HDRI AND IMAGE BASED LIGHTING

Master Project

Chong Deng
I7913026
NCCA

Abstract

This report illustrates the process of create Image Based Lighting (IBL) with HDR images. This program has been implemented by OpenGL and C++, and the shaders have been implemented by GLSL. Cube map has been used to create the background and the environment map of the object. Median cut algorithm has been implemented in the program, and then used to sample the environment map, and calculate the light colour and light position as well. In this report, the pipeline of the framework and the main algorithm will be demonstrated. In order to display the HDRI properly, some post-processing effects are used.

Keywords: *Image based lighting, Cube mapping, HDRI, Real-time*

Chapter 1 Introduction

In nowadays, more and more film production prefers to use seamless integration of CG objects and real scene. Image based lighting provides a good way. Image based lighting (IBL) is a process of illuminating the scene and CG objects by using the light of real world. In the IBL process, the light, which is sampled from a picture of the real world, is used to relight the CG scene. This technique can create realistic rendered appearances and combine the computer-generated objects with real scenes properly. So more and more films, like "*Iron man2*", *"The terminator","Avatar",* use this technique to get the attractive effect.

Image-based lighting is also popularly being used in video games and vitual reality. Because one of the advantages is that the photorealism result can be achieved easily without creating the complex real-world physical interactions. The only requirement is some simple photographs. That's why almost all of modern rendering softwares, such as Maya, Houdini, offer image-based lighting options, although the exact terminology may vary.

1.1 Objective

In this project, HDR images are used to create the scene and the environment map. Users can switch among different maps to get different scenes and different lighting effects. The camera can be moved to observe different angles of the object. Users can translate the object to see the change of the specular and diffuse light. And also users can select among different primitives and obj files. Different shaders are used to perform on the object to get different look. In the post-process, users can switch between different tone mapping methods, which are scurve, Rahman Retinex and bloom. The parameters of each tone mapping method can be adjusted.

1.2 Application used

To perform this program, the following applications were used:

QtCreator – the IDE used to implement the application.

HDRshop – the software developed at the USC Institute, used for generate textures.

Doxygen – used to generate the documentation of the program.

1.3 Libraries used

NCCA Graphics Library – developed by Jon Macey, and is based on OpenGL 3.x and QT and has a dependency on boost_1_44_0

Chapter 2 Previous work

Image-based lighting is based on the foundation of texture mapping and reflection mapping, which was first discussed by Jim Blinn and Martin Newell[1] in 1976. Image based lighting can be considered as an extension of environment mapping. As Debevec explored this area a lot, it becomes a more and more popular topic in the area of relighting.

Most research about image-based lighting can be refered to Debevec's website [2] and the book "High dynamic range imaging"[3]. They provide plenty of details of development and the main principle of making image based lighting.

2.1 Image based lighting

At SIGGRAPH 98, Paul Debevec released a paper, "Rendering Synthetic Objects into Real Scenes",[4] this is the first paper which introduced the image-based lighting officially. In this paper, Debevec described the general Method of image-based lighting as the figure showed below:

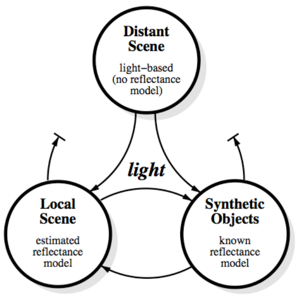


Figure 1 The general model of image based lighting[4]

In order to represent a scene, Debevec constructed a new light-based model and measure the illumination at the synthetic object location, and then composite the three components using the global illumination with different rendering technique.

In 2002, Paul Debevec published a tutorial "Image Based Lighting" [5] in IEEE computer graphics and applications. In this article, he demonstrated the basic steps of IBL.

First, he introduced several ways to capture an omnidirectional image of the real world. The common ways are to use a fisheye lens, which has 180-degree field of view, or a mirror ball, which has a reflective surface. Both of them can capture the surrounding environment easily. When the image is obtained, the next step is to apply it to the scene as an environment image. And then simulate the light from the environment illuminating the CG object.

In this paper, Debevec explained how to generate IBL to light CG objects in Radiance, which is running in Unix and for free. Pictures below are the scenes generated from the Radiance. The code refers to the Article.

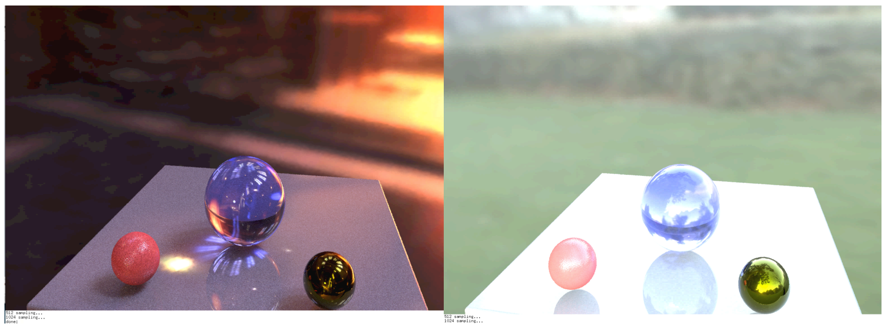


Figure 2 Radiance generated scene

Like the pictures show above, IBL can make the objects appear to belong in the scene. The light, which illuminates the object, looks like from the background. There are two important processes of IBL. One is omnidirectional photography. The light in the real world is infinite from every direction, and contributes to the objects, so the photography should be omnidirectional. And this is widely known as light probe image. After we capture the images using the ways shown above, we can use HDRShop[22] to generate the images to get proper version, cubic environment, or latitude/longitude, or angular map. The next is global illumination. It is always implemented by ray tracing. [1,6] This method shows how the light goes from the light source, reflect among the surfaces, refract into an object. Finally, It can generate the CG object appearance. The book "High Dynamic Range Imaging", which is published in 2005, shows many details about IBL, including how to generate light probe image, how to sample the image, how to map the omnidirectional image, how the global illumination renderer computes IBL images, how to simulate shadows and scene object inter-reflection.

2.2 Sample methods

Sampling is very important in the IBL. Accurate light information will be got if proper sampling method is used. There are also many methods of sampling the light probe images. Structured importance sampling, introduced by Agarwal, Ramamoorthi, Belongie and Jensen in the year 2003, represented the distant natural illumination on surfaces.[7]

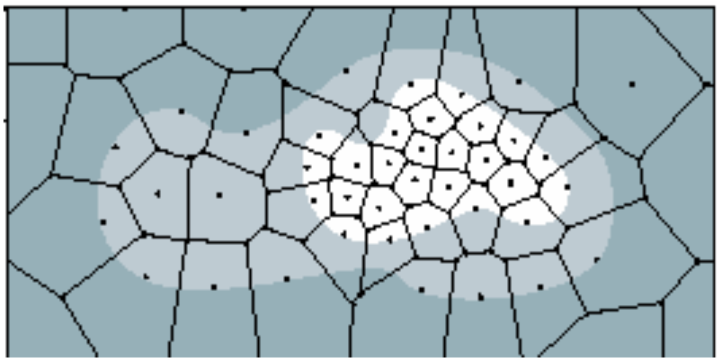


Figure 3 Structured Importance Sampling of Environment Maps, 2003[7]

At SIGGRAPH 2004, Fast Hierarchical Importance Sampling with Blue Noise Properties was introduced by Victor [8]. In this method, they create a large number of sample points, which are numbered using the Fibonacci number system. This technique is also the importance based, and can be used in a large area of computer graphic applications.

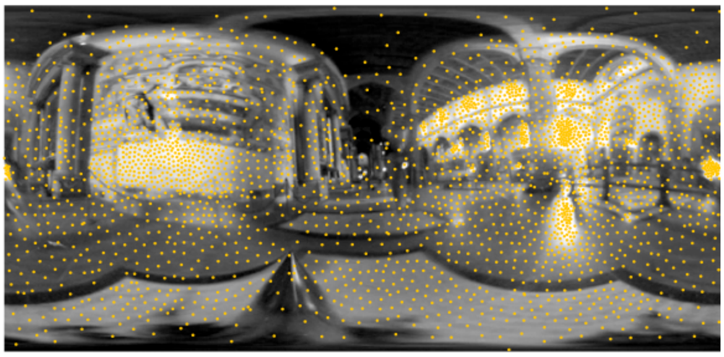
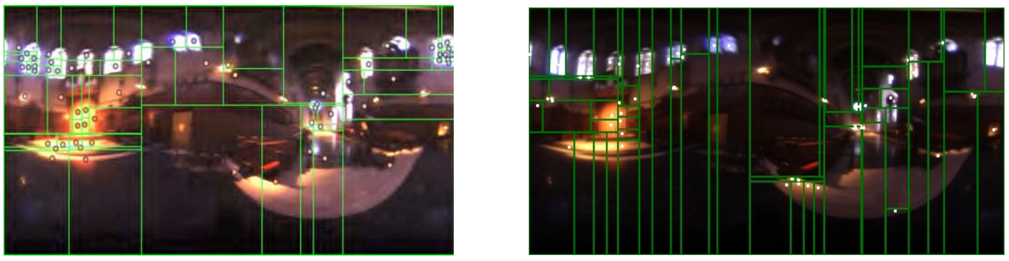


Figure 4 Fast Hierarchical importance sampling[8]

In the year 2005, Debevec presented a method called median cut algorithm for lighting probe sampling. This algorithm used a summed area table to cut the image into regions of equal energy, and was widely used in many applications. [9]

In the year 2009, Viriyothai and Debevec updated the algorithm together. This algorithm is based on the median cut but make the maximum of two sub-region's variance minimized. In this case, the sample lights will have never been cut through, when they are placed in the same region. [10]



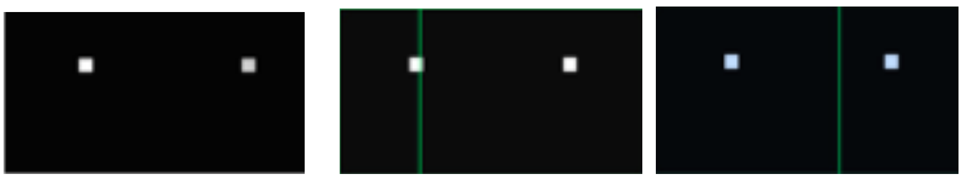


Figure 5 Median cut and Variance cut (the middle one shows that median cut may behave improperly when two bright light are placed in the same region)[9,10]

In this project, Median Cut Method will try to be used to sample the cube environment map directly.

