UNIVERSITY OF TEESSIDE

Real Time Graphics

Paul Demeulenaere – G7105735

Paul Demeulenaere
06/05/2008

Chapter 1 - Scenario

The subject of this assignment was:

"All action takes place within a theme park ride. This ride is themed around a rocket launch into outer space. You are placed within an enclosed room (the launch zone) through which the ride's train (the rocket) will pass. The train will approach the room and come to a complete stop. A pre-launch sequence will begin during which the lighting will build the suspense and then … whoosh the rocket launches at high speed. The action should capture the ride's detail and the thrill of the launch."

I have not really respected this aspect of this project; my application is more a demonstration of a set of shader than a scripted scene. I would like to thank Martin Johnson for his geometry. I have animated this scene with a displacement of the craft and one of the two lights follows a circle. These animations are stored in a XML file; it is possible to change these transformations without recompile. I have chosen the API of DirectX 9 because it was a leak in my computer graphics skills. The shader language used is the C for Graphics from NVidia.

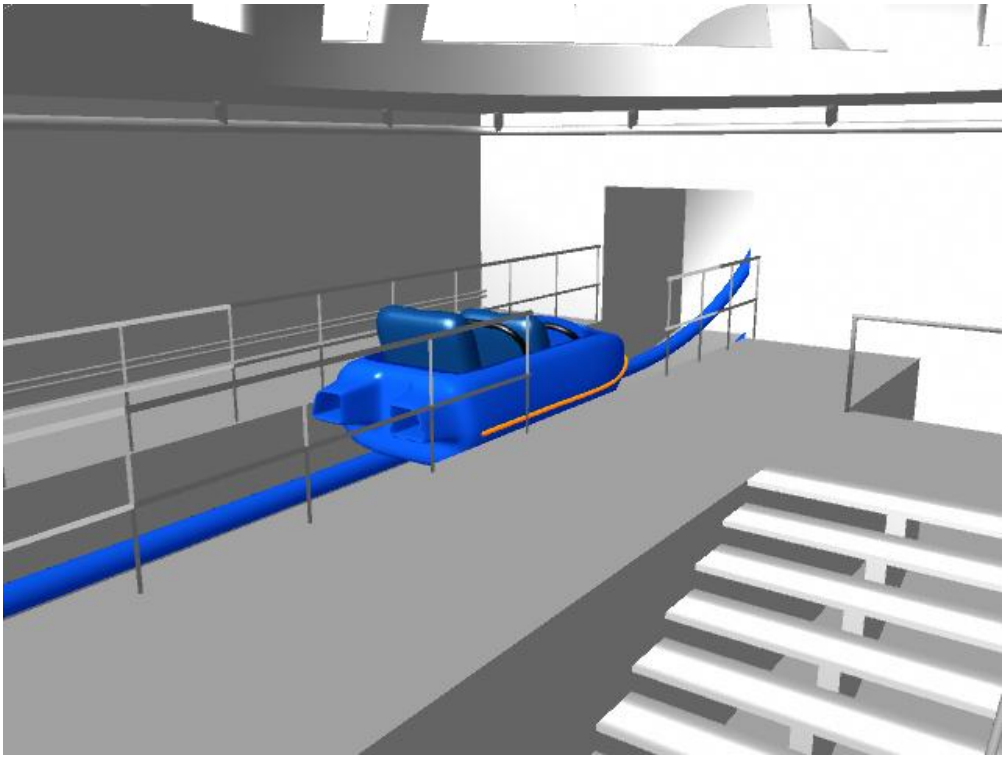


Figure 1: Original scene

Chapter 2 - Analysis

1. Lighting

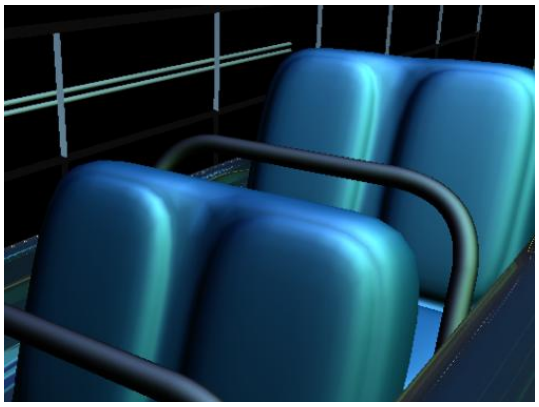
The lighting is based on the Phong shading. This model of lighting is based on three different components: the ambient, the diffuse and the specular. It is a simplification of the real physical effects, there is only a few parameters considered: the normal of the vertex or the fragment, the view vector which represent the direction to the eye and the light vector.

For each light, the "Phong reflection model" is applied:

$$I = k_a i_a + k_d (L.N) i_d + k_s (R.V)^{\propto} i_s$$

It is possible to see the three component evocated previously in this equation, first, $k_a i_a$ represent the ambient $k_d (L.N) i_d$ is the value of the diffuse and $k_s (R.V)^{\propto} i_s$ give the specular. It is interesting to see that the diffuse depend only of the light vector and the normal, and the specular is a combination of reflect and view vector.

