SSDM: Screen Space Displacement Mapping

Robin Lobel, November 2008

Fig 1. SSDM applied to RenderMonkey's Ninja Head model

Abstract

This paper present a new approach to displacement/parallax mapping, using screen space as a reference to achieve very fast and global effect. It runs realtime on 2004 hardware, and only requires DX9/Shaders 2.0b.

1. Introduction and related works

Displacement Mapping is a way to add details on a polygonal surface.
Until 2000, it was mostly an offline technique, due to the large amount of computation needed and insufficient processing power.
The idea to use pixel shaders (as found in GPUs) to compute displacement first appeared in 2001 with Parallax Mapping [1].
The technique was later improved with Parallax Occlusion Mapping (2004) [2].
The same year, a precise though slower displacement mapping algorithm was also developed [3].
SSDM is a new approach based on screen space, offering several advantages: faster than POM, handles objects sides and computation cost independent to scene complexity.

2. The idea behind SSDM

Screen Space Displacement Mapping was inspired by SSAO, Screen Space Ambient Occlusion (2007) [4].
Screen Space algorithms render the geometry in a first pass, then a second pass is applied for the effect, based on the first render. This approach has the advantage of being non-redundant: it only applies where needed (rendered pixels). It also have the advantage of providing informations on the surrounding, as the whole scene has been rendered before the effect is applied.
Those two points allows SSDM to render fast whatever the scene complexity, and to provide displacement effect on objects sides, which other methods such as parallax mapping/parallax occlusion mapping does not.

2.1 Memory requirements

This implementation use multiple pass after the geometric pass, at different level of details.
It requires 2 mip-mapping pyramids (A/B) based on screen resolution.

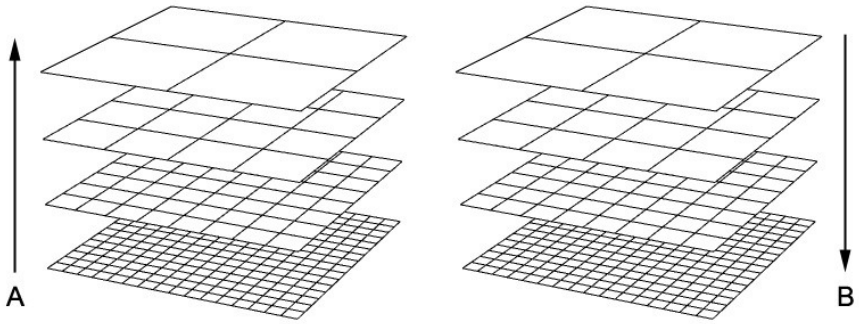

Fig 2. Two mip-mapping pyramids

Theses pyramids uses Half Float Luminance-Alpha texture format (or RGBA8 for wider compatibility with older cards), thus requires *screen resolution*2 pyramids*1.33 (mipmapping)*4* bytes of GPU memory. For instance, for a Full HD resolution (1920*1080), it uses as few as 22MB.

2.2 Displacement vectors

The first pass uses the first level (screen resolution) of pyramid A.
One or more frame buffers are used to render base geometry, providing informations such as diffuse color, normals, specular, etc... What we're interested in is getting the projected displacement vector, which usually equals to *projected normal vector*displacement map*.
We store the 2 coordinates of this vector in the root of pyramid A, using half-float luminance/alpha or BG/RA is we use a BGRA8 texture format.

Then the mip-map pyramid is built, either automatically if the GPU supports it, or manually by averaging 4 texels for 1 upper-level texel.
It's not necessary to build the entire pyramid, a good compromise is to go up 3-4 levels, which reduce a lot computation time, while the risk of losing small details is minimized.

2.3 Displacement

The second step starts with the highest built level of pyramid A, and will only write into pyramid B, at the same level. The idea is to sample around the current texel to see if it is inside the pointed vector area.

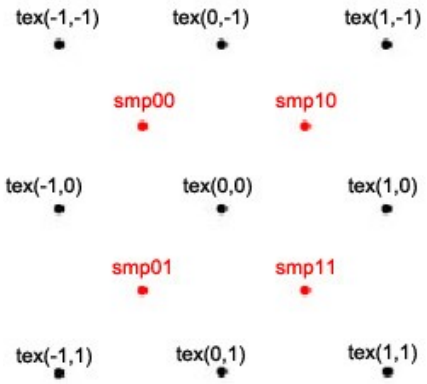

Fig 3. Sampling around center texel

We sample four times around the center texel (*smp00,smp10,smp01,smp11*), the four corners of the center texel area; so the resulting samples are a mix of center and surrounding texels.

Pyramid A is a set of projected displacement vectors, so are the four samples.
As we start with a high level in the pyramid, the surrounding texels are supposed to point roughly around the current texel.

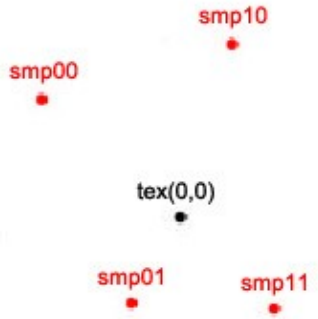

Fig 4. The actual position pointed by samples *smp00,smp10,smp01,smp11*

Then we determine the exact barycenter of this area by averaging *smp00,smp10,smp01,smp11.*
We store this barycenter in the current texel (pyramid B), and we go down one level in both pyramids.

The same process is repeated until we reach level zero of both pyramids, except that we reuse the result stored in pyramid B, one level higher, as our starting point for research (in other words the four samples are taken around the previous calculated barycenter, not according to the current raster position).
Note that in case of no displacement, the previous and new barycenter is equal to the current raster position.

