Spherical Billboards and their Application to Rendering Explosions

Tamás Umenhoffer* László Szirmay-Kalos† Gábor Szijártó‡

Department of Control Engineering and Information Technology
Budapest University of Technology, Hungary

Planar Spherical Planar Spherical

Figure 1: Comparison of the classical planar and the proposed spherical billboards

ABSTRACT

This paper proposes an improved billboard rendering method that splats particles as aligned quadrilaterals similarly to previous techniques, but takes into account the spherical geometry of the particles during fragment processing. The new method can eliminate billboard clipping and popping artifacts of previous techniques, happening when the participating medium contains objects, or the camera flies into the volume. This paper also discusses how to use spherical billboards to render high detail explosions made of fire and smoke at high frame rates.

CR Categories: I.3.7 [Three-Dimensional Graphics and Realism]

Keywords: Billboards, particle systems, shader programming

1 INTRODUCTION

Participating media [2] or volumetric models are often represented by *particle systems* [12]. The basic idea of particle systems is to build complex models from a large population of simple entities, whose geometric extent is small, but which possess certain properties used in animation (position, speed, acceleration) and in rendering (e.g. color, opacity). Particle systems can be used to simulate and render natural phenomena, including fire, smoke, explosions, fog, cloud, etc [13, 17, 4, 10, 5, 9].

Formally, a *particle system* is a discretization of a continuous medium, which allows us to replace differentials of the equations governing the motion and light scattering by finite differences during simulation and rendering. In this paper we focus on real-time rendering such systems, which means the solution of the volumetric rendering equation. In the continuous volume the optical properties on a single wavelength are described by the *emission radiance*, *density* τ, i.e. the reciprocal of the expected free run length of a photon, *albedo* a, i.e. the probability that the collided photon is not absorbed, and *phase function* $P(\vec{\omega}', \vec{\omega})$, i.e. the probability density of the photon direction $\vec{\omega}$ after collision, provided that the photon came from direction $\vec{\omega}'$ before collision and is reflected.

To solve the volumetric rendering equation efficiently, the domain is discretized using a particle system. A particle represents a spherical neighborhood where the volume is locally homogeneous, thus the particle neighborhood can be represented by a single set of optical properties. Denoting the length of the ray segment intersecting the sphere of particle j by Δs_j, and the *density*, *albedo*, and *phase function* of this particle by τ_j, a_j, P_j, respectively, the discretization of the volumetric rendering equation leads to the following equation expressing *outgoing radiance* $L(j, \vec{\omega})$ of particle j at direction $\vec{\omega}$:

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

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

$$\alpha_j = 1 - e^{-\int_{\Delta s_j} \tau(s) ds} = 1 - e^{-\tau_j \Delta s_j} \quad (2)$$

is the *opacity*, which equals to the decrease of radiance caused by this particle due to *extinction* (i.e. the sum of absorption and outscattering), L_j^e is the *emission radiance*, and

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

is the total contribution of *in-scattering* from all possible directions $\vec{\omega}'$ of directional sphere Ω'. Equation 1 can be interpreted as follows. Radiance I of the light ray in direction $\vec{\omega}$ is decreased because photons may collide with the media. The probability that collision occurs in a particle sphere equals to opacity α_j, which also expresses the amount of participating media enclosed by the particle. On the other hand, the intensity is increased by the in-scattering term $\alpha_j \cdot a_j \cdot C_j$, which means that photons traveling in other directions $\vec{\omega}'$ collide and are scattered into direction $\vec{\omega}$, and by emission $\alpha_j \cdot (1-a_j) \cdot L_j^e(\vec{\omega})$ of the participating media enclosed by the particle. Here $a_j \cdot P_j(\vec{\omega}', \vec{\omega})$ is the subcritical probability density that

*umitomi@freemail.hu
†szirmay@iit.bme.hu
‡szijarto.gabor@freemail.hu

the photon is scattered into direction $\vec{\omega}$ after collision, provided it came from direction $\vec{\omega}'$. We have to consider all incoming directions, therefore the in-scattering term contains an integral in the directional domain. Since particles are small, multiple scattering inside a particle sphere is ignored.

Having approximated the in-scattering term somehow [19], the volume can efficiently be rendered from the camera using *alpha blending*. The in-scattering term and the emission of a particular particle are attenuated according to the total opacity of those particles which are between the camera and this particle. This requires particles be sorted in the view direction before sending them to the frame buffer in back to front order. At a given particle, the evolving image is decreased according to the opacity of the particle and increased by its in-scattering and emission terms (equation 1).