2.3 Shadows

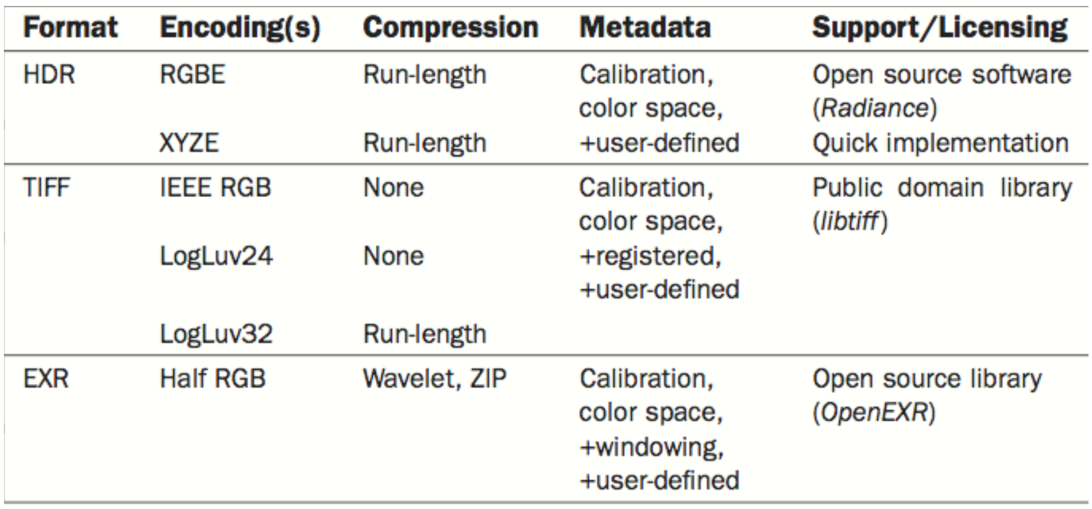
Once the light was captured in the environment map, it is well realized that how to cast shadows in a real scene. This is the most notable effects rather than the appearance of the objects in the environment. In the IBL case, the light is coming from infinite far away, so the shadow should need to match the potential light directions, as a result, the shadow should be very soft. [11] To generate the shadow, one method is to use shadow maps and provide the associated benefits. The other is to use ray tracing. One approach is to take layered attenuation maps to achieve the interactive rendering first and then apply the coherence-based ray tracing of depth images which is not interactive but can remove the limitation of the first methods. [12] Soft shadows can be generated by radiosity[13]. And many methods have been extended from the basic radiosity method. [14,15]

Chapter 3 Technical backgrounds

3.1 HDR

HDR pixel values can cover the full range of light in a scene; therefore, HDR image contains a high dynamic range of intensity. It is good to use it as a light source to illuminating the computer-generated objects and scenes. And if we use suitable rendering algorithms, we can accurately simulate how the CG object and the scene would look like as they were illuminated by the light, which is coming from the real word.

The following are the format of the HDR image:



Format	Encoding(s)	Compression	Metadata	Support/Licensing
HDR	RGBE	Run-length	Calibration, color space,	Open source software (*Radiance*)
	XYZE	Run-length	+user-defined	Quick implementation
TIFF	IEEE RGB	None	Calibration, color space,	Public domain library (*libtiff*)
	LogLuv24	None	+registered, +user-defined	
	LogLuv32	Run-length		
EXR	Half RGB	Wavelet, ZIP	Calibration, color space, +windowing, +user-defined	Open source library (*OpenEXR*)

Figure 6 Format of established HDR images[3]

The HDR format is widely known as Radiance picture format, such as *.hdr, .exr*. It has been used in particularly for HDR photography and image-based lighting. In this project, format *.hdr* is used to create the background and the environment map as well. The pixel data has a 4-byte RGBE encoding. And the bits are like:

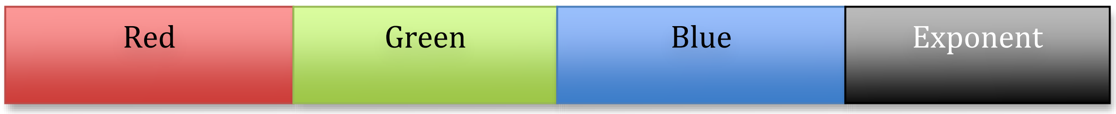


Figure 7 HDRI component

For each component there is a formula to convert the RGBE to the colour RGB, which is supported by the display devices. Suppose the components of HDRI is

(R_H, G_H, B_H, E), and the components of screen-supported colour is (R, G, B), then we can have equations as below: [3]

$$E=[\log_2(\max(R_H, G_H, B_H)) + 128]$$

$$R = \left\lceil \frac{R_H \times 256}{2^{E-128}} \right\rceil$$

$$G = \left\lceil \frac{G_H \times 256}{2^{E-128}} \right\rceil$$

$$B = \left\lceil \frac{B_H * 256}{2^{E-128}} \right\rceil$$

Equation 1 the equations of RGBE to RGB

3.2 Median Cut Algorithm

For the sample of environment map, in this project, Median cut Algorithm is chosen. As it is said above, this sample method was introduced by Paul Debevec in 2005; the main algorithm is like this:

Step1: place the original light probe image into a region list as a single region.

Step2: For each region in the region list, get the longest dimension and subdivide the region in order to divide the light energy evenly.

Step3: If the iteration is less than n, then go back to step2.

Step4: finally, put a light source in the center of each region, and set the colour as the sum of pixel values within the region.

To calculate the light energy, we can use the formula below. Suppose the light energy is Y, then we can get the representation of light energy:

$$Y=0.2125R+0.7154G+0.0721B$$

Equation 2 the equations of calculating illumination

In order to compensate the over-represent regions near the pole; the pixels of probe image should first be scaled by $\cos\varnothing$, where $\varnothing$ should be the pixel's angle of inclination. [9]

3.3 Environment map (Cube mapping)

Although there is many kinds of texture mapping methods can be chosen, like equirectangular texture mapping, lat-long texture mapping and cube mapping.

Comparing to sphere mapping, cube mapping solves many problems like image distortion, viewpoint dependency, and also computational efficiency. Another advantage of using the cube mapping is that it provides a large capacity to support real-time rendering of reflection. [11]

As a result, in this project, cube mapping is used. Generally, six side images are needed for the six faces of cube surface. However, In this case, only one vertical cross texture is needed for both background and the reflection of object.

So as to simulate the environment, which is considered to be infinitely far away, cube maps are used to create the reflections. And Cube mapping reflection is a standard part of real-time graphics recently. The reflection model displays the illumination of a surface with reflection of the environment. The reflection value is obtained by looking up the environment map.

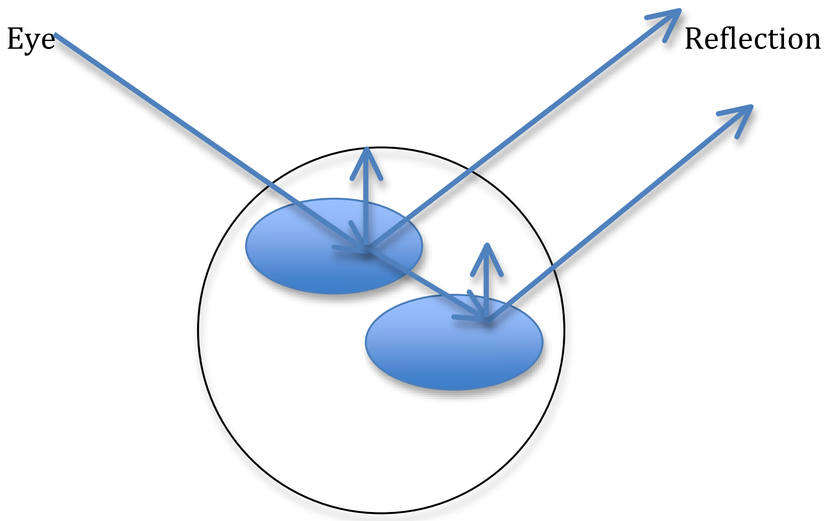


Figure 8 the Reflection model

Considering the vector of eyes is V, and the reflection vector is R, the normal of the object is N, then we can get the reflection formula like:

$$R=V+2(V\bullet N)\,N$$

Equation 3 reflection formula

3.4 Frame Buffer Object

Frame Buffer Object is the OpenGL extension. It provides an interface to create additional non-displayable frame buffer objects. By using this, an OpenGL application can render the output to the application-created frame buffer object. It has colour buffers, depth buffers, stencil buffer and accumulation buffer. It is totally controlled by OpenGL. There are two types of frame buffer-attachable images. One is called texture images. If an image of a texture object were attached to a frame buffer, OpenGL would use "render to texture". If an image of a render buffer object were attached to a frame buffer, then OpenGL would perform "off-screen rendering".

Frame Buffer Object also provides an efficient switching method. Switching framebuffer-attachment images is much faster than switching between FBOs. It use glFramebufferTexture2DEXT() to switch between texture objects, while using glFramebufferRenderbufferEXT() to switch between render buffer objects.

In order to do tone mapping, the first pass needs to be stored as an input so as to generate the second pass. In this case, frame buffer object is used to store the first pass on which the image-based lighting is applied. The first pass has been saved as a texture and used by the tone mapping shaders.

Chapter 4 Implements

In this project, HDR images are used to represent the background as well as the environment map of the object. A diffuse convolution is used as an environment-map to light a diffuse object, while a specular convolution is to use as an environment-map to light a shiny object. Pre-computing of the convolution has been done, as this process would take quite a long time. But the All of these images are format of *.hdr.* And use cube-mapping method to apply the environment map.

