////////////////////////////////////////////////////////water\_waves\_caustic\_gpu\_algorithm//////////////////////////////////////////////////////////////////////

VS:1>cube\_map\_blur.vs

#version 120

void main()

{

gl\_TexCoord[0] = gl\_MultiTexCoord0;

gl\_Position = gl\_Vertex;

}

2>cube\_map\_hblur.fs

#version 120

uniform samplerCube CubeMap;

uniform vec3 Offset;

void main()

{

float Color = 0.0;

for(float i = -3.0; i <= 3.0; i += 1.0)

{

Color += textureCube(CubeMap, Offset \* i + gl\_TexCoord[0].stp).r \* (4.0 - abs(i));

}

gl\_FragColor = vec4(Color / 16.0, 0.0, 0.0, 1.0);

}

3>cube\_map\_vblur.fs

#version 120

uniform samplerCube CubeMap;

uniform vec3 Offset;

void main()

{

float Color = 0.0;

for(float i = -3.0; i <= 3.0; i += 1.0)

{

Color += textureCube(CubeMap, Offset \* i + gl\_TexCoord[0].stp).r \* (4.0 - abs(i));

}

gl\_FragColor = vec4(vec3(smoothstep(0.0, 0.333333, Color / 16.0)), 1.0);

}

4>photon.vs

#version 120

uniform sampler2D WaterHeightMap, WaterNormalMap;

uniform mat4x4 PhotonsWorldToTextureMatrices[6];

uniform vec3 LightPosition, CubeMapNormals[6];

vec3 IntersectCubeMap(vec3 Position, vec3 Direction)

{

vec3 Point;

for(int i = 0; i < 6; i++)

{

float NdotR = -dot(CubeMapNormals[i], Direction);

if(NdotR > 0.0)

{

float Distance = (dot(CubeMapNormals[i], Position) + 1.0) / NdotR;

if(Distance > -0.03)

{

Point = Direction \* Distance + Position;

if(Point.x > -1.001 && Point.x < 1.001 && Point.y > -1.001 && Point.y < 1.001 && Point.z > -1.001 && Point.z < 1.001)

{

break;

}

}

}

}

return Point;

}

void main()

{

gl\_TexCoord[0].st = vec2(gl\_Vertex.x \* 0.5 + 0.5, 0.5 - gl\_Vertex.z \* 0.5);

vec3 Position = gl\_Vertex.xyz;

Position.y += texture2D(WaterHeightMap, gl\_TexCoord[0].st).g;

vec3 Normal = normalize(texture2D(WaterNormalMap, gl\_TexCoord[0].st).rgb);

vec3 LightDirection = normalize(Position - LightPosition);

vec3 LightDirectionRefracted = refract(LightDirection, Normal, 0.750395);

vec3 IntersetionPoint = IntersectCubeMap(Position, LightDirectionRefracted);

int MaxAxis = 0;

float Axes[6] = float[](IntersetionPoint.x, -IntersetionPoint.x, IntersetionPoint.y, -IntersetionPoint.y, IntersetionPoint.z, -IntersetionPoint.z);

for(int i = 1; i < 6; i++)

{

if(Axes[i] > Axes[MaxAxis])

{

MaxAxis = i;

}

}

gl\_TexCoord[0] = PhotonsWorldToTextureMatrices[MaxAxis] \* vec4(IntersetionPoint, 1.0);

gl\_Position = gl\_TexCoord[0] \* 2.0 - 1.0;

}

5>photon.fs

#version 120

uniform sampler2D PhotonsTexture;

void main()

{

gl\_FragColor = vec4(texture2D(PhotonsTexture, gl\_TexCoord[0].st).rgb + 0.0625, 1.0);

}

6>poolsky.vs

#version 120

void main()

{

gl\_TexCoord[0].stp = vec3(gl\_Vertex.x, -gl\_Vertex.yz);

gl\_Position = gl\_ModelViewProjectionMatrix \* gl\_Vertex;

}

7>poolsky.fs

#version 120

uniform samplerCube PoolSkyCubeMap, PhotonsCubeMap;

void main()

{

gl\_FragColor = textureCube(PoolSkyCubeMap, gl\_TexCoord[0].stp) + textureCube(PhotonsCubeMap, gl\_TexCoord[0].stp);

}

8>water.vs

#version 120

uniform sampler2D WaterHeightMap;

varying vec3 Position;

void main()

{

gl\_TexCoord[0].st = vec2(gl\_Vertex.x \* 0.5 + 0.5, 0.5 - gl\_Vertex.z \* 0.5);

Position = gl\_Vertex.xyz;

Position.y += texture2D(WaterHeightMap, gl\_TexCoord[0].st).g;

gl\_Position = gl\_ModelViewProjectionMatrix \* vec4(Position, 1.0);

}

9>water.fs

#version 120

uniform sampler2D WaterNormalMap;

uniform samplerCube PoolSkyCubeMap, PhotonsCubeMap;

uniform vec3 LightPosition, CubeMapNormals[6], CameraPosition;

varying vec3 Position;

vec3 IntersectCubeMap(vec3 Position, vec3 Direction)

{

vec3 Point;

for(int i = 0; i < 6; i++)

{

float NdotR = -dot(CubeMapNormals[i], Direction);

if(NdotR > 0.0)

{

float Distance = (dot(CubeMapNormals[i], Position) + 1.0) / NdotR;

if(Distance > -0.03)

{

Point = Direction \* Distance + Position;

if(Point.x > -1.001 && Point.x < 1.001 && Point.y > -1.001 && Point.y < 1.001 && Point.z > -1.001 && Point.z < 1.001)

{

break;

