Disco Room

Shader Programing Project

Made by: Philipp Petzold, Gunnar Busch and Hadeer Khalifa

Project Goal

The Goal of this project was to develop shaders suitable for a disco room 3D model. The main idea was to control the real-time shaders with a sound visualizer that would produce bass values which will control the shaders’ parameters ex :(texture, diffuse light…etc.).

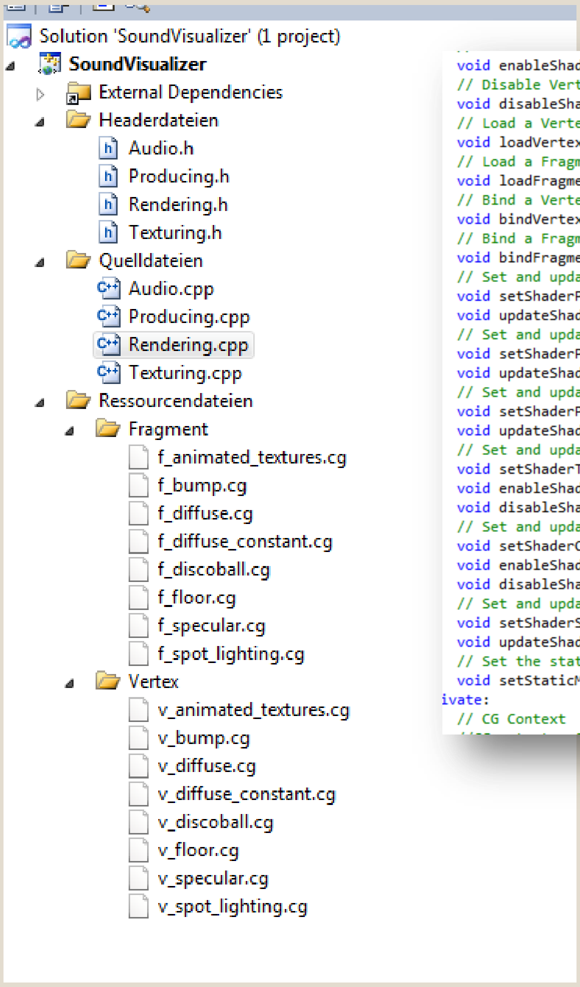
Disco Room

Techniques & Algorithms:

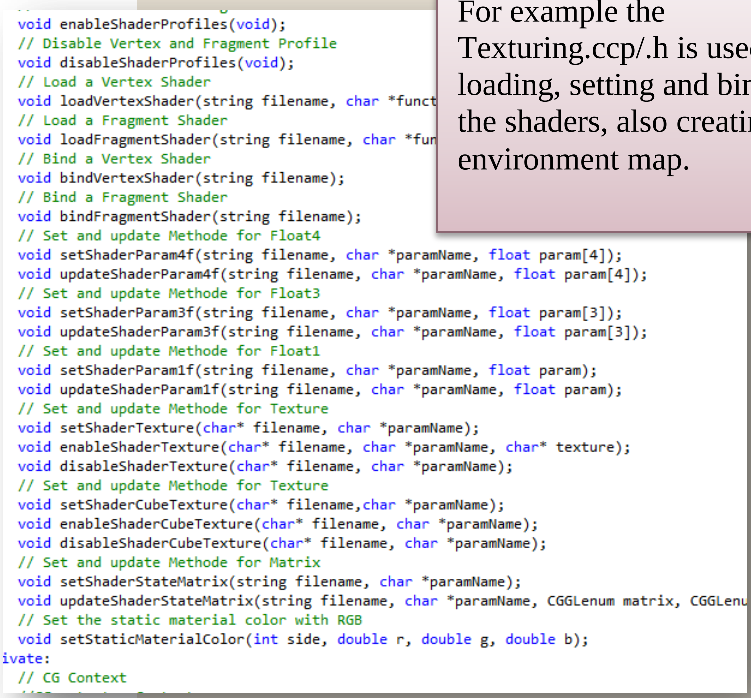
The programs contain two parts the openGL cpp files that contains the 3D modeling and scene setting with the shaders' initializations. The other part of the project is the CG shader, where almost every object has its own shader.

