ReRendering

Mikkel Adamsen <adamsen@diku.dk>

24. september 2007

Indhold

1 Introduction

Figur 1: A small scene of a mill, with some lightsources. The model is created by Ghost A/S

One of the main bottlenecks of doing computer animated films is the lighting design. Using the traditional shading method, which we will refer to as forward rendering, every time you manipulate the lights, it can take hours to see the result. Especially in large scenes that contains tens of millions of polygons. To solve this problem We will use a technique called deferred shading. Deferred shading uses a deep-framebuffer, to cache intermediate data. In a later pass we can illuminate our scene, by calculating the light of each pixel in the deep-framebuffer, using the data in the deep-framebuffer. To test out the method we have create a small lighting editor called Cinematic, that can create results as can be seen in figure 1.

2 Previous Work

Deferred shading was first introduced in 1988 by [2]. They proposed a new way of doing graphics hardware. Their idea was to have a pipline that produce pixels with a z-depth a normal and a color. The pixels then goes in to another pipeline that for every light source shade the pixel and adds the result together. Even though he doesn't mention it by name this is the first mention of the idea of deferred shading.

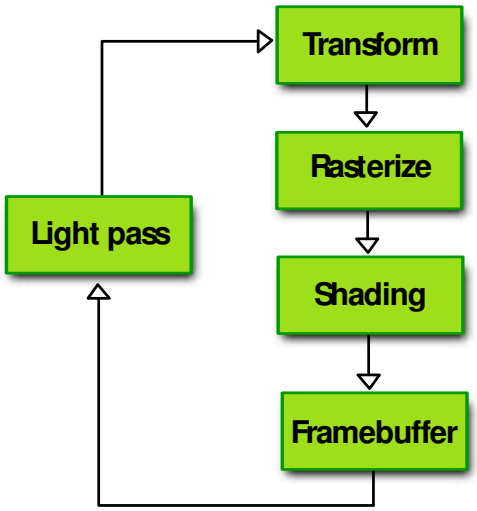
In [10] they want to enhance the recognition of shapes and patterns in 3D rendererd images. In order to optimize the enhancement process they use geometric buffers (G-buffers). Each G-buffer contains a geometric property such as world coordinate x or normal vector y. By using G-buffers they seperate geometric processes (projection and hidden surfaces) from shading which is performed as a postprocess. They can then tryout different types of image enhancements, without processing the geometry again.

In [5] they propose pixelflow a high speed image generation architecture. They mention deferred shading as a way to avoid transformation and frame-buffer-access bottlenecks. Deferred shading is also mentioned in [4] where they discuss ways to do real-time programmable shading. They also mention deferred shading as an optimization.

The first step in using deferred shading for cinematic lighting design is in [3]. They create a deep-framebuffer inspired by G-buffers to cache intermediate data. Unfortunately the material system was not flexible enough to make it useful for artist and studios.

In 2004 with the advances of real time graphics hardware and with the advent of programmable shaders on graphics hardware [8] creates a relighting system using programmable shaders that archives interactive rates. Indepentdently of [8] pixar creates a similar system called Lpics [7]. A the time of writing Lpics had already been used for two computer-animated feature films.

Finally in 2007 [9] improves on the earlier work done in [2004]. They enhance the relighting techinque with a way to do antialiasing, motion blur and transparency using a indirect framebuffer that decouples shading samples from final pixels.



Figur 2: The pipeline for forward rendering.

3 Background

To find out why we need a rerender engine, we must first look at how lighting is done on graphics hardware with normal forward rendering. Then we will look at how lighting is normally done in a production pipeline of a computer-animated films to find out what is expected of a lighting system.

3.1 Forward Rendering

A typical 3d scene contains a viewer, a set of meshes and set of lightsources. Usually a mesh is a collection of triangles. A mesh could also be made out of quads or polygons, or even determined by af beziér surface, but it is converted to triangles before it is rendered. The usual way to render such a scene is done with forward rendering. We go through the pipeline as can be seen in figure 2 and look at a couple of the stages.

3.1.1 Transformation and projection

Transformations are used to move an object, change the size or rotate it. An object is transformed by transforming each vertex of the mesh. After a local tranformation of the object it is transformed into view space. In view space each vertex is projected on to the view-plane. The process of projection is to transform a point from one coordinate system of n dimension into a coordinatesyste of less than n dimensions. On graphics hardware this process is controled in the vertex shader.