}

}

}

}

return vec3(Point.x, -Point.yz);

}

void main()

{

vec3 Normal = normalize(texture2D(WaterNormalMap, gl\_TexCoord[0].st).rgb);

vec3 Direction = normalize(Position - CameraPosition);

if(CameraPosition.y > 0)

{

vec3 ReflectedColor = textureCube(PoolSkyCubeMap, IntersectCubeMap(Position, reflect(Direction, Normal))).rgb;

vec3 IntersectionPoint = IntersectCubeMap(Position, refract(Direction, Normal, 0.750395));

vec3 RefractedColor = textureCube(PoolSkyCubeMap, IntersectionPoint).rgb + textureCube(PhotonsCubeMap, IntersectionPoint).rgb;

vec3 LightDirectionReflected = reflect(normalize(Position - LightPosition), Normal);

float Specular = pow(max(-dot(Direction, LightDirectionReflected), 0.0), 128);

gl\_FragColor.rgb = mix(ReflectedColor, RefractedColor, -dot(Normal, Direction)) + Specular;

}

else

{

Normal = -Normal;

vec3 IntersectionPoint = IntersectCubeMap(Position, reflect(Direction, Normal));

vec3 ReflectedColor = textureCube(PoolSkyCubeMap, IntersectionPoint).rgb + textureCube(PhotonsCubeMap, IntersectionPoint).rgb;

vec3 DirectionRefracted = refract(Direction, Normal, 1.332631);

if(DirectionRefracted.x == 0.0 && DirectionRefracted.y == 0.0 && DirectionRefracted.z == 0.0)

{

gl\_FragColor.rgb = ReflectedColor;

}

else

{

vec3 RefractedColor = textureCube(PoolSkyCubeMap, IntersectCubeMap(Position, DirectionRefracted)).rgb;

gl\_FragColor.rgb = mix(ReflectedColor, RefractedColor, -dot(Normal, Direction));

}

}

}

10>wateradddrop.vs

#version 120

void main()

{

gl\_TexCoord[0] = gl\_Vertex;

gl\_Position = gl\_Vertex \* 2.0 - 1.0;

}

11>wateradddrop.fs

#version 120

uniform sampler2D WaterHeightMap;

uniform float DropRadius;

uniform vec2 Position;

void main()

{

vec2 vh = texture2D(WaterHeightMap, gl\_TexCoord[0].st).rg;

float d = distance(gl\_TexCoord[0].st, Position);

gl\_FragColor = vec4(vh.r, vh.g - 4.0f \* max(DropRadius - d, 0.0), 0.0, 0.0);

}

12>waterheightmap.vs

#version 120

void main()

{

gl\_TexCoord[0] = gl\_Vertex;

gl\_Position = gl\_Vertex \* 2.0 - 1.0;

}

13>waterheightmap.fs

#version 120

uniform sampler2D WaterHeightMap;

uniform float ODWHMR;

void main()

{

vec2 vh = texture2D(WaterHeightMap, gl\_TexCoord[0].st).rg;

float force = 0.0;

force += 0.707107 \* (texture2D(WaterHeightMap, gl\_TexCoord[0].st - vec2(ODWHMR, ODWHMR)).g - vh.g);

force += texture2D(WaterHeightMap, gl\_TexCoord[0].st - vec2(0.0, ODWHMR)).g - vh.g;

force += 0.707107 \* (texture2D(WaterHeightMap, gl\_TexCoord[0].st + vec2(ODWHMR, -ODWHMR)).g - vh.g);

force += texture2D(WaterHeightMap, gl\_TexCoord[0].st - vec2(ODWHMR, 0.0)).g - vh.g;

force += texture2D(WaterHeightMap, gl\_TexCoord[0].st + vec2(ODWHMR, 0.0)).g - vh.g;

force += 0.707107 \* (texture2D(WaterHeightMap, gl\_TexCoord[0].st + vec2(-ODWHMR, ODWHMR)).g - vh.g);

force += texture2D(WaterHeightMap, gl\_TexCoord[0].st + vec2(0.0, ODWHMR)).g - vh.g;

force += 0.707107 \* (texture2D(WaterHeightMap, gl\_TexCoord[0].st + vec2(ODWHMR, ODWHMR)).g - vh.g);

force \*= 0.125;

vh.r += force;

vh.g += vh.r;

vh.g \*= 0.99;

gl\_FragColor = vec4(vh, 0.0, 0.0);

}

14>waternormalmap.vs

#version 120

void main()

{

gl\_TexCoord[0] = gl\_Vertex;

gl\_Position = gl\_Vertex \* 2.0 - 1.0;

}

15>waternormalmap.fs

#version 120

uniform sampler2D WaterHeightMap;

uniform float ODWNMR, WMSDWNMRM2;

void main()

{

float y[4];

y[0] = texture2D(WaterHeightMap, gl\_TexCoord[0].st + vec2(ODWNMR, 0.0)).g;

y[1] = texture2D(WaterHeightMap, gl\_TexCoord[0].st + vec2(0.0, ODWNMR)).g;

y[2] = texture2D(WaterHeightMap, gl\_TexCoord[0].st - vec2(ODWNMR, 0.0)).g;

y[3] = texture2D(WaterHeightMap, gl\_TexCoord[0].st - vec2(0.0, ODWNMR)).g;

vec3 Normal = normalize(vec3(y[2] - y[0], WMSDWNMRM2, y[1] - y[3]));

gl\_FragColor = vec4(Normal, 1.0);