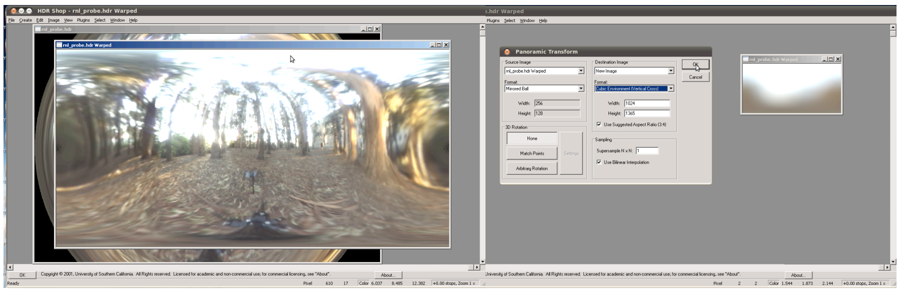
4.1 Diffuse convolution using HDRShop

For doing diffuse convolution, a latitude/longitude format light probe image should be loaded into HDRshop. Those images can be downloaded easily from Paul Debevec's website.

The size of the image, which used to be converted, should not be too large. So once the image is loaded, do resize to make sure that the convolution can be finished in a reasonable time. In this process, size of conversion is chosen to be 252×127.

Then use diffuse convolution (image->panorama->diffuse convolution) to generate the diffuse map. Use the size we adjust above, it will still take 1536 seconds to convert the image to diffuse convolution.

After that, go to image->panorama->choose the cubic environment (vertical cross) as the format of destination image. This will be used in cube mapping later. And finally, those images are saved into *.hdr* format.



Panoramic Transform

Source Image: rnl_probe.hdr Warped
Format: Mirrored Ball
Width: 256
Height: 128

Destination Image: New Image
Format: Cubic Environment (Vertical Cross)
Width: 1024
Height: 1365
☑ Use Suggested Aspect Ratio (3:4)

Sampling
Supersample N x N: 1
☑ Use Bilinear Interpolation

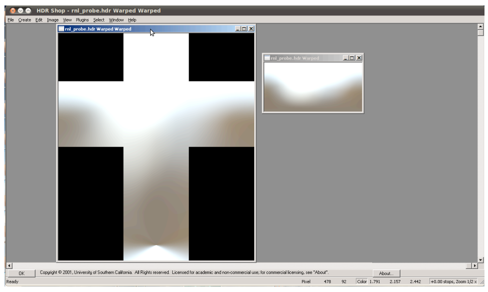


Figure 9 production of diffuse map using HDRShop

4.2 HDRI loading

There are lots of *.hdr* format images can be downloaded from website, that's why this format is chosen in this project. To find a way to load this format cost such a long time. And fortunately, there is an implement of load HDRI images method has been found on the Internet. [16] The method has been modified to suit for the class. The implement of HDRI components converted to the screen-support components is like this:

```
float convertComponent(int expo, int val)
{
float v = val / 256.0f;
float d=(float) pow(2, expo);
return v*d;
}
```

List 1 Code for convert the RGB component

The RGB has been stored in a float pointer and the code is as follows:

```
void workOnRGBE(RGBE *scan, int len, float *cols)
{
  while(len-- >0)
        {
          int expo = scan[0][E]-128;
          cols[0] = convertComponent(expo, scan[0][R]);
          cols[1] = convertComponent(expo, scan[0][G]);
          cols[2] = convertComponent(expo, scan[0][B]);
          cols+=3;
          scan++;
        }
}
```

List 2 Code for store the float value

By doing this all of the colour information has been stored in the float pointer, which will be used in the cube mapping and median cut. We can tell from the code that, the colour channel was stored separately. Col[0], col[3],col[6]…stored red channel; col[1],col[4],col[7]…stored green channel; and col[2], col[5],col[8]…stored blue channel, the rest may be deduced by analogy. So when the HDR image is loaded, it is easy to get the colour information.

4.3 Cube-mapping

The traditional cube mapping is to store 6 different sides of a cube map as six square textures and then project them onto six faces of a cube. In this project just one vertical crossing cube map is used for both background creating and environment map. In the program a function called memcpy() is used to split the vertical crossing map into six-sided maps and store them separately in cubemap array. So as it can be used when binding the texture. [17]

4.4 Mediancut

In this part, mediancut algorithm is implemented as follows:

Create the lights array and region array and initialize them at first. Next step is to load the HDR image and get the width and height, and decide which is the longest dimension.

Create a region list, and add the image to the region list, then get the long dimension of the region, and scan it and then calculate the sum of the longest dimension, and then get the median number from the sum, which we calculated in advance.

Then we use the median number to do cut. This will create two new regions. And this will be a loop until the size of the regions up to the iteration n, which is given by the user.

When all the cut was done, create lights in the center of the region and calculate the colour of the light using the average of the colour in the current area, and save the picture as .png format finally (still fail to save it in *.hdr* format, so use *.png* instead temporarily).

At the beginning of the project, median cut algorithm was supposed to use in two ways, first is to sample the cube map and finally get the subdivided image. And to be used as diffuse map after it has been blurred. Secondly, several lights can be created based on the image so that they can cast the shadows properly. But the effect was not very good. So the traditional way has been chosen finally, but the median cut part has been conserved for the preparation of the future work. Judging from the output image and the output data of the light's position and colour. The code refers to the NCCA median cut algorithm, but modified to fit for the HDR format.

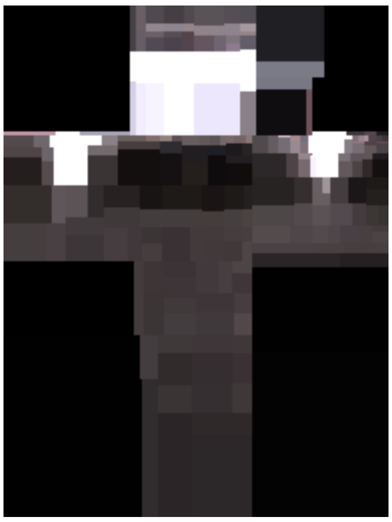


Figure 10 the image which is done by median cut

4.5 GLSL and shaders

GLSL is OpenGL Shading language, which is designed specifically for OpenGL. In GLSL vertex shader and fragment shader are connected together into the OpenGL pipeline. In the project, GLSL is used to define the lights travel methods, in order to decide the appearance of the object, and also used for the tone mapping and post-process.

To specify the usage of the shaders, a class for managing the shader has been used. In this class, there are lots of function for shaders to create a new program, access the .vs and .fs files, and attach the source to the program and so on. In the application parts, different shader objects will be created and then call proper functions to generate the shaders. Set parameter functions have not been packaged in this class; this will make it easier to input the relative values from interface.

4.5.1 Object Shader

Several kinds of different shaders are defined here; users can choose different ones and apply them to different objects.

4.5.1.1 Reflection shader:

This shader is the simplest one; just one cube map has been used to simulate the perfect reflective surface. In this shader, per-pixel method has been used. The model is as follows:

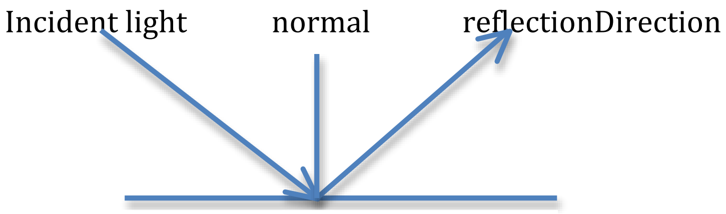


Figure 11 A perfect reflection model

As the model described, each view vector coming from eyes, will achieve the surface and then reflect to the environment. The normal is the normal of the surface. The reflectionDirection can be calculated using the GLSL built-in function

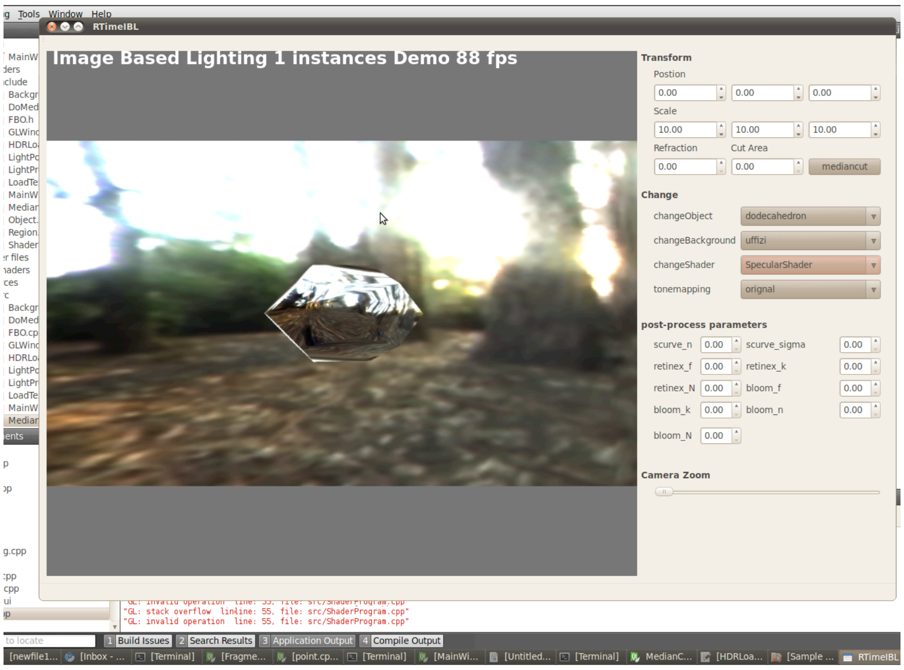
```
Reflect (eyeDir, normal);
```

List 3 GLSL Reflect function

The direction of the reflection will be transfer to fragment shader, and this vector will be used as an index to find the proper pixel of the cube map.

```
textureCube(specMap, ReflectDir);
```

List 4 GLSL the texture mapping function in fragment shader



Postion: 0.00, 0.00, 0.00
Scale: 10.00, 10.00, 10.00
Refraction: 0.00
Cut Area: 0.00

Change
changeObject: dodecahedron
changeBackground: uffizi
changeShader: SpecularShader
tonemapping: orignal

post-process parameters
scurve_n: 0.00 scurve_sigma: 0.00
retinex_f: 0.00 retinex_k: 0.00
retinex_N: 0.00 bloom_f: 0.00
bloom_k: 0.00 bloom_n: 0.00
bloom_N: 0.00

```
"GL: stack overflow  linline: 55, file: src/ShaderProgram.cpp"
"GL: invalid operation  line: 55, file: src/ShaderProgram.cpp"
```

Figure 12 reflection shader

4.5.1.2 Diffuse shader:

Diffuse reflection is due to the summing up of the many subsurface reflection. But in this case, this shader used one diffuse map to simulate the diffuse reflection. The diffuse map is converted by HDRshop.