3.1.2 Rasterization

For each triangle of the mesh, we have three corresponding points projected on the view plane. Rasterization is the process of filling out the triangle defined by the three points, with fragments. Each fragment we rasterize correspondce to a pixel in the framebuffer. When using graphics hardware, this process is taken care of by the hardware and we can't control it.

3.1.3 Shading and lighting

Each pixel must now be shaded. Shading is the process of determine a color. This can be done in a simple way by just assign it a color, or using a physical based model such as torrance-sparrow. Most shaders works with the combination of a lightsource and shades the surface deeping on the angle and distant to the light source.

3.2 Production pipeline

The lighting system should fit in to the production pipeline of a computer-animated film. We therefore need to take a closer look at the production pipeline. The production pipeline consist of five stages modeling, texturing animation shading and lighting.

3.2.1 Modeling

All scenes, characters, props and everything else that can be seen in an animated film, must be create by a modeller. The modeler creates objects by displacing vertices. Models can also be created by using beziér surfaces. To get an interactive feedback, the moddeler can use the graphics hardware to render a low tesselated version while creating the object. Then in the final rendering a higher tesselation can be used.

3.2.2 Texturing

Texturing is the process of applying an image to a surface. By using textures we can add more detail to the models.

3.2.3 Animation

An important aspect of an animated film is the animation. The animators can animate the models by moving the objects, or parts of it and setting keyframes for each movement. It is important that the animator can get a realtime playback of his animation to finess the small subtleties that goes into making the motion believable. This can be archived by using the graphics hardware. If the models are to complex, low tesselated versions can be used.

3.2.4 Shading

Shaders are written by programmers. The shaders are often much more complex than the shaders that are used with graphics hardware.

3.2.5 Lighting

Lighting is an extremely important aspect of movie making as it adds to the depth and richness of the visual experience. The lighting artist creates the final look of each scene by moving and positioning light sources throughout the scene. The lights are fine tuned by setting variables such as color, intensity and attenuation. Often, hundreds of lights are used to simulate the light bouncing of surfaces to create natural looking scenes.

3.3 The Problem with lighting design

on a live-action film set the lights are moved and positioned by grips and the result can be seen immediatly by the cinematographer. When doing lighting in computer-animated films, the scene must be rendered again each time the lighting artist moves a light. Having to wait tens of minutes or mulitple hours, this is a major bottelneck in the production pipeline. Even small changes like changing the color takes a really long time and limits the productivity of the lighting artist. Since lighting is in the final stage of the production pipeline this is often done under extreme time pressure and can hurt the qualty of the final shot.

4 Rerendering

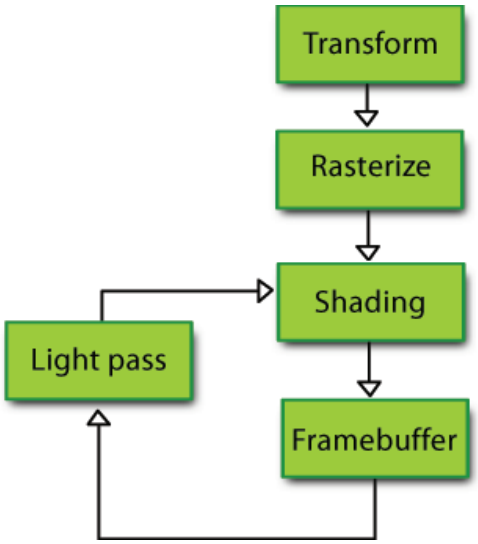
What we learned from looking at the production pipeline is that we could see that all the stages could be done at interactive speeds with the help of graphics hardware, except the shading and lighting stage. This creates a bottelneck in the production pipeline. We will look at a way to solve this problem. First we wil look at how we can have great lighting design at interactive speeds. Then we will look at how shading could be handled.