}

////////////////////////////Source////////////////////////////

#define WMR 128 // water mesh resolution

#define WHMR 128 // water height map resolution

#define WNMR 256 // water normal map resolution

#define PCMR 512 // photon cube map resolution

class COpenGLRenderer

{

protected:

int Width, Height;

mat3x3 NormalMatrix;

mat4x4 ModelMatrix, ViewMatrix, ViewMatrixInverse, ProjectionMatrix, ProjectionBiasMatrixInverse;

protected:

CTexture PoolSkyCubeMap;

GLuint WaterHeightMaps[2], WHMID, WaterNormalMap, PhotonsTexture, PhotonsTempCubeMaps[2], PhotonsCubeMap, PhotonsVBO, PoolSkyVBO, WaterVBO, FBO;

CShaderProgram WaterAddDropProgram, WaterHeightMapProgram, WaterNormalMapProgram, PhotonProgram, CubeMapHBlurProgram, CubeMapVBlurProgram, PoolSkyProgram, WaterProgram;

int PhotonsCount, QuadsVerticesCount;

public:

bool WireFrame, Pause;

float DropRadius;

public:

CString Text;

public:

COpenGLRenderer();

~COpenGLRenderer();

bool Init();

void Render(float FrameTime);

void Resize(int Width, int Height);

void Destroy();

void AddDrop(float x, float y, float DropRadius);

void AddDropByMouseClick(int x, int y);

};

#include "opengl\_21\_tutorials\_win32\_framework.h"

// ----------------------------------------------------------------------------------------------------------------------------

CBuffer::CBuffer()

{

SetDefaults();

}

CBuffer::~CBuffer()

{

Empty();

}

void CBuffer::AddData(void \*Data, int DataSize)

{

int Remaining = BufferSize - Position;

if(DataSize > Remaining)

{

BYTE \*OldBuffer = Buffer;

int OldBufferSize = BufferSize;

int Needed = DataSize - Remaining;

BufferSize += Needed > BUFFER\_SIZE\_INCREMENT ? Needed : BUFFER\_SIZE\_INCREMENT;

Buffer = new BYTE[BufferSize];

memcpy(Buffer, OldBuffer, OldBufferSize);

delete [] OldBuffer;

}

memcpy(Buffer + Position, Data, DataSize);

Position += DataSize;

}

void CBuffer::Empty()

{

delete [] Buffer;

SetDefaults();

}

void \*CBuffer::GetData()

{

return Buffer;

}

int CBuffer::GetDataSize()

{

return Position;

}

void CBuffer::SetDefaults()

{

Buffer = NULL;

BufferSize = 0;

Position = 0;

}

// ----------------------------------------------------------------------------------------------------------------------------

int gl\_max\_texture\_size = 0, gl\_max\_texture\_max\_anisotropy\_ext = 0;

// ----------------------------------------------------------------------------------------------------------------------------

CTexture::CTexture()

{

Texture = 0;

}

CTexture::~CTexture()

{

}

CTexture::operator GLuint ()

{

return Texture;

}

bool CTexture::LoadTexture2D(char \*FileName)

{

CString DirectoryFileName = ModuleDirectory + FileName;

int Width, Height, BPP;

FIBITMAP \*dib = GetBitmap(DirectoryFileName, Width, Height, BPP);

if(dib == NULL)

{

ErrorLog.Append("Error loading texture " + DirectoryFileName + "!\r\n");

return false;

}

GLenum Format = 0;

if(BPP == 32) Format = GL\_BGRA;

if(BPP == 24) Format = GL\_BGR;

if(Format == 0)

{

ErrorLog.Append("Unsupported texture format (%s)!\r\n", FileName);

FreeImage\_Unload(dib);

return false;

}

Destroy();

glGenTextures(1, &Texture);

glBindTexture(GL\_TEXTURE\_2D, Texture);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

if(GLEW\_EXT\_texture\_filter\_anisotropic)

{

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT, gl\_max\_texture\_max\_anisotropy\_ext);

}

glTexParameteri(GL\_TEXTURE\_2D, GL\_GENERATE\_MIPMAP, GL\_TRUE);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, Width, Height, 0, Format, GL\_UNSIGNED\_BYTE, FreeImage\_GetBits(dib));

glBindTexture(GL\_TEXTURE\_2D, 0);

FreeImage\_Unload(dib);

return true;

}

bool CTexture::LoadTextureCubeMap(char \*\*FileNames)

{

int Width, Height, BPP;

FIBITMAP \*dib[6];

bool Error = false;

for(int i = 0; i < 6; i++)

{

CString DirectoryFileName = ModuleDirectory + FileNames[i];

dib[i] = GetBitmap(DirectoryFileName, Width, Height, BPP);

if(dib[i] == NULL)

{

ErrorLog.Append("Error loading texture " + DirectoryFileName + "!\r\n");

Error = true;

}

}

if(Error)

{

for(int i = 0; i < 6; i++)

{

FreeImage\_Unload(dib[i]);

}

return false;

}

GLenum Format = 0;

if(BPP == 32) Format = GL\_BGRA;

if(BPP == 24) Format = GL\_BGR;

if(Format == 0)

{

ErrorLog.Append("Unsupported texture format (%s)!\r\n", FileNames[5]);

for(int i = 0; i < 6; i++)

{

FreeImage\_Unload(dib[i]);

}

return false;

}

Destroy();

glGenTextures(1, &Texture);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, Texture);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

if(GLEW\_EXT\_texture\_filter\_anisotropic)

{

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT, gl\_max\_texture\_max\_anisotropy\_ext);

}

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_GENERATE\_MIPMAP, GL\_TRUE);

for(int i = 0; i < 6; i++)

{

glTexImage2D(GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, 0, GL\_RGBA8, Width, Height, 0, Format, GL\_UNSIGNED\_BYTE, FreeImage\_GetBits(dib[i]));