Particle system rendering methods usually splat particles onto the screen [12, 21]. Splatting substitutes particles with semi-transparent, camera-aligned rectangles, called *billboards* [14].

1.1 Billboard clipping and popping artifacts

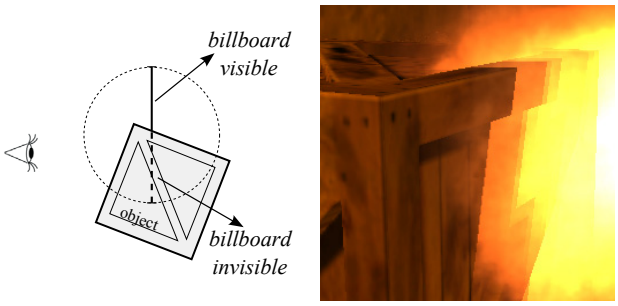

Figure 2: Billboard clipping artifact. When the billboard rectangle intersects an opaque object, transparency becomes spatially discontinuous.

Billboards are planar rectangles having no extension along one dimension. This can cause artifacts when billboards intersect opaque objects making the intersection of the billboard plane and the object clearly noticeable (figure 2). The core of this *clipping artifact* is that a billboard fades those objects that are behind it according to its transparency as if the object were fully behind the sphere of the particle. However, those objects that are in front of the billboard plane are not faded at all, thus transparency changes abruptly at the object–billboard intersection.

On the other hand, when the camera moves into the media, billboards also cause *popping artifacts*. In this case, the billboard is either behind or in front of the front clipping plane, and the transition between the two stages is instantaneous. The former case corresponds to a fully visible, while the latter to a fully invisible particle, which results in an abrupt change during animation (figure 3).

The issue of billboard clipping artifacts has been addressed in [6, 7] where the splitting of the intersected billboards is proposed. Instead of using the z-buffer to identify the visible fragments, a back to front rendering is executed, which draws two billboards instead of a single intersected billboard, one before, and the other after the object rendering. While this approach eliminates artifacts in case of a single object, it gets unmanageable for many included objects.

If billboard pixels are either fully visible or fully opaque, then another possibility exists to mix billboards and conventional 3D objects. To avoid incorrect clipping, billboards are equipped with per pixel depth information as happens, for example in *nailboards* [15] and 2.5D impostors [18, 20].

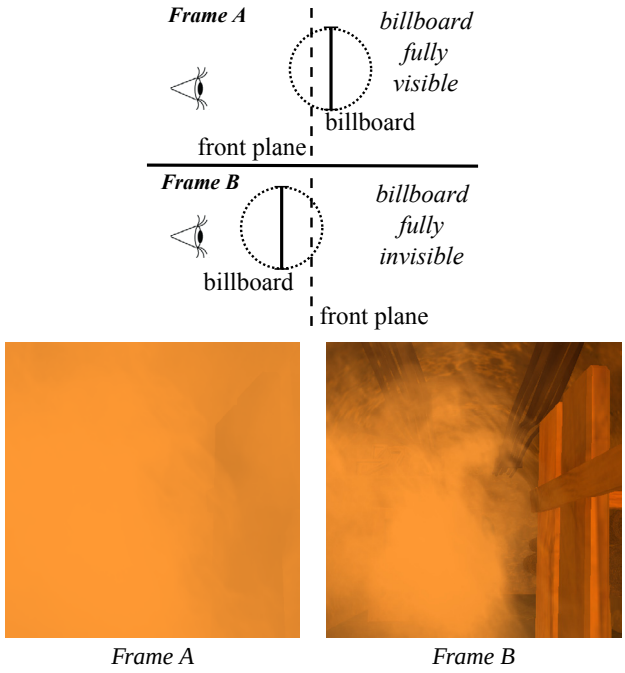

Figure 3: Billboard popping artifact. Where the billboard gets to the other side of the front clipping plane, the transparency is discontinuous in time (the figures show two adjacent frames of an animation).

In this paper we propose a novel solution for including objects into the participating medium without billboard clipping and popping artifacts. The basic idea of spherical billboards is introduced in section 2. Unlike nailboards, the proposed method can also handle semi transparent billboards needed for natural phenomena, such as fire, smoke, clouds, and explosions. Instead of associating depth textures to billboards, we obtain two depth values analytically, one for the front, and another one for the back surfaces, and compute the opacity according to the real distance the light travels inside the particle. The proposed idea is used then to render realistic explosions in real-time in section 3. In our explosion rendering system fire and smoke particles are animated and colored in a physically plausible way similarly to [10, 11].

