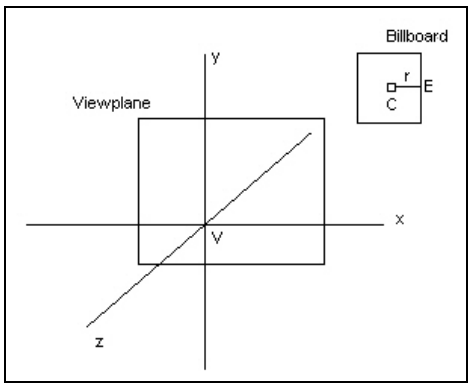

Figure 3: Error in billboard alignment.

We first consider E for billboards aligned with the viewplane. In this case, E is located at the point

$E_\perp = (C_x + r, C_y, C_z)$ and projects to

$$E_\perp^* = \left(\frac{N(C_x + r)}{-C_z}, \frac{NC_y}{-C_z} \right). \tag{9}$$

For billboards aligned with the viewpoint, E lies a distance r along the vector u orthogonal to (*0, 1, 0*) and the vector from the center of the billboard to the viewpoint. V being the viewpoint, u = (*0, 1, 0*) x (V - C), and after normalization, $u^* = \left(\frac{-C_z}{\sqrt{(C_z^2 + C_x^2}}, 0, \frac{-C_x}{\sqrt{(C_z^2 + C_x^2}} \right)$. For billboards aligned with the viewpoint, E lies at $E_\bullet = C + ru^*$ and projects to

$$E_\bullet^* = \left(\frac{N(C_x + ru_x^*)}{-C_z - ru_z^*}, \frac{NC_y}{-C_z - ru_z^*} \right). \tag{10}$$

3.1.1 Horizontal projection error

Taking the difference of the x-components of $E_\perp^*$ and $E_\bullet^*$, the horizontal projection error on the viewplane is

$$e_x = E_{\perp x}^* - E_{\bullet x}^* = \frac{Nr[C_x^2 - C_z\sqrt{C_z^2 + C_x^2} + rC_x - C_z^2]}{-C_z[-C_z\sqrt{C_z^2 + C_x^2} + rC_x]}. \tag{11}$$

As can be expected, the horizontal projection error depends on the nearplane distance, the size of the billboard, and the location of the billboard.

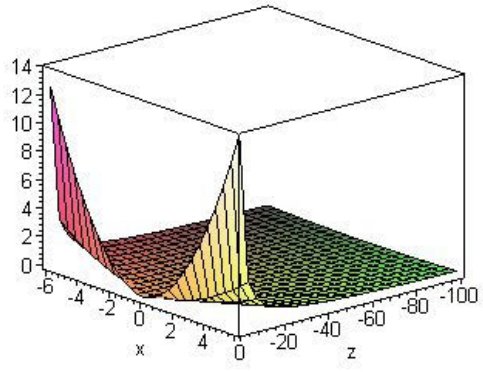

Figure 4: Plot of e_x as C_x and C_z are allowed to vary.

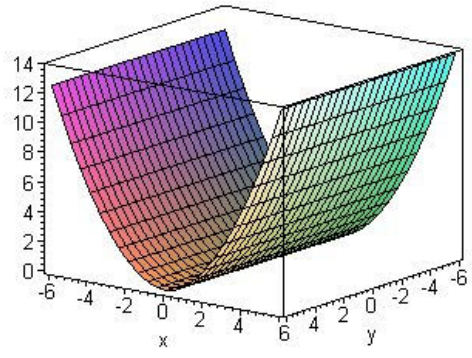

Figure 5: Plot of e_x as C_x and C_y are allowed to vary.