4.1 Lighting Design

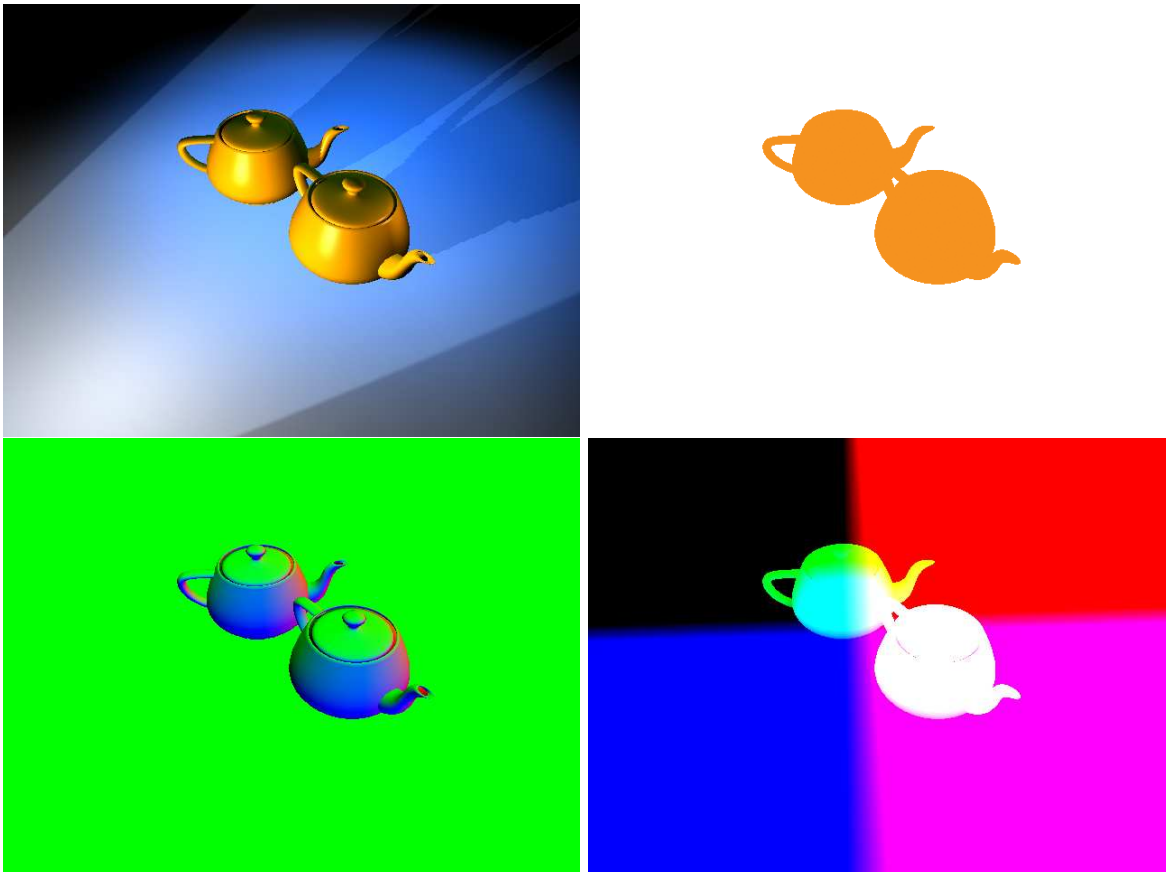
A system must be created to allow for interactive lighting design. It should have the ability to add, remove and positioning light sources. It should allow the lighting artist to adjust light parameters such as the color, intensity and shape of the light, while getting the result instantaniously.

In [3] they discovered that since the lighting was done in the final stage, all characters had been transformed and the camera was sat in a fixed position. Each time a light was adjustet and the shot was rendered, all the excact same transformation and projection calculations were recalculated needlessly. As a solution to this problem they suggest to cashed the intermediate data used for calculating shading and lighting in a deep-framebuffer, inspired by the G-buffer [10]. They used graphics hardware to accelerate the process, but at the time, there was no programmable shaders and the system was too inflexible to create the advanced shaders used in computer animated films.

In [7] Pixar created Lpics that is based on [3] but uses programable shaders. The basic idea is to cache the intermediate data used for the lighting pass in a deep-framebuffer, then for each light we have a light pass where we lit the object. This gives us a new pipeline that can be seen in figure 3



Figur 3: The pipeline for deferred shading.



Figur 4: Top-left: The final image. Bottom-left: The normals. Top-right: The Diffuse color. Bottom-right: The position in world space.

4.1.1 Transform and rasterize

This stage is done exactly the same way as it was done in Forward rendering.

4.1.2 Deep-framebuffer

The idea of the deep-framebuffer is that we do not want to recalculate the data that we need for the lighting pass. We do this by first rendering the scene as normally, but instead of rendering out to the screen we render to multiple rendertargets. The data that we need to cached, is the normal and the position of the point. We can also save material properties such as the diffuse color, the specuclar contribution or the specular roughness. An example of the different layers in the deep-framebuffer can be seen in figure 4.

The graphics hardware limits how many parameters we can save in the deep-framebuffer. If we run out of space there is some common techniques to help compress the number of data. If we Store pixel normals in view space, we only need to save 2 values instead of 3. In view space the z component will always be positive and we know every normal is a unit vector, which means that we can get the z component as $z = \sqrt{1 - x^2 - y^2}$.