2 Spherical billboards

Billboard clipping artifacts are solved by calculating the real path length a light ray travels inside a given particle since this length determines the opacity value to be used during rendering. The traveled distance is obtained from the spherical geometry of particles instead of assuming that a particle can be represented by a planar rectangle. However, in order to keep the implementation simple and fast, we still send the particles through the rendering pipeline as quadrilateral primitives, and take into account the spherical shape only during fragment processing.

To find out where opaque objects are during particle rendering, first these objects are drawn and the resulting depth buffer storing camera space z coordinates is saved in a texture.

Having rendered the opaque objects, particle systems are processed. The particles are rendered as quads perpendicular to axis z of the camera coordinate system. These quads are placed at the farthest point of the particle sphere to avoid unwanted front plane clipping. Disabling depth test is also needed to eliminate incorrect object–billboard clipping.

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 considering the saved depth values of opaque objects and the camera's front clipping plane distance. From the resulting interval we can compute the opacity for each fragment in such a way that both fully and partially visible particles are displayed correctly, giving the illusion of a volumetric medium. During opacity computation we assume that the density is uniform inside a particle sphere.

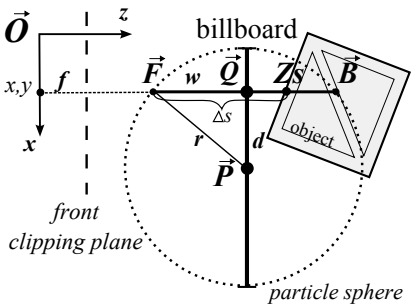

Figure 4: Computation of length Δs the ray segment travels inside a particle sphere of radius r and of center $\vec{P}$.

For the formal discussion of the processing of a single particle, let us use the notations of figure 4. We consider the fragment processing of a particle of center $\vec{P} = (x_p, y_p, z_p)$ and of radius r. The particle is rendered as a quad perpendicular to axis z. Suppose that the current fragment corresponds to the visibility ray cast through point $\vec{Q} = (x_q, y_q, z_q)$ of the quadrilateral. Although visibility rays start in the origin of the camera coordinate system and thus form a perspective bundle, for the sake of simplicity, we consider them as being parallel with axis z. Note that it would also be possible to trace rays not parallel with axis z, or even rotating billboards to be perpendicular at their centers with the viewing direction. However, we have found that the orthographic projection approximation is acceptable if the perspective distortion is not too strong, and requires the simplest formulae to implement. If the visibility ray is parallel with axis z, then the z coordinates of $\vec{P}$ and $\vec{Q}$ are identical, i.e. $z_q = z_p$. The distance between the ray and the particle center is

$$d = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2}.$$

The closest and the farthest points of the particle sphere on the ray from the camera are $\vec{F}$ and $\vec{B}$, respectively. The distances of these points from the camera can be obtained as

$$|\vec{F}| \approx z_p - w, \quad |\vec{B}| \approx z_p + w,$$

where $w = \sqrt{r^2 - d^2}$. The ray travels inside the particle in interval $[|\vec{F}|, |\vec{B}|]$.

Taking into account the front clipping plane and the depth values of opaque objects, these distances may be modified. First to eliminate popping artifacts, we should ensure that $|\vec{F}|$ is not smaller than the front clipping plane distance f, thus the distance the ray travels in the particle before reaching the front plane is not be included. Secondly we should also ensure that $|\vec{B}|$ is not greater than Z_s which is the stored object depth at the given pixel, thus the distance traveled inside the object is not considered.

From these modified distances we can obtain the real length the ray travels in the particle:

$$\Delta s = min(Z_s, |\vec{B}|) - max(f, |\vec{F}|).$$

2.1 "Gouraud shading" for spherical billboards

Assuming that the density is homogeneous inside a particle and using equation 2 to obtain the respective opacity value correspond to piece-wise constant finite-element approximation. While constant finite-elements might be acceptable from the point of view of numerical precision, their application results in annoying visual artifacts.

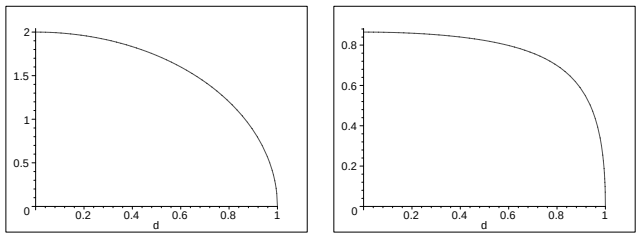

Figure 5: The accumulated density of a ray (left) and its seen opacity (right) as the function of the distance of the ray and the center in a unit sphere with constant, unit density.