}

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

for(int i = 0; i < 6; i++)

{

FreeImage\_Unload(dib[i]);

}

return true;

}

void CTexture::Destroy()

{

glDeleteTextures(1, &Texture);

Texture = 0;

}

FIBITMAP \*CTexture::GetBitmap(char \*FileName, int &Width, int &Height, int &BPP)

{

FREE\_IMAGE\_FORMAT fif = FreeImage\_GetFileType(FileName);

if(fif == FIF\_UNKNOWN)

{

fif = FreeImage\_GetFIFFromFilename(FileName);

}

if(fif == FIF\_UNKNOWN)

{

return NULL;

}

FIBITMAP \*dib = NULL;

if(FreeImage\_FIFSupportsReading(fif))

{

dib = FreeImage\_Load(fif, FileName);

}

if(dib != NULL)

{

int OriginalWidth = FreeImage\_GetWidth(dib);

int OriginalHeight = FreeImage\_GetHeight(dib);

Width = OriginalWidth;

Height = OriginalHeight;

if(Width == 0 || Height == 0)

{

FreeImage\_Unload(dib);

return NULL;

}

BPP = FreeImage\_GetBPP(dib);

if(Width > gl\_max\_texture\_size) Width = gl\_max\_texture\_size;

if(Height > gl\_max\_texture\_size) Height = gl\_max\_texture\_size;

if(!GLEW\_ARB\_texture\_non\_power\_of\_two)

{

Width = 1 << (int)floor((log((float)Width) / log(2.0f)) + 0.5f);

Height = 1 << (int)floor((log((float)Height) / log(2.0f)) + 0.5f);

}

if(Width != OriginalWidth || Height != OriginalHeight)

{

FIBITMAP \*rdib = FreeImage\_Rescale(dib, Width, Height, FILTER\_BICUBIC);

FreeImage\_Unload(dib);

dib = rdib;

}

}

return dib;

}

// ----------------------------------------------------------------------------------------------------------------------------

CShaderProgram::CShaderProgram()

{

SetDefaults();

}

CShaderProgram::~CShaderProgram()

{

}

CShaderProgram::operator GLuint ()

{

return Program;

}

bool CShaderProgram::Load(char \*VertexShaderFileName, char \*FragmentShaderFileName)

{

bool Error = false;

Destroy();

Error |= ((VertexShader = LoadShader(VertexShaderFileName, GL\_VERTEX\_SHADER)) == 0);

Error |= ((FragmentShader = LoadShader(FragmentShaderFileName, GL\_FRAGMENT\_SHADER)) == 0);

if(Error)

{

Destroy();

return false;

}

Program = glCreateProgram();

glAttachShader(Program, VertexShader);

glAttachShader(Program, FragmentShader);

glLinkProgram(Program);

int LinkStatus;

glGetProgramiv(Program, GL\_LINK\_STATUS, &LinkStatus);

if(LinkStatus == GL\_FALSE)

{

ErrorLog.Append("Error linking program (%s, %s)!\r\n", VertexShaderFileName, FragmentShaderFileName);

int InfoLogLength = 0;

glGetProgramiv(Program, GL\_INFO\_LOG\_LENGTH, &InfoLogLength);

if(InfoLogLength > 0)

{

char \*InfoLog = new char[InfoLogLength];

int CharsWritten = 0;

glGetProgramInfoLog(Program, InfoLogLength, &CharsWritten, InfoLog);

ErrorLog.Append(InfoLog);

delete [] InfoLog;

}

Destroy();

return false;

}

return true;

}

void CShaderProgram::Destroy()

{

glDetachShader(Program, VertexShader);

glDetachShader(Program, FragmentShader);

glDeleteShader(VertexShader);

glDeleteShader(FragmentShader);

glDeleteProgram(Program);

delete [] UniformLocations;

delete [] AttribLocations;

SetDefaults();

}

GLuint CShaderProgram::LoadShader(char \*FileName, GLenum Type)

{

CString DirectoryFileName = ModuleDirectory + FileName;

FILE \*File;

if(fopen\_s(&File, DirectoryFileName, "rb") != 0)

{

ErrorLog.Append("Error loading file " + DirectoryFileName + "!\r\n");

return 0;

}

fseek(File, 0, SEEK\_END);

long Size = ftell(File);

fseek(File, 0, SEEK\_SET);

char \*Source = new char[Size + 1];

fread(Source, 1, Size, File);

fclose(File);

Source[Size] = 0;

GLuint Shader = glCreateShader(Type);

glShaderSource(Shader, 1, (const char\*\*)&Source, NULL);

delete [] Source;

glCompileShader(Shader);

int CompileStatus;

glGetShaderiv(Shader, GL\_COMPILE\_STATUS, &CompileStatus);

if(CompileStatus == GL\_FALSE)

{

ErrorLog.Append("Error compiling shader %s!\r\n", FileName);

int InfoLogLength = 0;

glGetShaderiv(Shader, GL\_INFO\_LOG\_LENGTH, &InfoLogLength);

if(InfoLogLength > 0)

{

char \*InfoLog = new char[InfoLogLength];

int CharsWritten = 0;

glGetShaderInfoLog(Shader, InfoLogLength, &CharsWritten, InfoLog);

ErrorLog.Append(InfoLog);

delete [] InfoLog;

}

glDeleteShader(Shader);

return 0;

}

return Shader;

}

void CShaderProgram::SetDefaults()

{

VertexShader = 0;

FragmentShader = 0;

Program = 0;

UniformLocations = NULL;

AttribLocations = NULL;

}

// ----------------------------------------------------------------------------------------------------------------------------

CCamera::CCamera()

{

ViewMatrix = NULL;