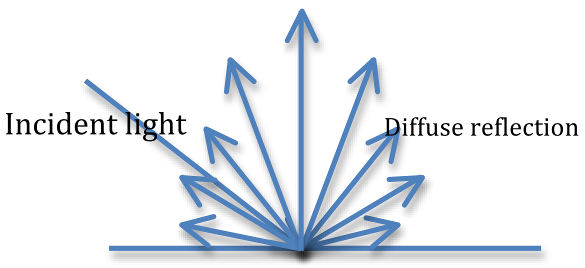


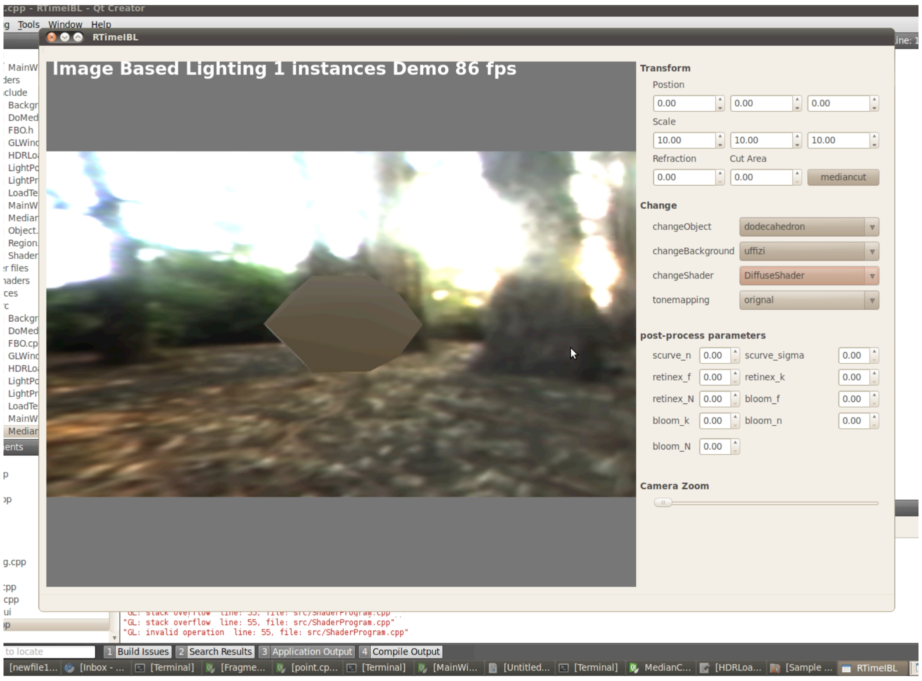
Figure 13 diffuse reflection model

As the model showed, the direction of diffuse reflection is massive. And this normal is changed according to the view. Therefore, to calculate the normal of diffuse reflection, the surface normal should be multiplied by the ModelView matrix, so:

$$Normal_{diffuse} = ModelView * Normal_{surface}$$

Equation 4 calculate the normal of the diffuse reflection

And then use the diffuse normal as an index to find the right pixel in the diffuse map.



Transform

Postion 0.00 0.00 0.00

Scale 10.00 10.00 10.00

Refraction 0.00 Cut Area 0.00

Change

changeObject dodecahedron

changeBackground uffizi

changeShader DiffuseShader

tonemapping orignal

post-process parameters

scurve_n 0.00 scurve_sigma 0.00

retinex_f 0.00 retinex_k 0.00

retinex_N 0.00 bloom_f 0.00

bloom_k 0.00 bloom_n 0.00

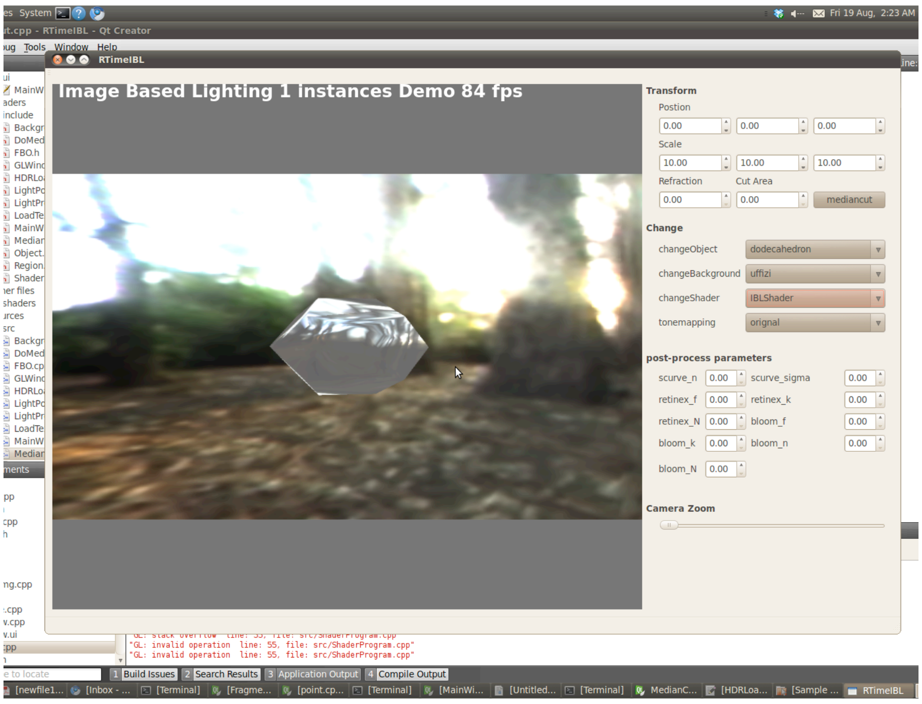
bloom_N 0.00

```
"GL: stack overflow  line: 55, file: src/ShaderProgram.cpp"
"GL: invalid operation  line: 55, file: src/ShaderProgram.cpp"
```

Figure 14 diffuse result

4.5.1.3 Reflective and diffuse shader:

This shader combines the specular and diffuse shaders above together.



Transform

Postion: 0.00, 0.00, 0.00

Scale: 10.00, 10.00, 10.00

Refraction: 0.00 Cut Area: 0.00 mediancut

Change

changeObject: dodecahedron

changeBackground: uffizi

changeShader: IBLShader

tonemapping: orignal

post-process parameters

scurve_n: 0.00 scurve_sigma: 0.00

retinex_f: 0.00 retinex_k: 0.00

retinex_N: 0.00 bloom_f: 0.00

bloom_k: 0.00 bloom_n: 0.00

bloom_N: 0.00

Camera Zoom

```
"GL: stack overflow  line: 55, file: src/ShaderProgram.cpp"
"GL: invalid operation  line: 55, file: src/ShaderProgram.cpp"
"GL: invalid operation  line: 55, file: src/ShaderProgram.cpp"
```

Figure 15 diffuse and reflect result

4.5.1.4 Fresnel equations and Fresnel term:

Flowing shaders have been using Fresnel equations.

The Fresnel equations were deduced by Augustin-Jean Frensel. It describes the behavior of the light moving between two different media of different refractive indices.

When the light transmit from one media to another, both the reflection and refraction may happen. The model can be described like this:

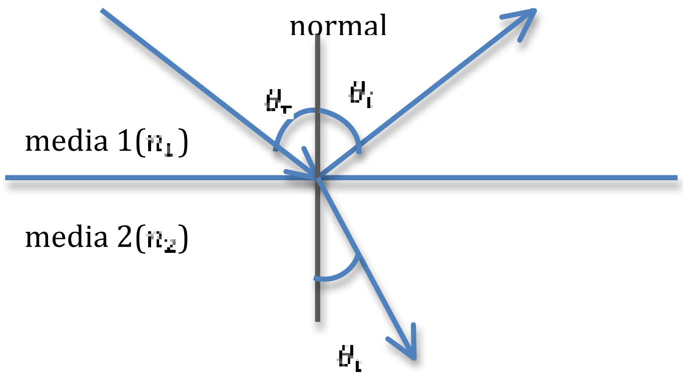


Figure 16 reflection and refraction model

As the figure shows above, we describe the angle between income light and normal as θ_r, and describe the angle between reflection light and normal as θ_i, the angle between refraction light and normal as θ_t, the refraction indices of the two media are n_1 and n_2. The relationship between incidence angles and the reflection angles is given by the law of reflection:

$$\theta_i = \theta_r;$$

Equation 5 incidence angle and reflection angle

And Snell's law defines the relationship between the incidence angles, refraction angles and two medias refractive indices:

$$\frac{\sin(\theta_i)}{\sin(\theta_t)} = \frac{n_2}{n_1}$$

Equation 6 incidence angle and reflection angle

In physics, the Fresnel equation is very complicated, so we just need to use a simple version to achieve the final effect. In this project, three values have been used to represent the Fresnel equation, they are fresnelScale, fresnelBias and fresnelPower. And the simple equation would look as below:

fresnelTerm = fresnelbias + fresnelscale * pow(1.0f+dot(-eyedirection, Normal), frenelPower); [17,18,19]

(where fresnelBias=1-fresnelscale)

Equation 7 fresnel equation

4.5.1.5 Refraction:

This shader has been using the fresnel term created as above to simulate the refraction of light. The main solution is to use two texture map, one is for reflection, the other is for refraction, finally use the fresnel term as the weight while doing the interpolation between this two textures. There is one important thing should be noticed here: the normal of refraction and reflection are opposite to each other.

$$Normal_{reflection} = -Normal_{refraction}$$

Equation 8 normal of reflection and refraction

And in GLSL, it provide a function to calculate the refraction direction, so

refract (-eyedirection, normal, ratio);