Figure 5 depicts the accumulated density ($\int_{\Delta s_j} \tau(s)ds$) and the respective opacity as a function of ray–particle distance d, assuming constant finite-elements. Note that at the contour of the particle sphere ($d = 1$) the accumulated density and the opacity become zero, but they do not diminish smoothly. The accumulated density has a significant derivative at this point, which makes the contour of the particle sphere clearly visible for the human observer.

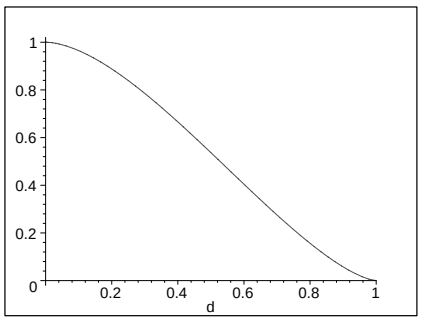

Figure 6: The accumulated density of a ray as the function of the distance of the ray and the center in a unit sphere with density function linearly decreasing with the distance from the particle center.

This artifact can be eliminated if we use piece-wise linear rather than piece-wise constant finite-elements, that is, the density is supposed to be linearly decreasing with the distance from the particle center. The accumulated density computed with this assumption is shown by figure 6. Note that this function is very close to a linear function, thus the effects of piece-wise linear finite elements can be approximated by modulating the accumulated density by this linear function. It means that instead of equation 2 we use the following formula to obtain the opacity of particle j:

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

where d is the distance between the ray and the particle center, and r_j is the radius of this particle. Note that this approach is very similar to the trick applied in radiosity methods. While computing the patch radiosities piece-wise constant finite-elements are used, but the final image is presented with Gouraud shading, which corresponds to linear filtering.

2.2 GPU Implementation of spherical billboards

The length the ray travels in a particle sphere and the corresponding opacity can be computed by a custom fragment shader program. The fragment program gets some of its inputs from the 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 (`scr`). The fragment program also gets uniform parameters, including the texture of the depth values of opaque objects (`Depth`), the density (`tau`), and the camera's front clipping plane distance (`f`). The fragment program calls the following function to compute the particle opacity at the shaded fragment:

```
float Opacity(float3 P, float3 Q, float r, float2 scr) {
  float alpha = 0;
  float d = length(P.xy - Q.xy);
  if(d < r) {
    float w  = sqrt(r*r - d*d);
    float F  = P.z - w;
    float B  = P.z + w;
    float Zs = tex2D(Depth, scr);
    float ds = min(Zs, B) - max(f, F);
    alpha = 1 - exp(-tau * (1-d/r) * ds);
  }
  return alpha;
}
```

With this simple calculation the shader program obtains the real ray segment length (`ds`) and computes opacity `alpha` of the 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 figures 1 and 7).

Figure 7: Particle system rendered with spherical billboards.

3 Rendering explosions

The proposed spherical billboard technique can replace planar billboards in all participating media rendering application. In this section, we consider a single example, the explosion rendering. Explosions consist of dust, smoke, and fire, which are modeled by specific particle systems. Dust and smoke absorb light, fire emits light. These particle systems are rendered separately, and the final result is obtained by compositing their rendered images.

3.1 Dust and smoke

Smoke particles are responsible for absorbing light in the fire. These particles typically have low albedo values ($a = 0.2, \tau = 0.4$). High albedo ($a = 0.9, \tau = 0.4$) dust that is swirling in the air, on the other hand, is added to improve the realism of the explosion (figure 8). When rendering dust and smoke we assume that these particles do not emit radiance so their emission term is zero. To calculate the in-scattering term, the length the light travels in the particle sphere, the albedo, the density, and the phase function (see equation 1) are needed. We use the Henyey-Greenstein phase function [8, 3]:

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

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

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

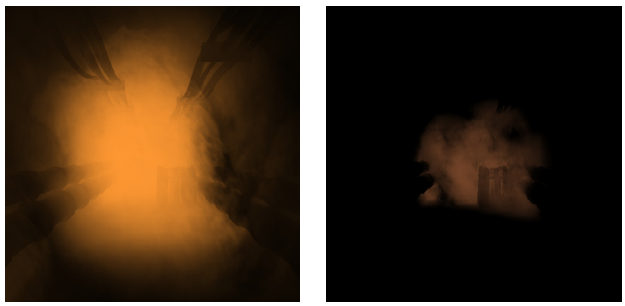

Figure 8: High albedo dust and low albedo smoke.