ViewMatrixInverse = NULL;

X = vec3(1.0f, 0.0f, 0.0f);

Y = vec3(0.0f, 1.0f, 0.0f);

Z = vec3(0.0f, 0.0f, 1.0f);

Position = vec3(0.0f, 0.0f, 5.0f);

Reference = vec3(0.0f, 0.0f, 0.0f);

}

CCamera::~CCamera()

{

}

void CCamera::Look(const vec3 &Position, const vec3 &Reference, bool RotateAroundReference)

{

this->Position = Position;

this->Reference = Reference;

Z = normalize(Position - Reference);

X = normalize(cross(vec3(0.0f, 1.0f, 0.0f), Z));

Y = cross(Z, X);

if(!RotateAroundReference)

{

this->Reference = this->Position;

this->Position += Z \* 0.05f;

}

CalculateViewMatrix();

}

void CCamera::Move(const vec3 &Movement)

{

Position += Movement;

Reference += Movement;

CalculateViewMatrix();

}

vec3 CCamera::OnKeys(BYTE Keys, float FrameTime)

{

float Speed = 5.0f;

if(Keys & 0x40) Speed \*= 2.0f;

if(Keys & 0x80) Speed \*= 0.5f;

float Distance = Speed \* FrameTime;

vec3 Up(0.0f, 1.0f, 0.0f);

vec3 Right = X;

vec3 Forward = cross(Up, Right);

Up \*= Distance;

Right \*= Distance;

Forward \*= Distance;

vec3 Movement;

if(Keys & 0x01) Movement += Forward;

if(Keys & 0x02) Movement -= Forward;

if(Keys & 0x04) Movement -= Right;

if(Keys & 0x08) Movement += Right;

if(Keys & 0x10) Movement += Up;

if(Keys & 0x20) Movement -= Up;

return Movement;

}

void CCamera::OnMouseMove(int dx, int dy)

{

float Sensitivity = 0.25f;

Position -= Reference;

if(dx != 0)

{

float DeltaX = (float)dx \* Sensitivity;

X = rotate(X, DeltaX, vec3(0.0f, 1.0f, 0.0f));

Y = rotate(Y, DeltaX, vec3(0.0f, 1.0f, 0.0f));

Z = rotate(Z, DeltaX, vec3(0.0f, 1.0f, 0.0f));

}

if(dy != 0)

{

float DeltaY = (float)dy \* Sensitivity;

Y = rotate(Y, DeltaY, X);

Z = rotate(Z, DeltaY, X);

if(Y.y < 0.0f)

{

Z = vec3(0.0f, Z.y > 0.0f ? 1.0f : -1.0f, 0.0f);

Y = cross(Z, X);

}

}

Position = Reference + Z \* length(Position);

CalculateViewMatrix();

}

void CCamera::OnMouseWheel(float zDelta)

{

Position -= Reference;

if(zDelta < 0 && length(Position) < 500.0f)

{

Position += Position \* 0.1f;

}

if(zDelta > 0 && length(Position) > 0.05f)

{

Position -= Position \* 0.1f;

}

Position += Reference;

CalculateViewMatrix();

}

void CCamera::SetViewMatrixPointer(float \*ViewMatrix, float \*ViewMatrixInverse)

{

this->ViewMatrix = (mat4x4\*)ViewMatrix;

this->ViewMatrixInverse = (mat4x4\*)ViewMatrixInverse;

CalculateViewMatrix();

}

void CCamera::CalculateViewMatrix()

{

if(ViewMatrix != NULL)

{

\*ViewMatrix = mat4x4(X.x, Y.x, Z.x, 0.0f, X.y, Y.y, Z.y, 0.0f, X.z, Y.z, Z.z, 0.0f, -dot(X, Position), -dot(Y, Position), -dot(Z, Position), 1.0f);

if(ViewMatrixInverse != NULL)

{

\*ViewMatrixInverse = inverse(\*ViewMatrix);

}

}

}

// ----------------------------------------------------------------------------------------------------------------------------

CCamera Camera;

// ----------------------------------------------------------------------------------------------------------------------------

COpenGLRenderer::COpenGLRenderer()

{

WHMID = 0;

WireFrame = false;

Pause = false;

DropRadius = 4.0f / 128.0f;

Camera.SetViewMatrixPointer(&ViewMatrix, &ViewMatrixInverse);

}

COpenGLRenderer::~COpenGLRenderer()

{

}

bool COpenGLRenderer::Init()

{

// ------------------------------------------------------------------------------------------------------------------------

bool Error = false;

// ------------------------------------------------------------------------------------------------------------------------

if(!GLEW\_ARB\_texture\_non\_power\_of\_two)

{

ErrorLog.Append("GL\_ARB\_texture\_non\_power\_of\_two not supported!\r\n");

Error = true;

}

if(!GLEW\_ARB\_texture\_float)

{

ErrorLog.Append("GL\_ARB\_texture\_float not supported!\r\n");

Error = true;

}

if(!GLEW\_EXT\_framebuffer\_object)

{

ErrorLog.Append("GL\_EXT\_framebuffer\_object not supported!\r\n");

Error = true;

}

// ------------------------------------------------------------------------------------------------------------------------

char \*PoolSkyCubeMapFileNames[] = {"pool\\right.jpg", "pool\\left.jpg", "pool\\bottom.jpg", "pool\\top.jpg", "pool\\front.jpg", "pool\\back.jpg"};

Error |= !PoolSkyCubeMap.LoadTextureCubeMap(PoolSkyCubeMapFileNames);

// ------------------------------------------------------------------------------------------------------------------------

Error |= !WaterAddDropProgram.Load("wateradddrop.vs", "wateradddrop.fs");