List 4 refract function used in GLSL

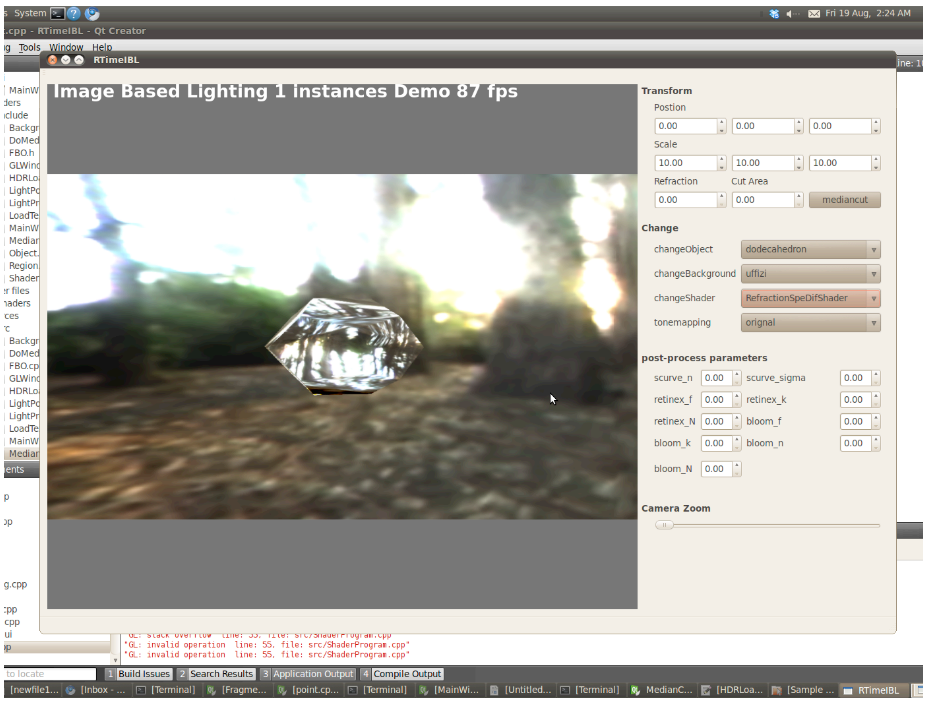
is used in vertex shader and in this case, ratio is the indices of refraction

In the fragment shader,

gl_FragColor=mix(vec4 (refrCol,1.0), vec4(specCol, 1.0), Ratio);

List 5 use ratio which is generated by frensel to get the right pixel in the environment map.

is used to get the final pixel colour.



Transform
Postion 0.00 0.00 0.00
Scale 10.00 10.00 10.00
Refraction 0.00
Cut Area 0.00
mediancut

Change
changeObject dodecahedron
changeBackground uffizi
changeShader RefractionSpeDifShader
tonemapping orignal

post-process parameters
scurve_n 0.00 scurve_sigma 0.00
retinex_f 0.00 retinex_k 0.00
retinex_N 0.00 bloom_f 0.00
bloom_k 0.00 bloom_n 0.00
bloom_N 0.00

Camera Zoom

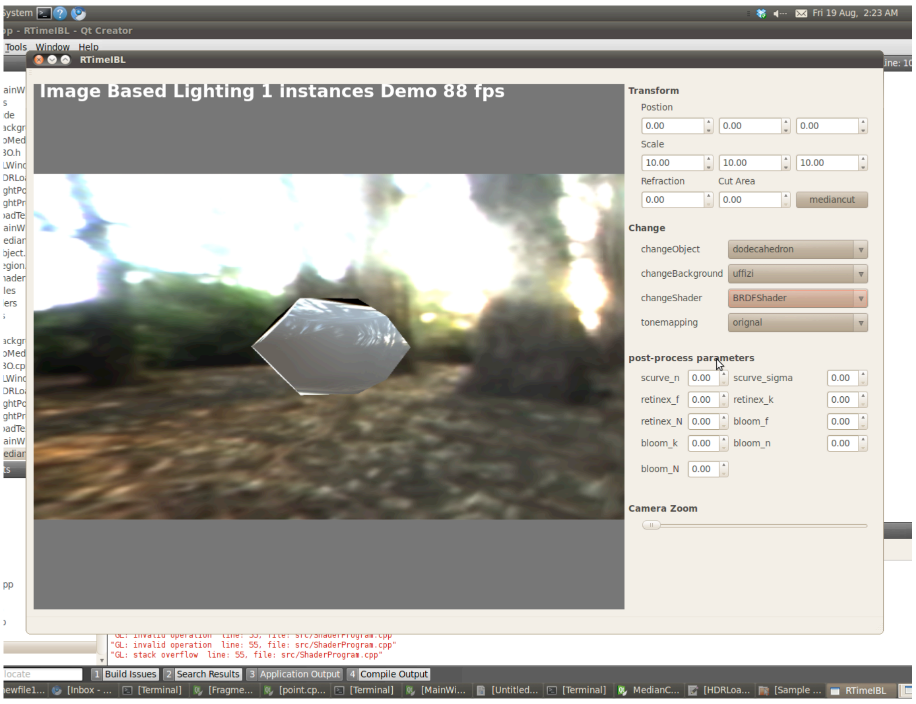
```
"GL: invalid operation  line: 55, file: src/ShaderProgram.cpp"
"GL: invalid operation  line: 55, file: src/ShaderProgram.cpp"
```

Figure 17 reflection and refraction model

4.5.1.6 Reflection and diffuse using fresnel:

This part is almost the same as the one described above.

The difference is to use fresnel as the weight to interpolation between the diffuse texture and the specular texture.



Transform

Postion		
0.00	0.00	0.00
Scale		
10.00	10.00	10.00
Refraction	Cut Area	
0.00	0.00	mediancut

Change

changeObject	dodecahedron
changeBackground	uffizi
changeShader	BRDFShader
tonemapping	orignal

post-process parameters

scurve_n	0.00	scurve_sigma	0.00
retinex_f	0.00	retinex_k	0.00
retinex_N	0.00	bloom_f	0.00
bloom_k	0.00	bloom_n	0.00
bloom_N	0.00		

Camera Zoom

```
"GL: invalid operation  line: 55, file: src/ShaderProgram.cpp"
"GL: stack overflow  line: 55, file: src/ShaderProgram.cpp"
```

Figure 18 Reflection and diffuse using fresnel

4.5.1.7 Chromatic distortion:

Chromatic distortion is a kind of aberration, which is a failure of the lens to focus all the colours on the same point. It happens because different wavelength of light has different refractive index. And the refractive index is inversely proportional to the wavelength. So it can be implemented easily, dealing with the RGB channel separately.

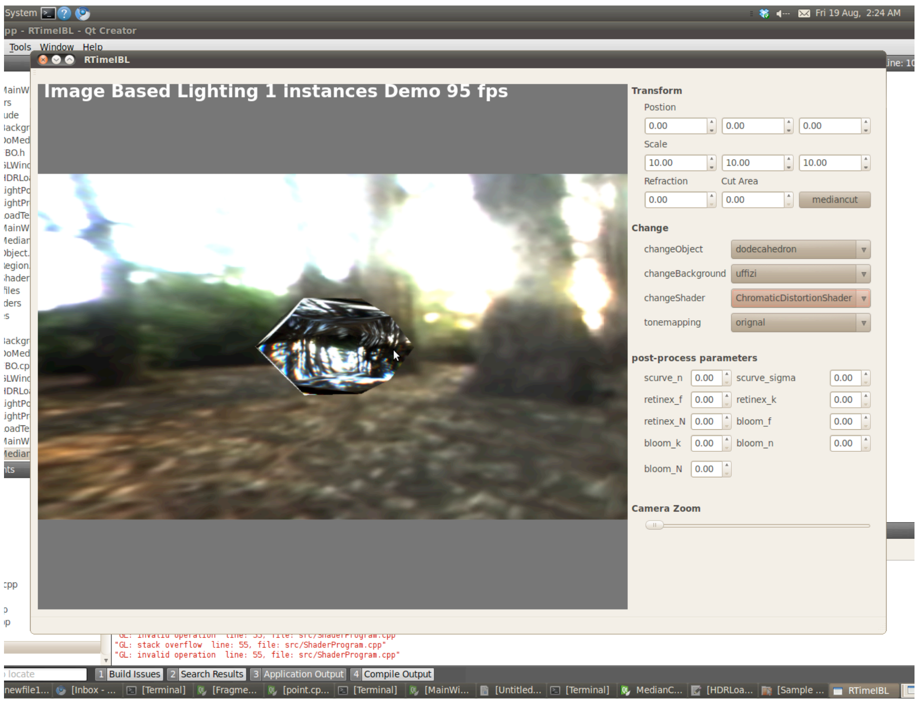
Here the refraction indices *Eta* should be defined as a vector, which contains the different value of different colour channel refraction indices. Take red channel as an example:

```
RefractR=vec3(refract(EyeDir, cNormal, Era.r));
```

List 6 deal with refraction direction of the red channel in .vs

```
refrCol.r = vec3(textureCube(Map,RefractR)).r
```

List 7 find red channel value the proper pixel in .fs



Transform
Postion 0.00 0.00 0.00
Scale 10.00 10.00 10.00
Refraction 0.00 Cut Area 0.00

Change
changeObject dodecahedron
changeBackground uffizi
changeShader ChromaticDistortionShader
tonemapping orignal

post-process parameters
scurve_n 0.00 scurve_sigma 0.00
retinex_f 0.00 retinex_k 0.00
retinex_N 0.00 bloom_f 0.00
bloom_k 0.00 bloom_n 0.00
bloom_N 0.00

```
"GL: stack overflow  line: 55, file: src/ShaderProgram.cpp"
"GL: invalid operation  line: 55, file: src/ShaderProgram.cpp"
```

Figure 19 Chromatic distortion

4.5.2 Tonemapping

HDR colours are limited to display on the normal device screen; in this case, tonemapping is used to map HDR images to LDR images, so that the image can be displayed on the screen. In this project, the HDR image has already converted to the LDR when it is loaded. There are two methods to do tonemapping, one is s-curve which is the global tonemapping, the other is Rahman Retinex which is local tonemapping

4.5.2.1 S-curve (global tonemapping)

S-curve is the global tonemapping method. It just uses one nonlinear curve to affect each pixel of the picture.

The represent curve can be described as following equation:

$$\frac{R}{R_{max}} = \frac{I^n}{(I^n + sigma^n)}$$

Equation 9 S-Curve equation

In this equation, R is the response and the value is between 0 and R_{max}. I is the intensity of the light, n is a sensitivity exponent which generally between 0.7 and 1.0 sigma is a constant, and it controls the x axis of the curve.