In order to maintain high frame rates, the number of particles should be limited, which may compromise high detail features. To cope with this problem, the number of particles is reduced while their radius is increased. The variety needed by the details is added by perturbing the opacity values computed by spherical billboards. Each particle has a unique, time dependent perturbation pattern. The perturbation is extracted from a grey scale texture, called *detail image*, which depicts real smoke or dust (figure 9). The perturbation pattern of a particle is taken from a randomly placed, small quad shaped part of this texture. This technique has been used for a longer time by off-line renderers of the motion picture industry [1]. As time advances this texture is dynamically updated to provide variety in the time domain as well. Such animated 2D textures can be obtained from real world videos and stored as a 3D texture since inter-frame interpolation and looping can automatically be provided by the graphics hardware's texture sampling unit [11].

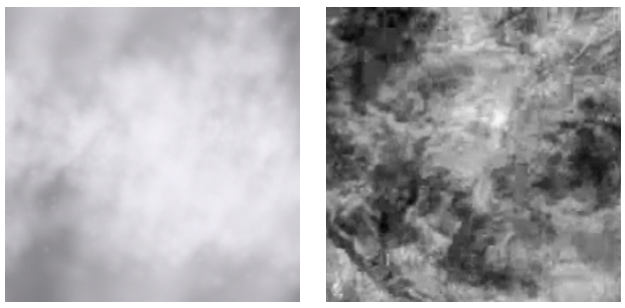

Figure 9: Images from real smoke and fire video clips, which are used to perturb the billboard fragment opacities and temperatures.

3.2 Fire

Fire is modeled as a black-body radiator rather than participating medium, i.e. the albedo in equation 1 is zero, so only the emission term is needed. The color characteristics of fire particles are determined by the physics theory of black-body radiation. For wavelength λ, the emitted radiance of a black-body can be computed by Planck's formula:

$$L^e(\lambda) = \frac{2C_1}{\lambda^5(e^{C_2/(\lambda T)} - 1)}$$

where $C_1 = 3.7418 \cdot 10^{-16} W m^2$, $C_2 = 1.4388 \cdot 10^{-2} m^\circ K$, and T is the absolute temperature of the radiator [16]. Figure 10 shows the spectral radiance of black-body radiators at different temperatures. Note that the higher the temperature is, the more blueish the color gets. For different temperature values, the RGB components can be obtained by integrating the spectrum multiplied by the color matching functions. These integrals can be precomputed and stored in a texture. To depict realistic fire, the temperature range of $T \in [2500^\circ K, 3200^\circ K]$ needs to be processed (see figure 11).

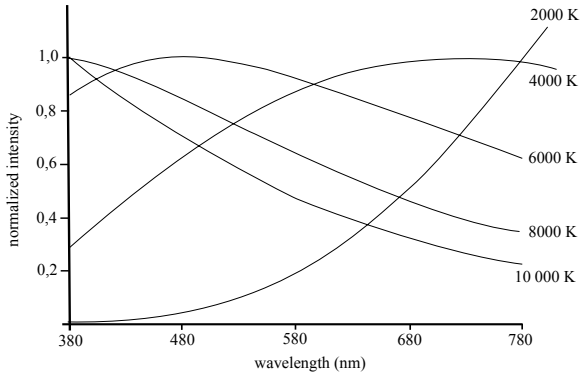

Figure 10: Black-body radiator spectral distribution

Figure 11: Black-body radiator colors from $0^\circ K$ to $10000^\circ K$. Fire particles belong to temperature values from $2500^\circ K$ to $3200^\circ K$.

The opacity of fire particles is calculated by the spherical billboards. High detail features are added to fire particles similarly to smoke and dust particles. However, now not the opacity, but the emission radiance should be perturbed. We could use a color video and take the color samples directly from this video, but this approach would limit the freedom of controlling the temperature range and color of different explosions. Instead, we decided to store the temperature variations in the detail texture (figure 9), and its stored temperature values are scaled and are used for color computation on the fly. A randomly selected, small quadrilateral part of a frame in the detail video is assigned to a fire particle to control the temperature perturbation of the fragments of the particle billboard. The temperature is scaled and a bias is added if required. Then the resulting temperature is used to find the color of this fragment in the black-body radiation function.