With the default pipeline, the lighting is computed for each vertex and then, interpolated on the fragment. Thanks to shader's language, it is possible to compute the lighting for each pixel, it is slower because there is more fragment than vertex but the effect is nicer.



```
float3 diffContrib;
float3 specContrib;
/*float3 Rn = normalize(Ln + Vn);
diffContrib = dot(Nn,Ln)* Lamp0Color;
specContrib = diffContrib * pow(dot(Nn,Rn),SpecExpon)*Ks*Lamp0Color;*/
//OR
float3 Hn = normalize(Ln + Vn);
float4 litV = lit(dot(Ln,Nn),dot(Hn,Nn),SpecExpon);
float3 color=IN.color.rgb;
diffContrib = litV.y * color ;
specContrib = litV.z*litV.y * Ks * color;
```

2. Bump Mapping

The bump mapping is a very common effect in computer graphics but it is very powerful to give an effect of bump, it is very simple: before the lighting computation in the pixel shader, the normal is perturbed by another texture which can be call “normal map”.

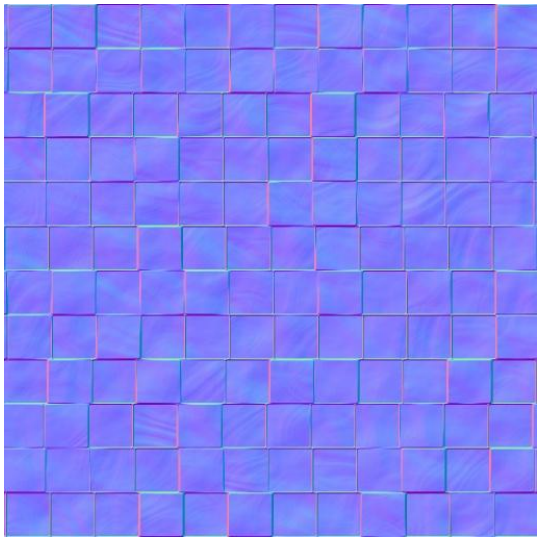


Figure 4: Normal map

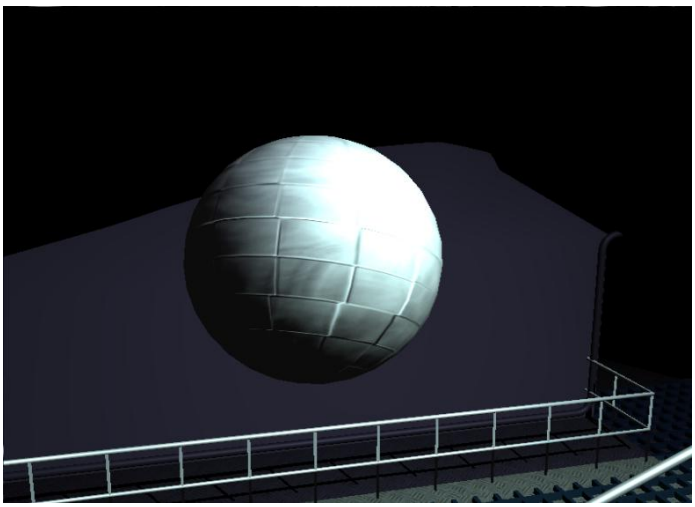


Figure 5: Result of the bump mapping

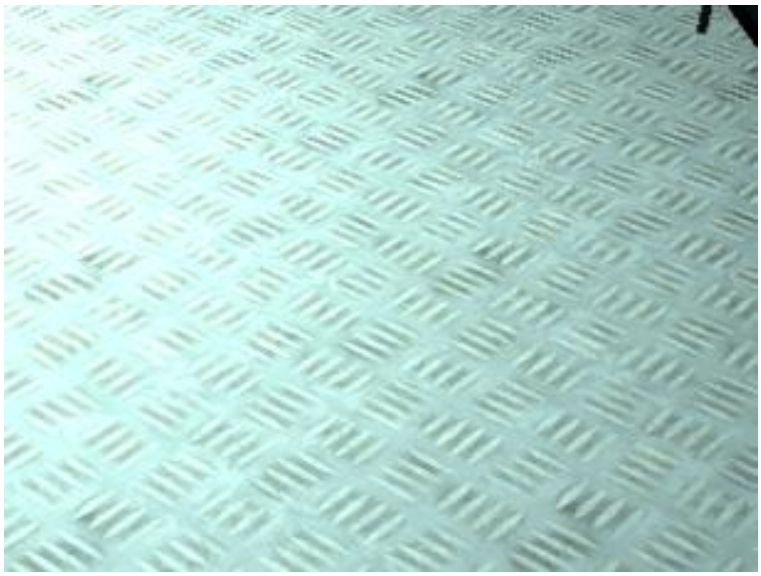


Figure 3: Without Bump Mapping

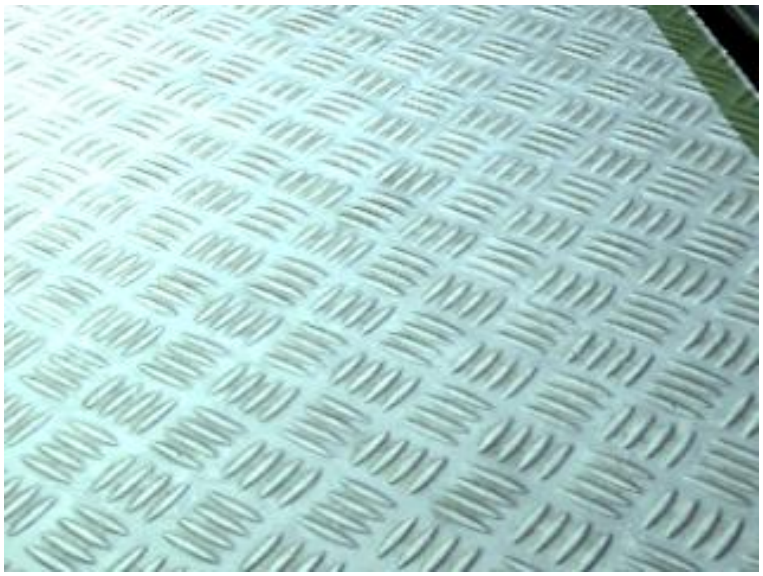


Figure 2 : With Bump Mapping

3. Reflection