Figure 4 is a plot of the e_x as we allow the location of the billboard to vary in the x and z directions. The error increases rapidly as the billboard approaches the viewplane, and as the billboard moves farther away, the error approaches zero. This effect is illustrated in Figures 6 and 7 and is due mainly to perspective foreshortening; the projected area of the billboard on the viewplane decreases as the billboard recedes. Error is also seen to increase as the billboard moves horizontally away from the viewpoint as seen from the camera. This occurs because a viewpoint oriented billboard

farther away from the viewpoint in the x-direction will require a greater rotation to be aligned with the viewpoint than one that is closer. This increases the disparity between viewplane and viewpoint oriented billboard counterparts.

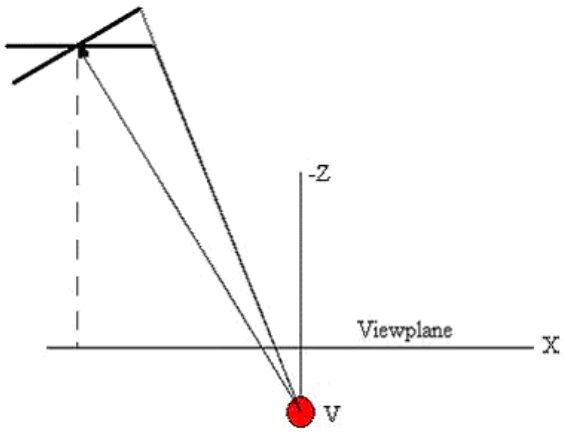

Figure 6: Error for distant billboard.

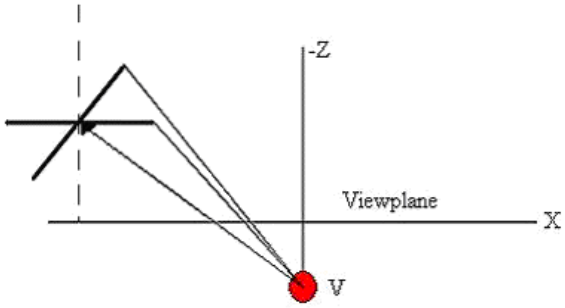

Figure 7: Error for near billboard.

Figure 5 plots the horizontal projection error behavior as a billboard is allowed to move freely in a plane parallel to the viewplane. Again, error increases as the billboard retreats horizontally from the viewpoint for the same reason as in Figure 4. It is apparent in this figure that the y-component of the billboard position has no effect on the horizontal projection error. This is due to our choice of E, which remains static under y-translation for both types of billboard alignment. Since a vertical translation in camera-space has no effect on horizontal projection, the horizontal projection error at E is unaffected as well.

3.1.2 Vertical projection error

We now consider the vertical projection error. Taking the difference of the y-components of $E_{\perp}^{*}$ and $E_{\bullet}^{*}$ we have

$$e_y = E_{\perp y}^{*} - E_{\bullet y}^{*} = \frac{NrC_xC_y\sqrt{C_z^2 + C_x^2}}{C_z[C_z\sqrt{C_z^2 + C_x^2} - rC_x]}. \tag{12}$$

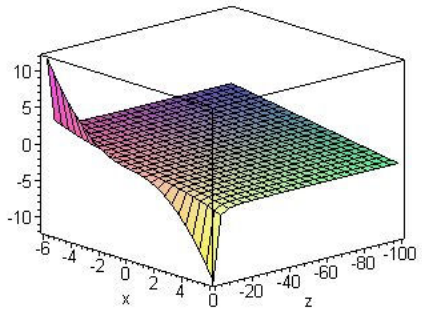

Figure 8: Plot of e_y as C_x and C_z are allowed to vary.

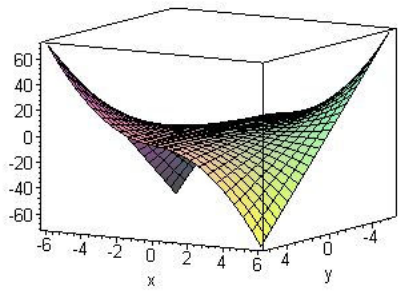

Figure 9: Plot of e_y as C_x and C_y are allowed to vary.