Error |= !WaterHeightMapProgram.Load("waterheightmap.vs", "waterheightmap.fs");

Error |= !WaterNormalMapProgram.Load("waternormalmap.vs", "waternormalmap.fs");

Error |= !PhotonProgram.Load("photon.vs", "photon.fs");

Error |= !CubeMapHBlurProgram.Load("cube\_map\_blur.vs", "cube\_map\_hblur.fs");

Error |= !CubeMapVBlurProgram.Load("cube\_map\_blur.vs", "cube\_map\_vblur.fs");

Error |= !PoolSkyProgram.Load("poolsky.vs", "poolsky.fs");

Error |= !WaterProgram.Load("water.vs", "water.fs");

// ------------------------------------------------------------------------------------------------------------------------

if(Error)

{

return false;

}

// ------------------------------------------------------------------------------------------------------------------------

vec3 LightPosition = vec3(0.0f, 1.0f, 0.0f);

vec3 CubeMapNormals[6] = {

vec3(-1.0f, 0.0f, 0.0f),

vec3(1.0f, 0.0f, 0.0f),

vec3(0.0f, -1.0f, 0.0f),

vec3(0.0f, 1.0f, 0.0f),

vec3(0.0f, 0.0f, -1.0f),

vec3(0.0f, 0.0f, 1.0f),

};

mat4x4 BiasScaleMatrix = scale(1.0f / 2.0f, 1.0f / 3.0f, 1.0f) \* BiasMatrix;

mat4x4 PhotonsWorldToTextureMatrices[6] = {

translate(0.0f / 2.0f, 0.0f / 3.0f, 1.0f) \* BiasScaleMatrix \* mat4x4(vec4(0.0f, 0.0f, -1.0f, 0.0f), vec4(0.0f, 1.0f, 0.0f, 0.0f), vec4(1.0f, 0.0f, 0.0f, 0.0f), vec4(0.0f, 0.0f, 0.0f, 1.0f)),

translate(1.0f / 2.0f, 0.0f / 3.0f, 1.0f) \* BiasScaleMatrix \* mat4x4(vec4(0.0f, 0.0f, 1.0f, 0.0f), vec4(0.0f, 1.0f, 0.0f, 0.0f), vec4(-1.0f, 0.0f, 0.0f, 0.0f), vec4(0.0f, 0.0f, 0.0f, 1.0f)),

translate(0.0f / 2.0f, 1.0f / 3.0f, 1.0f) \* BiasScaleMatrix \* mat4x4(vec4(1.0f, 0.0f, 0.0f, 0.0f), vec4(0.0f, 0.0f, -1.0f, 0.0f), vec4(0.0f, 1.0f, 0.0f, 0.0f), vec4(0.0f, 0.0f, 0.0f, 1.0f)),

translate(1.0f / 2.0f, 1.0f / 3.0f, 1.0f) \* BiasScaleMatrix \* mat4x4(vec4(1.0f, 0.0f, 0.0f, 0.0f), vec4(0.0f, 0.0f, 1.0f, 0.0f), vec4(0.0f, -1.0f, 0.0f, 0.0f), vec4(0.0f, 0.0f, 0.0f, 1.0f)),

translate(0.0f / 2.0f, 2.0f / 3.0f, 1.0f) \* BiasScaleMatrix \* mat4x4(vec4(-1.0f, 0.0f, 0.0f, 0.0f), vec4(0.0f, 1.0f, 0.0f, 0.0f), vec4(0.0f, 0.0f, -1.0f, 0.0f), vec4(0.0f, 0.0f, 0.0f, 1.0f)),

translate(1.0f / 2.0f, 2.0f / 3.0f, 1.0f) \* BiasScaleMatrix \* mat4x4(vec4(1.0f, 0.0f, 0.0f, 0.0f), vec4(0.0f, 1.0f, 0.0f, 0.0f), vec4(0.0f, 0.0f, 1.0f, 0.0f), vec4(0.0f, 0.0f, 0.0f, 1.0f))

};

// ------------------------------------------------------------------------------------------------------------------------

glUseProgram(WaterHeightMapProgram);

glUniform1f(glGetUniformLocation(WaterHeightMapProgram, "ODWHMR"), 1.0f / (float)WHMR);

glUseProgram(0);

glUseProgram(WaterNormalMapProgram);

glUniform1f(glGetUniformLocation(WaterNormalMapProgram, "ODWNMR"), 1.0f / (float)WNMR);

glUniform1f(glGetUniformLocation(WaterNormalMapProgram, "WMSDWNMRM2"), 2.0f / (float)WNMR \* 2.0f);

glUseProgram(0);

glUseProgram(PhotonProgram);

glUniform1i(glGetUniformLocation(PhotonProgram, "WaterHeightMap"), 0);

glUniform1i(glGetUniformLocation(PhotonProgram, "WaterNormalMap"), 1);

glUniform1i(glGetUniformLocation(PhotonProgram, "PhotonsTexture"), 2);

glUniform3fv(glGetUniformLocation(PhotonProgram, "LightPosition"), 1, &LightPosition);

glUniform3fv(glGetUniformLocation(PhotonProgram, "CubeMapNormals"), 6, (float\*)CubeMapNormals);

glUniformMatrix4fv(glGetUniformLocation(PhotonProgram, "PhotonsWorldToTextureMatrices"), 6, GL\_FALSE, (float\*)PhotonsWorldToTextureMatrices);

glUseProgram(0);

glUseProgram(PoolSkyProgram);

glUniform1i(glGetUniformLocation(PoolSkyProgram, "PoolSkyCubeMap"), 0);