OPENGL

Since the scenes have a lot of objects and several shaders were used, therefore it was decided to make our own engine for the openGL/ shaders, were it will be easier to implement and use.



```
Solution 'SoundVisualizer' (1 project)
  SoundVisualizer
    External Dependencies
    Headerdateien
      Audio.h
      Producing.h
      Rendering.h
      Texturing.h
    Quelldateien
      Audio.cpp
      Producing.cpp
      Rendering.cpp
      Texturing.cpp
    Ressourcendateien
      Fragment
        f_animated_textures.cg
        f_bump.cg
        f_diffuse.cg
        f_diffuse_constant.cg
        f_discoball.cg
        f_floor.cg
        f_specular.cg
        f_spot_lighting.cg
      Vertex
        v_animated_textures.cg
        v_bump.cg
        v_diffuse.cg
        v_diffuse_constant.cg
        v_discoball.cg
        v_floor.cg
        v_specular.cg
        v_spot_lighting.cg
```



```
 void enableShaderProfiles(void);
 // Disable Vertex and Fragment Profile
 void disableShaderProfiles(void);
 // Load a Vertex Shader
 void loadVertexShader(string filename, char *funct
 // Load a Fragment Shader
 void loadFragmentShader(string filename, char *fun
 // Bind a Vertex Shader
 void bindVertexShader(string filename);
 // Bind a Fragment Shader
 void bindFragmentShader(string filename);
 // Set and update Methode for Float4
 void setShaderParam4f(string filename, char *paramName, float param[4]);
 void updateShaderParam4f(string filename, char *paramName, float param[4]);
 // Set and update Methode for Float3
 void setShaderParam3f(string filename, char *paramName, float param[3]);
 void updateShaderParam3f(string filename, char *paramName, float param[3]);
 // Set and update Methode for Float1
 void setShaderParam1f(string filename, char *paramName, float param);
 void updateShaderParam1f(string filename, char *paramName, float param);
 // Set and update Methode for Texture
 void setShaderTexture(char* filename, char *paramName);
 void enableShaderTexture(char* filename, char *paramName, char* texture);
 void disableShaderTexture(char* filename, char *paramName);
 // Set and update Methode for Texture
 void setShaderCubeTexture(char* filename,char *paramName);
 void enableShaderCubeTexture(char* filename, char *paramName);
 void disableShaderCubeTexture(char* filename, char *paramName);
 // Set and update Methode for Matrix
 void setShaderStateMatrix(string filename, char *paramName);
 void updateShaderStateMatrix(string filename, char *paramName, CGGLenum matrix, CGGLenu
 // Set the static material color with RGB
 void setStaticMaterialColor(int side, double r, double g, double b);
ivate:
 // CG Context
```

For example the Texturing.ccp/.h is used for loading, setting and binding the shaders, also creating environment map.

Producing.h/.cpp

Is the CPP file responsible for creating the gl objects.

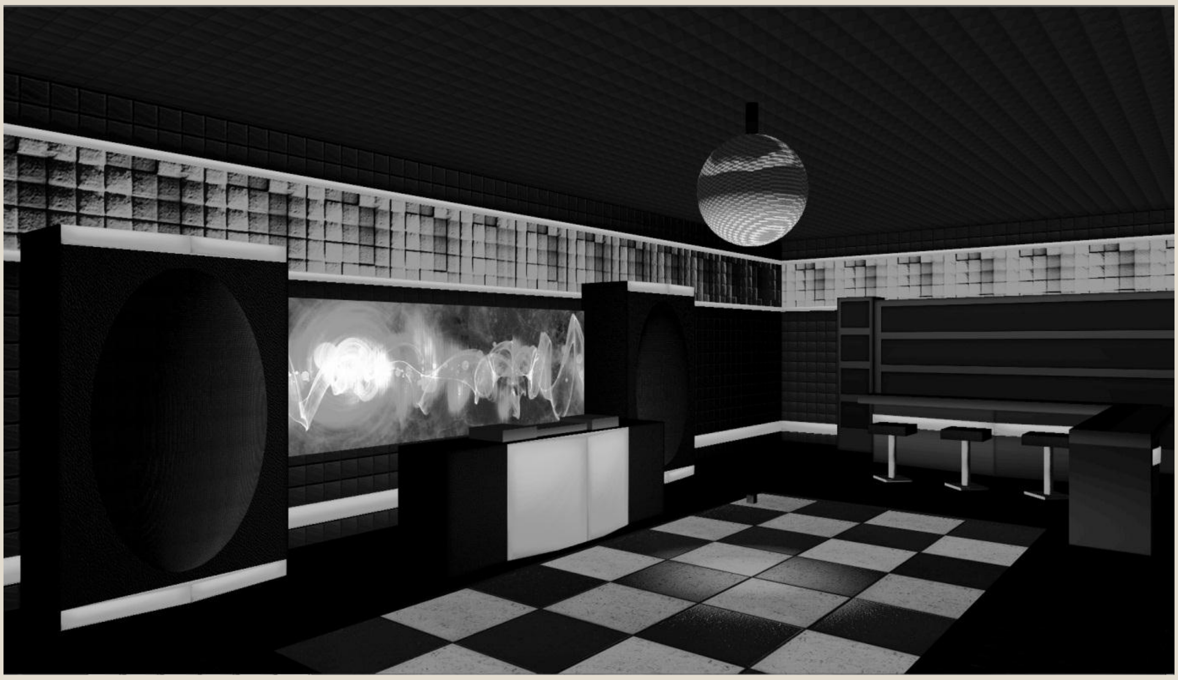
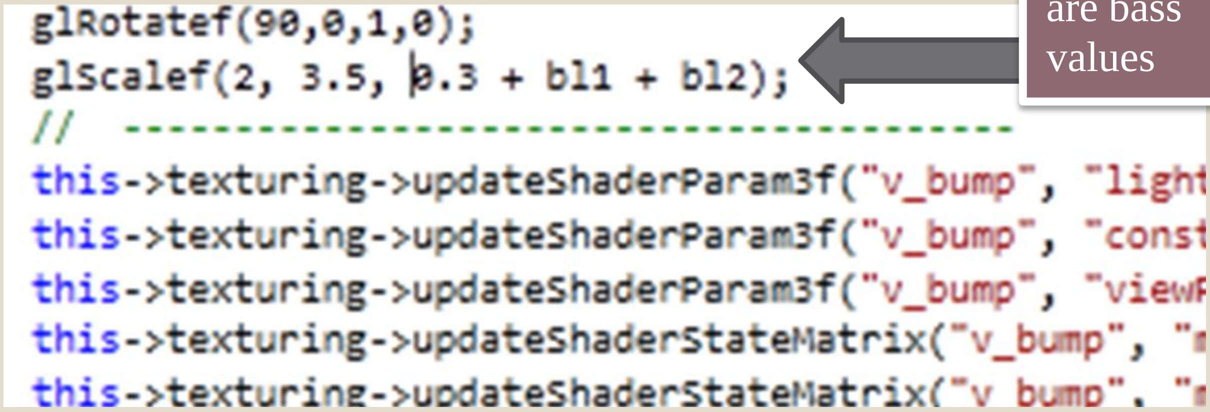