4.1.3 Light pass

For each light we have a light pass. We render a quad that has the same screen size as the deep-framebuffer. We give the deep-framebuffer to the pixelshader as a texture. For each pixel we then do a lookup in the deepframebuffer to the variables, such as the position and normal, that we need. We then use these to do the lighting as we would do in the forward rendering.

4.2 Shading

Shaders are used to get objects to look more detailed. A shader can be anything from a simple texture to an advanced subsurface scattering skin shader. In computer animated films, renderes such as Renderman or mental ray are used to create the final image. These renderes both have their own shader system, where you create a shader program to determine the surface color of the object. These renderes however are very slow and can not be used for doing lighting desing at interactive speeds.

This is where the rerendering can help, but to get any useful result, the image created in the rerenderer, must be as close as possible to the image created in the final renderer. This means that we have use the same shaders to get a result that is as close as possible. This is a problem because we can not use Renderman shading language or mental ray shading language on graphics hardware. We must instead somehow convert it to for example a Cg shader. In [8] they descripe a method for automatically transform a renderman shader to a Cg shader. In [7] they write that, to get the optimal result the shaders must be converted manually, to optimize the shader as best as possible. Cinematic is only meant for simple lighting design testing, and we have decided only to support a single default shader for all objects.

Another problem with shading is that we can't tell objects in the deep frame buffer apart. In [8] they write a material ID into each pixel of the deep frame buffer. In the light pass they then have a lightshader for each material and each shader only shades pixel that has its material ID. This is a potentiel problem for the performance if they have many materials, since they have to do a pass for each material.

5 Lights

For the lighting we used the light shader described in [6] which is based on the über-shader described in [1]. The über-shader gives us a lot of tools to manipulate a lightsource.

5.1 Selection

A problem in doing lighting design in the real world is that sometimes we want different light on different objects. This is a hard problem, because light is hard to control. In the virtual world it is alot easier. If we used forward rendering it would be, to simply have a list of objects that should be lit for each light source. Then for each light pass we would only render the objects in that list. But with deferred shading it is a bit more difficult.

The problem is that we tell the objects apart, once they have been rendered into the deep-framebuffer. We could try to solve it the same way as we did with shading. By giving every object an ID and render the ID into the deep-framebuffer. If we use an 32 bit buffer to store the ID's, we should have sufficient ID's for even very complex scenes with millions of objects. But the problem is in the light pass, the light source should have a list of obects that should be lit by that light source. This list should then be sent to the GPU as a list of uniforms but there are no current hardware that can handle such a large list of uniforms.

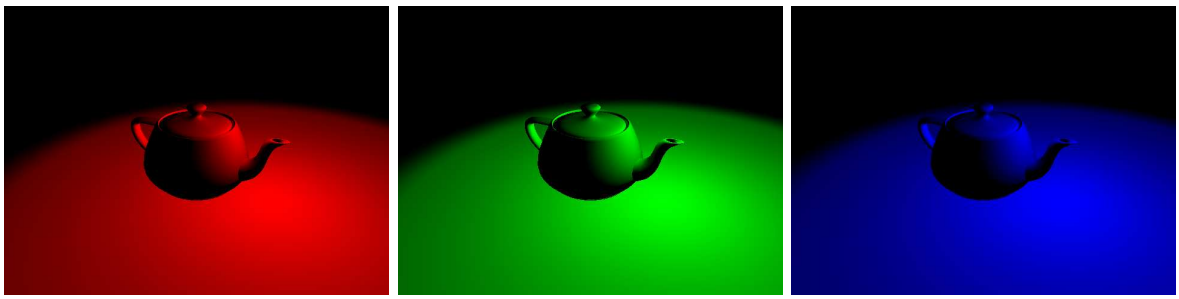
Instead we can create a new 2 bit buffer for each light source. We will call this buffer a light mask. For each light, each object that should be lit by that light source should be renderered from the perspective of the camera into the buffer, with a value of 1. Then in the light pass we only lit a pixel if it has a value of 1 in the lightmask. This lightmask does not have to be recomputed every frame but only when a object is included or excluded from the list of objects that are lit by the light source. If an object is included only the new object is added. If an object is excluded the entire mask would have to be recreated, but since we only exclude lights once in a while it will be acceptable.