The reflection and refraction in real time graphic are, the most of time, based on a cube map. A cube represent an environment, it is set of six render for each face of the cube. Dynamic cube map are costly due to these six render, it is possible to optimize this technique rendering only element which has been changed between two render and disabling some costly effects.

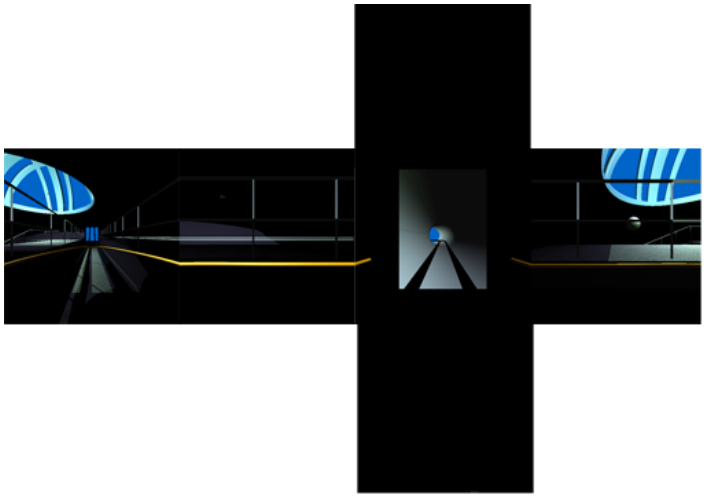


Figure 6: Cube map generated

To apply this cube map as a reflection, in the fragment shader, we use the reflect function which depend of the view vector and the normal. The value of the reflexion is combined with a Fresnel value, the Fresnel value should be used to set reflection and refraction value depending of the view vector in a glass effect, but, here, it is to increase the reflection on the edges to give a nicer effect.

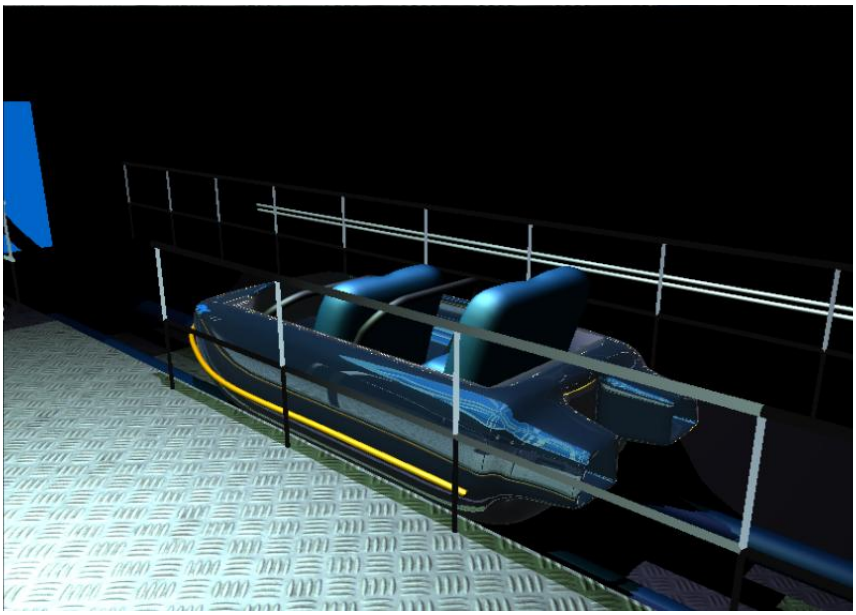


Figure 7: Result of the scene with reflexion on the craft

4. Shadow Map

The shadows are essentials in real time graphic, shadows gives realistic effect. There are some techniques to render shadow in real time. The shadow mapping is less accurate than shadow volume but, this technique is really faster. The shadow mapping is based on the rendering of the depth buffer from the light view. The depth is rendered into a texture and projected with the view matrix of the light on the object shadowed. This part will explain this technique step by step.

