5.1

Spherical Billboards for Rendering Volumetric Data

Tamás Umenhoffer, László Szirmay-Kalos, and Gábor Szíjártó

Budapest University of Technology and Economics

Introduction

Many types of natural phenomena are participating media [Blinn82]; examples include fire, smoke, explosions, fog, and clouds. Participating media can be simulated by animating and rendering a particle system [Reeves83]. A particle is an animation and rendering primitive whose geometric extent is small, but that possesses certain properties such as position, speed, color, and opacity.

Formally, a particle system is a discretization of a continuous medium, where each particle represents a spherical neighborhood in which the volume is locally homogeneous. This formulation allows us to replace differentials of the equations governing the motion and light scattering with finite differences during simulation and rendering.

In this article we focus on rendering such systems in real-time; that is, we solve the volumetric rendering equation. The discretization of the volumetric rendering equation thus leads to the following equation expressing *outgoing radiance* $L(j,\vec{\omega})$ of particle j at direction $\vec{\omega}$ (with the *density*, *albedo*, and *phase function* of this particle given by τ_j, a_j, and P_j, respectively, and the length of a ray segment intersecting the sphere of particle j given by Δs_j):

$$L(j,\vec{\omega}) = I(j,\vec{\omega})\cdot(1-\alpha_j) + \alpha_j \cdot a_j \cdot C_j + \alpha_j \cdot (1-a_j)\cdot L_j^e(\vec{\omega}), \qquad (5.1.1)$$

where $I(j,\vec{\omega})$ is the *incoming radiance* from direction $\vec{\omega}$,

$$\alpha_j = 1 - e^{-\int_{\Delta s_j} \tau(s)ds} = 1 - e^{-\tau_j \Delta s_j} \qquad (5.1.2)$$

is the *opacity* that expresses the decrease of radiance caused by this particle due to *extinction*, L_j^e is the emission radiance, and

$$C_j = \int_{\Omega'} I(j,\vec{\omega}') \cdot P_j(\vec{\omega}',\vec{\omega})\, d\vec{\omega}' \tag{5.1.3}$$

is the total contribution from *in-scattering*.

Splatting [Schaufler95] [Reeves83] [Wei02] is one solution to this problem. It renders particles as semi-transparent, camera-aligned rectangles, also called billboards. When such a billboard is rasterized, the in-scattering term attenuates according to the total opacity of the particles that are between the camera and the current billboard. To obtain a reasonable approximation to the in-scattering term thus requires billboards to be sorted and rendered in back to front order. Rendering a billboard adds its in-scattering and emission terms (Equation 5.1.1) to the frame buffer.

Billboard Clipping Artifact

Billboards are planar rectangles with no extension along one dimension. This two-dimensional representation causes artifacts when billboards intersect opaque objects (see Figure 5.1.1). The root cause of this clipping artifact is that billboards assume that all objects are fully behind the sphere of the particle. Objects in front of the billboard plane are not affected at all, and thus transparency changes abruptly at all object–billboard intersections.

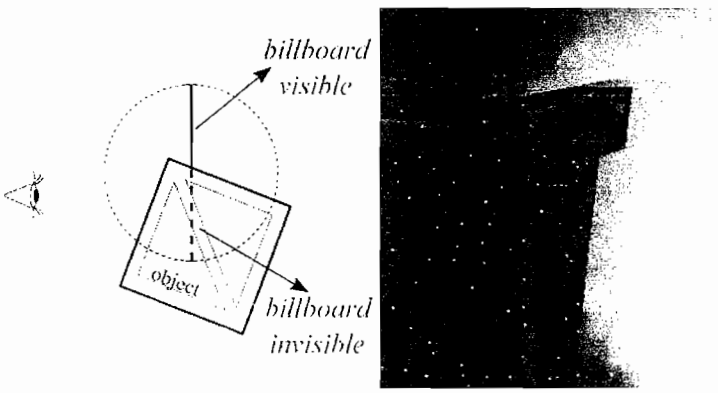

FIGURE 5.1.1 A billboard clipping artifact. When the billboard rectangle intersects an opaque object, transparency becomes spatially discontinuous.

