pages 146-149; definitive version available from http://dx.doi.org/10.1109/VS-GAMES.2013.6624242

Smoke and Shadows: Rendering and Light Interaction of Smoke in Real-Time Rendered Virtual Environments

Christopher J. Bass
Faculty of Engineering and Computing
Coventry University
Coventry, UK

Eike Falk Anderson
The National Centre for Computer Animation
Bournemouth University
Poole, UK

***Abstract*—Realism in computer graphics depends upon digitally representing what we see in the world with careful attention to detail, which usually requires a high degree of complexity in modelling the scene. The inevitable trade-off between realism and performance means that new techniques that aim to improve the visual fidelity of a scene must do so without compromising the real-time rendering performance.**

We describe and discuss a simple method for realistically casting shadows from an opaque solid object through a GPU (graphics processing unit) based particle system representing natural phenomena, such as smoke.

***Keywords*—Computer Graphics, Virtual Environments, Natural Phenomena.**

I. Introduction

A successful virtual environment as depicted in modern computer games depends on immersing the user in a convincing virtual world and lighting plays an important element in the creation of convincing 3D virtual worlds. Real-time speeds are also essential so that the user can interact with the virtual environment without any delays, which would break the user's immersion. There are a variety of well-known existing lighting techniques that are capable of attaining real-time performance. Many of these are designed for the lighting of opaque solid objects, but when confronted with gaseous phenomena such as smoke, which is translucent, these lighting models become inadequate. While there are other methods for simulating these phenomena [1], a frequently used technique for representing such insubstantial substances are particle systems.

Advances in computer graphics hardware and 3D graphics APIs (application programming interfaces) have freed software developers from using the fixed function graphics pipeline. With the arrival of the programmable pipeline, developers can now utilise the GPU in new ways, allowing new effects to be created to improve graphics in applications such as computer games. New and more accurate illumination models than previously possible, as well as particle systems, can now be implemented through the use of programmable shaders, resulting in the creation of more realistic virtual environments. The addition of programmable geometry shaders to the already existing vertex and pixel shaders allows for a range of new effects as geometry shaders can generate new geometry on the fly by creating and appending new primitives to the shader's output stream. Input and output streams are not required to have the same topology, so the geometry shader can be used to turn single input points into billboards (camera aligned rectangular textured primitives).

When particle systems are used, their shadowing helps to define their position and shape, with different kinds of shadowing effects resulting from different shadowing techniques:

- *Cast shadows* are shadows that are cast by a particle system and that are visible on other solid objects in the virtual scene.
- *Self-shadows* are shadows that are cast by particles and that affect other particles in the same particle system by overshadowing them.
- *External shadows* are shadows that are cast by solid models in the virtual scene onto a particle system, placing the particles in shadow.

The use of shadow maps is a common approach to implementing cast shadows from particle systems. Particles are often represented as translucent billboards, therefore requiring shadowing techniques to be implemented differently from shadows cast from solid opaque objects, and their translucency becomes an issue. These difficulties can make shadows look unrealistic and therefore a more advanced shadowing approach may be required. Cast shadows often tie in with the self-shadowing of a particle system where the resulting self-shadowing information, often stored as a shadow map, can be reused to create the cast shadows.

Extending our previous work [2], in this paper we describe a simple method for casting external shadows onto a volume of 'smoke' particles, simulated using a dynamic GPU-based particle system. This method renders particles as billboards and is capable of running at interactive frame rates that would allow it to be implemented within a computer game or other real-time rendered virtual environment.

Published in Proceedings of VS-GAMES 2013: 5th International Conference on Games and Virtual Worlds for Serious Applications, pages 146-149; definitive version available from http://dx.doi.org/10.1109/VS-GAMES.2013.6624242

II. Related Work

The use of shadow volumes [3] is a common method for adding shadows to solid objects in a scene. Shadow volumes can be created from the solid object geometry by extruding the back face of the model to infinity. Pixels inside the shadow volume are rendered as shaded whereas those outside are not, which is usually determined via a stencil buffer implementation, such as 'depth fail' [4] which is also known as 'Carmack's reverse' [5].

Shadow maps [6] provide an alternative method for adding shadows to a scene. For this, first a height map is generated from the light source's point of view, mapping the distance from the light source to all the occluding shadow objects. Then, during the rendering of the scene, a comparison is made between the distance of the pixel to the light and the value that is stored in this height map. If the distance to the light is greater than the value stored in the height map then the pixel is in shadow and rendered accordingly.