Figure 8 plots e_y when C_y is held constant. As can be seen in the plot, any change in C_x that moves the billboard away from the viewpoint results in an increase in the vertical projection error. This occurs for the same reason as mentioned above for the horizontal projection error. Also, the error is increased when the billboard moves toward the viewplane in

the z-direction. As a particle moves towards the viewplane, the disparity in the alignment between viewplane and viewpoint oriented billboard counterparts will increase, increasing the vertical projection error.

Figure 9 shows the vertical projection error as a billboard moves around in a plane parallel to the viewplane. The error is zero when the center of the billboard lies on the z-axis and increases as the billboard moves away from it. The vertical error is zero when C_x is zero since $E_\perp$ is equal to $E_\bullet$ in this case for any C_y. To see why there is no vertical projection error when C_y is zero, recognize that in this case, $E_\perp$, $E_\bullet$, and the viewpoint all lie in the xz-plane.

As both C_x and C_y move away from zero, we see an increase in the vertical projection error. As C_x moves away from the z-axis the vertical error increases due to the increasing difference between $E_{\perp z}$ and $E_{\bullet z}$ which is a result of the viewpoint oriented billboard requiring a further rotation to remain aligned with the viewpoint.

Increases in C_y will also result in an increase in the vertical error as seen in Figure 9. Although spatially difficult to visualize, mathematically, since $E_{\perp y}$ and $E_{\bullet y}$ will always be equal for our choice of E, we have $e_y = \frac{Ny}{-E_{\perp z}} - \frac{Ny}{-E_{\bullet z}}$ from equation (8). This is the difference of two linear functions of y, explaining the linear increase in vertical projection error seen in the y-direction.

3.3 Parallax error

We now consider the *parallax error* produced by orienting billboards parallel to the viewplane instead of the viewpoint. While the projection error provides insight into the distance onscreen between $E_\perp^*$ and $E_\bullet^*$, the parallax error provides insight into how the user-perceived depth will be different between the two methods.

The parallax of a certain point on a viewpoint oriented billboard is the difference in the projections between the left and right eye views. The parallax error for our technique is the difference between the parallax for a point on a viewpoint oriented billboard and the parallax for the corresponding point on a viewplane oriented billboard. Mathematically, this is equivalent to the difference in the projection errors for the left and right eye viewpoints. Since the analysis of the projection error we performed previously was dependent only on the position of a billboard relative to the location of the viewpoint, we can express the parallax error for a billboard with center C as the difference in the projection error of a billboard at $C + (-I/2, 0, 0)$ and a billboard at $C + (I/2, 0, 0)$, where I is the inter-ocular distance. This is the difference of two translated projection error surfaces.

Figures 10-13 plot the horizontal and vertical parallax error for a constant inter-ocular distance. The surfaces were created by taking each projection error surface discussed previously and subtracting the surface translated by $(-I/2, 0, 0)$ from the surface translated by $(I/2, 0, 0)$, or the right eye projection error minus the left eye projection error.

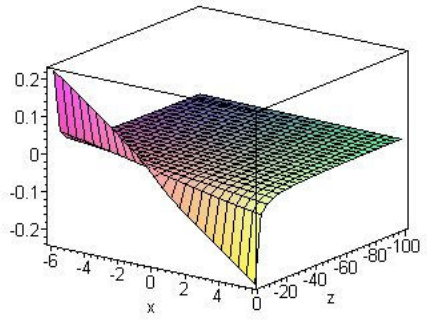

Figure 10: Horizontal parallax error as C_x and C_z vary.

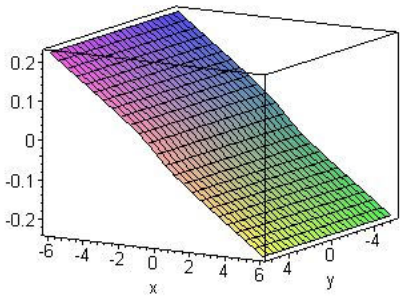

Figure 11: Horizontal parallax error as C_x and C_y vary.