5.2 Color

Color is the most noticeable property of the light. The human brain can perceive light waves with wavelength in the visual spectrum. Each wavelength correspond to a color. We tend to give meaning to color of light. For example blue colors are considered cold and orange colors are perceived as varm. It is therefore important to have control of the color, to set the right mood for a scene. To give further control we have weights for ambient, diffuse and specular illumnation, similar to the OpenGL fixed function pipeline. An example can be seen in figure 5.

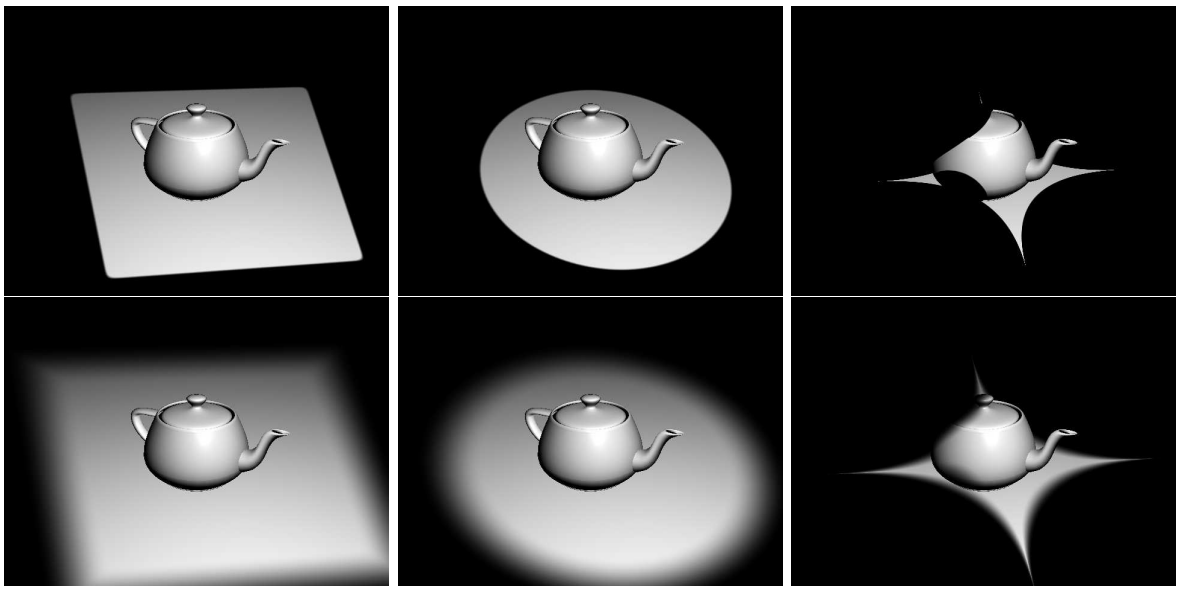
5.3 shape

In live-action films cinematographers use spotlights with barndoors to shape the light. In computer animated films we want to give the lighting artist the same abilites to control the area lit by the light. In Cinematic we have two types of lights: Omni lights and spot lights. The omni lights can be controlled by setting an attenuation based on the distant from the light source to the point that shold be lit. For spot lights we use superellipses to simulate the



Figur 5: The teapot scene with 3 different colored lights.

effect of barndoors. A super ellipse can be varied in to the shape of a square, circle and a star-like shape, by only changing a single variable, this makes it very easy to control. For both the attenuation and superellipse a fallof area can be controlled. This gives the light at softer look. In figure 6 we can se three different settings, with and without a soft fallout.



Figur 6: Three different shape settings, with and without a soft fallout.

5.4 Shadows

We can use standard methods for calculating shadows. Some of the most popular methods for shadows are stencil volume shadows, shadow maps and raytraced shadows. Stencil volume shadows only works with two manifold objects and raytracing is to slow for realtime. Shadowmaps has problems with bias, and they can give very pixilated shadows, if the shadow area are to large compared to the shadow map.

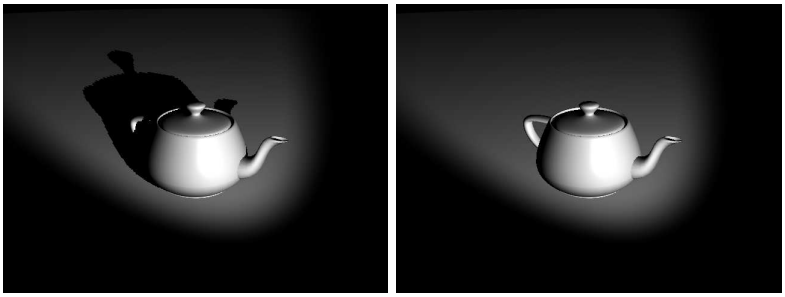
But we have choosen shadowmaps to represent shadows in Cinematic, because they work with all types of meshes and are very fast. When we create the shadowmap we have to render the scene from the perspective of the light source. Which means that every time we move the

light source we have to remake the shadowmap. This is as big problem with deferred shading since the whole idea is that the scene is to complex to render in real time and now we have to anyway if we want shadows.