Billboard Popping Artifact

When the camera moves into the simulated medium, billboards also cause popping artifacts. In this case, the billboard is either behind or in front of the front clipping plane; transitioning between these two states is instantaneous. The particle thus switches from fully visible to fully invisible, resulting in an abrupt change (see Figure 5.1.2).

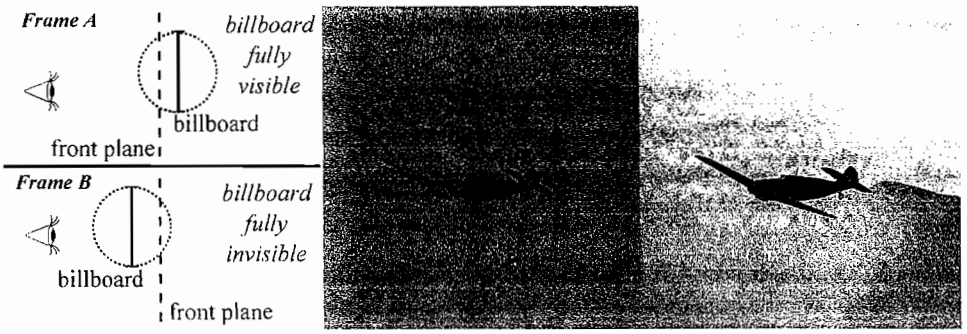

FIGURE 5.1.2 A billboard-popping artifact. When the billboard switches sides of the front clipping plane, the transparency is discontinuous in time (the figure shows two adjacent frames in an animation).

To solve these problems of clipping and popping, we introduce spherical billboards in the section "Spherical Billboards." We apply the proposed idea to render realistic explosions in the section "Rendering Explosions." Figure 5.1.3 shows a few frames of an animation that uses this technique to render explosions.

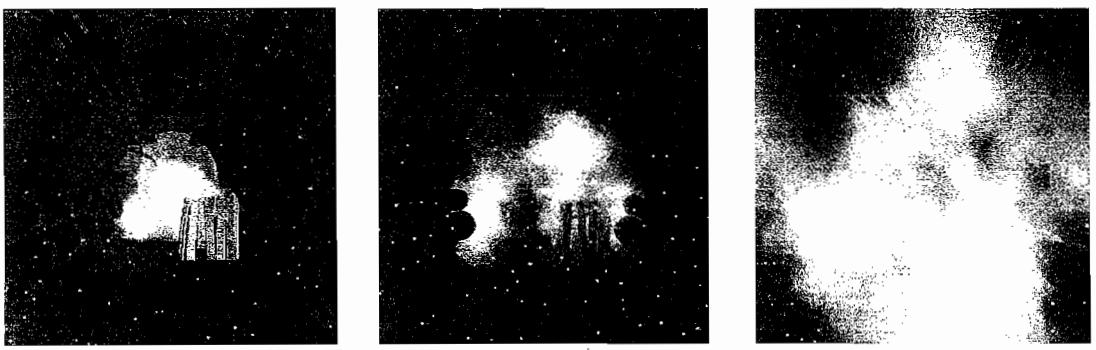

FIGURE 5.1.3 Rendered frames from an animation sequence showing an explosion.

Spherical Billboards

To solve the billboard-clipping artifact we calculate the path length a light ray travels inside a given particle. This length determines the correct opacity value during rendering. Instead of representing particles as planar rectangles, we assume particles to be spherical. To keep an implementation simple and fast, however, we still render particles as quadrilateral primitives and only take their spherical shape into account during fragment processing.