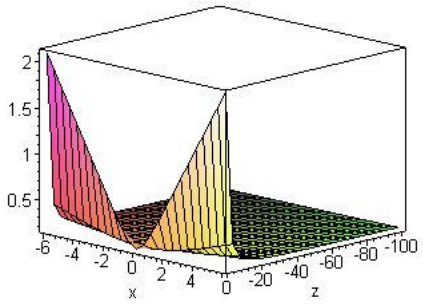

Figure 12: Vertical parallax error as C_x and C_z vary.

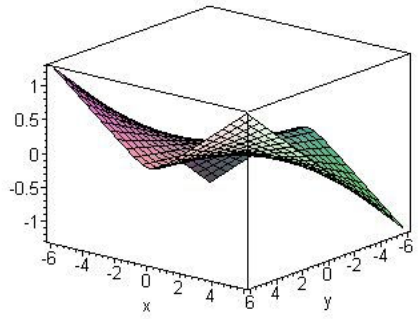

Figure 13: Vertical parallax error as C_x and C_y vary.

The plots show which billboard positions will yield the greatest parallax error. The exact error will depend on the inter-ocular distance, but these plots give intuition as to the overall behavior of the error. The parallax error seen in these plots is due to each billboard having a different position relative to the left and right viewpoints and therefore a different projection error. In general, the billboard positions for which the projection error is increasing most rapidly will have the greatest parallax error. Referencing the projection error surfaces, we see this is indeed the case. There is little parallax error for billboards distant from the viewplane due to the inter-ocular distance being insignificant in comparison to the distance between the billboard and the stereo viewpoints. Also, billboards which lie near the z-axis also show little parallax error due to the disparity in alignment between viewpoint and viewplane oriented billboards being insignificant in this case.

3.4 Error discussion

We found that for billboards close to the viewplane, the projection and parallax error incurred could be substantial. At the same time, this does not lead to viewer discomfort or result in any visual side effects. This can be attributed to the nature of the image; the particles in use lack well-defined boundaries and are subject to a high degree of occlusion by other particles. Additionally, the points on a billboard most distant from the center which are subject to the greatest projection error are also those points with the least effect on the final image. This is due to the exponential decline of the Gaussian kernel. Also, while it is true that the rendered image of a viewplane oriented billboard will appear slightly warped in comparison to its viewpoint oriented counterpart in cases of large projection error, the viewer is presented only with viewplane oriented billboards, keeping the observed geometry consistent and avoiding parallax error of the type that can lead to viewer discomfort. In short, our method will results in an observed geometry which is different from the original geometry depending on the amount of projection error.

4. RESULTS

We illustrate the method for several scenes where we vary the number of particles as well as particle density, color, and radius. In the figures, r is the radius given to each particle, ρ the density, and c the grayscale color in the range [*0, 1*]. In all scenes, each particle present in the scene was given the same radius, color, and density. A 3D model of a water tower has been included to give the viewer intuition as to how variations in the parameters affect visibility. Particles were placed with a uniformly random distribution within a 3D bounding box around the water tower. As seen in the images, the water tower and any particles in the scene correctly occlude one another. The anaglyph in Figure 17 was computed using the least squares technique proposed by Eric Dubois[9] and analyzed in Sanders[10] for red/cyan filters.

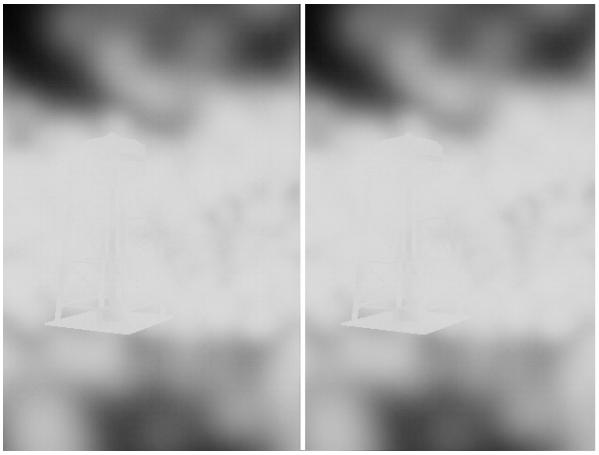

Figure 14: Clouds, 1500 particles, r =0.37, ρ = 0.4, c = 0.86.

