Fast rendering of opacity-mapped particles using DirectX 11 tessellation and mixed resolutions

Jon Jansen
jjansen@nvidia.com

Louis Bavoil
lbavoil@nvidia.com

February 2011

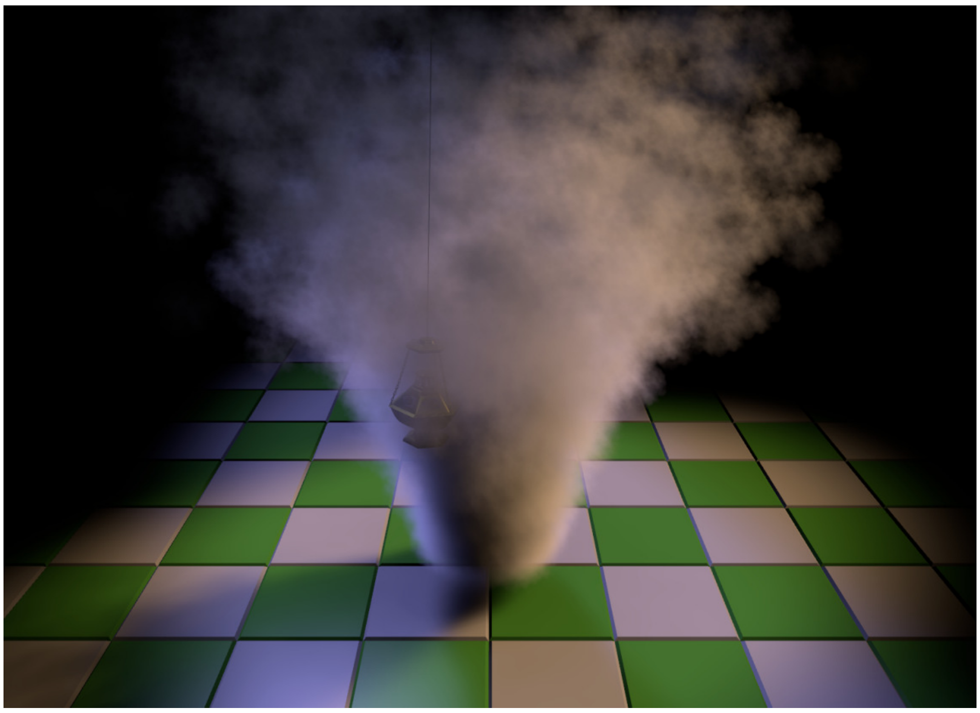
Document Change History

Version	Date	Responsible	Reason for Change
1	22 Feb 2011	Jon Jansen	Initial release

Fast rendering of opacity-mapped particles using DirectX 11 tessellation and mixed resolutions

What the sample shows

This sample application demonstrates efficient rendering of dense particle plumes using four light sources, with self-shadowing applied to all four sources.



The sample uses the following techniques for rendering and acceleration:

- Fourier Opacity Mapping, for rendering self-shadowing effects
- DX11 Tessellation, for acceleration
- Mixed-resolution Particle Rendering with Nearest-Depth Upsampling, for acceleration

Fourier Opacity Mapping

The particles are not opaque, so we cannot use conventional opaque shadow-mapping techniques. Instead, we need to use a technique which takes account of variable translucency, and which is able to model the gradual extinction of light as it travels into the plume from the light source.

The sample uses Fourier Opacity Mapping for this. Fourier Opacity Mapping approximates the variable density of the plume using a truncated Fourier series. Provided the opacity is low and varies smoothly, and as long as sufficient Fourier series terms are used in the approximation, Fourier Opacity Mapping will produce plausible and stable self-shadowing effects.

The technique is implemented in two passes:

1. Render the opacity map – this is conceptually similar to conventional opaque shadow mapping, but instead of depths, we store Fourier series coefficients in a 4xMRT opacity map (i.e. 16 coefficients in total – see Figure 1).
2. Render the particles – again, this is conceptually similar to opaque shadow mapping, but instead of reading and comparing depths, we read Fourier series coefficients and we use the coefficients to model how much light has been extinguished on its journey from source.