In [7] they solve this problem by simplifying the meshes when rendering the shadowmap. This will not create correct shadows, but by softening the shadow in the edges there wont be that big of visually difference. In cinematic, we do not support simplifying meshes and therefore have a limit to how complex a scene can be if shadows are wanted.

5.4.1 Shadow selection

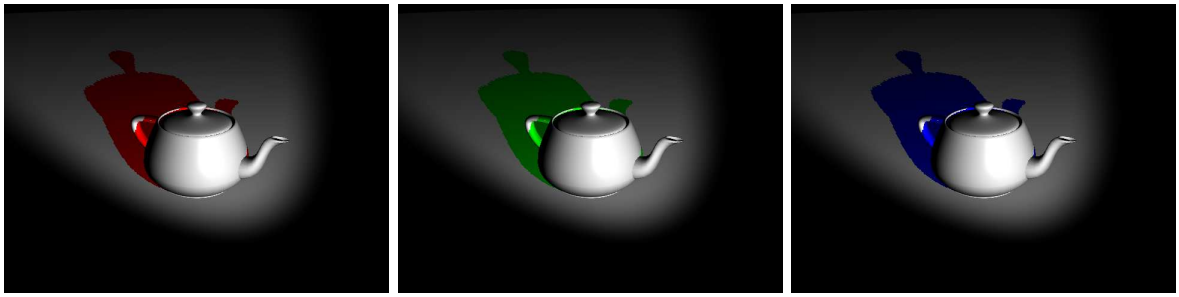
When doing a lighting setup, the stardard way to do it is a 3 point light setup. The 3 lights consist of a key light, a fill light and back light. Usually you want your key light and your backlight to cast shadows but not your fill light. We therefore give the light artist the ability to turn of shadows as can be seen on figure 7.



Figur 7: The teapot scene with and without shadow.

5.4.2 Hue

When looking at snow on a clear day, the snow seems to have blueish colored shadows. This is because the snow is only lit by the blue sky in the shadow area. To mimic the effect the lighting artist has been given the ability the change the hue of the shadow. The slight variation of the shadow hue can make the difference between a great looking shadow and a fake loooking one. An example can be seen in figure 8



Figur 8: Three different colored shadows.

5.4.3 Darkness

With the direct lighting model we get very very dark shadows. This happens because we dont get the indirect illumination that we see in the real world. To mimic this effect [6] suggest to have the ability to adjust the density of the shadow. The use the diffuse contribution of the surface to color the shadow region. This method let them maintain the gradients that make the shadow beliveable. They also switch off the specular term to avoid highlights in the shadow area.

6 Light Caching

When we deal with direct lighting the lights are linear independent. This means that to get the light contribution from multiple light sources, all we have to do is to compute the light contribution from each individual light source and then add them together. This makes it very simple to represent different types of lightsources. In Cinematic we currently have a point light and a spot light.

In each lightpass we find out which light type it is, and use the shader corresponding to that light source. To save the final light contribution we can use a ping-pong method, where we use two rendertargets, where we read from one and write to the other and in each pass we switch the two. An easier way to do it though is to use blending with the source and destination sat to ONE. This will the add the new pixel value on to the value in the framebuffer.

Since we can add lights, by simply adding the final pixels we should also be able to remove a light source by substracting the light source from the framebuffer. If we can remove and add lightsources to the framebuffer we only need to render the pass of the light source we are manipulating, by first substracting the ligth contribution from the light cache, apply the modification to the ligth source and then add the new light contribution.

But we have a problem with using the framebuffer to cache the light, since it is only 8 bit pr color channel. Which means that if we add two white light sources it will overflow and be clamped. If we then substract one of them, we will have no light left. If we instead use a rendertarget to cache our combined light, we can have 32 bit float value pr color channel. With this we should be able to cache many lights.