glUniform1i(glGetUniformLocation(PoolSkyProgram, "PhotonsCubeMap"), 1);

glUseProgram(0);

glUseProgram(WaterProgram);

glUniform1i(glGetUniformLocation(WaterProgram, "WaterHeightMap"), 0);

glUniform1i(glGetUniformLocation(WaterProgram, "WaterNormalMap"), 1);

glUniform1i(glGetUniformLocation(WaterProgram, "PoolSkyCubeMap"), 2);

glUniform1i(glGetUniformLocation(WaterProgram, "PhotonsCubeMap"), 3);

glUniform1f(glGetUniformLocation(WaterProgram, "ODWMS"), 1.0f / 2.0f);

glUniform3fv(glGetUniformLocation(WaterProgram, "LightPosition"), 1, &LightPosition);

glUniform3fv(glGetUniformLocation(WaterProgram, "CubeMapNormals"), 6, (float\*)CubeMapNormals);

glUseProgram(0);

// ------------------------------------------------------------------------------------------------------------------------

WaterAddDropProgram.UniformLocations = new GLuint[2];

WaterAddDropProgram.UniformLocations[0] = glGetUniformLocation(WaterAddDropProgram, "DropRadius");

WaterAddDropProgram.UniformLocations[1] = glGetUniformLocation(WaterAddDropProgram, "Position");

CubeMapHBlurProgram.UniformLocations = new GLuint[1];

CubeMapHBlurProgram.UniformLocations[0] = glGetUniformLocation(CubeMapHBlurProgram, "Offset");

CubeMapVBlurProgram.UniformLocations = new GLuint[1];

CubeMapVBlurProgram.UniformLocations[0] = glGetUniformLocation(CubeMapVBlurProgram, "Offset");

WaterProgram.UniformLocations = new GLuint[1];

WaterProgram.UniformLocations[0] = glGetUniformLocation(WaterProgram, "CameraPosition");

// ------------------------------------------------------------------------------------------------------------------------

glGenTextures(2, WaterHeightMaps);

vec4 \*Heights = new vec4[WHMR \* WHMR];

for(int i = 0; i < WHMR \* WHMR; i++)

{

Heights[i] = vec4(0.0f, 0.0f, 0.0f, 0.0f);

}

for(int i = 0; i < 2; i++)

{

glBindTexture(GL\_TEXTURE\_2D, WaterHeightMaps[i]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT, gl\_max\_texture\_max\_anisotropy\_ext);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA16F, WHMR, WHMR, 0, GL\_RGBA, GL\_FLOAT, Heights);

glGenerateMipmapEXT(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, 0);

}

delete [] Heights;

// ------------------------------------------------------------------------------------------------------------------------

glGenTextures(1, &WaterNormalMap);

vec4 \*Normals = new vec4[WNMR \* WNMR];

for(int i = 0; i < WNMR \* WNMR; i++)

{

Normals[i] = vec4(0.0f, 1.0f, 0.0f, 1.0f);

}

glBindTexture(GL\_TEXTURE\_2D, WaterNormalMap);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT, gl\_max\_texture\_max\_anisotropy\_ext);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA16F, WNMR, WNMR, 0, GL\_RGBA, GL\_FLOAT, Normals);

glGenerateMipmapEXT(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, 0);

delete [] Normals;

// ------------------------------------------------------------------------------------------------------------------------

glGenTextures(1, &PhotonsTexture);

vec4 \*Colors = new vec4[PCMR \* 2 \* PCMR \* 3];

for(int i = 0; i < PCMR \* 2 \* PCMR \* 3; i++)

{

Colors[i] = vec4(0.0f, 0.0f, 0.0f, 1.0f);

}

glBindTexture(GL\_TEXTURE\_2D, PhotonsTexture);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, PCMR \* 2, PCMR \* 3, 0, GL\_RGBA, GL\_FLOAT, Colors);

glBindTexture(GL\_TEXTURE\_2D, 0);

delete [] Colors;

// ------------------------------------------------------------------------------------------------------------------------

glGenTextures(2, PhotonsTempCubeMaps);

for(int i = 0; i < 2; i++)

{

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PhotonsTempCubeMaps[i]);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

for(int i = 0; i < 6; i++)

{

glTexImage2D(GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, 0, GL\_RGBA8, PCMR, PCMR, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, NULL);

}

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

}

// ------------------------------------------------------------------------------------------------------------------------

glGenTextures(1, &PhotonsCubeMap);

Colors = new vec4[PCMR \* PCMR];

for(int i = 0; i < PCMR \* PCMR; i++)

{

Colors[i] = vec4(0.0f, 0.0f, 0.0f, 1.0f);

}

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PhotonsCubeMap);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT, gl\_max\_texture\_max\_anisotropy\_ext);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

for(int i = 0; i < 6; i++)

{

glTexImage2D(GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, 0, GL\_RGBA8, PCMR, PCMR, 0, GL\_RGBA, GL\_FLOAT, Colors);

}

glGenerateMipmapEXT(GL\_TEXTURE\_CUBE\_MAP);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

delete [] Colors;

// ------------------------------------------------------------------------------------------------------------------------

glGenBuffers(1, &PhotonsVBO);

int PMR = PCMR, PMRP1 = PMR + 1;

PhotonsCount = PMRP1 \* PMRP1;

vec3 \*Photons = new vec3[PhotonsCount];

float WMSDPMR = 2.0f / (float)PMR;

int i = 0;

for(int y = 0; y <= PMR; y++)

{

for(int x = 0; x <= PMR; x++)

{

Photons[i++] = vec3((float)x \* WMSDPMR - 1.0f, 0.0f, 1.0f - (float)y \* WMSDPMR);