Figure 1 : BW environment of the disco room

Shown above in Figure 1 is the 3D objects (Models) of the disco room, the entire scene is modeled using openGL polygons and triangles.
The scene contains the following objects:

1. Ground floor.
2. Ceiling.
3. Right, left and back walls.
4. 2 bars.
5. 2 shelves.
6. 4Speakers.
7. DJ table.
8. Turn tables.
9. Coach.
10. Bar seats.
11. Disco Ball.
12. Light source.

Animated Speakers



```
glRotatef(90,0,1,0);
glScalef(2, 3.5, 0.3 + bl1 + bl2);
//  ----------------------------------------
this->texturing->updateShaderParam3f("v_bump", "light
this->texturing->updateShaderParam3f("v_bump", "const
this->texturing->updateShaderParam3f("v_bump", "viewP
this->texturing->updateShaderStateMatrix("v_bump", "
this->texturing->updateShaderStateMatrix("v_bump", "
```

bl1 and bl2 are bass values

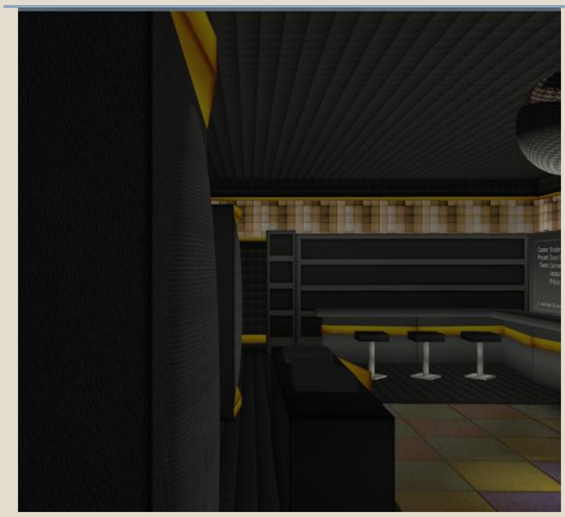


Figure 2 : Low bl1 & b12

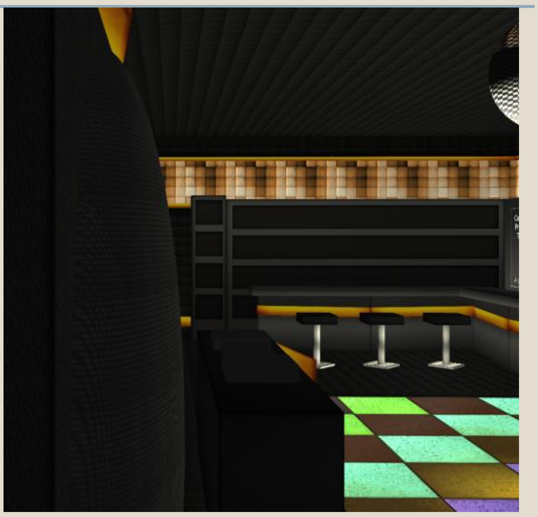


Figure 3 : high bl1 & bl2

CG Shader

The reason we chose CG shader programming, because we felt more comfortable using it, also it was the one of main shader languages taught in class. However we realized after using the language, that it was not a good idea, because there are not enough resources online and it’s a bit outdated, therefore we came to the conclusion, that using cgFX or GLSL would have been a more convenient platform (language) for creating the disco room shaders, because of its broader capabilities and recent available resources.