The fragment program gets fire particle position in camera space (`P`), the shaded billboard point in camera space (`Q`), the particle radius (`r`), the screen coordinates of the shaded point (`scr`), and the position of the detail image in the texture (`detail`) and the starting time of the animation in `time`. The fragment program also gets uniform parameters, including the texture of the fire video (`FireVideo`), the black body radiation function of figure 11 (`BBRad`), the depth values of opaque objects (`Depth`), the density (`tau`), the camera's front clipping plane distance (`f`), and the temperature scale and bias in `T1` and `T0`, respectively. The fragment program calls the `Opacity` function of subsection 2.2 and computes the color of the fire particle with the following Cg program:

```
float alpha = Opacity(P, Q, r, scr);
float3 detuvw = detail + float3(0,0,time);
float T = T0 + T1 * tex3D(FireVideo, detuvw);
return float4(tex1D(BBRad, T).rgb, 1) * alpha;
```

4 Layer Composition

To combine the particle systems together and with the image of opaque objects, a layer composition method has been used. This way we should render the opaque objects and the particle systems into separate textures, and then compose them. This leads to three rendering passes: one pass for opaque objects, one pass for dust, fire, and smoke, and one final pass for composition. The first pass computes both the color and the depth of opaque objects.

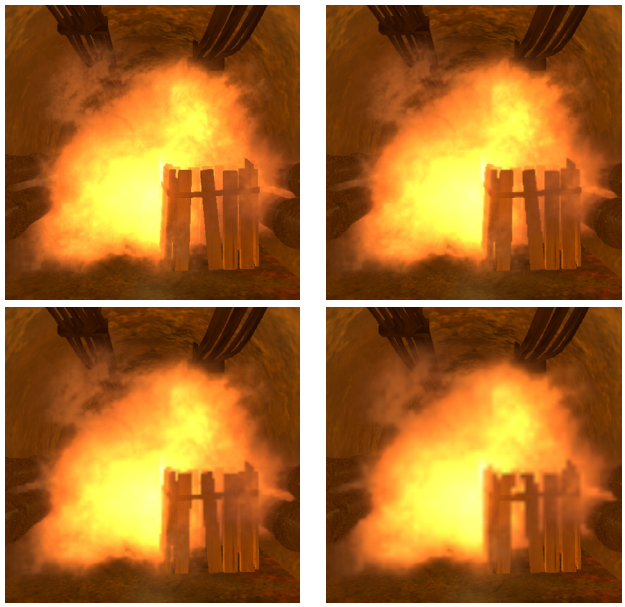

Figure 12: Particles rendered to render targets of different resolutions. Upper left: particle render target with screen resolution (35 FPS). Upper right, lower left, lower right: render target with half (60 FPS), quarter (110 FPS) and one eights (185 FPS) of the screen resolution.

One great advantage of rendering the participating medium into a texture is that we can use floating point blending. Another advantage is that this render pass may have smaller resolution rendering target than the final display resolution, which considerably speeds up rendering since blending needs a huge amount of pixel processing power to overdraw a pixel many times (see figure 12).

To enhance realism, we also simulated heat shimmering that distorts the image [11]. This is done by rendering particles of a noisy

texture. This noise is used in the final composition as u, v offset values to distort the image (figure 13). With the help of multiple render targets, this pass can also be merged in the pass of rendering of fire particles.

The final effect that could be used through composition is motion blur, which can easily be done with blending, letting the new frame fade into previous frames. The complete rendering process is shown in figure 14.

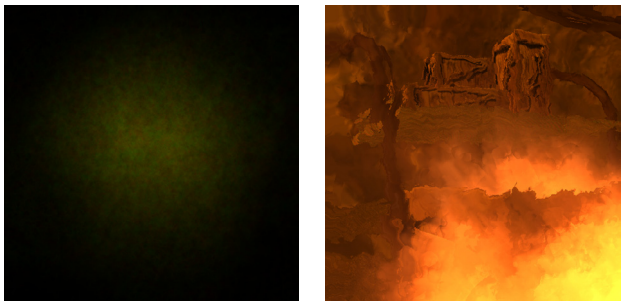

Figure 13: Heat noise texture and the final distorted image.

