Frosty and the flamethrowers in EDAN35 High Performance Computer Graphics

John David Olovsson* Einar Holst†

Lund University
Sweden

Abstract

1 Introduction

This project takes the form of a very simple 3D computer game with a tower defence theme. The graphics algorithms have been the main focus of the project since the start. The project was chosen so that it would be possible to demonstrate these algorithms while at the same time providing an entertaining setting.

The project specifically focuses on two graphics algorithms that were of much interest. These were *parallax occlusion mapping* and the *fire effect* which was based on the effect used in the nVidia Vulcan demo[Hubert 2007].

Parallax occlusion mapping was chosen in order to investigate a more advanced replacement for simpler methods such as *bump mapping* which do not give a good enough result for highly displaced surfaces.

Creating a realistic fire effect in real time is a very interesting subject for which there does not seem to exist many good publications. The nVidia Vulcan demo is one of the available sources and also happens to give a reasonably realistic result with a lower performance impact then expected.

2 Algorithms

2.1 Parallax occlusion mapping

Parallax occlusion mapping is a graphics algorithm that uses very simple ray tracing within a height map texture in order to generate a correct parallax displacement effect as well as occlusion upon the geometry surface. The algorithm also provides for seamless integration with self shadowing by using the same simple ray tracing method. This allows for rendering very complex scenes in real time without needing an excessive amount of actual geometry detail.

The parallax occlusion mapping in this project was also combined with bump mapping and phong shading.

Implementation The starting point for this algorithm is a standard phong shader with bump map support. The first additional requirements of the parallax occlusion mapping is the height map texture which specifies the displacement *into* the surface. It is important to note that most of the calculations in the algorithm should be done in the tangent space of the surface. Especially the *view vector* and *light vector* needs to be transformed into that space.

With the help of the view vector it is possible to calculate how far along the surface the view vector would have to travel from intersecting the actual surface to where it intersects the bottom of the height map. This distance depends on the angle between the view vector and the surface.

With the knowledge of this maximum offset along the surface it is possible to choose a set of evenly spaced sample points along this offset vector. These sample points will be the points on the height map where the view vector is checked for intersection against the height map. The first intersection will be used as the actual position when looking up the color from the diffuse texture as well as the normal from the bump map.

Since the maximum offset will be greater the closer the view vector is to being parallel with the surface it would not be optimal to use the same amount of sample points every time. Instead, the amount of sample points will vary with the angle of the view vector so that there are few sample points when looking straight down into the surface and many sample points while looking far along the surface.

After determining the point of intersection of the view vector it is possible to do another trace, but this time towards the light source. This makes it possible to determine if the point is in shadow or in light.

Simply deciding if the fragment is in shadow or in light makes enables the use of *hard shadows*. In order to use *soft shadows* with this technique it is neccessary to determine how much the *light ray* is below the height map. This is done by taking samples along the light vector in the same manner as for the view vector, but this time determining how close the samples are to the height map. The closer they pass the height map the darker the fragment.

2.2 Fire effect

The fire in the nVidia Vulcan demo is based on using relatively large *particle* quads which are rendered with a fire animation of 256×256 pixels with 64 frames per animation. Three distinct animations were used[1]. There were two fire animations and one smoke animation.

The animation was provided in the form of a $256 \times 256 \times 256$ 3D volume texture where the animation would progress along the Z-direction.

The emitter for the fire particles was configurable for different sizes, density, particle life time and so forth. The particles were also influenced by random variations to these and other parameters. Additionally the particles were always rendered facing the camera.

Implementation The 3D volume texture used in the nVidia Vulcan demo is distributed in a format which is not supported by the `RenderChimp` framework. In fact, it seems to be a very obscure format which it generally not supported by anything. Fortunately someone who worked with a similar project has managed to convert it into a Direct3D volume texture available in a `.dds` format. Unfortunately there are not many utilities that support volume textures in this format either. As a result of this it was required to implement a Direct3D tool which would load the texture, render each *slice* onto a Direct3D surface and finally saving these surfaces to a `PNG` file. Given the resulting 256 frames of the animation as separate `PNG` files it was trivial to combine

