Rendering transparent objects with deferred shading

Christian Magnerfelt <magnerf@kth.se>

DD143X

12th of February 2012

1 Introduction

Deferred Shading[i] is a GPU rendering method that's considered to be the mainstream GPU rendering method today when it comes to real-time graphics in major computer games. Despite being introduced by Saito and Takehashi[ii] in 1990, deferred shading was not made possible until the introduction of programmable shaders and multiple rendering target in 2002 with DirectX9[iii]. The first game ever to have deferred shading was DOOM III released by Carmac[iv] & id software in 2004 and many games that followed used similar or modified version of deferred shading.

Before deferred shading however the often used GPU rendering method was front-rendering. The front-rendering method sends all the geometric and light-data to the shaders and all the light computations is then performed. This often result in waste of shading. If there are many lights in a scene there is bound to be fragments, which light is calculated but the fragment itself is not visible in the final screen. In term of upper bound complexity the front rendering methods have a O(lights * objects) complexity in per-pixel lighting which is pretty bad for high number of lights and objects.

What deferred shading actually does is reducing the complexity upper-bound to O(lights + objects) for per-pixel lighting. The basic idea is that you do a geometry pass which filters out only the visible geometry. This geometry is stored in a G-Buffer which contains for example normals, diffuse & specular material properties and of course the depth for each pixel. By doing this only the visible fragments are stored in the G-Buffer's depth buffer filtered by the z-buffer algorithm. From this G-buffer it's possible to perform per-pixel lighting, in a second pass, in screen-space in a 1 to 1 ration between light and fragment. This lowers the previous upper bound of complexity as mentioned earlier.

Despite being are very effective GPU rendering for scenes with many lights the method comes with a cost. Compared to front rendering the G-buffer has to be stored in the video memory. Lets say that you have a screen resolution of 1280 x 720 pixels. If the G-Buffer size per pixel is about 16 bytes, with normals, diffuse, specular, depth components, the total size would be around 15 mb which for

consoles is a lot since video memory is often the bottleneck on such systems. There exist a modification of the deferred shading method which is called deferred lighting which reduces the size of the G-Buffer. Our document however will only focus on deferred shading.

Deferred Shading is not compatible with anti-aliasing and transparency rendering and special work-around have to be used in order to support anti-aliasing and transparency. Regarding transparency, the reason why it does not work is that the G-Buffer only holds information for one fragment per pixel. This makes it impossible to represent transparent objects as it would imply that there could be multiple fragments per pixel.

The scope of this project is to explore methods used to render transparent objects and examine how these methods could be implemented in a deferred shader. We will look at older methods such as Blending, Alpha-test[v] and front rendering but also newer methods within Order-Independent Transparency[vi] such as depth peeling[vii], stochastic transparency[viii] and adaptive transparency[ix] .

2 Problem statement

How could we combine methods for rendering transparent objects with deferred shading and what do we expect to gain from doing so? Which of these methods suit best for combining with deferred shading?

3 Approach

We'll start by looking at already existing methods commonly used with deferred shading. However it's expected that there is not much research in this field. However within the field of rendering transparent objects there is a lot of research as rendering transparent objects is still difficult even without a deferred shader. So looking into research regarding rendering transparent objects is high priority. To start of methods such as front-rendering, alpha-test and blending should be explored as they are methods that have been around and used for a while so there should be sufficient research is this field. It's expected that current deferred shaders use at least one of these methods.

Later we should also explore newer order-Independent transparency methods such as depth-peeling, stochastic transparency and adaptive-transparency. These are methods that previously was impossible on the current hardware which now is possible.

It should be expected that there is not as much research on these methods due to being published only recently as they can only be used on newer graphics cards which support Shader Model 5.0 which requires D3D11 and OpenGL 4.1[x]

These methods may also not be compatible with deferred shading. If so there should be an explanation why the method does not work and if it's possible work-around the issue. However the focus of this document is to study methods that can be used with deferred shading.

The design for each methods on how to implement with deferred shaders should not be in the greatest detail but at least open up possibilities for further research for specific implementations of transparency rendering with deferred shading.

4 Time plan

Task	Week
Examine already existing methods such as front-rendering, blending and alpha-test	7
Examine Order-Independent transparency methods such as depth peeling	8
Examine never Order-Independent transparency methods such as stochastic and adaptive-transparency (SM5.0 , D3D11)	8,9
Validate which methods could be implemented in a deferred shader and if so construct a design for how these methods could be implemented in a deferred shader	9,10,11
Summarize and finish report	12,13,14

5 References

i Thomas Akenine-Möller, Eric Haines, Naty Hoffman “Deferred Shading” Real-Time Rendering Third Edition 2008 p. 279 - 283

ii Saito, T. and Takahashi, T. Comprehensible rendering of 3-D shapes. In: Proceedings of the 17 th annual conference on computer graphics and interactive techniques (SIGGRAPH 90). Vol. 24, Issue 4. New York: The ACM Press, 1990. p. 197-206.

iii DirectX Development Center <http://msdn.microsoft.com/en-us/directx>

iv Carmack Plan File for Doom III <http://www.pvcmuseum.com/gpu/john-carmack-plan-file.htm> 2003.01.29

v Thomas Akenine-Möller, Eric Haines, Naty Hoffman “Transparency, Alpha and Compositing” Real-Time Rendering Third Edition 2008 p. 134 - 141

vi Jason Yang, Jay McKee "Real-time order independent transparancy and indirect illumination using Direct3D 11" SIGGGRAPH 2010 <http://advances.realtimerendering.com/s2010/Yang,%20McKee%20-%20OIT%20and%20Indirect%20Shadows(SIGGRAPH%202010%20Advanced%20RealTime%20Rendering%20Course).pdf>

vii Louis Bavoil, Kevin Myers "Order Independet transparency with Dual Depth Peeling" Nvidia February 2008 <http://developer.download.nvidia.com/SDK/10/opengl/src/dual_depth_peeling/doc/DualDepthPeeling.pdf>

viii Eric Enderton, Erik Sintorn, Peter Shirley, David Luebke "Stochastic Transparency" ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games February 2010 <http://www.nvidia.com/docs/IO/88888/StochasticTransparency_I3D2010.pdf>

ix Marco Salvi "Road to real-time order independent transparancy" SIGGRAPH 2011 <http://bps11.idav.ucdavis.edu/talks/10-roadToRealTimeOIT-Salvi-BPS2011.pdf>

x"OpenGL wiki" OpenGL 2012.02.12 <http://www.opengl.org/wiki/Main_Page>