Opacity Calculation

To find out where opaque objects are during particle rendering, the algorithm first renders all objects of the scene and saves the camera-space distance to the origin of the visible points. These saved values are real Euclidean distances $\sqrt{x^2+y^2+z^2}$ for a point having (x,y,z) coordinates in camera space. We call the texture storing these distance values a distance map. Then we render the particles as quads placed at the farthest point from the camera of the particle sphere (to avoid unwanted front plane clipping) and rotated toward the camera via the vertex shader. To eliminate incorrect object–billboard clipping, we also disable the depth test.

When rendering a fragment of the particle, we compute the interval the ray travels inside the particle sphere in camera space. This interval is obtained by considering the saved distance values of opaque objects and the camera's front clipping plane distance (see also Equation 5.1.4 below). From the resulting interval, we compute the opacity for each fragment in such a way that both fully and partially visible particles display correctly, resulting in the illusion of a volumetric medium. During opacity computations we assume that the density inside a particle sphere is uniform.

We use the notations of Figure 5.1.4: a particle of center $\bar{P}=(x_p,y_p,z_p)$ renders as a quad perpendicular to the view ray, and a ray intersects a point $\bar{Q}=(x_q,y_q,z_q)$ of the quadrilateral. All coordinates are in camera space where the eye is at origin $\bar{O}$. The radius of the particle sphere is r. The closest point on the ray from the particle center is $\bar{C}$, which $\bar{Q}$ closely approximates as long as the particle's distance to the camera is large compared to the particle's size. Another way to formulate this approximation is to say that we assume that the angle $(\bar{O},\bar{Q},\bar{P})$ is close to 90 degrees.

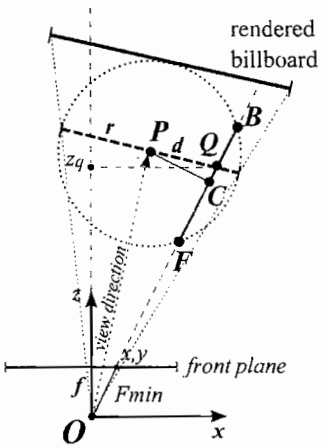

FIGURE 5.1.4 Computation of the length of the ray segment traveling inside a particle sphere in camera space.

The distance d between the ray and the particle center is

$$d \approx \left|\vec{P} - \vec{Q}\right|.$$

The ray thus travels inside the particle in the interval $\left[\left|\vec{F}\right|, \left|\vec{B}\right|\right]$, where $\left|\vec{F}\right| \approx \left|\vec{Q}\right| - \omega$, $\vec{B}\left| \approx \right|\vec{Q}\right| + \omega$, and $\omega = \sqrt{r^2 - d^2}$ as long as neither the front clipping plane nor the scene objects intersect the particle sphere.

To take the front clipping distance f into account, we ensure that $\left|\vec{F}\right|$ is no smaller than $F_{\min} = \frac{f}{z_q}\left|\vec{Q}\right|$.

For the ray passing through point $\vec{Q}$, the distance D_s of the visible surface is stored in the distance map. To take scene objects into account, we thus ensure that $\left|\vec{B}\right|$ is no greater than D_s. These distances allow us to compute a very good approximation of the distance a ray travels inside a particle sphere:

$$\Delta s = \min(Z_{s_i}, \left|\vec{B}\right|) - \max(F_{\min}, \left|\vec{F}\right|). \tag{5.1.4}$$

If we assume that the density inside a particle is homogeneous, although the accumulated opacity at the contour of a particle sphere does become zero, it does not diminish smoothly, making the contour of a particle sphere clearly visible for a human observer. We can eliminate this artifact if the density decreases linearly with distance from the particle center. Instead of Equation 5.1.2, we modulate the accumulated density by a linear function and use the following formula to obtain the opacity of particle j:

$$\alpha_j \approx 1 - e^{-\tau_j (1 - d/r_j) \Delta s_j}, \tag{5.1.5}$$

where d is the distance between the ray and the particle center, and r_j is the radius of this particle.