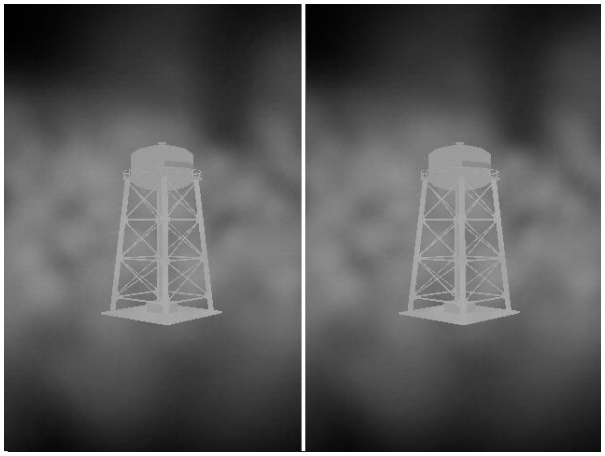

Figure 15: Mist, 1000 particles, r = 0.45, ρ = 0.15, c = 0.6.

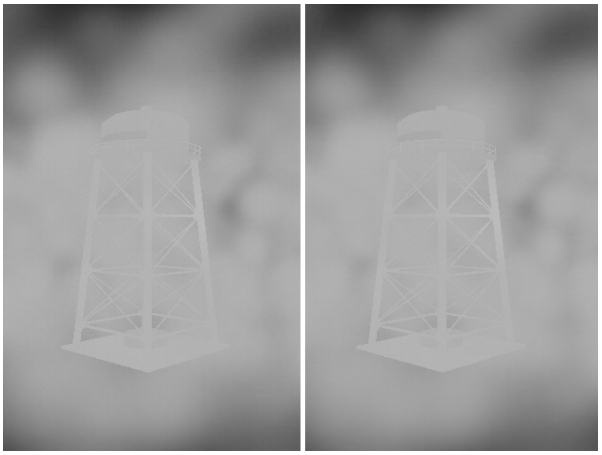

Figure 16: Fog, 1200 particles, r = 0.45, ρ = 0.4, c = 0.7.

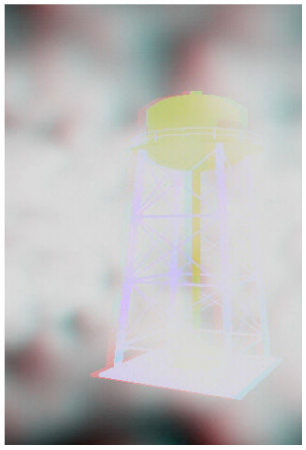

Figure 17: Anaglyph of fog.

4.1 Performance

We found that in order to achieve the "wispy" appearance of gaseous phenomena as seen in the images above, the particles needed to have a relatively large screen-space extent when close to the camera. Smaller particles did not adequately fill the space of the volume, yielding a scene that did not look realistic. Increasing the screen-space extent of the particles adversely affects rendering time however, but offers the advantage of requiring fewer particles for a realistic looking effect.

To test the performance of the technique for scenes similar to those above, we rendered full-screen anaglyph frames with the left and right views superimposed onscreen. A 1.9 GHz Pentium® 4 machine with a 64MB NVidia® GeForce™ 4 graphics card was used for all tests. We collected frame rates as the viewer moved around within the phenomena, which is much more expensive than rendering the phenomena at a distance due the amount of overlapping particle area onscreen. Filling the bounding box with 1000 particles of radius 0.4, we obtained an average frame rate of around 50 frames/sec. We then stressed the performance of the technique with a very dense scene. We used 2000 particles of radius 0.4 and obtained an average frame rate of around 14 frames/sec. In a third scene with 2000 particles of radius 0.32, the average frame rate was about 20 frames/sec.