As it is said previously, we have to render the depth map of each directional light. A projection and view matrix is necessary. The projection matrix is simple; the field of view value define the size of the light. The view matrix is a “look at” with the position, direction and up of the light. The scene is rendered with these matrices, it is possible to directly get the depth map thank to depth buffer, but, in this project, a shader rendering the depth has been implemented. It is important to keep value of depth between 0 and 1 because it is store in a texture.

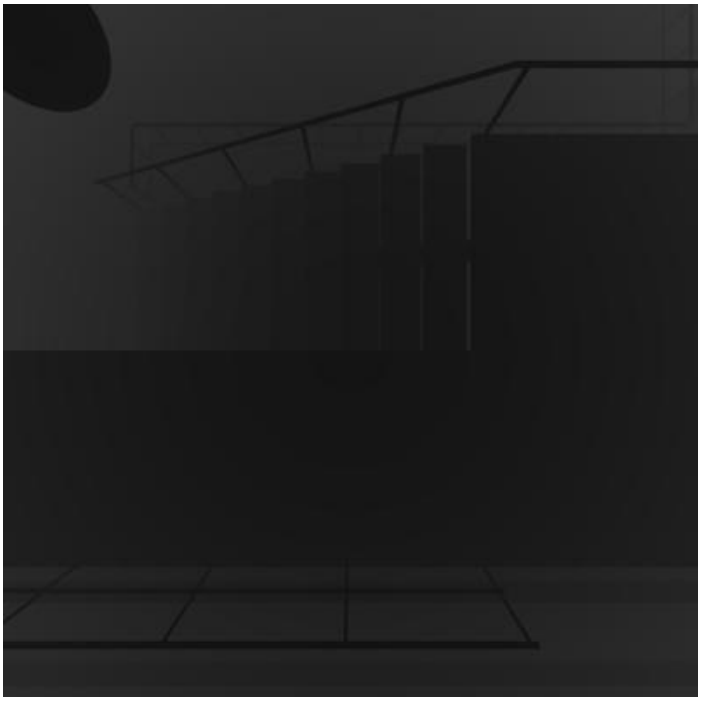


Figure 8: shadow map from light2

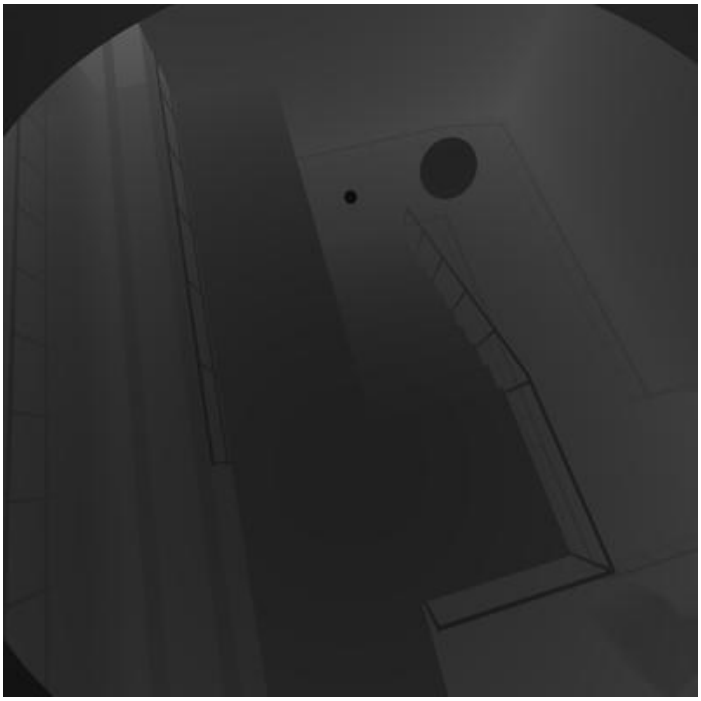


Figure 9: shadow map from light1

Then the depth map texture is projected on object shadowed. To generate correct texture textures coordinate, the position of the vertex in object space is multiplied by the world transformation matrix of the object, then, the view and projection matrices of the light. The result is stored in a texture coordinate channel for the next step in the pixel shader.