Finally we reach level zero with each texel displaced, pointing to its source texel.
In this final pass, we can reuse other frame buffers to displace accordingly the other informations (diffuse, specular, normals, etc).

3. Results

Theses results were obtained using a GeForce 8800GTS, at 1080p resolution (1920x1080) .
The software used to prototype the shader was AMD RenderMonkey.
SSDM prototype was first written using GLSL (OpenGL Shading Language), then rewrote using HLSL (DirectX Shading Language). Surprisingly, HLSL results were faster than GLSL; so the framerates indicated here are obtained using the HLSL version.

3.1 POM and SSDM speed comparison

Fig 5. 1080p fullscreen quad
POM 3.0 : 80fps
SSDM : 160fps

Fig 6. 1080p fullscreen torus
POM 3.0 : 40fps
SSDM : 160fps

Note that SSDM computation time only depends on screen resolution, not scene complexity.

3.2 POM and SSDM object side comparison

Fig 7. Object sides: Parallax Occlusion Mapping (POM) and SSDM

Another advantage of SSDM over POM: it can handle objects sides displacement.

3.3 Compressed and mipmapped formats

Fig 8. Using DXT5 compression and mip-mapping

DXT5 is a RGBA compression format, which cut by two thirds the required memory.
It works well with SSDM, so we can use RGB to store diffuse information, and A to store the displacement. A 4096*4096 texture uses only 21MB.
Mip-mapping and anisotropy can also be used to improve appearance.

3.4 Other results

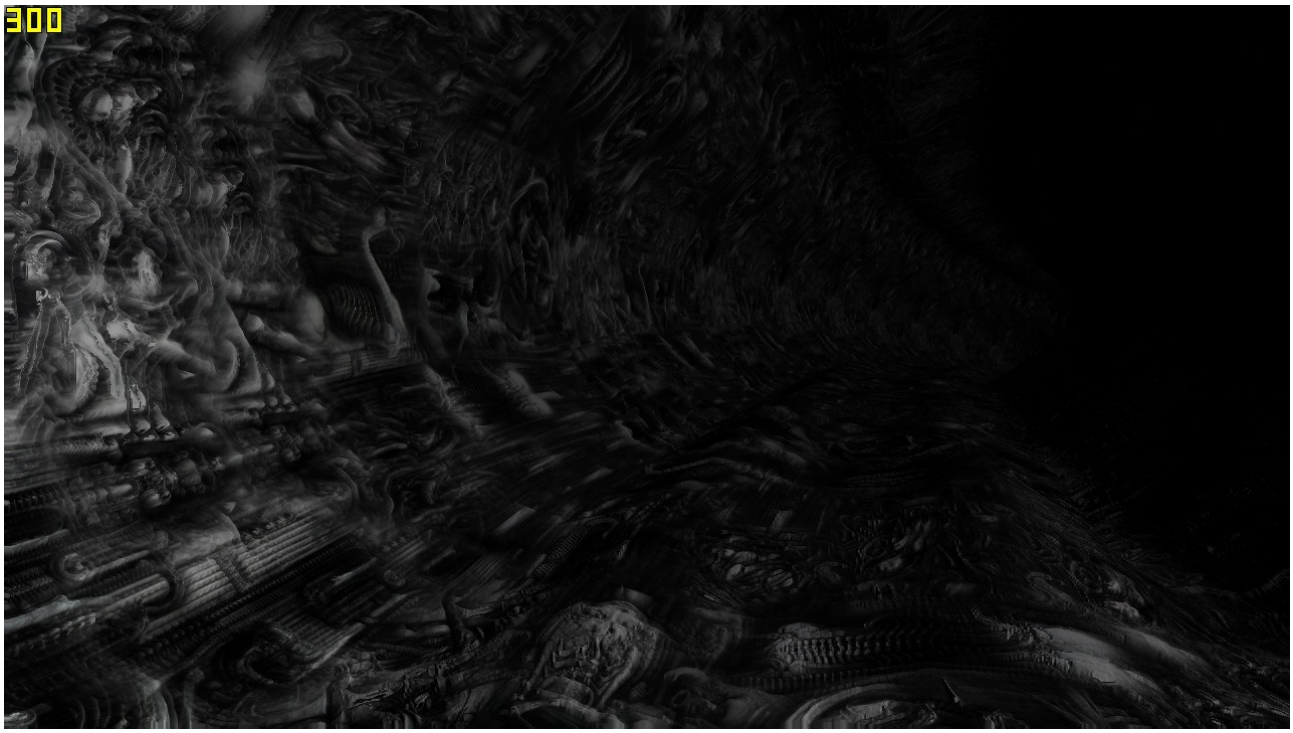

Fig 8. 500 million polygons equivalent scene, 300fps at 720p resolution (1280x720).


Fig 9. case of overlap

Alpha and beta videos are also available online [5].

4. Limitations and future development

Some cases are not well handled: when the displacement has too high frequencies, and some cases of overlapments.
This could be improved respectively by building the pyramid on fewer levels, do a grid displacement using vertex shader on top level, then going down to level zero using the method described in this paper. This would avoid losing smaller details.
For the overlapping problems, the algorithm could be enhanced by doing multiple displacement pass. The method is pretty fast, so there's definitely room for improvement.

References

[1] http://en.wikipedia.org/wiki/Parallax_mapping
[2] http://en.wikipedia.org/wiki/Parallax_occlusion_mapping
[3] http://www.vimeo.com/8850109 , http://www.vimeo.com/8850154 , http://www.vimeo.com/8850225
[4] http://en.wikipedia.org/wiki/Screen_Space_Ambient_Occlusion
[5] http://www.vimeo.com/1900352 , http://www.vimeo.com/3106401