During testing we found that as the viewer moved into the volume, he would encounter a point where maximum overlap would occur, causing the lowest frame rates. Rendering in the vicinity of this point, the average frame rate was around

15 frames/sec for the first test case, 10 frames/sec for the second, and 11 frames/sec for the third. We note that the rendering can exploit parallel computation, which will be a subject of future research.

5. CONCLUSIONS AND AREAS OF FUTURE RESEARCH

Using splatting, a technique for rendering volumes using signal reconstruction, it is possible to render gaseous atmospheric phenomena such as clouds, fog, and mist in real-time and in stereo. Volumes are represented as points around which a 3D Gaussian reconstruction kernel is centered. Each of these kernels can be thought of as a separate particle which, due to rotational symmetry, can be represented faithfully as a texture-mapped, alpha-blended quad. Representing volumes as points provides an efficient representation for storage, and the speed with which volumes can be rendered is completely dependent on the speed with which the graphics hardware can render texture-mapped alpha-blended polygons, allowing rendering to occur in real-time. The technique allows views from inside the volume to be rendered with particles properly occluding other objects. Additionally, orienting billboards to face the viewplane instead of the viewpoint requires that only one rotation be computed per frame for all billboards. Due to the nature of the image, visual artifacts are avoided and no view discomfort results.

When inside the volume, the screen-space extent of particles is often very large, slowing rendering time considerably. Dynamically generated imposters have been used successfully in monoscopic methods to increase performance[3,4]. The applicability of imposters to stereo rendering is something that should be investigated.

Truly realistic clouds and fog take into account the scattering of light through the medium as well as the effects self-shadowing. A lighting model such that described by Harris[3] could be incorporated into our method with lighting computations being performed in preprocess. Each particle could be assigned a color based on this model and would be rendered with proper illumination at run-time.

REFERENCES

1. Lee Westover, “Interactive Volume Rendering”, *Proceedings of the Chapel Hill Workshop on Volume Visualization*, May 1989.
2. Lee Westover, “Footprint Evaluation for Volume Rendering”, SIGGRAPH, pg 367 – 376, 1990.
3. Mark Harris, Anselmo Lastra, “Real-Time Cloud Rendering” Eurographics, **Vol. 20**, issue 3, 2001.
4. Niniane Wang, “Let There Be Clouds!”, CMP Media LLC Game Developer, pg 34, 2004.
5. G. Schaufler, “Dynamically Generated Imposters”, GI Workshop *Modeling – Virtual Worlds – Distributed Graphics*, pg 129 – 136, 1995.
6. Manoj K. Patel, David F. McAllister, “Combining Motion Blur and Stereo”, *Proc. SPIE*, **Vol. 1669**, pg 71 – 83, 1992.
7. Jetentre Borse, David F. McAllister, “Real-time Image-based Rendering for Stereo Views of Vegetation”, *Proc. SPIE*, **Vol. 4660**, pg 292-299, 2002. http://research.csc.ncsu.edu/stereographics/
8. Kajiya, Mij, Brian Von Herzen, “Ray Tracing Volume Densities“, *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, pg 165 -174, 1984.
9. Eric Dubois, "A Projection Method to Generate Anaglyph Stereo Images," *Proc. IEEE Int. Conf. Acoustics Speech Signal Processing*, **Vol. 3**, pg 1661-1664, IEEE, Salt Lake City, UT, May, 2001.
10. W. Sanders, David F. McAllister, "Producing Anaglyphs from Synthetic Images," *Proc. SPIE,* **Vol. 5006**, pg 348-358*,* 2003. http://research.csc.ncsu.edu/stereographics/
11. D. F. McAllister (Ed.), *Stereo Computer Graphics and other True 3D Technologies,* Princeton U. Press, Princeton, NJ, 1993.
12. D.F. McAllister, "3D Displays," *Wiley Encyclopedia on Imaging*, Jan, pg 1327-1344, 2002.