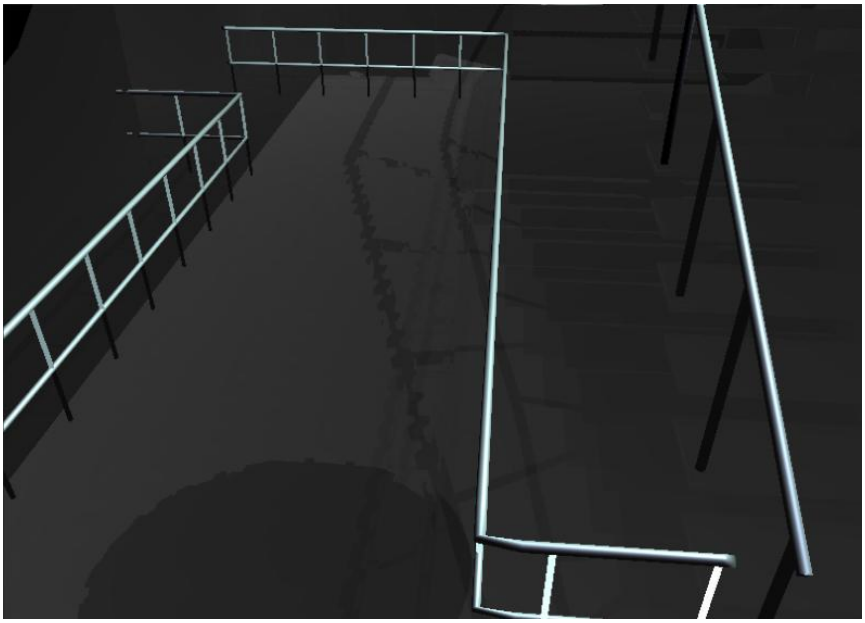


Figure 10: Shadow map projected

In the fragment shader, thanks to the shadows, knowing the position of the light and the fragment, it is possible to identify if this pixel is in shadow or not, if the depth value in the shadow map is different than the computed distance from the light, an object hide the fragment for the light, so, the pixel is in shadow.

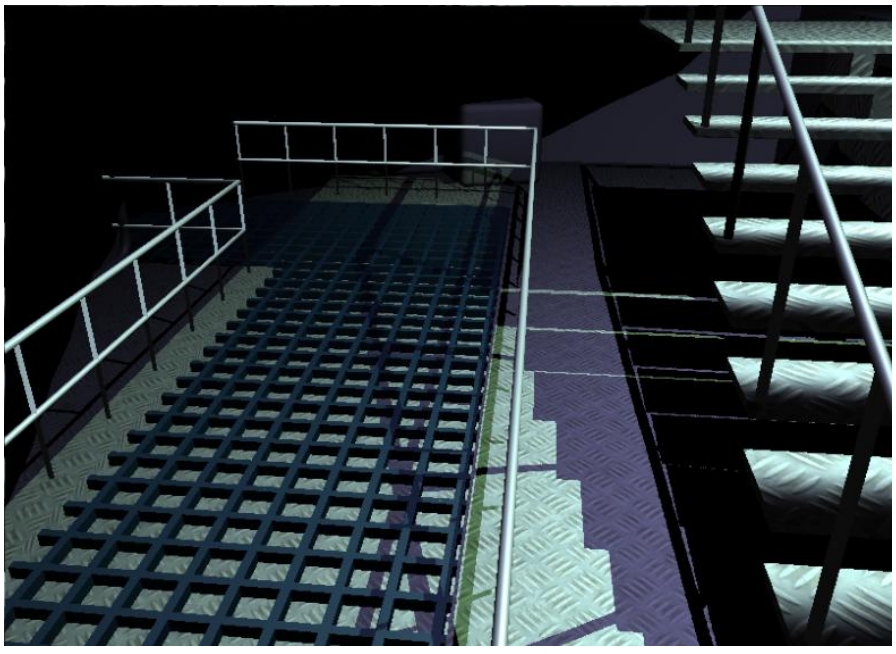


Figure 11: The scene shadowed and textured with two lights

5. Post Render Effects

In this application, the scene is never directly rendered. A texture target is created before the display and this texture is applied to a plane of the size of the screen with particular shaders. So it is possible to easily add some additional effects directly on the render. The first one is an approach to the motion blur effect. In fact, it is simply a mix of the n (five in this sample) last renders with the last one more visible than the others.