In the project, S-curve has affect each pixel chanel, and n is the pixel size. So in the fragment shader, the output colour can be implement as follows:

```
gl_FragColor.r=pow(colour.r, n)/(pow(colour.r,n)+pow(sigma,n));
gl_FragColor.r=pow(colour.g, n)/(pow(colour.g,n)+pow(sigma,n));
gl_FragColor.r=pow(colour.b, n)/(pow(colour.b,n)+pow(sigma,n));
```

List 8 implement of scurve in different channels

4.5.2.2 Rahman Retinex(local tonemapping)

Rahman Retinaex is a local tonemapping method. In this method, the local pixel will be computed according to the pixel itself as well as the pixel, which is surrounding to it. This method will use a blur filter to compress the pixels. And it will act on each channel of the pixel seperately. The equation this project used is as follows:

$$w_n = \frac{(N - n + 1)^f}{\sum_{m=0}^{N}(N - m + 1)^f}$$

$$I_d(x, y) = e^{\sum_{n=0}^{N} w_n \cdot (\log(I_W(x,y)) - k \cdot \log(I_{W,n}^{blur}(x,y)))}$$

Equation 10 the equation of Rahman Retinaex

In this equation, f controls the weight of each scales, and decides the blurred image which plays a important role, generally from -0.3 to 0.3 in steps of 0.1. The value k represents the relative weight of the blurred image.

4.5.3 Blooming

After doing those tone mappings, it is time to add some special effects. A bloom effect is used to glow the very bright areas in the image. This can display an image, which is bright than it actually is. Bloom effect simulates effects of human eyes. When part of environment was viewed by our eyes, it is always brighter than rest of the area. Also, there is a glow effect around the area. So it can make the image much more realistic. This effect can be implemented using a filter, which falls off quickly. The bright pixels can contribute energy to the other pixels around them. The filter falls off quickly so that it just make the extremely bright pixels contribute to other pixels in a sudden without changing the other area. Finally the different blurred texture is combined together and adds to the original one in order to get the bloom effect.

In this project, in order to simplify the process, just one texture is used. In bloom shader, the filter used is similar as the one used in the Rahman Retinex shader. And use smoothstep function to calculate the bright parts of the image. This post-process is based on the s-curve.

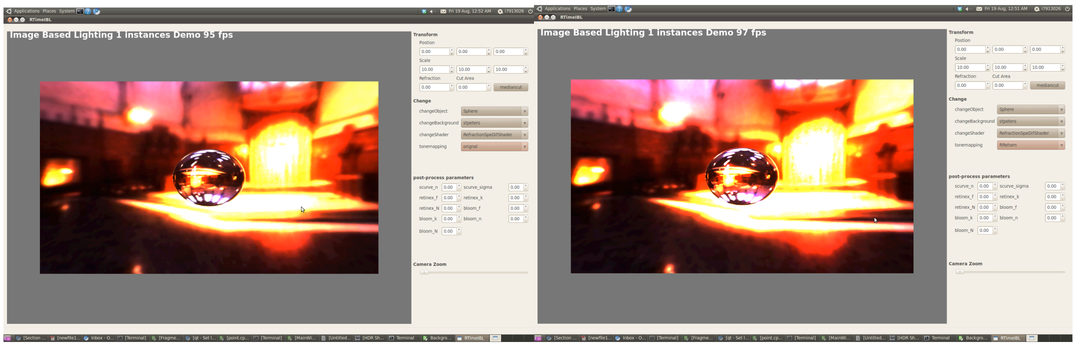


Image Based Lighting 1 instances Demo 95 fps

Fri 19 Aug, 12:52 AM i7913026

Transform
Postion 0.00 0.00 0.00
Scale 10.00 10.00 10.00
Refraction 0.00 Cut Area 0.00 mediancut
Change
changeObject: Sphere
changeBackground: stpeters
changeShader: RefractionSpeDifShader
tonemapping: orignal
post-process parameters
scurve_n 0.00 scurve_sigma 0.00
retinex_f 0.00 retinex_k 0.00
retinex_N 0.00 bloom_f 0.00
bloom_k 0.00 bloom_n 0.00
bloom_N 0.00
Camera Zoom

Image Based Lighting 1 instances Demo 97 fps

Fri 19 Aug, 12:51 AM i7913026

Transform
Postion 0.00 0.00 0.00
Scale 10.00 10.00 10.00
Refraction 0.00 Cut Area 0.00 mediancut
Change
changeObject: Sphere
changeBackground: stpeters
changeShader: RefractionSpeDifShader
tonemapping: RReham
post-process parameters
scurve_n 0.00 scurve_sigma 0.00
retinex_f 0.00 retinex_k 0.00
retinex_N 0.00 bloom_f 0.00
bloom_k 0.00 bloom_n 0.00
bloom_N 0.00
Camera Zoom

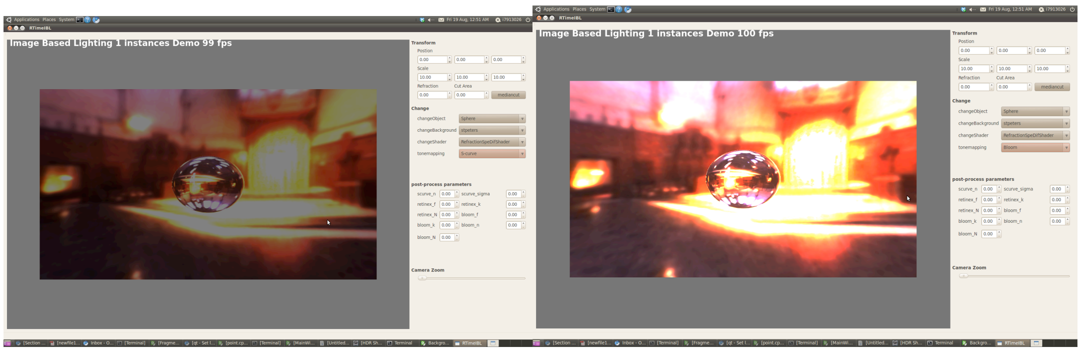


Image Based Lighting 1 instances Demo 99 fps

Transform
Postion 0.00 0.00 0.00
Scale 10.00 10.00 10.00
Refraction 0.00 Cut Area 0.00 mediancut

Change
changeObject Sphere
changeBackground stpeters
changeShader RefractionSpeDifShader
tonemapping S-curve

post-process parameters
scurve_n 0.00 scurve_sigma 0.00
retinex_f 0.00 retinex_k 0.00
retinex_N 0.00 bloom_f 0.00
bloom_k 0.00 bloom_n 0.00
bloom_N 0.00

Camera Zoom

Image Based Lighting 1 instances Demo 100 fps

Transform
Postion 0.00 0.00 0.00
Scale 10.00 10.00 10.00
Refraction 0.00 Cut Area 0.00 mediancut

Change
changeObject Sphere
changeBackground stpeters
changeShader RefractionSpeDifShader
tonemapping Bloom

post-process parameters
scurve_n 0.00 scurve_sigma 0.00
retinex_f 0.00 retinex_k 0.00
retinex_N 0.00 bloom_f 0.00
bloom_k 0.00 bloom_n 0.00
bloom_N 0.00

Camera Zoom

Figure 20 orginal shader, Rahman Retinex shader, scurve shader, bloom shader

4.6 frame buffer object

In this project, frame buffer object is used to render the frame to the texture. The pipeline is as follows:

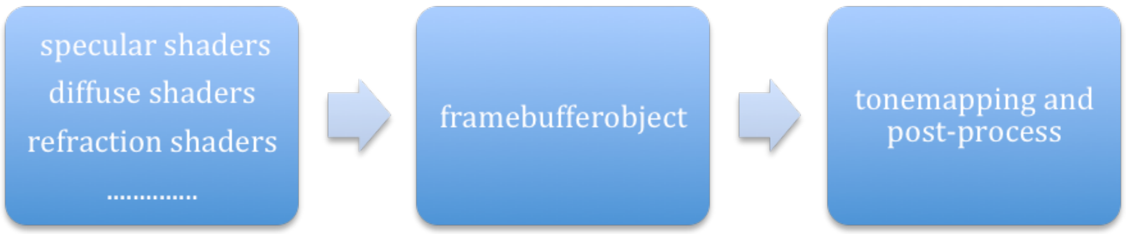


Figure 21 tonemapping using frame buffer object

Use IBL shaders to render the first pass and put them into framebufferobject, and then use tonemapping and post-process shaders to generate the first pass. This will be output to the screen finally. When using bloom shader, the normal method is to render the bright area of the image into framebufferobject and use filter to blur the pass. In this project, the two steps are combined together. However, in this place, the frame buffer object was still used because bloom effect should use after the scurve shader.

4.7 Interaction guide:

Users can choose different primitives and change the value of position and scale. Users can also change different shaders to observe different materials. Different tone mapping method and post process can be choose as well.