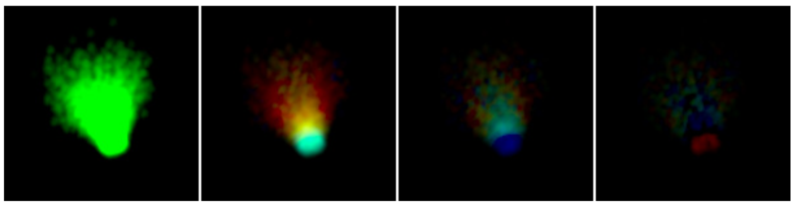


Figure 1 – a Fourier Opacity Map packed into 4xMRT

For further details on Fourier Opacity Mapping, see (Jansen & Bavoil, 2010)

DX11 Tessellation

We rely on the accumulation of many layers of particles to form the impression of a plume, and this gives rise to challenging fill-related workloads.

If we perform per-pixel lighting and self-shadowing calculations, the sample is typically bottlenecked in pixel-shader work. However, if we perform per-*vertex* lighting and self-shadowing, we find that the results are under-sampled and important features are missed or mis-represented (see Figure 3).

Our solution is to use DX11 tessellation to calculate lighting and self-shadowing at an intermediate rate in the Domain Shader.

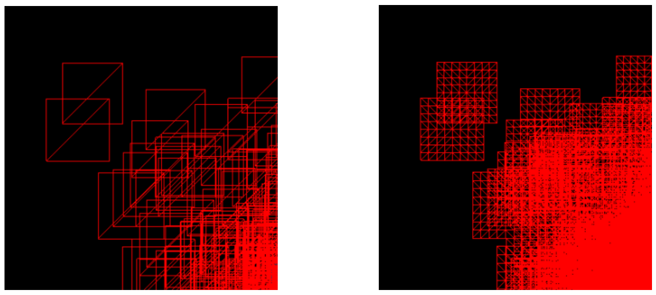


Figure 2 – untessellated (left) vs. tessellated (right) particle billboards

In the sample, we use a simple scheme where the tessellation level is inversely proportional to camera distance and where the results are just indistinguishable from those obtained using per-pixel calculations. This delivers a significant speedup over the non-tessellated per-pixel approach, with no perceptible change in image quality.

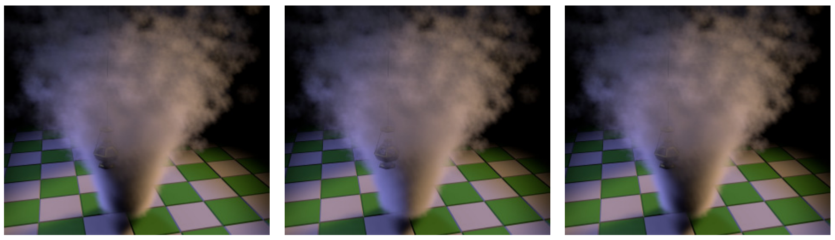


Figure 3 – pixel lighting (left, ground truth), vertex lighting (middle), tessellated lighting (right)

Nearest-Depth Upsampling

Motivation

DX11 tessellation helps to remove much of the pixel-shader burden, but it does nothing to unload the remaining fill-dependent parts of the DX11 pipeline, especially blending in the output-merger stage. We can mitigate this remaining bottleneck by rendering the particles at lower (e.g. half or quarter) resolution.

Rendering particles in low resolution tends to generate aliasing artifacts at the edges between low-resolution particles and full-resolution opaque objects.

For scenes where the opaque objects interacting with the particles are not too thin, convincing results can be achieved with a simple low-resolution render of the particles (typically half resolution or quarter resolution), followed by a depth-aware upsampling filter taking as input the full-resolution opaque depths, the low-resolution opaque depths and the low-resolution particle colors.

Low-Resolution Pass

During the low-resolution rendering pass, the particles need to be depth tested against the depths of the opaque objects that have been rendered so far. There are two orthogonal ways to do this.