GPU Implementation of Spherical Billboards

A custom fragment shader program executes an appropriate function that efficiently evaluates the length a ray travels in a particle sphere and the corresponding opacity. This function needs the following inputs, which are calculated in a corresponding vertex shader: the particle position in camera space (`P`), the shaded billboard point in camera space (`Q`), the particle radius (`r`), and the screen coordinates of the shaded point (`screenCoord`). The function uses uniform parameters, such as the texture of the distance values of opaque objects (`Dist`), the density (`tau`), and the camera's front clipping-plane distance (`f`). The Cg function to determine the opacity is the following:

```
float Opacity(float3 P, float3 Q, float r, float2 screenCoord)
{
    float alpha = 0;
    float d = length(P.xy - Q.xy);
    if(d < r)
```

```
    {
      float Ql = length(Q);
      float fMin = f * Ql / Q.z;
      float w = sqrt(r * r - d * d);
      float F = Ql - w;
      float B = Ql + w;
      float Ds = tex2D(Dist, screenCoord).r;
      float ds = min(Ds, B) - max(fMin, F);
      alpha = 1 - exp(-tau * (1 - d / r) * ds);
      }
}
```

The shader program thus obtains the real ray segment length (`ds`) and computes the opacity `alpha` of a given particle that controls blending of the particle into the frame buffer. The consideration of the spherical geometry during fragment processing eliminates clipping and popping artifacts (see Figure 5.1.5).

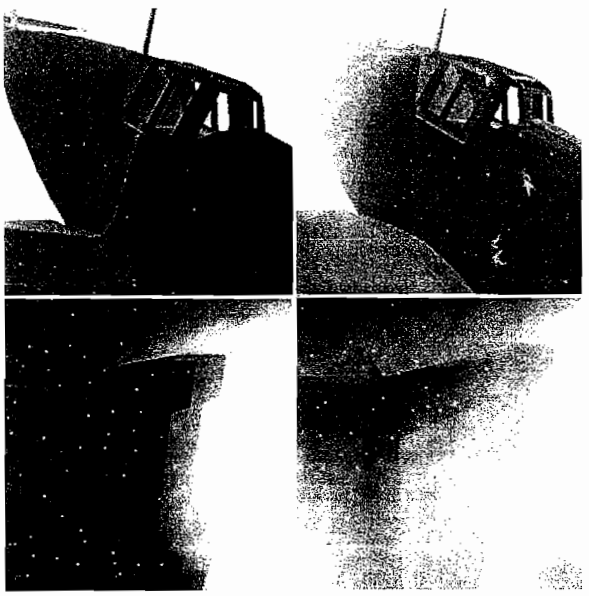

FIGURE 5.1.5 Particle system rendered with planar (left) and spherical (right) billboards.

Rendering Explosions

We now apply spherical billboards to rendering realistic explosions. An explosion consists of dust, smoke, and fire. All these effects are modeled with specific particle systems. Dust and smoke absorb light; fire emits light. We render these particle systems separately and then composite their rendered images to obtain the final result.

Dust and Smoke

Smoke particles absorb light in a fire. These particles typically have low albedo and high density values ($a=0.2, \tau=0.25$). On the other hand, high-albedo and low-density ($a=0.95, \tau=0.03$) dust further improves the realism of explosions (see Figure 5.1.6 for examples).