4.8 Result:

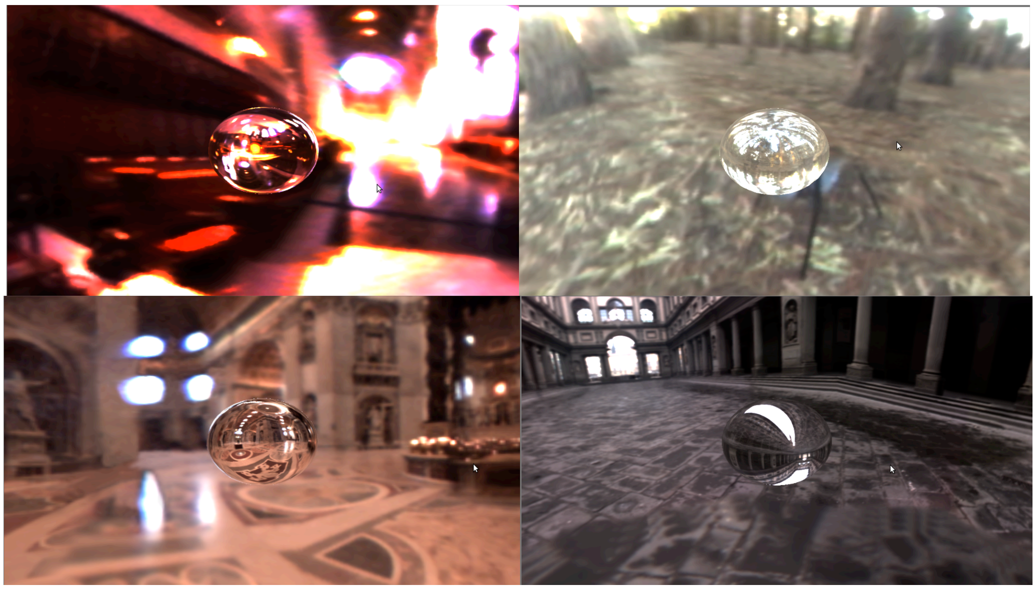


Figure 22 four different scence same object

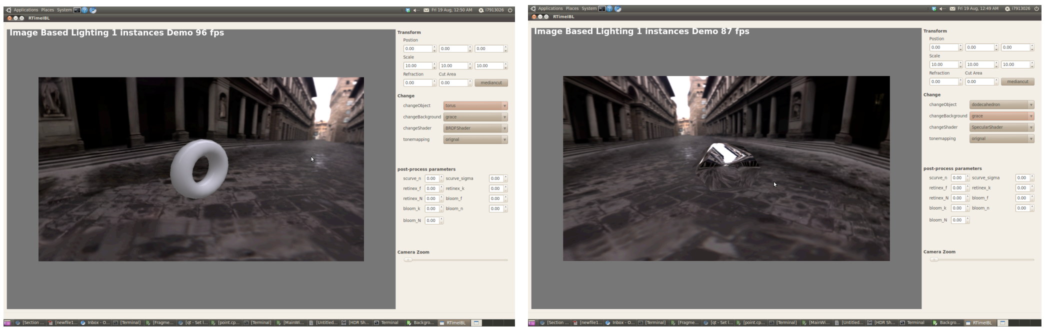


Image Based Lighting 1 instances Demo 96 fps

Transform
Postion 0.00 0.00 0.00
Scale 10.00 10.00 10.00
Refraction 0.00 Cut Area 0.00 mediancut

Change
changeObject torus
changeBackground grace
changeShader BRDFShader
tonemapping orignal

post-process parameters
scurve_n 0.00 scurve_sigma 0.00
retinex_f 0.00 retinex_k 0.00
retinex_N 0.00 bloom_f 0.00
bloom_k 0.00 bloom_n 0.00
bloom_N 0.00

Camera Zoom

Image Based Lighting 1 instances Demo 87 fps

Transform
Postion 0.00 0.00 0.00
Scale 10.00 10.00 10.00
Refraction 0.00 Cut Area 0.00 mediancut

Change
changeObject dodecahedron
changeBackground grace
changeShader SpecularShader
tonemapping orignal

post-process parameters
scurve_n 0.00 scurve_sigma 0.00
retinex_f 0.00 retinex_k 0.00
retinex_N 0.00 bloom_f 0.00
bloom_k 0.00 bloom_n 0.00
bloom_N 0.00

Camera Zoom

Figure 23 other primitives

Chapter 5 Conclusion

5.1 Achievement overview:

In this project, several things have been achieved.

First, this project use the *.hdr* format directly, which has lots of resources on Debevec's website. This project has implemented loading of *.hdr* format. And use mediancut algorithm to divide the image into proper religions, and finally determine the vitual lights position and colour.

This project uses the cross map directly rather than use six different side images. Cube mapping performed very well for both the skybox and the objects.

Different image based lighting shaders have been implemented. Some of them are based on fresnel equation to get the good looking result.

Tone mapping and post-process shaders such as s-curve, bloom, Rahman Retinex has been performed and it allows users to adjust the relative parameters to get different outputs.

Users can zoom in and out the camera to observe the scene. Objects can be moved to different places to get the different lighting results.

5.2 Improvements:

There indeed some parts of the application can be improved:

Now, median cut algorithm can manage the regions of the image, and also can get the lights positions and colours but some errors happen prevent the light to cast the shadows. And when use bloom shader, and choose the primitive as teapot or dodecahedron, some black square would appear around the object, it might be caused by the blur filter, and will be fixed in the furture.

5.2.1 Shadows:

Relatively, the implement of shadow is much more complicated. But shadow plays a very important part of the lighting. This application will use lights to cast shadows, based on the median cut method, with the technique of global illumination.

5.2.2 Interreflection between objects:

The program is just carrying out the lighting of single object. In order to get more attractive result, light spreading among objects will be implemented in the future work.

5.2.3 Deferred lighting:

This project took a lot of time to combine the *hdr* format with the program and also paid much more attention to the HDR and shader writing, ignoring the very important lighting technique, which called deferred lighting. Deferred lighting can be used to scale with more lights in order to affect the scene. The information can be saved to texture instead of doing the actual lighting calculation. The lighting will only be calculated as long as the geometry is rendered.

5.2.4 The usage of images:

In the project, two maps are used to represent the diffuse light and the specular light. And diffuse map is generated from specular map using HDRshop. Although this will improve the speed of computation, it is still important to do it in the program. Because if the diffuse map was created by the program, users can import any other random HDR image as environment map.

5.2.5 More effects:

More effect can be added when doing the post-process, such as depth of field, glow, glare and so on.

To conclude, according to this project, many techniques have been learned. But there are many interesting parts of image-based lighting are ignored due to the time, such as the implement of the shadow, the global illumination using IBL. This will be accomplished in the future.

Chapter 6 References

[1] Blinn J., Newll M., 1976,Texture and reflection in computer generated images. Communication of the ACM 19(10): 542-547.

[2] Debevec P., Debevec's website

Available from http://ict.debevec.org/~debevec/ [Accessed 19 August 2011]

[3] Reinhard E., Ward G., Pattanaik S., Debevec P., 2005, High dynamic Range Image, The Morgan Kaufmann series in compyter graphics, San Francisco.

[4] Debevec P., Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In SIGGRAPH 98, July 1998.

[5] Debevec P., 2002, Image---based Lighting, *IEEE Computer Graphics and Applications,* P26---34

[6] Yu Y.,Z., Debevec P., Malik J., Hawkins T., 1999,Inverse Global illumination: Recovering Reflectancd Models of Real Scenes from Photographs, SIGGRAPH 99, 26th conference on Computer graphics and interactive techniques.

[7] Agarwal S., Ramamoorthi R., Belongie S., Jensen,W.H., Stuctured Importance Sampling of Environment Maps, Siggraph 2003.

[8] Hensley J., Scheuermann T., Coombe G., Singh M., and Lastra A., 2005, Fast Summed---Area Table Generation and its Applications, EuroGraphics 2005 (v24)

[9] Debevec P., 2005, A Median Cut Algorithm for Light Probe Sampling *(SIGGRAPH Poster 2005)*

[10] Viriyothai, K. Debevee P., 2009,Variance Minimization light probe sampling, SIGGRAPH 2009, New Orleans, Louisiana, August 3–7, 2009.

[11] GPU Gems, NVDIA Developer zone, [Accessed 19 August 2011]

[12] Agrawala M., Ramamoorthi R., Heirich A., Moll L., 2000,Efficient Image-Based Methods for Rendering softShadows.

[13] Alexander Keller. Instant radiosity. In Computer Graphics (ACM SIGGRAPH '97 Proceedings), volume 31, pages 49– 56,1997.

[14] George Drettakis and François X. Sillion. Interactive update of global illumination using a line-space hierarchy. In Pro- ceedings of SIGGRAPH 97,

Computer Graphics Proceedings, Annual Conference Series, pages 57–64, Los Angeles, Cali- fornia, August 1997.

[15]Xavier Granier and George Drettakis. Incremental updates for rapid glossy global illumination. *Computer Graphics Forum,* 20(3):268–277, 2001.

[16] HDRLoader, http://www.graphics.cornell.edu/~bjw/rgbe.html, [Accessed 19 August 2011]

[17] split the cubemap code
http://etudiant.univ-mlv.fr/~rdamon/IMAC3/openGL/td6/src_exo2

[18] Gamedev.net, (http://www.gamedev.net/topic/427702-diffuse--fresnel-term/) [Accessed 19 August 2011]

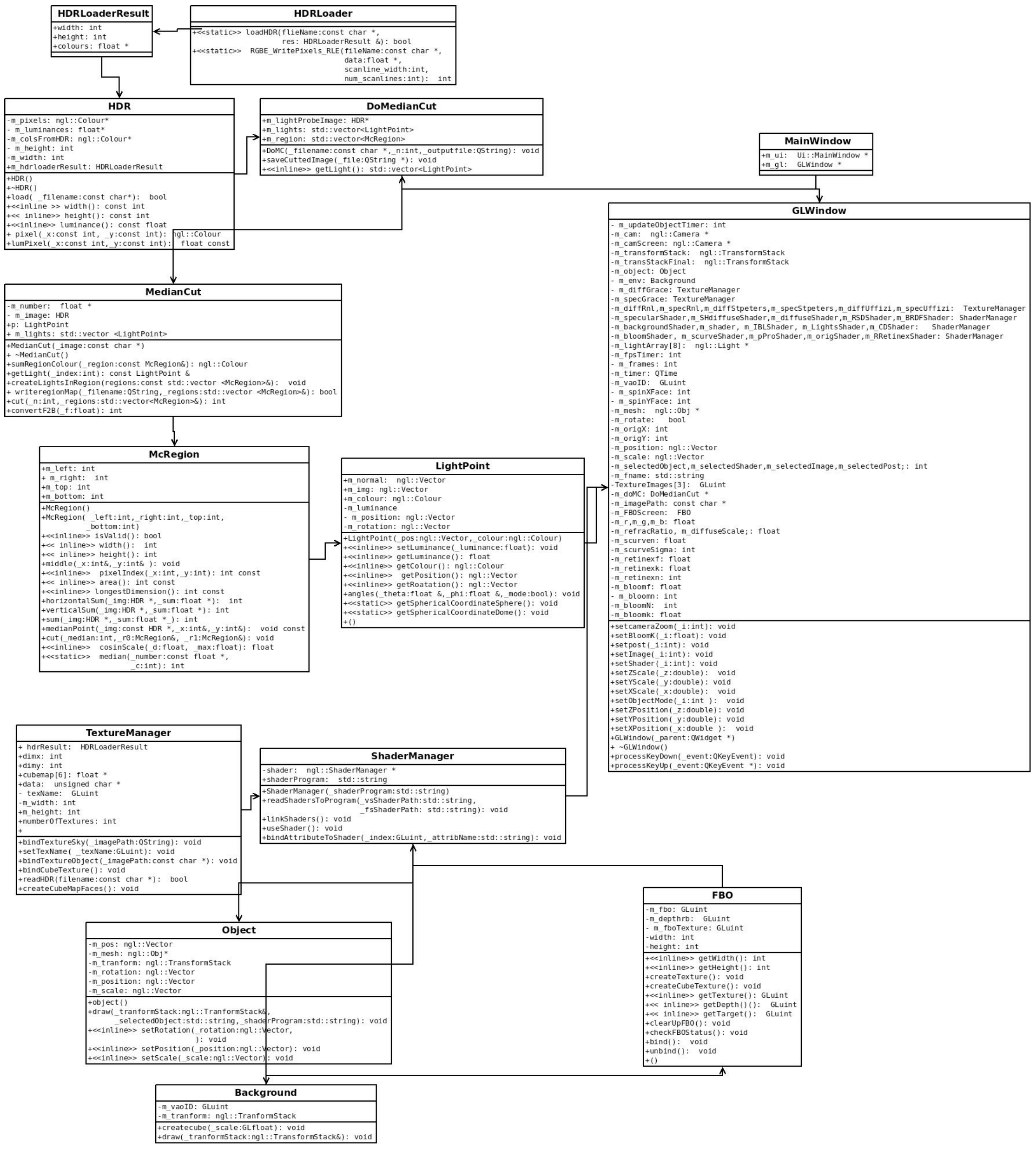
[19]nullah blog (http://blog.csdn.net/nullah/article/details/5553668) [Accessed 19 August 2011]