First, the least invasive way may be to downsample the full-resolution hardware depth buffer into a low-resolution hardware depth buffer. In DirectX 11, we can do this using a pixel shader to explicitly Load() a particular sample from the full-resolution depth buffer. This is particularly useful when the technique is used with MSAA because it allows fine control over the choice of sample.

Second, the depth test can be performed in the particle's pixel shader by varying the output alpha based on the depth difference between the particle's depth and the opaque depth (soft particles). To maximize performance, the compared depth can be fetched from a low-resolution texture. Various filters may be used for generating the low-resolution depth texture, such as an average or a max filter. We have found that plain hardware point filtering works well with our upsampling filter.

At the end of this pass, the low-resolution color buffer should contain the particle's color in RGB and the particle's visibility $(1\text{-alpha}_0).(1\text{-alpha}_1)\ldots(1\text{-alpha}_n)$ in the destination alpha channel.

Bilinear Upsampling

Now that we have a low-resolution off-screen color buffer available for the particles, we need to upscale this buffer and blend it over the full-resolution color buffer. We do this by rendering a full-screen quad with a pixel shader, and blending the particle's RGBA low-resolution colors over the full-resolution color buffer using the same blend equations as in (Cantlay, 2007).

One way of upscaling the low-resolution particle's colors is to simply fetch the low-resolution texture with hardware bilinear filtering. For particles with smooth textures, plain bilinear filtering may cause jaggies and halos at pixels for which low-resolution particles are occluded by full-resolution opaque objects (see Figure 4).

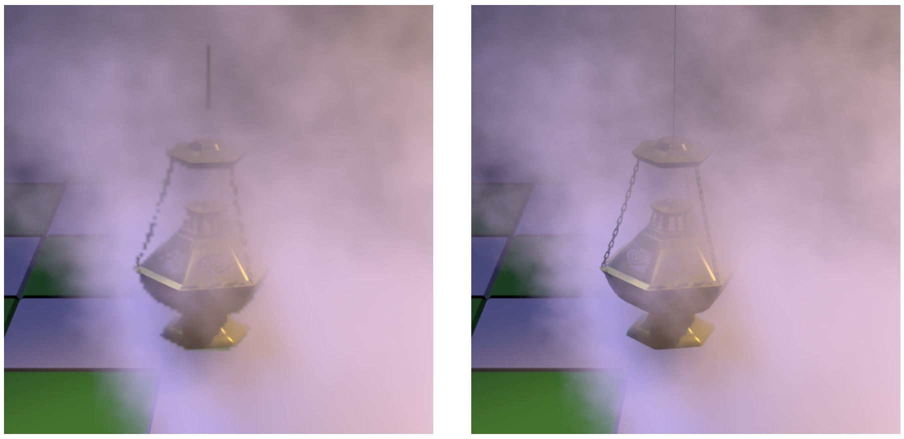


Figure 4 - quarter-resolution particles with plain bilinear upsampling (left), full-resolution particles (right)

This problem may be reduced in some situations by rendering the low-resolution particles with a low-resolution depth buffer downscaled using a MAX filter but this approach does not remove all halo artifacts (Cantlay, 2007).

Nearest-Depth Filter

The idea of nearest-depth filtering is similar to cross bilateral filtering (Eisemann and Durand, 2004) (Petschnigg et al., 2004), the main difference being that only one of the low-resolution color samples is used in the result, not a weighted average. This makes the filter more robust and faster.

The nearest-depth upsampling filter fetches the 2x2 low-resolution depths in the bilinear footprint of the current full-resolution pixel and compares these 4 depths with the full-resolution depth of the current pixel. Then the filter computes which of these four low-resolution depths is nearest to the full-resolution depth and returns the corresponding low-resolution color for that sample. Note that the 2x2 low-resolution depths can be fetched using a single GatherRed instruction with DirectX 11. The nearest-depth filter can reconstruct high-quality edges if the resolution of the low-resolution rendering pass is high enough to capture the opaque-geometry features (see Figure 5).