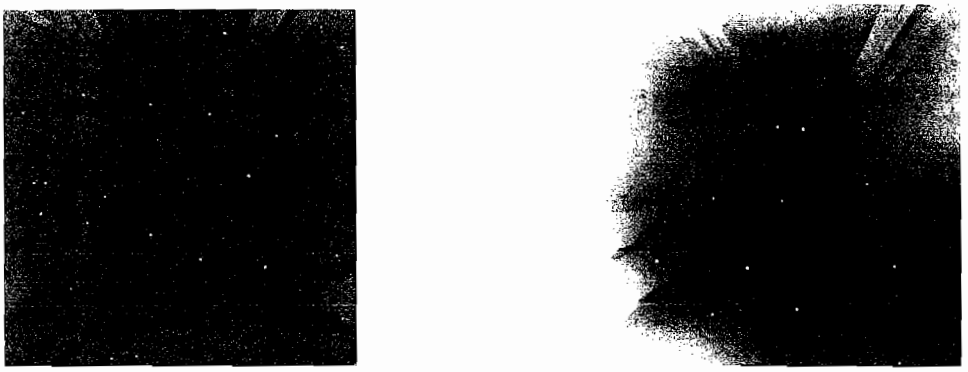

FIGURE 5.1.6 Low-albedo smoke (left) and high-albedo dust.

When rendering dust and smoke, we assume that these particles do not emit radiance; that is, their emission term is zero. To calculate their in-scattering term, the length that light travels inside a particle sphere, the albedo, the density, and the phase function (see Equation 5.1.1) are needed. We use the Henyey-Greenstein phase function [Henyey40] [Cornette92]:

$$P(\vec{\omega}',\vec{\omega})=\frac{1}{4\pi}\cdot\frac{3(1-g^2)\cdot(1+(\vec{\omega}'\cdot\vec{\omega})^2)}{2(2+g^2)\cdot(1+g^2+2g(\vec{\omega}'\cdot\vec{\omega}))^{3/2}} \tag{5.1.6}$$

where $g\in(-1,1)$ is a material property describing how strongly a material scatters forward or backward. To speed up rendering, these function values are fetched from a 2D texture addressed by $\cos\theta=\vec{\omega}'\cdot\vec{\omega}$ and g, respectively. We find that setting g to constant zero gives satisfactory results for both smoke and dust.

The length that light travels inside a smoke or dust particle is computed by the proposed spherical billboard method.

In order to maintain high frame rates, the number of particles must be limited, which compromises high-detail features. To address this problem, we increase the particles' radii as we decrease their number. To increase variety, we perturb the opacity values computed by the spherical billboards. Each particle has a unique, time-dependent perturbation pattern. This perturbation derives from a greyscale texture called a *detail image*, which depicts real smoke or dust (see Figure 5.1.7). The perturbation pattern of a particle is taken from a randomly placed, small quad-shaped part of this texture. (Off-line renderers use a similar technique for the same purpose [Apodaca00].) To

provide variety in the time domain, this texture is updated as time advances. These animated 2D textures are obtained from real-world videos and are best stored as a volume texture: a GPU's texture sampling unit can then provide interframe interpolation and looping.

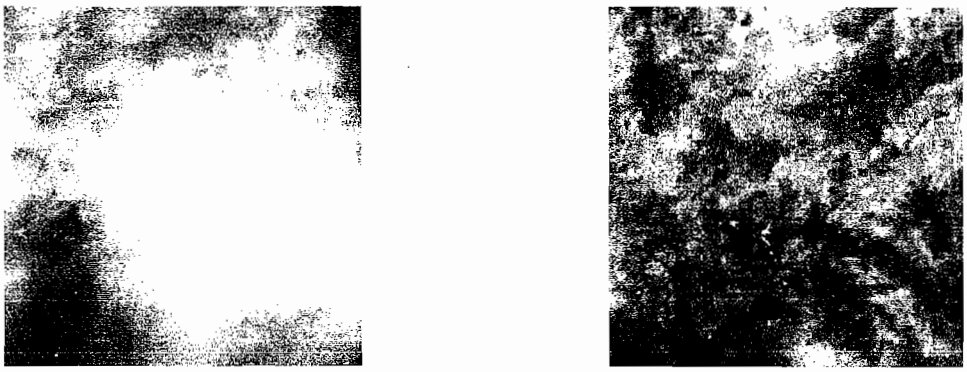

FIGURE 5.1.7 Images from real smoke and fire video clips, which are used to perturb the billboard fragment opacities and temperatures.