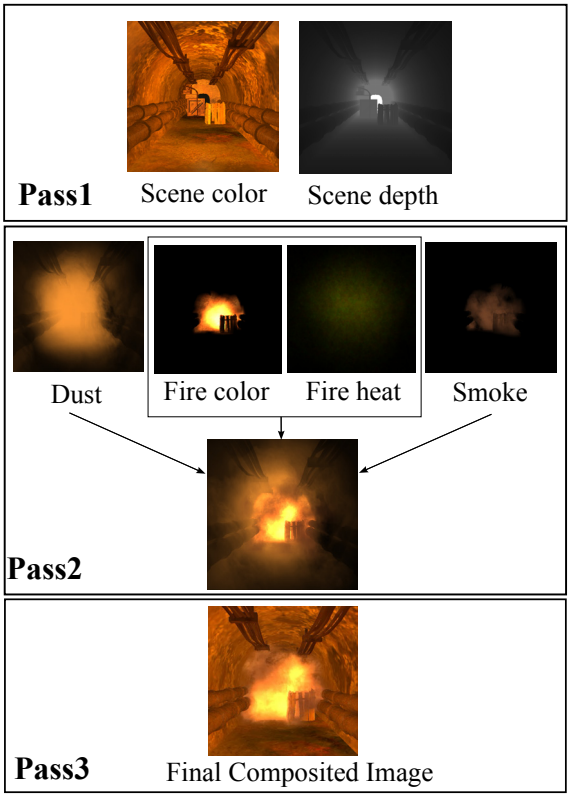

Figure 14: Rendering algorithm

5 Results

The presented algorithm has been implemented in OpenGL/Cg environment on an NV7800GT graphics card. The dust, the fire, and the smoke consist of 16, 135, and 112 animated particles, respectively. Note that these small number of larger particles can be simulated very efficiently, but thanks to the opacity and temperature perturbation, the high frequency details are not compromised. The modeled scene consists of 16800 triangles. The scene is rendered with per pixel Phong shading. During particle simulation we detected collisions between particles and a simplified scene geometry. Our algorithm offers real-time rendering speed (70 FPS) in 512×512 resolution windowed mode, providing high details (see figure 15). The frame rate strongly depends on the number of overwritten pixels. For comparison, the scene without the particle system is rendered at 370 FPS, and the classic billboard rendering method would run at about 80 FPS. This means that the performance lost due to the more complex spherical billboard calculations can be regained by decreasing the render target resolution during particle system drawing.

Frame rate data measured with different particle numbers are shown by table 1, having set the resolution of the render target for the opacity to the half of the final image resolution. Note that when higher number of particles are used, the billboard sizes are decreased proportionally. This means that only the particle simulation and vertex processing time are proportional to the number of particles, the fragment processing time remains constant.

particles	0	263	526	1052	1578
Planar	370	80 FPS	60 FPS	45 FPS	40 FPS
Spherical	370	70 FPS	56 FPS	40 FPS	35 FPS

Table 1: Frame rates with different particle numbers

6 Conclusion

This paper proposed to consider particles as spheres rather than planar billboards during fragment processing, while still rendering them as billboards. Spherical billboards eliminated billboard clipping and popping artifacts. The paper also introduced an efficient method to display explosions composed of fire, smoke, and dust. The discussed method also used post rendering effects and runs at high frame rates.

Acknowledgements

This work has been supported by OTKA (T042735), GameTools FP6 (IST-2-004363) project, and by Hewlett-Packard and the National Office for Research and Technology (Hungary).

References

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

[2] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH '82 Proceedings*, pages 21–29, 1982.

[3] W. Cornette and J. Shanks. Physical reasonable analytic expression for single-scattering phase function. *Applied Optics*, 31(16):31–52, 1992.

[4] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *ACM SIGGRAPH 2001*, pages 15–22, 2001.

[5] B. E. Feldman, J. F. O'Brien, and O. Arikan. Animating suspended particle explosions. In *ACM SIGGRAPH 2003*, pages 708–715, 2003.

[6] M. Harris and A. Lastra. Real-time cloud rendering. *Computer Graphics Forum*, 20(3), 2001.

[7] M. J. Harris. Real-time cloud rendering for games. In *Game Developers Conference*, 2002.