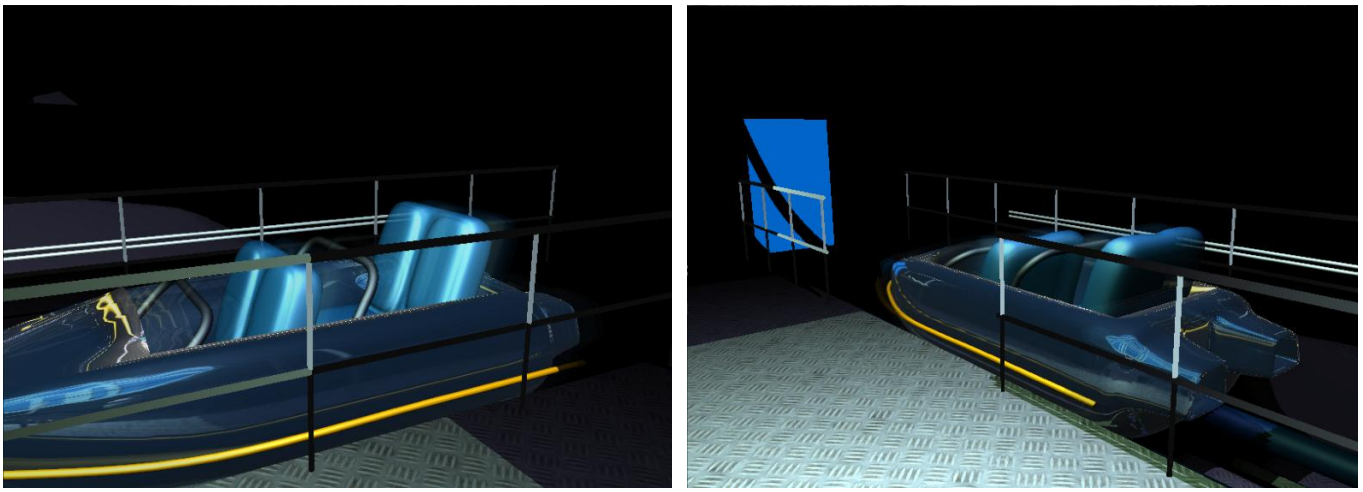


Figure 12: Motion Blur

It is very simple to implement other post render effects. The following part will explain how a particular post render effect has been implemented. The idea is to simulate the display of a cathode ray tube (CRT) television. First, it is necessary to quickly describe this technology. The display of the screen is given by little component which are called luminophore; there are three component red green and blue really close. At a normal distance from the TV, the colours seem to be mixed.

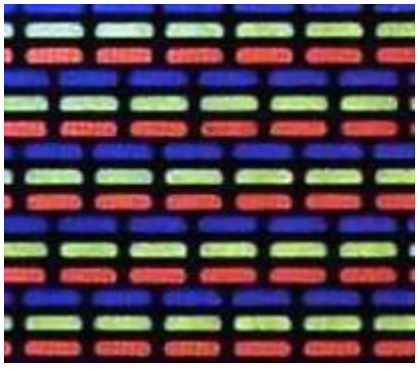


Figure 14: Luminophores

Figure 13: TV sample

To approach of this effect, two techniques has been combined, first, we have to use a texture which going to represents these luminophore. This texture will loop and is repeated to have small but visible pixels. The real pixels are multiply by the value of this luminophore texture. Then, a displacement map is defined to deform the screen to give the effect of deformed display near to the edges in fact; the displacement map will modify texture coordinate before the multiplication by the luminophores.