Fire

We model fire as a black body radiator rather than a participating medium; that is, we set the albedo in Equation 5.1.1 to zero, so that only the emission term is required. The color characteristics of fire particles are thus determined by the physics theory of black body radiation. Planck's formula computes the emitted radiance of a black body for wavelength λ:

$$L_{e,\lambda}(x) = \frac{2C_1}{\lambda^5(e^{C_2/(\lambda T)} - 1)} \tag{5.1.7}$$

where $C_1 = 3.7418 \cdot 10^{-16} Wm^2$, $C_2 = 1.4388 \cdot 10^{-2} m^\circ K$, and T is the absolute temperature of the radiator [Siegel81]. The RGB components for different temperature values are obtained by integrating the spectrum multiplied by the color-matching functions. These integrals are precomputed and stored in a texture.

Similar to smoke and dust particles, we add detail to fire particles by perturbing their opacity, as well as their emission radiance. Sampling color directly from a video does not allow us to control the temperature range or color for different explosions. We instead store temperature variations in a detail texture (see Figure 5.1.7); the stored temperature values are scaled and used for color computation on the fly. A randomly selected, small quadrilateral part of the detail video is assigned to each fire particle to control the temperature perturbation of the fragments of a particle billboard. The resulting temperature thus determines the color of a fragment according to the above black body radiation function.

A fragment program to compute fire particles receives as input a fire particle position in camera space (P), a shaded billboard point in camera space (Q), a particle radius (r), screen coordinates of the shaded point (screen), a position of the detail image in the texture (detail) (with starting time in its z coordinate), and the current time of the animation (time). This fragment program also reads uniform parameters, including the texture of the fire video (FireVideo), the black body radiation (BBRad) map, and the temperature scale (T1) and bias (T0). The fragment program calls the Opacity() function, which computes the opacity using the spherical billboard method described above and then calculates the color of a fire particle using the following shader program:

```
float alpha = Opacity(P, Q, r, screen);   // opacity of particle
float3 detuvw = detail + float3(0,0,time);// texcoords of the
                                             detail
float fireVideoSample = tex3D(FireVideo, detuvw).r; //video sample
float T = T0 + T1 * fireVideoSample;      // perturbed temperature
alpha *= fireVideoSample;                   // perturbed opacity
return float4(tex1D(BBRad, T).rgb, 1) * alpha; // emitted color
```

Layer Composition

To combine the above smoke, dust, and fire particle systems with the image of opaque objects, we use a layer composition method. We render the opaque objects and the particle systems into separate textures and then blend the particle texture on top of the opaque objects. This approach thus consists of three rendering passes: one pass to render the opaque objects; one pass to render the dust, smoke, and fire; and one final pass for composition. The first pass computes both the color and the depth of the opaque objects.

One advantage of rendering a participating medium into a texture is that this render pass may use a lower-resolution rendering target than the final display resolution, which speeds up rendering since rendering billboard particles typically overdraws a single pixel multiple times and uses blending (see Figure 5.1.8).

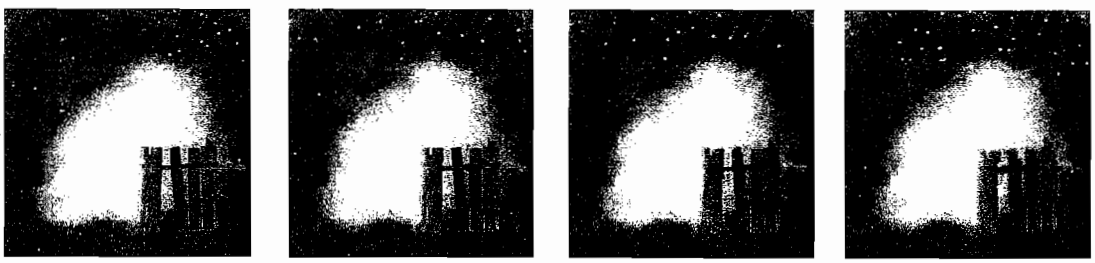

FIGURE 5.1.8 Effect of render-target resolution on rendering throughput. As render-target resolution decreases from full-screen resolution (left) to one-eighth the screen resolution (right), rendering performance increases. On each picture the frame buffer has resolution 512×512. The render-target resolution and fps changes are (from left to right): 512×512, 30 FPS; 256×256, 40 FPS; 128×128, 50 FPS; 64×64, 60 FPS.