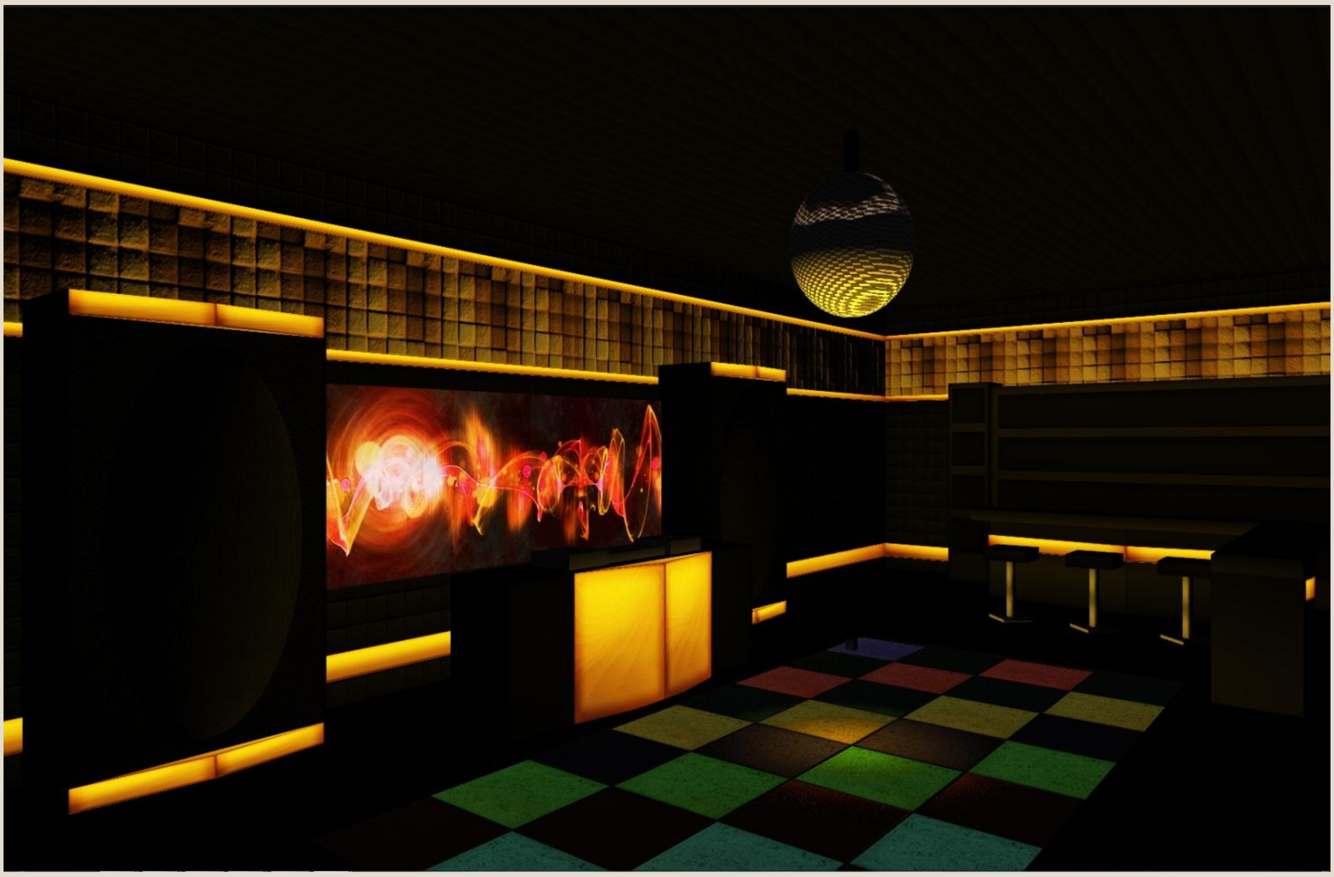
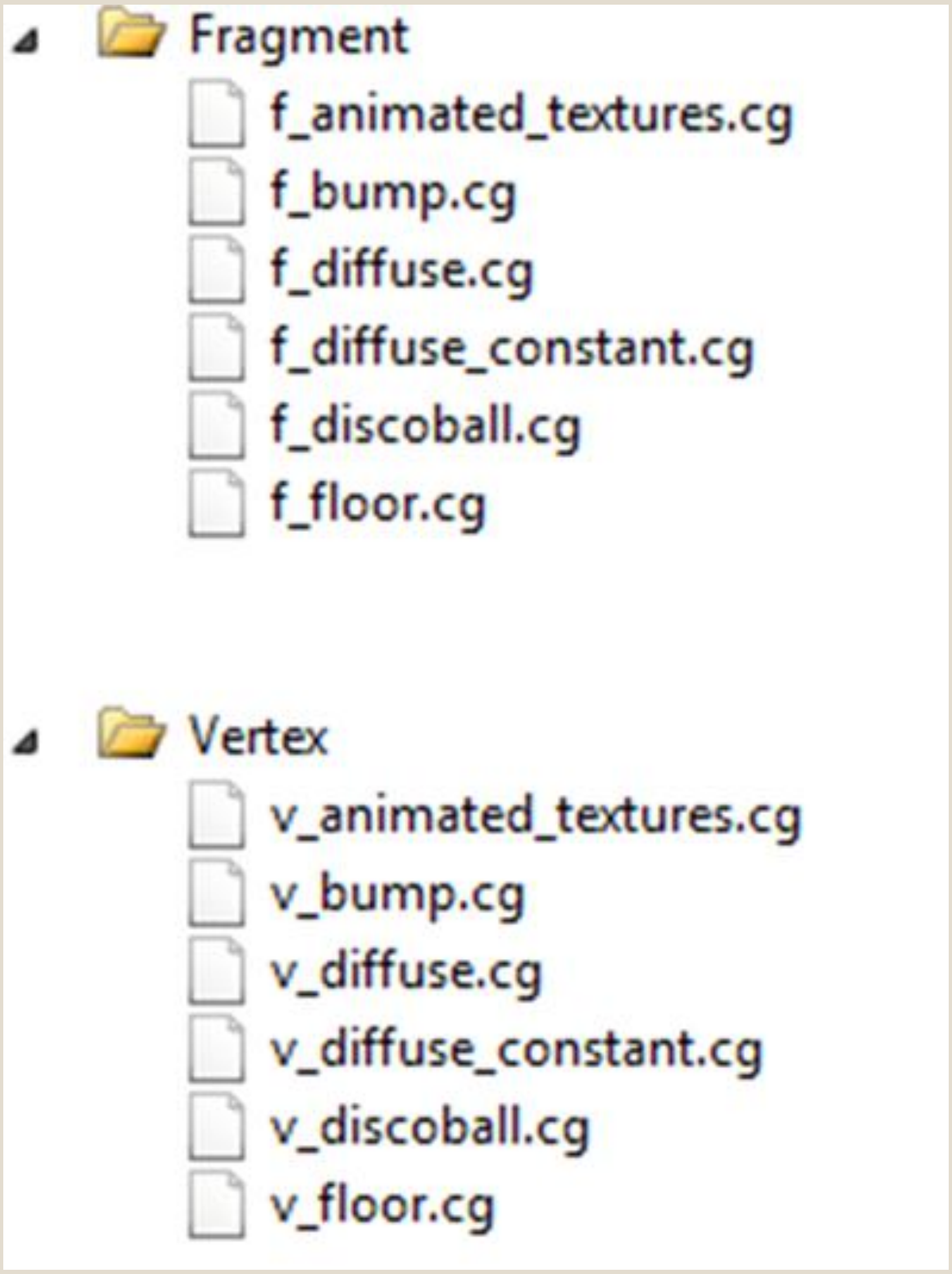