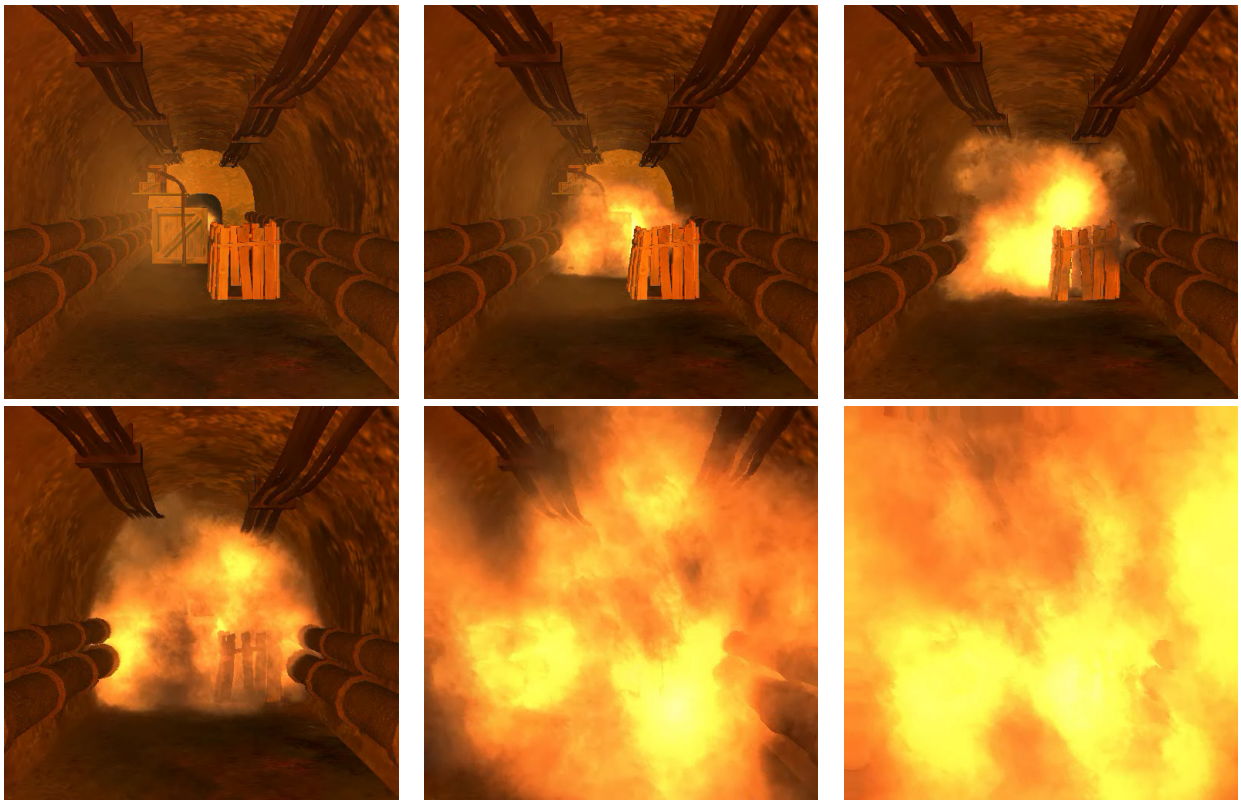

Figure 15: Rendered frames from the animation sequence.

[8] G. Henyey and J. Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal*, 88:70–73, 1940.

[9] J. Krüger and R. Westermann. GPU simulation and rendering of volumetric effects for computer games and virtual environments. *Computer Graphics Forum*, 24(3):685–693, 2005.

[10] D. C. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire. In *ACM SIGGRAPH 2002*, 2002.

[11] H. Nguyen. Fire in the vulcan demo. In Fernando R., editor, *GPU Gems*, pages 359–376. Addison-Wesley, 2004.

[12] W. T. Reeves. Particle systems - techniques for modelling a class of fuzzy objects. In *SIGGRAPH '83 Proceedings*, pages 359–376, 1983.

[13] H. Rushmeier, A. Hamins, and M. Y. Choi. Volume rendering of pool fire data. *IEEE Computer Graphics and Applications*, 15(4):62–67, 1995.

[14] G. Schaufler. Dynamically generated impostors. In *I Workshop - Virtual Worlds - Distributed Graphics*, pages 129–136, 1995.

[15] G. Schaufler. Nailboards: A rendering primitive for image caching in dynamic scenes. In *Eurographics Workshop on Rendering*, pages 151–162, 1997.

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

[17] J. Stam and E. Fiume. Depiction of fire and other gaseous phenomena using diffusion processes. In *ACM SIGGRAPH 95*, pages 129–136, 1995.

[18] G. Szijártó. 2.5 dimensional impostors for realistic trees and forests. In Kim Pallister, editor, *Game Programming Gems 5*, pages 527–538. Charles River Media, 2005.

[19] L. Szirmay-Kalos, M. Sbert, and T. Umenhoffer. Real-time multiple scattering in participating media with illumination networks. In *Proceedings of the 2005 Eurographics Symposium on Rendering*, pages 277–282, 2005.

[20] T. Umenhoffer and L. Szirmay-Kalos. Real-time rendering of cloudy natural phenomena with hierarchical depth impostors. In *Eurographics Conference. Short papers.*, 2005.

[21] X. Wei, W. Li, K. Mueller, and A. Kaufman. Simulating fire with texture splats. In *IEEE Visualization '02*, 2002.