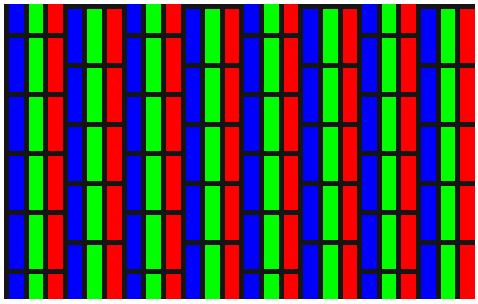


Figure 15: The texture for luminophores

Figure 16: The displacement map

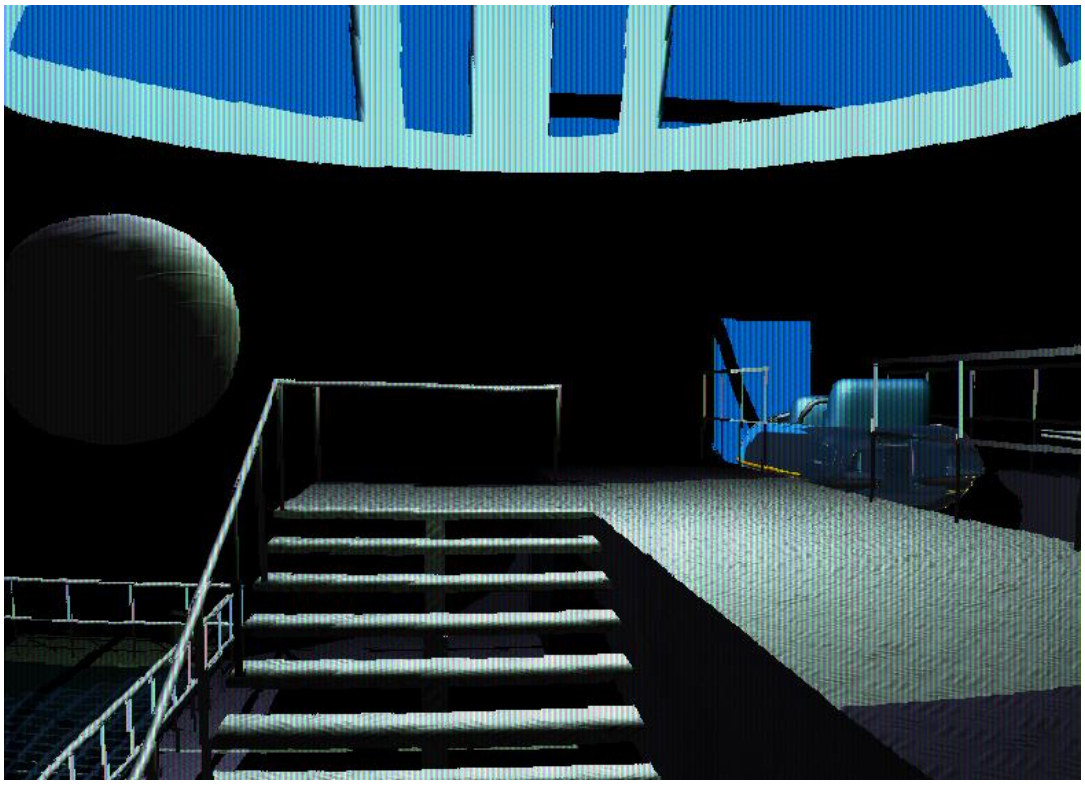


Figure 17: render of the TV Screen effect

Chapter 3 - Evaluation

The application is almost fully settable without recompile the project because XML files have been massively used in this project. The scene comes from a TSF file, shaders and these parameters (resolutions of cube map and shadow map too) are stored in other XML file, and this last define also the shader by objects. The animations are given by another XML file. It isn't possible to choose XML files loaded with the execution of the application but it could be very easy to do.

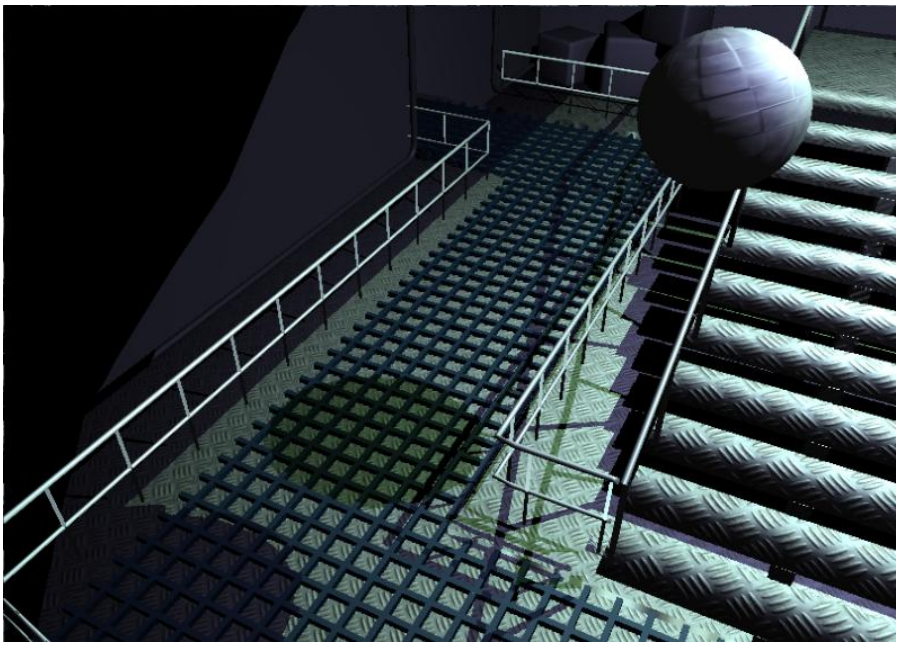
Beyond the shaders, the application is designed to simplify the development. There is a shader class, an abstract shader parameter class with a lot of child, such as Matrix, Material, Light or simply Float and especially a shader manager which contain information to send for each object. These classes have been really useful during the development of shaders, it help me to easily set the scene without any waste of time. There is still an aspect that I haven't developed professionally; the leak in memory, I should have done my delete and release functions during all the development process.