Figure 4 : Shadered SceneList of Shaders

CG Shader List



- Fragment
  - f_animated_textures.cg
  - f_bump.cg
  - f_diffuse.cg
  - f_diffuse_constant.cg
  - f_discoball.cg
  - f_floor.cg
- Vertex
  - v_animated_textures.cg
  - v_bump.cg
  - v_diffuse.cg
  - v_diffuse_constant.cg
  - v_discoball.cg
  - v_floor.cg

Animated Textures Shader (f_animated_textures/ v_animated_textures)

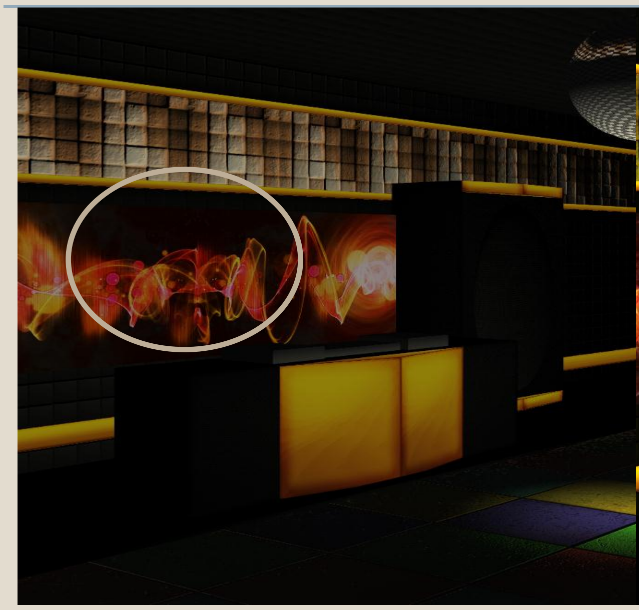


Figure 5 : Scene with default texture dark

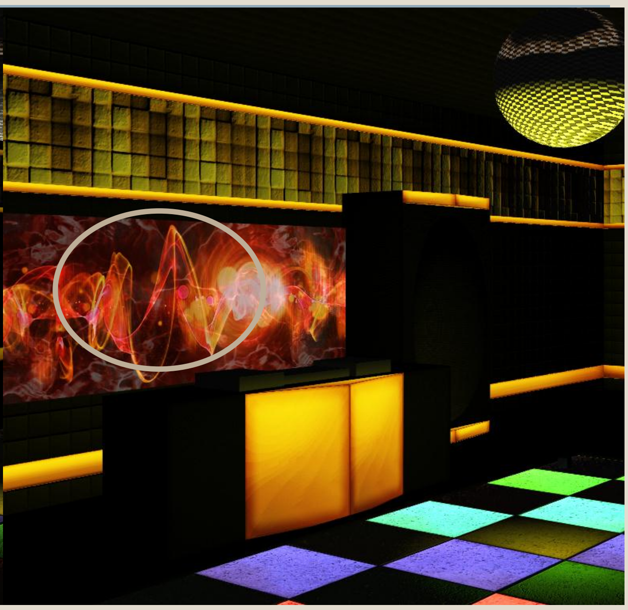


Figure 6 : Scene with bright texture controlled by music

The figures above shows the animated texture shader is used for the picture behind the DJ table animated by the bass values. The animated texture shader mixes in general three given textures and animates them. The first one, the main texture, will be mapped as normal and will be animated to the x-axes. The second texture will be linear interpolated with the first on by a given sound depending value and is animated along the y-axe. Also the third one will be linear interpolated with both before, but it’s nearly transparent – and it’s animated along both axes.

Diffuse Constant Shader (f_diffuse_constant / v_diffuse_constant)