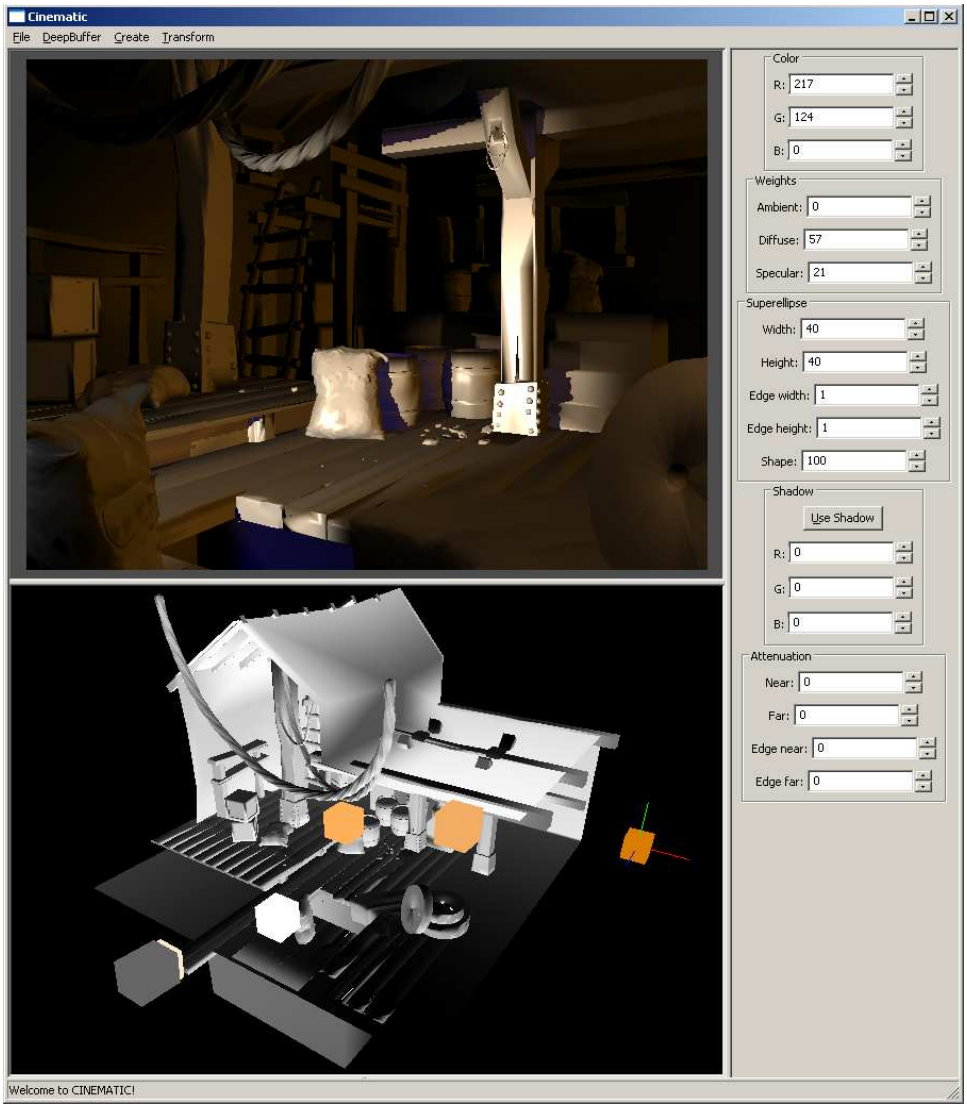
7 Transparency

Transparancy is a big problem in rerendering. The problem is that we can only represent one point in the world pr. pixel in the deep-framebuffer. With transparent objects we often have more than one point in the world that project into the same pixel in the deep-framebuffer.

In games that use deferred shading, they have the same problem with transparent objects occluding opaque or other transparent objects. To solve the problem they first render all the opaque objects with deferred shading and then render all transparent objects with normal forward rendering.

In a rerender engine we can't do it this way, if the transparent objects are very complex. Instead wo could render the scene in to multiple layers. To do this we could use a raytracer, where for each ray we find the intersections of the ray with the scene. Then we sort them based on the distant from the view point. Then we insert the points into each layer starting with the opaque point furthest away from the view point. Then in each lighting pass we simply light each layer and blend them together. The only drawback is that in scenes with many transparent objects we will have many layers and we could loose the realtime performance.