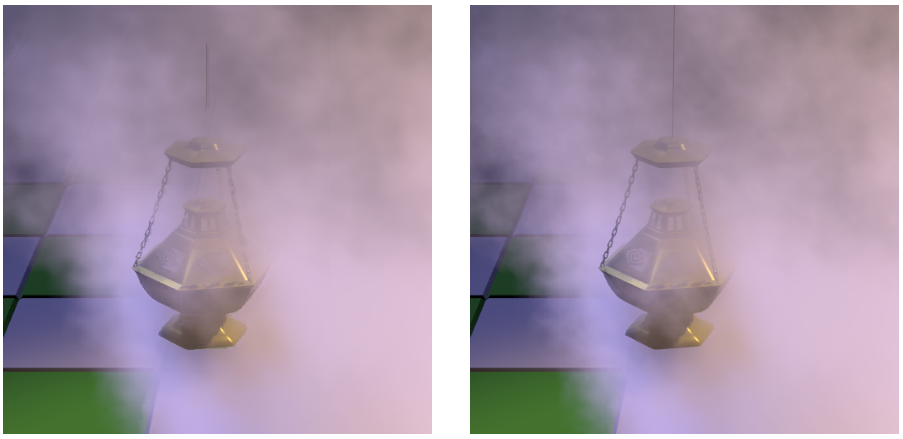


Figure 5 – quarter-resolution particles with nearest-depth filter (left), full-resolution particles (right)

Edge Detection

An issue with the nearest-depth filter is that because it uses only one low-resolution color sample per group of 2x2 texels, if may generate blocky artifacts at non-edge pixels. For these non-edge pixels, bilinear filtering usually works well. Therefore a solution is to use nearest-depth filtering for edge pixels and bilinear filtering for non-edge pixels. Edge pixels can be simply detected by re-using the depth differences that have been computed in the nearest-depth computation: we can classify the current pixel to be a non-edge pixel if all of the absolute depth differences between the full-resolution depth and the low-resolution depths are smaller than a given depth threshold.

The resulting combined filter looks like this, where Z00, Z10, Z01, Z11 are the 2x2 low-resolution linear depths, ZFull is the full-resolution linear depth, and NearestUV is the texture coordinate of the nearest-depth low-resolution sample:

```
if (abs(Z00 - ZFull) < g_DepthThreshold &&
    abs(Z10 - ZFull) < g_DepthThreshold &&
    abs(Z01 - ZFull) < g_DepthThreshold &&
    abs(Z11 - ZFull) < g_DepthThreshold)
{
    return g_LoResColor.Sample(g_SamplerBilinear, LoResUV);
}
else
{
    return g_LoResColor.Sample(g_SamplerNearest, NearestUV);
}
```

This edge-detection test may decide to use the point-sampling branch although there is no opaque-particle interaction at this pixel, which may cause blocky artifacts at these pixels. To minimize the artifacts, the application may compute the minimum and maximum view-space depths for the current particle system's bounding box and clamp the low-resolution depths to this view-space depth range before performing the edge detection.

Such a filter was used in the game Batman: Arkham Asylum, with quarter-resolution or half-resolution rendering depending on the particle systems.

References

Cantlay, I. (2007). High-Speed, Off-Screen Particles. In H. Nguyen, *GPU Gems 3*.

Eisemann, E., & Durand, F. (2004) Flash Photography Enhancement via Intrinsic Relighting. *ACM Trans. Graph. (SIGGRAPH) 23, 3,* (pp. 673–678).

Jansen, J., & Bavoil, L. (2010). Fourier Opacity Mapping. *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, (pp. 165-172).

Kim, T.-Y., & Neumann, U. (2001). Opacity shadow maps. *Proceedings of the 12th Eurographics Workshop on Rendering*, (pp. 177-182).

Petschnigg, G., Agrawala, M., Hoppe, H., Szeliski, R., Cohen, M., & Toyama, K. (2004). Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. (SIGGRAPH) 23, 3,* (pp. 664–672).

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com