The complete rendering process is shown in Figure 5.1.9.

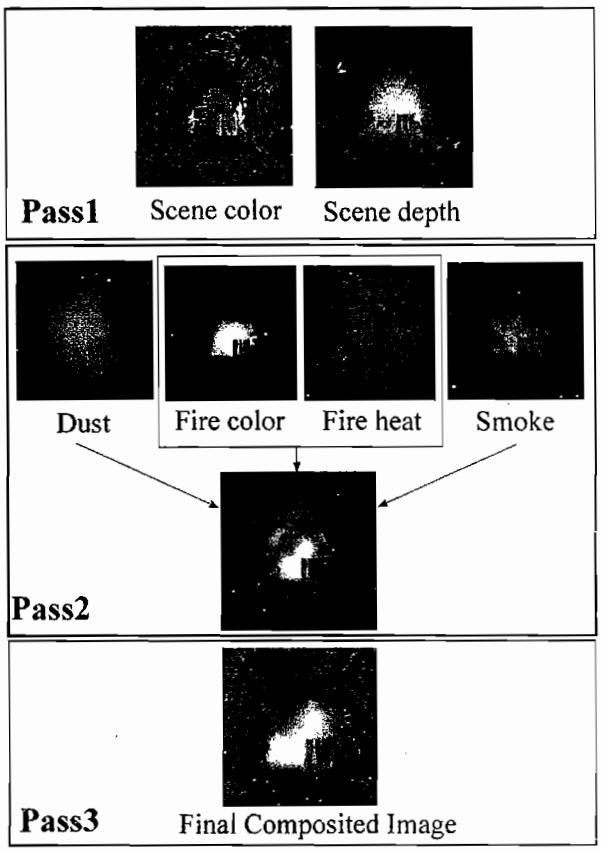

FIGURE 5.1.9 The rendering algorithm.

Conclusion

We introduce an improved billboard rendering method that splats particles as camera-aligned quadrilaterals but takes into account the spherical geometry of the particles during fragment processing. This new method eliminates the billboard clipping and popping artifacts of previous techniques. We show how to apply this new technique to render high-detail explosions consisting of fire, dust, and smoke at high frame rates. Please refer to this article's folder on the CD-ROM for tutorial files.

ON THE CD

References

[Apodaca00] Apodaca, A. A. and L. Gritz. *Advanced RenderMan: Creating CGI for Motion Picture.* Academic Press, 2000.

[Blinn82] Blinn, J. F. "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces." *SIGGRAPH '82 Proceedings*, 1982: pp. 21–29.

[Cornette92] Cornette, W. and J. Shanks. "Physical Reasonable Analytic Expression for Single-Scattering Phase Function." *Applied Optics, 31*(16), (1992): pp. 31–52.

[Henyey40] Henyey, G. and J. Greenstein. "Diffuse Radiation in the Galaxy." *Astrophysical Journal*, 88, (1940): pp. 70–73.

[Reeves83] Reeves, W. T. "Particle Systems: Techniques for Modelling a Class of Fuzzy Objects." *SIGGRAPH '83 Proceedings*, (1983): pp. 359–376.

[Schaufler95] Schaufler, G. "Dynamically Generated Impostors." *I Workshop: Virtual Worlds*. Distributed Graphics, 1995: pp. 129–136.

[Siegel81] Siegel, R. and J. R. Howell. *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corp., Washington, D.C., 1981.

[Wei02] Wei, X., W. Li, K. Mueller, and A. Kaufman. "Simulating Fire with Texture Splats." *IEEE Visualization '02*, 2002.