The main techniques for the simulation of smoke can be broken down into different categories and include particle-based Lagrangian systems, grid-based Eulerian methods, and hybrids. The particle system Lagrangian approach involves modelling smoke as a group of particles that are born in an initial position and state and then move according to a set of rules or equations until they are destroyed or reset. A good survey of computer graphics representations for smoke is presented by Stopford [7].

Particle systems are well suited to the SIMD (single instruction multiple data) architecture of GPUs as a single instruction can be used to simultaneously govern many particles of the same particle system. The parallel processing capabilities of modern GPUs provide a vast improvement to simulation speeds for most particle systems and for smoke simulation via a particle system a GPU implementation is advisable to reap the benefits of parallel processing. A common method for a GPU implementation involves using textures with particle data stored in its colour channels. This approach was used by Latta [8] to create a GPU-based particle system containing roughly one million particles. Kolb et al. [9] built on this by adding support for collision detection with geometry. Now with the geometry shader and the stream out functionality of modern graphics hardware [10], particles can be stored as vertices with simulation operations performed in the geometry shader. Latta [11] presents a useful overview on particle systems, briefly describing the main methods for implementing them on CPU and GPU.

A particle system which represents smoke should interact with external shadows from other objects in the scene so as to look realistic and properly integrated into the scene, which it otherwise would not (Fig. 1). Most methods for self-shadowing of particle systems require particles to be sorted along an axis so that the opacity and shadowing information can be accumulated for each particle in the correct order. This process is computationally demanding but there are optimisations, which can reduce this complexity, e.g. those

Fig. 1. GPU vertex based 'smoke' particle system without any form of shadowing.

employed by Green [12]. Deep shadow maps [13] can be used to add self-shadowing to particle systems with translucent particles. Whereas regular shadow maps store a single depth value for each pixel, a deep shadow map stores a 'visibility function' which estimates the amount of light that passes through at different depths. Unfortunately deep shadow maps are computationally expensive and therefore unsuitable for a demanding real-time solution. Opacity shadow maps [14] on the oher hand provide a real-time solution for self-shadowing. They are distributed throughout the volume and face towards the light source, each rendered from the point of view of the light source and accumulating the alpha values of the particles contained up to the depth of the opacity map. These opacity maps store opacity values at different depths through the volume which can then be used to calculate the shadowing at different depths when rendering the volume.

III. Shadowed Smoke Rendering Method

We use a very simple particle system that stores particles as vertices – using vertex buffers for storage – to simulate smoke. The system, which does not attempt to accurately simulate smoke, uses a seeding buffer for initialisation and particles are spawned in a grid from a ground plane, drifting upwards along the y-axis before being removed from the simulation. The particle system runs entirely on the GPU, i.e. particles are created and destroyed in the geometry shader, so there is no need for CPU intervention. Input and output buffers are used afterwards to store the particle system simulation data which are swapped each frame. Once the particle system has been updated, the remaining scene objects are updated.

In our attempts to add external shadows cast onto the particle system we have experimented with both shadow mapping and shadow volume techniques. We found shadow mapping to be the more suitable solution for our purposes,

Published in Proceedings of VS-GAMES 2013: 5th International Conference on Games and Virtual Worlds for Serious Applications, pages 146-149; definitive version available from http://dx.doi.org/10.1109/VS-GAMES.2013.6624242