*e-mail: dt07jo1@student.lth.se
†dt07eh2@student.lth.se

[1] A fourth animation was included in the Vulcan demo, but used for other purposes.

these frames into one combined texture using the `ImageMagick` *montage* command line tool using the following UNIX command:

```
montage flames%d.png[0-255] -geometry
'256x256+0+0' -background transparent -tile '64'
flames.png
```

The resulting image file had a resolution of 16384×1024 pixels and occupied approximately 50 MB of disk space in uncompressed form. Unfortunately neither `RenderChimp`, `SDL` nor `OpenGL` could support images of this size. In order to load the image it first had to be resized in such a way that each frame in the animation had a resolution of 64×64 pixels. This could then be loaded and as mentioned in [Hubert 2007] it still yields a good looking result.

To create a good fire effect it is imperative to have access to some form of particle system. This is done by using an `Emitter` which can handle a static amount of particles through a `USAGE_STREAM` vertex array. Particles in the emitter have all their vertices collapsed into their *position* in order to simplify keeping them aligned towards the camera. In the vertex shader these vertices are translated outwards based on their texture coordinates and the rotation of the particle.

The `Emitter` class is then wrapped into a `FireEmitter` class which provides the particles with the behavior of fire particles and deals with spawning and moving particles, randomizing their appearance and turning them into smoke. This class also sorts all of the particles prior to rendering them so that the *alpha blending* can work properly.

In order to provide maximum flexibility for the fire shader it accepts inputs in the form of the `size` of the fire particle, the `rotation` of the particle, the current normalized `time` of the animation, a specification of which `animation` to display.

For the particle to behave as expected it is neccessary for them to operate in *world space* rather then model space. This allows the particles to act independently of the transforms that are applied to the emitter.

The fire particles emitted from the flame thrower are spawned using a small size and then gradually grow as they move away from the emitter. Additionally they have a randomized maximum size, life time, animation speed, rotation and fire animation type. This gives the impression of a firece and volent fire which is appropriate for a flame thrower.

3 Results

The parallax occlusion mapping performs a varying amount of sampling steps in he height map depending on the angle of the view vector. When looking straight into the surface then it only needs very few samples[2] and will run reasonably fast.

This project uses parallax occlusion mapping on the entire game world terrain. The performance penalty of this is however somewhat managed since everything in the world except the road has the maximum height, which means that no matter what the view vector is the algorithm will always find the intersection in the first sample.

Generally however this algorithm is much more expensive then regular bump mapping in that it uses significantly more texture lookups and it also uses dynamic flow control. This is indicated by the frame rates. When using plain bump mapping the frame rate is 30 frames per second on the test computer while parallax occlusion mapping brings it down to 10 frames per second even when viewing the surface directly from above.

This particular scene does not have a big enough parallax occlusion mapped surface to cover the entire screen when viewed from a small angle to yield a good benchmark.

[2]The implementation used in this project it uses 2 samples when looking straight into the surface.

Combining this algorithm with regular bump mapping and phong shading is trivial and gives a very good looking result. Self shadowing is also relatively straight forward as long as fairly crude approximations are allowed. In either case the end result looks good.

The flame throwers in the projects use a particle system of up to 100 quad particles each. In the current version the particle engine is somewhat suboptimal and uses six vertices for each quad particle which yields a total of 600 vertices for each particle emitter. These are all updated once during every physics update and all of the particles in each system are sorted after their distance to the viewer for each rendered frame. With ten separate flame thrower towers, all of which have this exact specification, this sums up to 1000 particles and 6000 vertices.

In the full scene, the flame throwers reduce the frame rate from ten frames per second to five frames per second on the testing computer. This is measured when the map is using parallax occlusion mapping. If regular bump mapping is used instead then the flame throwers reduce the framerate from 30 frames per second down to seven frames per second.

4 Discussion

The parallax occlusion mapping gives a very good looking result which can compete with even the most detailed geometry, but at the expense of more texture lookups. This project has not attempted to represent the same surface as detailed geometry. Thus no comparison can be made to this alternative.

It is however very useful to be able to represent detailed surfaces as very simple planes. The entire game world in the project is just one big flat plane.

