The SoundVisualizer

The idea…

- Shader that interacts with a song
- The bass – frequency as input to the fragment shader
- Value influences the lighting of a disco-room

OpenGL - Application

```
// Bump Vertex
this->texturing->loadVertexShader("v_bump", "vfunc_bump");
// ---- PARAMS ------------------------------------------------------------
this->texturing->setShaderParam4f("v_bump", "constantColor", lightColor_1);
this->texturing->setShaderParam3f("v_bump", "lightPos", lightPosition_1);
this->texturing->setShaderParam3f("v_bump", "viewPos", view);
this->texturing->setShaderStateMatrix("v_bump", "m2w");
this->texturing->setShaderStateMatrix("v_bump", "m2wIT");
this->texturing->setShaderStateMatrix("v_bump", "mvp");
// ------------------------------------------------------------------------

// Bump Fragment
this->texturing->loadFragmentShader("f_bump", "ffunc_bump");
// ---- PARAMS ------------------------------------------------------------
this->texturing->setShaderParam4f("f_bump", "globalAmbient", globalAmbientLight);
this->texturing->setShaderParam4f("f_bump", "materialAmbient", defaultMaterialAmbient);
this->texturing->setShaderParam4f("f_bump", "materialEmissive", defaultMaterialEmissive);
this->texturing->setShaderParam3f("f_bump", "materialDiffuse", defaultMaterialDiffuse);
this->texturing->setShaderParam3f("f_bump", "materialSpecular", defaultMaterialSpecular);
this->texturing->setShaderParam1f("f_bump", "emissiveBoost", emissiveBoost);
this->texturing->setShaderParam1f("f_bump", "materialShiniess", defaultShiniess);
this->texturing->setShaderTexture("f_bump", "colorTex");
this->texturing->setShaderTexture("f_bump", "normTex");
// ------------------------------------------------------------------------
```

Load and compile a shader program

retrieve and initialize the shader params

```
// Shader container
struct cgShader
{
    // Name of the program
    char* filename;
    // CGprogram
    CGprogram cgProgram;
    // CGparams
    vector<CGparameter> cgParam;
    // Name of the params
    vector<char*> cgParamName;
};
```

OpenGL - Application

```
//  OBJECT: Wall - Side left
this->texturing->bindVertexShader("v_bump");
this->texturing->bindFragmentShader("f_bump");
//  ----------------------------------------
glTranslatef(-60.2,0,65);
glScalef(1,30,120);
//  ----------------------------------------
this->texturing->updateShaderParam3f("v_bump", "lightPos", lightPosition_1);
this->texturing->updateShaderParam3f("v_bump", "constantColor", lightColor_1);
this->texturing->updateShaderParam3f("v_bump", "viewPos", view);
this->texturing->updateShaderStateMatrix("v_bump", "m2w", CG_GL_MODELVIEW_MATRIX, CG_GL_MATRIX_IDENTITY);
this->texturing->updateShaderStateMatrix("v_bump", "m2wIT", CG_GL_MODELVIEW_MATRIX, CG_GL_MATRIX_INVERSE_TRANSPOSE);
this->texturing->updateShaderStateMatrix("v_bump", "mvp", CG_GL_MODELVIEW_PROJECTION_MATRIX, CG_GL_MATRIX_IDENTITY);
this->texturing->updateShaderParam4f("f_bump", "globalAmbient", globalAmbientLight);
this->texturing->updateShaderParam4f("f_bump", "materialAmbient", defaultMaterialAmbient2);
this->texturing->enableShaderTexture("f_bump", "colorTex", "../Assets/wall2.bmp");
this->texturing->enableShaderTexture("f_bump", "normTex", "../Assets/wall2_norm.bmp");
//  ----------------------------------------
this->producing->drawCube();
//  ----------------------------------------
this->texturing->disableShaderTexture("f_bump", "colorTex");
this->texturing->disableShaderTexture("f_bump", "normTex");
```

} bind the shader programs

} transformations

} update shader params and bind textures

} draw object

} unbind textures

OpenGL - Application

```
FMOD::Channel *channel = audio->getChannel(1);
channel->getSpectrum( bassL, SPECLEN, 0, FMOD_DSP_FFT_WINDOW_BLACKMAN );
```

bassL { [0] 0.0 - 1.0 [1] 0.0 - 0.8 [2] 0.0 - 0.5 [3] 0.0 - 0.3 }

The Floor Shader

VERTEX SHADER:

```
OUT.color    = constantColor;
OUT.texCoord = texCoord;
OUT.lightDir = lightPos;
OUT.viewDir  = viewPos;

OUT.position  = mul(mvp, float4(position,1));
OUT.positionW = mul(m2w, float4(position,1));

float3 T = cross(normal,float3(1.0,0.0,0.0));
T = normalize(T);
OUT.tangentW  = normalize(mul(m2wIT, float4(T,0))).xyz;
OUT.normalW   = mul(m2wIT, float4(normal,0)).xyz;
OUT.binormalW = cross(OUT.normalW, OUT.tangentW).xyz,
```