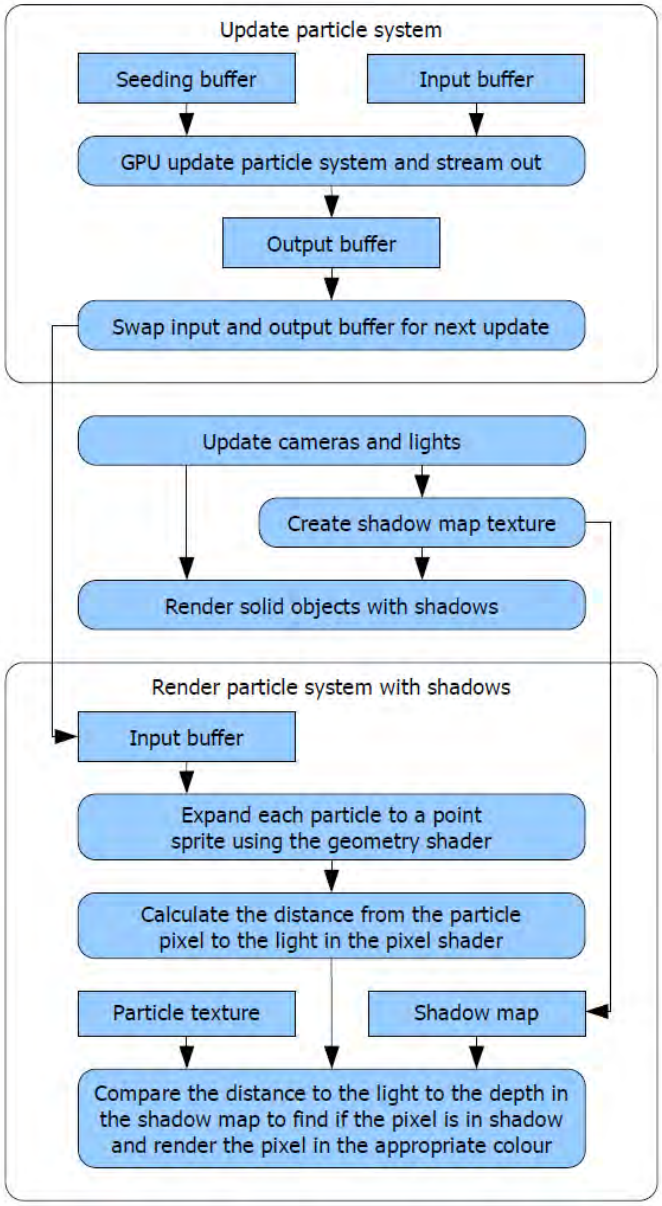

Fig. 2. Flowchart of the processes involved in rendering a frame with our technique.

as this requires only a simple operation to determine if a particle is located in shadow or not and as it is trivial to pass a shadow map as a texture input when rendering our GPU vertex based particle system.

When rendering a frame, a shadow map is created and the scene objects are rendered with shadows. The shadow map is then reused when rendering the particle system to apply external shadowing to the particle system. We then sample and compare the depth in the shadow map to the actual depth of the particle in the pixel shader to determine if a particle pixel is located in shadow and accordingly render it with the appropriate colour. Fig. 2 shows a flowchart describing the processes that take place for rendering a single frame using this technique.

Our initial implementation [2] suffered from aliasing artefacts resulting in rough pixellated edges of the shadows. As we were using shadow mapping, our method also suffered from these artefacts throughout the 3D shadow, as there are hard edges where the shadow cuts through particles. Soft particles [15] are a technique to remove the hard edges that occur where a 2D billboard intersects 3D geometry. The original soft particles technique uses the scene depth buffer when rendering particles and as the distance between the particle and the depth buffer shortens, alpha blending is used to blend the particle out. If integrated with our technique, soft particles could be used to alpha blend particle pixels which intersect 3D geometry, but using the original soft particles technique would not remove artefacts caused by 2D (billboard) particles intersecting our shadow map. To overcome this we have developed a similar, simple solution which uses the depths stored within the shadow map to blend particle pixels.

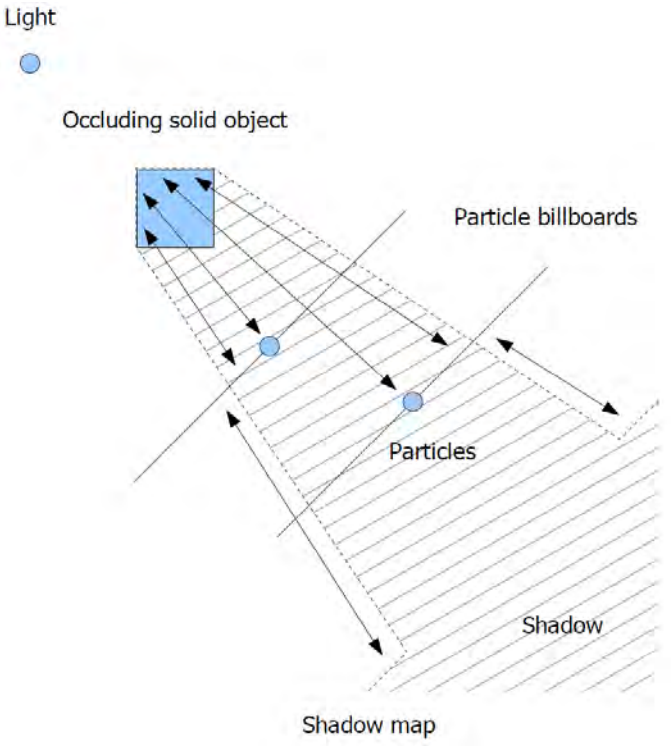

Fig. 3. Soft particles technique used in our system.