[20]Sousa T., Crytek GPU Gems 2 ,Generic Refraction Simulation
(http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter19.html) [Accessed 19 August 2011]

[21] yangdelong, Fresnel and Chromatic aberration shader
http://blog.csdn.net/yangdelong/article/details/4427782 [Accessed 19 August 2011]

[22] HDRShop, Available from: http://gl.ict.usc.edu/HDRShop/ [Accessed 19 August 2011]

Chapter 7 Appendices



```
HDRLoaderResult
+width: int
+height: int
+colours: float *
```

```
HDRLoader
+<<static>> loadHDR(flieName:const char *,
                    res: HDRLoaderResult &): bool
+<<static>>  RGBE_WritePixels_RLE(fileName:const char *,
                                  data:float *,
                                  scanline_width:int,
                                  num_scanlines:int):  int
```

```
HDR
-m_pixels: ngl::Colour*
- m_luminances: float*
-m_colsFromHDR: ngl::Colour*
- m_height: int
-m_width: int
+m_hdrloaderResult: HDRLoaderResult
+HDR()
+~HDR()
+load( _filename:const char*):  bool
+<<inline >> width(): const int
+<< inline>> height(): const int
+<<inline>> luminance(): const float
+ pixel(_x:const int, _y:const int): ngl::Colour
+lumPixel(_x:const int,_y:const int):  float const
```

```
DoMedianCut
+m_lightProbeImage: HDR*
+m_lights: std::vector<LightPoint>
+m_region: std::vector<McRegion>
+DoMC(_filename:const char *,_n:int,_outputfile:QString): void
+saveCuttedImage(_file:QString *): void
+<<inline>> getLight(): std::vector<LightPoint>
```

```
MainWindow
+m_ui:  Ui::MainWindow *
+m_gl:  GLWindow *
```

```
GLWindow
- m_updateObjectTimer: int
-m_cam:  ngl::Camera *
-m_camScreen: ngl::Camera *
-m_transformStack:  ngl::TransformStack
-m_transStackFinal:  ngl::TransformStack
-m_object: Object
- m_env: Background
- m_diffGrace: TextureManager
-m_specGrace: TextureManager
-m_diffRnl,m_specRnl,m_diffStpeters,m_specStpeters,m_diffUffizi,m_specUffizi:  TextureManager
-m_specularShader,m_SHdiffuseShader,m_diffuseShader,m_RSDShader,m_BRDFShader: ShaderManager
-m_backgroundShader,m_shader, m_IBLShader, m_LightsShader,m_CDShader:   ShaderManager
-m_bloomShader, m_scurveShader,m_pProShader,m_origShader,m_RRetinexShader: ShaderManager
-m_lightArray[8]:  ngl::Light *
-m_fpsTimer: int
- m_frames: int
-m_timer: QTime
-m_vaoID:  GLuint
- m_spinXFace: int
- m_spinYFace: int
-m_mesh:  ngl::Obj *
-m_rotate:   bool
-m_origX: int
-m_origY: int
-m_position: ngl::Vector
-m_scale: ngl::Vector
-m_selectedObject,m_selectedShader,m_selectedImage,m_selectedPost;: int
-m_fname: std::string
-TextureImages[3]:  GLuint
-m_doMC: DoMedianCut *
-m_imagePath: const char *
-m_FBOScreen:  FBO
-m_r,m_g,m_b: float
-m_refracRatio, m_diffuseScale;: float
-m_scurven: float
-m_scurveSigma: int
-m_retinexf: float
-m_retinexk: float
-m_retinexn: int
-m_bloomf: float
- m_bloomn: int
-m_bloomN:  int
-m_bloomk: float
+setcameraZoom(_i:int): void
+setBloomK(_i:float): void
+setpost(_i:int): void
+setImage(_i:int): void
+setShader(_i:int): void
+setZScale(_z:double):  void
+setYScale(_y:double): void
+setXScale(_x:double):  void
+setObjectMode(_i:int ):  void
+setZPosition(_z:double): void
+setYPosition(_y:double): void
+setXPosition(_x:double ):  void
+GLWindow(_parent:QWidget *)
+ ~GLWindow()
+processKeyDown(_event:QKeyEvent): void
+processKeyUp(_event:QKeyEvent *): void
```

```
MedianCut
-m_number:  float *
- m_image: HDR
+p: LightPoint
+ m_lights: std::vector <LightPoint>
+MedianCut(_image:const char *)
+ ~MedianCut()
+sumRegionColour(_region:const McRegion&): ngl::Colour
+getLight(_index:int): const LightPoint &
+createLightsInRegion(regions:const std::vector <McRegion>&):  void
+ writeregionMap(_filename:QString,_regions:std::vector <McRegion>&): bool
+cut(_n:int,_regions:std::vector<McRegion>&): int
+convertF2B(_f:float): int
```

```
McRegion
+m_left: int
+ m_right:  int
+m_top: int
+m_bottom: int
+McRegion()
+McRegion( _left:int,_right:int,_top:int,
          _bottom:int)
+<<inline>> isValid(): bool
+<< inline>> width():  int
+<< inline>> height(): int
+middle(_x:int&,_y:int& ): void
+<<inline>>  pixelIndex(_x:int,_y:int): int const
+<< inline>> area(): int const
+<<inline>> longestDimension(): int const
+horizontalSum(_img:HDR *,_sum:float *):  int
+verticalSum(_img:HDR *,_sum:float *): int
+sum(_img:HDR *,_sum:float *_): int
+medianPoint(_img:const HDR *,_x:int&,_y:int&):  void const
+cut(_median:int,_r0:McRegion&, _r1:McRegion&): void
+<<inline>>  cosinScale(_d:float, _max:float): float
+<<static>>  median(_number:const float *,
                    _c:int): int
```

```
LightPoint
+m_normal:  ngl::Vector
+m_img: ngl::Vector
+m_colour: ngl::Colour
-m_luminance
- m_position: ngl::Vector
-m_rotation: ngl::Vector
+LightPoint(_pos:ngl::Vector,_colour:ngl::Colour)
+<<inline>> setLuminance(_luminance:float): void
+<<inline>> getLuminance(): float
+<<inline>> getColour(): ngl::Colour
+<<inline>>  getPosition(): ngl::Vector
+<<inline>> getRoatation(): ngl::Vector
+angles(_theta:float &,_phi:float &,_mode:bool): void
+<<static>> getSphericalCoordinateSphere(): void
+<<static>> getSphericalCoordinateDome(): void
+()
```

```
TextureManager
+ hdrResult:  HDRLoaderResult
+dimx: int
+dimy: int
+cubemap[6]: float *
+data:  unsigned char *
- texName:  GLuint
-m_width: int
+m_height: int
+numberOfTextures: int
+
+bindTextureSky(_imagePath:QString): void
+setTexName( _texName:GLuint): void
+bindTextureObject(_imagePath:const char *): void
+bindCubeTexture(): void
+readHDR(filename:const char *):  bool
+createCubeMapFaces(): void
```

```
ShaderManager
-shader:  ngl::ShaderManager *
+shaderProgram:  std::string
+ShaderManager(_shaderProgram:std::string)
+readShadersToProgram(_vsShaderPath:std::string,
                      _fsShaderPath: std::string): void
+linkShaders(): void
+useShader(): void
+bindAttributeToShader(_index:GLuint,_attribName:std::string): void
```

```
Object
-m_pos: ngl::Vector
-m_mesh: ngl::Obj*
-m_tranform: ngl::TransformStack
-m_rotation: ngl::Vector
-m_position: ngl::Vector
-m_scale: ngl::Vector
+object()
+draw(_tranformStack:ngl::TranformStack&,
      _selectedObject:std::string,_shaderProgram:std::string): void
+<<inline>> setRotation(_rotation:ngl::Vector,
                        ): void
+<<inline>> setPosition(_position:ngl::Vector): void
+<<inline>> setScale(_scale:ngl::Vector): void
```

```
FBO
-m_fbo: GLuint
-m_depthrb:  GLuint
- m_fboTexture: GLuint
-width: int
-height: int
+<<inline>> getWidth(): int
+<<inline>> getHeight(): int
+createTexture(): void
+createCubeTexture(): void
+<<inline>> getTexture(): GLuint
+<< inline>> getDepth()():  GLuint
+<< inline>> getTarget():  GLuint
+clearUpFBO(): void
+checkFBOStatus(): void
+bind():  void
+unbind():  void
+()
```

```
Background
-m_vaoID: GLuint
-m_tranform: ngl::TranformStack
+createcube(_scale:GLfloat): void
+draw(_tranformStack:ngl::TransformStack&): void
```