Also the self shadowing works well, at least for surfaces with somewhat low complexity. The results definitely look believable.

It did however turn out to be all but trivial to implement the shader for parallax occlusion mapping so that it updates the depth buffer correctly. That problem is not solved in this project. It is however not a critical problem in this project since it is just used for a relatively flat ground plane.

The amount of sample points during the intersection testing in the parallax occlusion mapping has been carefully tweaked to give the best visual result at the same time as providing acceptable performance. A much lower sampling rate would result in highly noticable visual artifacts while a much higher rate would result in too low overall performance. This does however depend on the resolution of the height map, the depth of the surface and generally what the height map looks like.

Much effort has been invested in tweaking the various parameters of the fire effect. Especially the animation speed, the particle creation rate and the particle size have required special attention.

It was not obvious that simply keeping the particles facing the camera would give a natural looking result, but it did.

Unfortunately it was not easy to use the fire animation from the nVidia Vulcan demo directly because of the unsupported file format. Many hours of work had to be wasted on extracting these into a useable format which could have been spent on more important work.

The original animation frames had a resolution of 256×256 pixels. It was not obvious that it would look convincing when this was downsized to 64×64 pixels, but on the scale that fires are used in this project the result was good enough.

5 Conclusion

Unfortunately there was not enough time in the project to implement a fully functional game. Thus the project was limited to what is most accurately described as a *tech demo*. There also was not

enough time to implement the additional algorithms that were considered for the project[3].

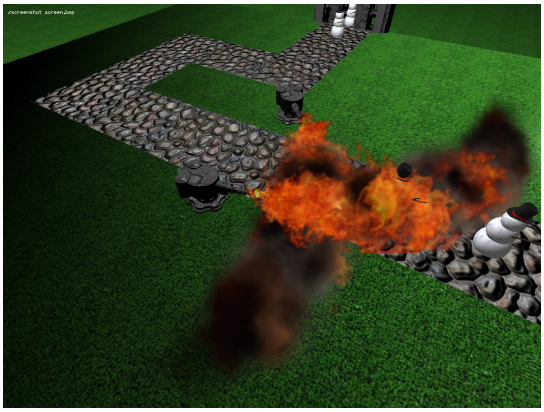

Figure 1: Close up screen shot of the fire effect.

Figure 2: Distant screen shot of the fire effect.

References

HUBERT, N. 2007. Chapter 6. Fire in the Vulcan Demo. In *GPU Gems*.

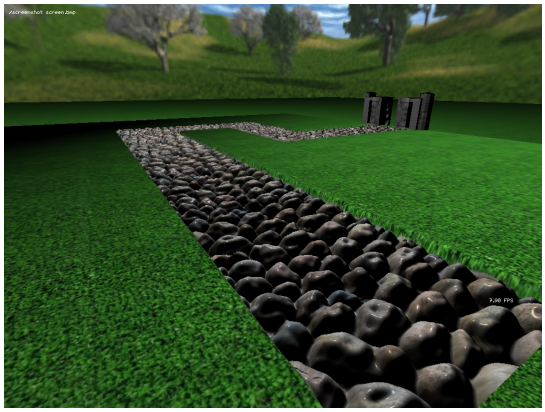

Figure 3: Parallax occlusion mapping applied to a cobbled stone road.

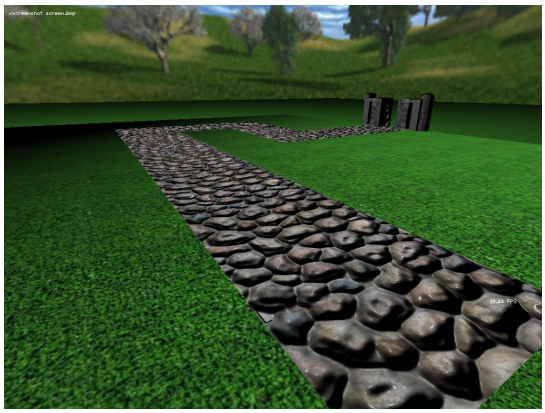

Figure 4: Standard bump mapping applied to a cobbled stone road.

[3] Initially the project was intended to include a grass animation algorithm and glow as well.