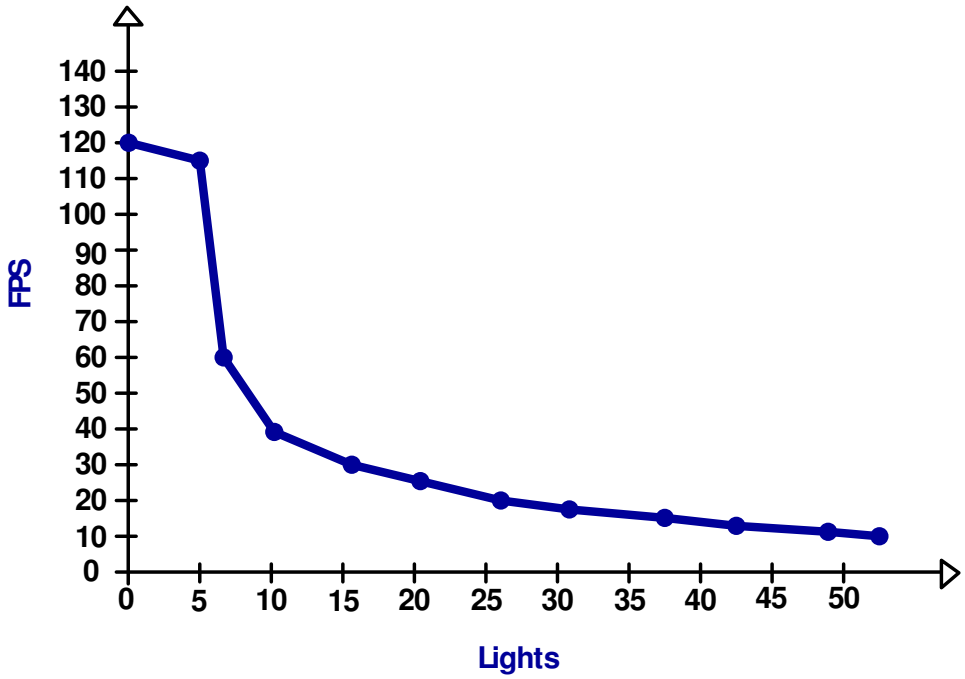
8 Results



Figur 9: A screenshot of Cinematic

To test a the principles of deferred shading we have created a small program called Cinematic. Cinematic is a simple program to test out light designs. Cinematic lets the use add, remove, position and rotate the light sources. In figure 9 we have a small screenshot of what the interface looks like. To the right in the image there is a menu that lets the user change the parameters of the light source. In the bottom there is a realtime view of the scene, that the user can fly around in to get a better view of the scene. This was included because it can be hard to position the lights if we can only see the scene from one angle. In the top view we see the final image, with all the light contributions.

On the graph in figure 10 we can see how Cinematic's framerate scale with the amount of light sources. We can see that we can't really have that many light. At 50 lights it drops



Figur 10: How the framerate scale with the amount of lights.

below interactive speed. This is manily because light caching has not been implemented, and we belive this would help the framerate considerbly.

9 The CD

On the CD-rom I have included all the source code used for Cinematic. There is also some videos of the program in function.

Litteratur

[1] Ronen Barzel. Lighting controls for computer cinematography. *J. Graph. Tools*, 2(1):1–20, 1997.

[2] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 21–30, New York, NY, USA, 1988. ACM Press.

[3] Reid Gershbein and Pat Hanrahan. A fast relighting engine for interactive cinematic lighting design. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 353–358, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[4] Anselmo Lastra, Steven Molnar, Marc Olano, and Yulan Wang. Real-time programmable shading. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 59–ff., New York, NY, USA, 1995. ACM Press.

[5] Steven Molnar, John Eyles, and John Poulton. Pixelflow: high-speed rendering using image composition. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 231–240, New York, NY, USA, 1992. ACM Press.

[6] Fabio Pellacini and Kiril Vidimce. Cinematic lighting. In *GPU Gems*. Addison Wesley, 2004.

[7] Fabio Pellacini, Kiril Vidimče, Aaron Lefohn, Alex Mohr, Mark Leone, and John Warren. Lpics: a hybrid hardware-accelerated relighting engine for computer cinematography. *ACM Trans. Graph.*, 24(3):464–470, 2005.

[8] Jonathan Ragan-Kelley. Practical interactive lighting design for renderman scenes. 2004.

[9] Jonathan Ragan-Kelley, Charlie Kilpatrick, Brian W. Smith, Doug Epps, Paul Green, Christophe Hery, and Frédo Durand. The lightspeed automatic interactive lighting preview system. *ACM Trans. Graph.*, 26(3):25, 2007.

[10] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 197–206, New York, NY, USA, 1990. ACM Press.