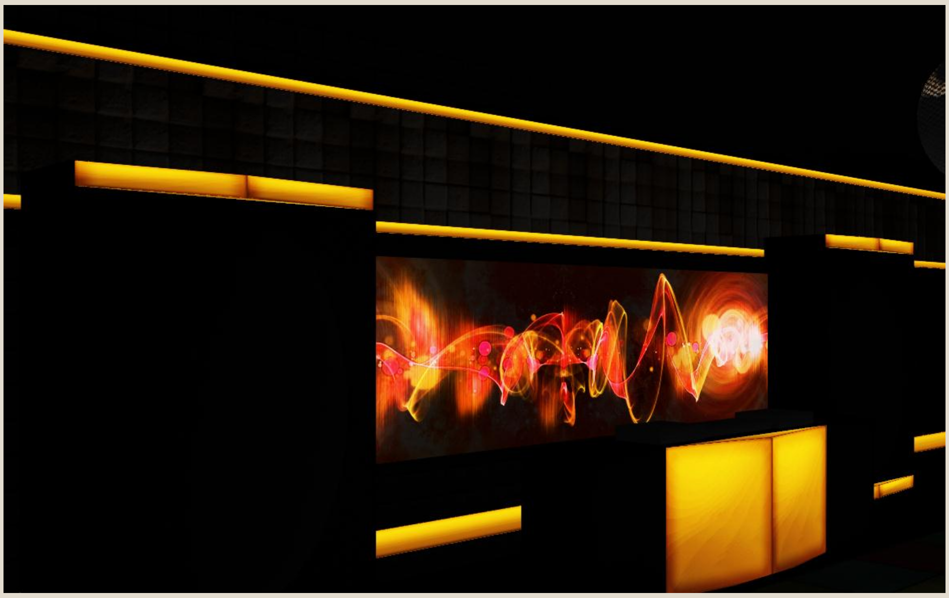


Figure 7 : Diffuse constant shader is the bright orange effect

The name of this shader might be a little bit confusing, because we don’t use the diffuse light term in it, but on the other hand this shader is used for simulating glowing objects. The shader simply consists of light color and a texture.

Diffuse Shader (f_diffuse / v_diffuse)

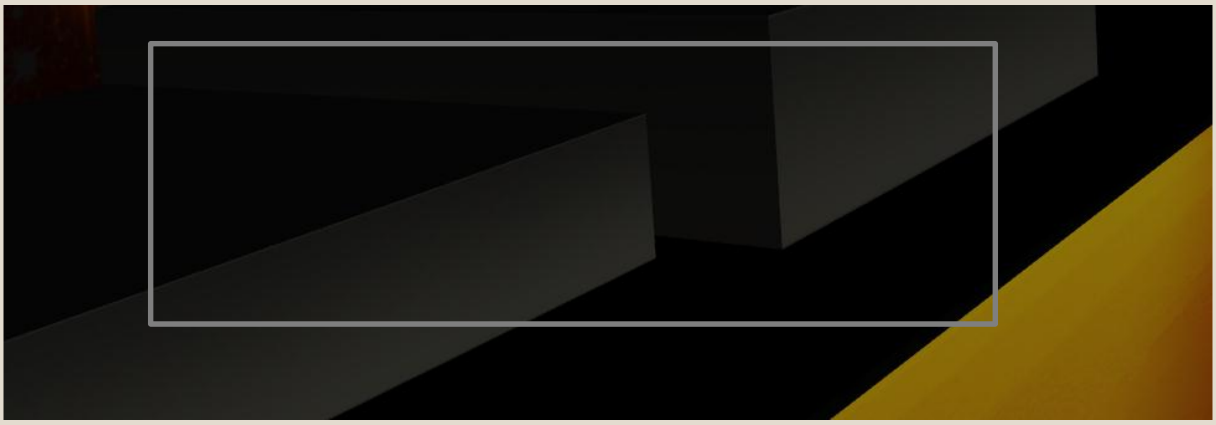


Figure 8 : Turn table

This shader has the typical light diffuse effect. It calculates the diffuse term by Lambert function and maps a given texture on it. Currently the diffuse shader is only used on the turn tables.

Disco ball Shader (f_discoball / v_discoball)

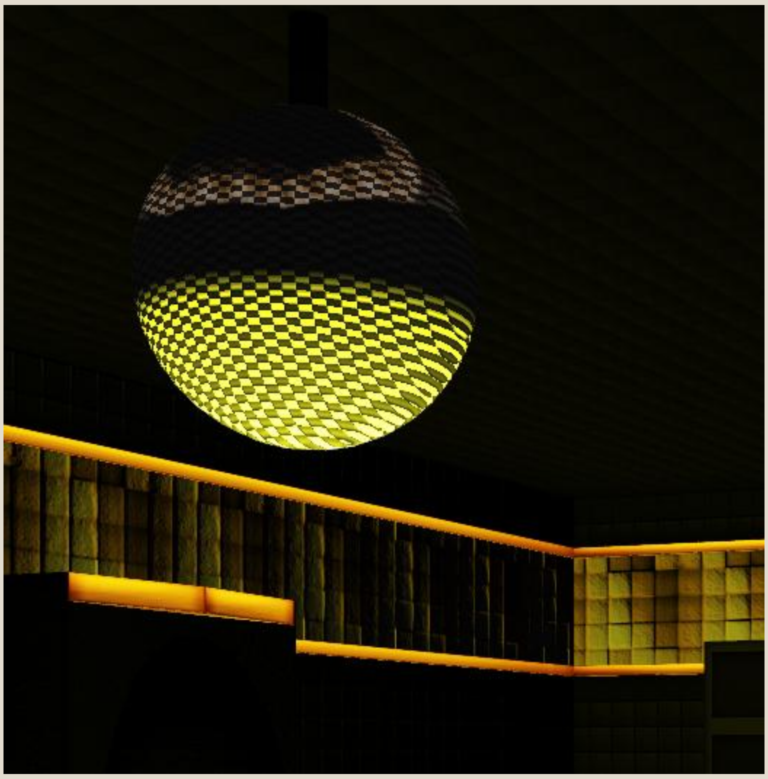


Figure 9 : Disco ball shader

This shader is costumed specially for the disco ball it contains diffuse lighting (lambert) and specular light (Blinn), reflection and refraction of the environment (the environment map was created in the Texturing.cpp), texture mapping and finally a pattern (tile) was created to define which part of the ball will have the reflective shader (mirror like) and the matte dark non reflective color.