Fig. 3 illustrates how we have implemented this modified soft shadow technique by using the shadow map depths. This technique supports particle billboards that are partially in shadow and partially lit. To overcome the limitations of our initial implementation we sample the shadow map in the vertex shader, calculate the distance between the particle position and the shadow map sample and pass the result to the pixel shader. In the pixel shader we then repeat a similar process to calculate the distance between the pixel and the shadow map sample. This then allows us to calculate the pixel shading using a simple weighting function between the two depths: as the depth values change, pixels can be smoothly shaded from being in full shadow to being fully lit without any hard edges appearing, as there is no longer a sharp cut-off between pixels in shadowed and lit areas.

Published in Proceedings of VS-GAMES 2013: 5th International Conference on Games and Virtual Worlds for Serious Applications, pages 146-149; definitive version available from http://dx.doi.org/10.1109/VS-GAMES.2013.6624242

IV. Results

Fig. 4 shows results using our initial technique. While running at real-time speeds on modern consumer graphics hardware, there are some artefacts as a hard edge appears at the cut-off between pixels located in shadowed and lit areas.

Fig. 4. Initial implementation with external shadowing effect. Artefacts (hard edges) within the 'smoke' are visible.

Fig. 5 shows results using our improved technique with our version of soft particles. While still running at real-time speeds, the artefacts are significantly reduced.

Fig. 5. Final results, including our soft particles (inspired by Lorach [15]), rendered at 124 frames per second on an ATI Radeon HD 5770 graphics card.

V. Summary and Future Work

In this paper we have presented a simple real-time 3D shadowing method for casting external shadows onto dynamic, translucent particle systems, suitable for use in virtual environments, such as computer games. Our technique is implemented on a GPU vertex based particle system using a typical billboarding approach for rendering particles to represent smoke, and improved by the addition of a soft particles technique for reducing hard edge artefacts. Not only does this system result in visually convincing scenes, but keeping the particle system on the GPU also allows the application to run at real-time speeds and frees the CPU up for other tasks.

Future work will likely concentrate on further reducing rendering artefacts, possibly by employing a different shadow mapping technique, such as PCSS (percentage-closer soft shadows) [16] to remove aliasing from the shadow map, which would have the added benefit of providing a varied soft edge to the 3D shadow depending on the distance from the light and the occluding model. Another avenue for exploration may be the use of volumetric particles for representing the smoke.

References

[1] K. Zhou, Z. Ren, S. Lin, H. Bao, B. Guo, and H.-Y. Shum, "Real-time smoke rendering using compensated ray marching," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, 2008, pp. 1–12.

[2] C. J. Bass and E. F. Anderson, "Real-time smoke rendering and light interaction," in *Eurographics 2010 - Posters*, A. Hast and I. Viola, Eds. Eurographics Association, 2010.

[3] F. C. Crow, "Shadow algorithms for computer graphics," *SIGGRAPH Comput. Graph.*, vol. 11, pp. 242–248, 1977.

[4] B. Bilodeau and M. Songy, "Real time shadows," in *Creativity 1999, Creative Labs Inc. Sponsored Game Developer Conferences*, 1999.

[5] J. Carmack. (2000) Carmack on shadow volumes. NVIDIA Developer Zone.

[6] L. Williams, "Casting curved shadows on curved surfaces," *SIGGRAPH Comput. Graph.*, vol. 12, pp. 270–274, 1978.

[7] D. Stopford. (2006) Representing smoke in computer graphics. Project Report, The National Centre for Computer Animation, Bournemouth University.

[8] L. Latta, "Building a million particle system," in *Game Developers Conference 2004*, 2004.

[9] A. Kolb, L. Latta, and C. Rezk-Salama, "Hardware-based simulation and collision detection for large particle systems," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 2004, pp. 123–131.

[10] D. Blythe, "The direct3d 10 system," *ACM Trans. Graph.*, vol. 25, pp. 724–734, 2006.

[11] L. Latta, "Everything about particle effects," in *Game Developers Conference 2007*, 2007.

[12] S. Green. (2012) Volumetric particle shadows. NVIDIA Developer Zone.

[13] T. Lokovic and E. Veach, "Deep shadow maps," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '00, 2000, pp. 385–392.

[14] T.-Y. Kim and U. Neumann, "Opacity shadow maps," in *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, 2001, pp. 177–182.

[15] T. Lorach, "Soft particles," NVIDIA DirectX 10 SDK, 2007.

[16] R. Fernando, "Percentage-closer soft shadows," in *ACM SIGGRAPH 2005 Sketches*, 2005.