The Floor Shader

NORMAL MAPPING:

```
float3x3 tangentMatrix = float3x3( normalize( IN.tangentW ),
                                   normalize( IN.binormalW ),
                                   normalize( IN.normalW ));

float3 normalTex = tex2D(normTex, IN.texCoord).xyz;
float3 normal = normalize(expand(normalTex));
normal = mul(normal,tangentMatrix);

float3 normLight = normalize(IN.lightDir - IN.positionW).xyz;
normLight = expand(normLight);

float3 normView = normalize(IN.viewDir - IN.positionW).xyz;
normView = expand(normView);
float3 normHalfangle = normalize(normLight + normView);
```

The Floor Shader

COLOR-TILING:

```
for(int x = 0; x <= tiles; x++)
{
   for(int y = 0; y <= 2*tiles; y++)
   {
      if(IN.texCoord.x > x*xFactor && IN.texCoord.x <= (x+1)*xFactor &&
         IN.texCoord.y > y*yFactor && IN.texCoord.y <= (y+1)*yFactor)
      {
         outColor = mixColor[arrCount];
         if(counter % 2 == rdm % 2) glow = 1;
      }
      counter++;
      arrCount++;
      if(arrCount == 5) arrCount = 0;
   }
}
```

```
}
```

The Floor Shader

SOUND-INTERACTION:

```
float3 Fcolor  = ((outColor + coloredTex * 0.8) * coloredTex) *
                 (soundInput * 5);

float3 Fcolor2 = ((outColor + coloredTex * 0.8) * coloredTex *
                  ((materialEmissive * emissiveBoost) +
                  (materialAmbient * globalAmbient) +
                   diffuseColor + specularColor));

Fcolor =  length(Fcolor2) < length(Fcolor) &&
          glow == 1 ? Fcolor : Fcolor2;
```

Animated Textures

Linear Interpolation:

```
float2 tex1Coords = float2(IN.texCoord.x + time / 1000,
                           IN.texCoord.y);

float2 tex2Coords = float2(IN.texCoord.x,
                           IN.texCoord.y + time / 1000);

float2 tex3Coords = float2(IN.texCoord.x + time / 1000,
                           IN.texCoord.y + time / 1000);

float4 pureTexture = lerp(tex2D(tex1, tex1Coords),
                          tex2D(tex2, tex2Coords), soundInput);
pureTexture = lerp(pureTexture, tex2D(tex3, tex3Coords), 0.1);
```

The Discoball

Vertex Shader:

```
float3 I = positionW - viewPos;

OUT.R = reflect(I, N);
I = normalize(I);

OUT.TRed   = refract(I, N, etaRatio.x);
OUT.TGreen = refract(I, N, etaRatio.y);
OUT.TBlue  = refract(I, N, etaRatio.z);

OUT.reflectionFactor = fresnelBias + fresnelScale *
                       pow(1+ dot(I, N), fresnelPower);
```

The Discoball

Fragement Shader - Lighting:

```
float3 Ln = normalize(IN.lightDir);
float3 Nn = normalize(IN.normal);
float diffuseLight = max(dot(Nn, Ln), 0);

float3 diffuse = pureTexture.rgb *  diffuseLight.xxx *
                 IN.lightcolor.xyz;

float3 Vn = normalize(IN.viewDir);
float3 Hn = normalize(Ln + Vn);
float specularLight = pow(max(dot(Nn, Hn), 0), shiniess);
if (diffuseLight <= 0) specularLight = 0;
float3 specular = specularLight.xxx * IN.lightcolor.xyz;
```

The Discoball

Fragement Shader – Enviroment Mapping:

```
float4 reflectedColor = texCUBE(environmentMap, IN.R);
float4 refractedColor;

refractedColor.r   = texCUBE(environmentMap, IN.TRed).r;
refractedColor.g  = texCUBE(environmentMap, IN.TGreen).g;
refractedColor.b  = texCUBE(environmentMap, IN.TBlue).b;
refractedColor.a  = 1;

float4 environment = lerp(refractedColor, reflectedColor,
                          IN.reflectionFactor);
```

The Discoball

Fragement Shader – Tiling:

```
float4 patternCheck = tex2D(pattern, IN.texCoord);
float glow = 0;

if(patternCheck.r == glow && patternCheck.g == glow && patternCheck.b == glow)
{
    float3 a = lerp(diffuse.rgb * (soundInput * 50 + 1) + specular.rgb * (soundInput * 50 +
1) +
                materialEmissive.rgb,environment, 0.6);
    float3 b = lerp(diffuse.rgb + specular.rgb  + materialEmissive.rgb, environment, 0.5);
    fragmentOut.rgb = length(a) > length(b) ? a : b;
}else{
    fragmentOut.rgb = diffuse.rgb + specular.rgb  + materialEmissive.rgb + environment * 0.1;
}
fragmentOut.a = 1
```