FLOOR & BUMP SHADER

The Bump and floor shaders are very similar the only difference between these shaders is, that in the bump-shader it is not reflective and it doesn’t have color-tiling, the sound-interaction implemented and also the “range-expand”-effect is not used, contrary the floor shader has all what was mentioned before. The floor shader is used for the colorful floor, while the bump shader is used for almost everything else.

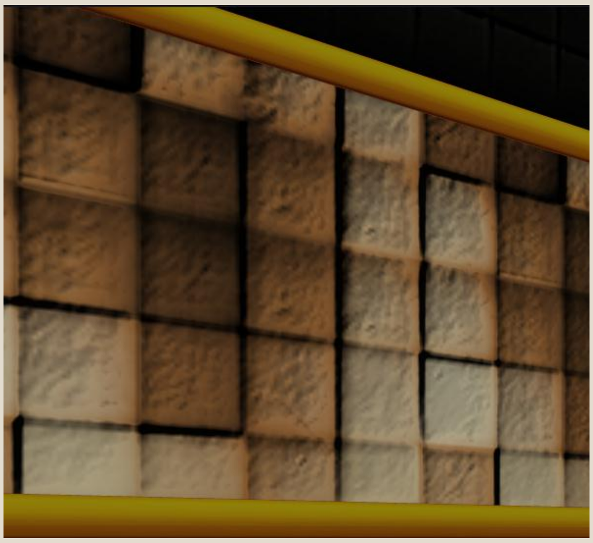


Figure 10 : Wall Bump Shader

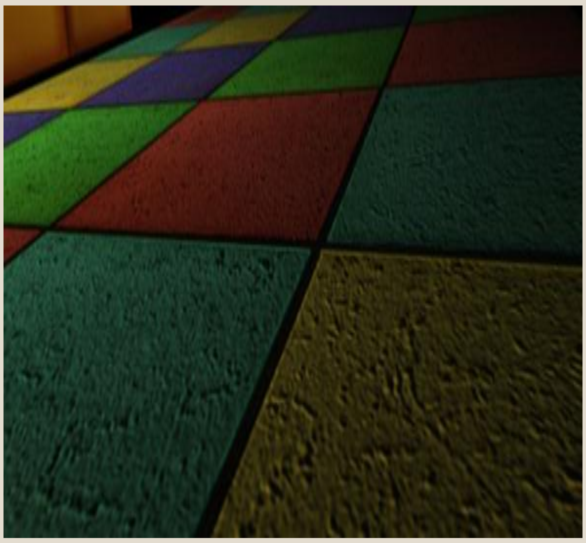


Figure 11 : Colored floor bump map shader

The Normal Map:

The normal mapping is used to add more detail to the surface of an object without requiring further tessellation of the mesh. Therefore a texture is generated which includes the encoded surface normal at every point. Every color channel is holding the information about one axis in the tangent space (RGB -> XYZ). The color-values are saved in the range of 0->1, therefore the normal which has a range from -1->1 has to be range-compressed by:

$$\text{color} = 0.5 * \text{normal} + 0.5$$

the expanding of the color-values is calculated like:

$$\text{normal} = 2 * (\text{color} - 0.5)$$

The Vertex Program:

In the vertex shader the parameters are calculated to world space and passed on to the fragment shader. The reason for not calculating the necessary vectors to tangent space is that we also wanted to implement environment mapping in the shaders. The cube map is in world space and therefore the vector of the reflected light has to be in the world space too, to make a lookup of the reflected color value. Therefore it’s easier to transform just the normal vector to world space by calculating the dot product of the orthonormalized rotation matrix and make all lighting calculations in the world space. In the vertex shader we also calculate a tangent, by making the cross product between the surface normal and a self-defined vector. We need the tangent to create the orthonormalized rotation matrix. The other required vectors are the surface normal and binormal.

The fragment shader:

In the fragment shader the normal is fetched in texture space, normalized and expanded and then transformed to world space by calculating the matrix product with the orthonormalized rotation matrix. Also the light, view and halfway vectors are calculated to define the lighting and reflection.
The light and view are expanded to create an effect that increases the lighting and the normal mapping.

Lighting

The diffuse term is calculated by this formula:

*Diffuse = diffuse-color * light-color * max (normal-vector · light-vector, 0)*

The dot product of the normal and the light vector defines the angle between them, therefore as smaller the angle is, the greater the dot product and the more light the surface will receive. If the surface is facing away from the light the dot product is negative and so it has to be clamped.
The specular term is calculated by the formula:

*Specular = specular-color * light-color * facing * (max (normal-vector · halfway-vector,0))^shininess*

When the angle between the halfway and the normal vector is small the specular is visible. The exponentiation is needed to ensure that the specular falls of when the angle is getting bigger. If the diffuse term is zero the specular also doesn’t appear, this ensures that surfaces that are facing away from the light don’t receive specular light.

Reflection

The reflection is calculated by calculating the incident ray which is going from the eye position to the surface. When it reached the surface the reflected ray is calculated. It has the same angle as the incident vector, based on the surface normal.
The reflected ray is calculated with the formula:

*Reflected-vector = incident-vector * 2.0 * normal * dot (normal, incident-vector)*

In the code we are changing the position in word space to increase the amount of reflection on the floor.