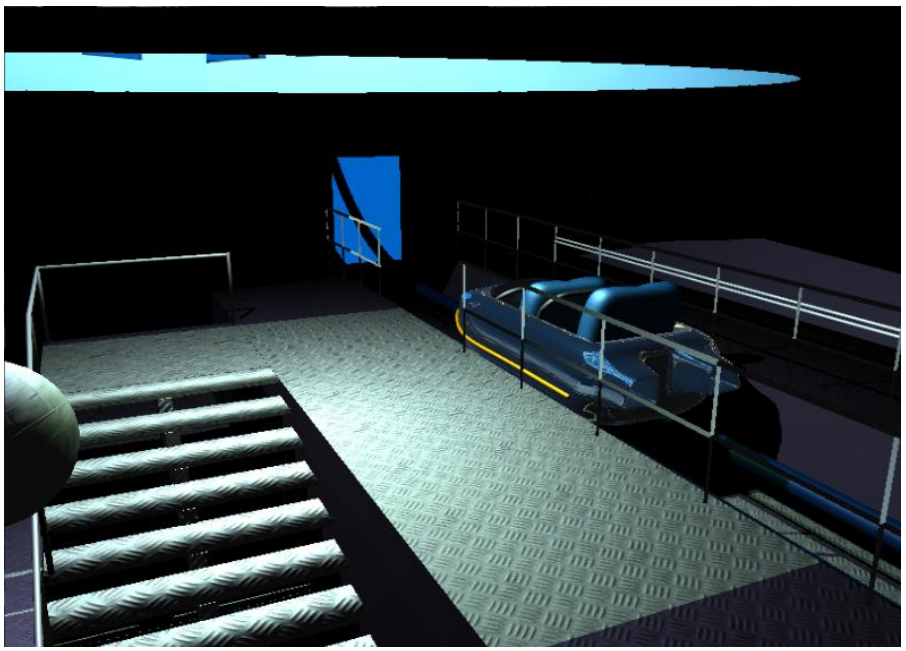
The implementation of shaders is less designed but some functions, like the illumination, are separated to ease potential modifications. The possibility offered by shader language hasn't completely been explored. Some effects are more complicated to implement than other, the shadow map is a good example of complex technique but the effect given is really impressive and the realism of the scene is improved. The reflexion by a cube map was an interesting research, I have added a Fresnel value on the reflexion, it isn't a realist effect but it could be interesting. The lighting and the bump mapping are common but the effect is right. I think the TV effect could be a good demonstration of the possibilities of render into a texture.

The main disadvantages of this application is the performances, I cannot hide that I haven't done any profiling to identify bottlenecks. I think that the main bottleneck can be found with the setting of the shaders on each frame, a lot of objects have the same shader and all matrices are computed for each object instead of order entities to optimize the render. A potential bottleneck could be the render of the cube map; it could be interesting to speed up the render disabling some effects. Even if the application hasn't been profiled, the frame rate is good with a correct computer.

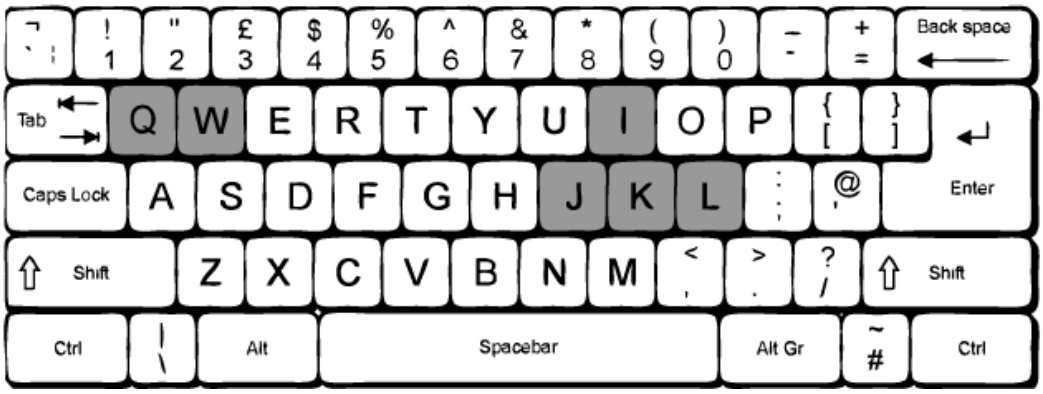
This project isn't perfect but it is a good representation of possibilities offered by customisable shaders. Shading language is essential in computer graphics, almost when the real time is expected. There is a lot of research about rendering techniques thanks to customisable shader, it is an interesting domain, this application isn't the best demonstration of these possibilities but I am satisfied of my work. It was a good exercise to get and improve my skills. It was my first approach of DirectX with Shaders language.

If I had to continue with the research of rendering with shader's language, I will explore the possibility of optimisation and improvement by deferred shading. These techniques could permit the utilisation of several lights without problems of performances; the high dynamic range is also a technique that I should study. The shadow map could also be rendered in deferred to create soft shadows into the screen space. Finally, it will be interesting to study the possibility of shader model 4.0 and DirectX 10 especially the geometry shaders.





Chapter 4 - Usage Instruction



- **Mouse left click and drag** : Rotate the camera
- **I, J, K, L :** Move the camera
- **Q :** Change the post render shader
- **W:** Stop or play animation

Chapter 5 - List of features

Feature	Level of implementation
Per Pixel Phong lighting	Up to four light – No particular bug
Bump mapping	Independent pixel shader so, binormal and tangent computed per fragment
Cube map Reflexion	Dynamic cube map generated, applied on the craft combined by a Fresnel value: There is more reflexion on the centre than the side.
Shadow mapping	Two shadow maps computed for two different lights mixed on the most on the object in the scene. The main bug is due to the precision of the depth map that should be in 32 bit float instead of 8bits.
Post Render effects	Scene rendered in a texture, possibility of different post effects, motion blur, cel-shading or black and white.
Designed shader implementation	Possibility to add object and add shader by XML files. The main problem is the optimisation; some matrices can be computed for each object, objects aren't group by shader.
Animation by key frame	The animations of the objects are stored in a XML file and then, interpolated along the time.