Color-Tiling

In the tiling loop the ranges of the tiles are defined in x and y direction.
*[x * xFactor ; (x+1)*xFactor] [y * yFactor ; (x+1)*yFactor]*
If the texture coordinate lies in the range of the current tile in the loop it is assigned a certain value from the color array. The color array contains 5 different color values. If the counter reaches the end of the array it’s set back.

Sound-Interaction

In the loop the glow is active to either the odd or the even tiles when the equation counter % 2 == random % 2 is for filled. The counter is raised up in every passing of the loop. The random value is an input and is calculated every 25 frames new. SO the glowing is steady for a while and is not flickering too much. There are two color values defined. The one is multiplied with an input-variable that is measuring the bass in the song that is played. The second color-value is the color that is assigned when the light is off. A condition ensures that always the brighter color value is assigned, so that when the sound is on the floor tiles are brightening, and when there is no sound it’s never dropping under the “normal” color value.

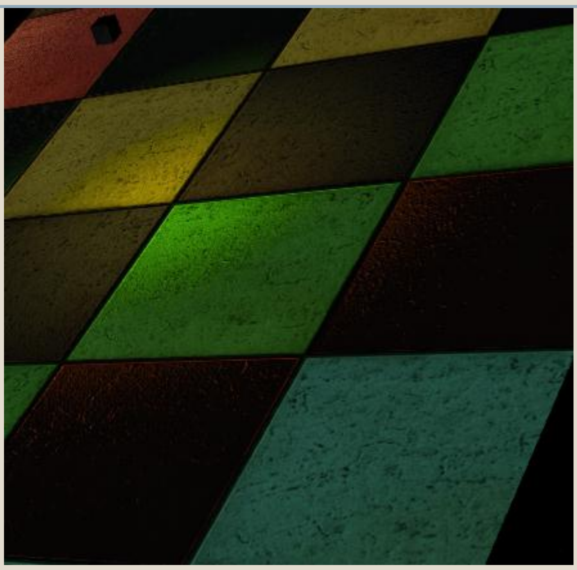


Figure 12 : Darker = less bass

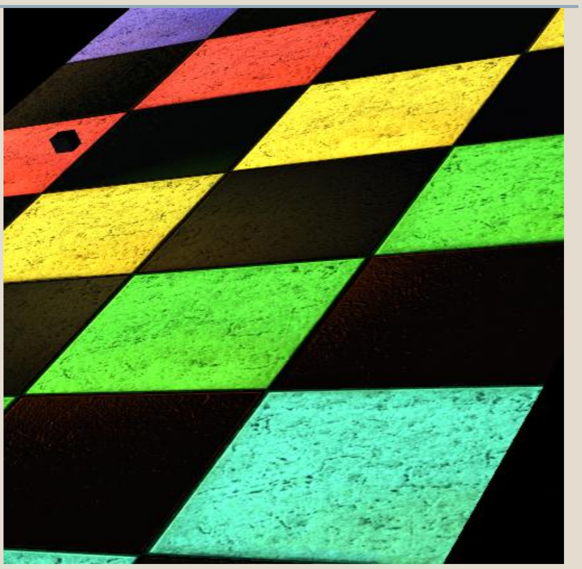
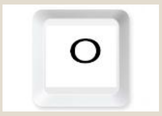


Figure 13 : Brighter = more bass

Interactions

One can travel inside the scene using both the mouse and keyboard.

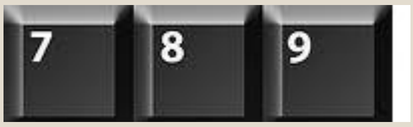
Light Interactions

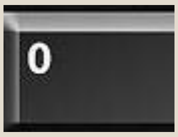
- 

By pressing the “o” character button the light will be switched on.
-

By pressing the “L” (l lowercase) character button the light will be switched back off.

Music

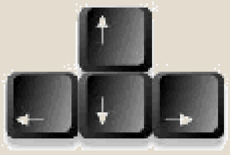
- 

Press any of those keys to enable the music and sound visualization effects. (Each key has a different song attached to it).
- 

To disable the music.

Camera Translation & rotation

1\.



1. By using the up and down arrow you can translate in the y axis up and down.
2. By using the left and right you can translate in the x-axis going left and right.

2\.

- By scrolling the scroll wheel you can translate in the z axis back and forth.
- By clicking the left mouse button and moving the mouse, you can rotate in the y-axis.
- By clicking the right mouse button and moving the mouse, you can rotate in the z-axis.

3. Automatic Camera Roaming through the room :
   -

     By pressing character “i” the camera will start roaming around the room (PS: if the roaming is on you won’t be able to control the camera).
   -

     By pressing character “k” the camera will stop roaming and one would be able to control the camera again.

Light Source Translation

- Z-axis ;Buttons :
  - w-> translate –z -axis.
  - S-> translate + z-axis.
- Y- axis ;Buttons :
  - r-> translate +y-axis.
  - f->translate –y-axis.
- X-axis; Buttons:
  - d->translate +x-axis.
  - A->translate –x-axis.

Reference & Sources

- http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter08.html
- http://3dgep.com/?p=1464
- http://www.blacksmith-studios.dk/projects/downloads/bumpmapping_using_cg.php