}

}

glBindBuffer(GL\_ARRAY\_BUFFER, PhotonsVBO);

glBufferData(GL\_ARRAY\_BUFFER, PhotonsCount \* 12, Photons, GL\_STATIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

delete [] Photons;

// ------------------------------------------------------------------------------------------------------------------------

glGenBuffers(1, &PoolSkyVBO);

float PoolSkyVertices[] =

{ // x, y, z, x, y, z, x, y, z, x, y, z

1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, -1.0f, // +X

-1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f, // -X

-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, // +Y

-1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, // -Y

1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, // +Z

-1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f // -Z

};

glBindBuffer(GL\_ARRAY\_BUFFER, PoolSkyVBO);

glBufferData(GL\_ARRAY\_BUFFER, 288, PoolSkyVertices, GL\_STATIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

// ------------------------------------------------------------------------------------------------------------------------

glGenBuffers(1, &WaterVBO);

int WMRP1 = WMR + 1;

vec3 \*Vertices = new vec3[WMRP1 \* WMRP1];

float WMSDWMR = 2.0f / (float)WMR;

for(int y = 0; y <= WMR; y++)

{

for(int x = 0; x <= WMR; x++)

{

Vertices[WMRP1 \* y + x].x = x \* WMSDWMR - 1.0f;

Vertices[WMRP1 \* y + x].y = 0.0f;

Vertices[WMRP1 \* y + x].z = 1.0f - y \* WMSDWMR;

}

}

CBuffer Quads;

for(int y = 0; y < WMR; y++)

{

int yp1 = y + 1;

for(int x = 0; x < WMR; x++)

{

int xp1 = x + 1;

int a = WMRP1 \* y + x;

int b = WMRP1 \* y + xp1;

int c = WMRP1 \* yp1 + xp1;

int d = WMRP1 \* yp1 + x;

Quads.AddData(&Vertices[a], 12);

Quads.AddData(&Vertices[b], 12);

Quads.AddData(&Vertices[c], 12);

Quads.AddData(&Vertices[d], 12);

}

}

glBindBuffer(GL\_ARRAY\_BUFFER, WaterVBO);

glBufferData(GL\_ARRAY\_BUFFER, Quads.GetDataSize(), Quads.GetData(), GL\_STATIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

QuadsVerticesCount = Quads.GetDataSize() / 12;

Quads.Empty();

delete [] Vertices;

// ------------------------------------------------------------------------------------------------------------------------

glGenFramebuffersEXT(1, &FBO);

// ------------------------------------------------------------------------------------------------------------------------

Camera.Look(vec3(0.0f, 1.0f, 2.5f), vec3(0.0f, -0.5f, 0.0f), true);

// ------------------------------------------------------------------------------------------------------------------------

srand(GetTickCount());

// ------------------------------------------------------------------------------------------------------------------------

return true;

}

void COpenGLRenderer::Render(float FrameTime)

{

// add drops --------------------------------------------------------------------------------------------------------------

if(!Pause)

{

static DWORD LastTime = GetTickCount();

DWORD Time = GetTickCount();

if(Time - LastTime > 100)

{

LastTime = Time;

AddDrop(2.0f \* (float)rand() / (float)RAND\_MAX - 1.0f, 1.0f - 2.0f \* (float)rand() / (float)RAND\_MAX, 4.0f / 128.0f \* (float)rand() / (float)RAND\_MAX);

}

}

// update water surface and generate photon cube map ----------------------------------------------------------------------

static DWORD LastTime = GetTickCount();

DWORD Time = GetTickCount();

if(Time - LastTime >= 16)

{

LastTime = Time;

// update water height map --------------------------------------------------------------------------------------------

glViewport(0, 0, WHMR, WHMR);

GLuint whmid = (WHMID + 1) % 2;

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, FBO);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_COLOR\_ATTACHMENT0\_EXT, GL\_TEXTURE\_2D, WaterHeightMaps[whmid], 0);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_DEPTH\_ATTACHMENT\_EXT, GL\_TEXTURE\_2D, 0, 0);

glBindTexture(GL\_TEXTURE\_2D, WaterHeightMaps[WHMID]);

glUseProgram(WaterHeightMapProgram);

glBegin(GL\_QUADS);

glVertex2f(0.0f, 0.0f);

glVertex2f(1.0f, 0.0f);

glVertex2f(1.0f, 1.0f);

glVertex2f(0.0f, 1.0f);

glEnd();

glUseProgram(0);

glBindTexture(GL\_TEXTURE\_2D, 0);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, 0);

glBindTexture(GL\_TEXTURE\_2D, WaterHeightMaps[whmid]);

glGenerateMipmapEXT(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, 0);

++WHMID %= 2;

// update water normal map --------------------------------------------------------------------------------------------

glViewport(0, 0, WNMR, WNMR);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, FBO);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_COLOR\_ATTACHMENT0\_EXT, GL\_TEXTURE\_2D, WaterNormalMap, 0);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_DEPTH\_ATTACHMENT\_EXT, GL\_TEXTURE\_2D, 0, 0);

glBindTexture(GL\_TEXTURE\_2D, WaterHeightMaps[WHMID]);

glUseProgram(WaterNormalMapProgram);

glBegin(GL\_QUADS);

glVertex2f(0.0f, 0.0f);

glVertex2f(1.0f, 0.0f);

glVertex2f(1.0f, 1.0f);

glVertex2f(0.0f, 1.0f);

glEnd();

glUseProgram(0);

glBindTexture(GL\_TEXTURE\_2D, 0);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, 0);

glBindTexture(GL\_TEXTURE\_2D, WaterNormalMap);

glGenerateMipmapEXT(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, 0);

// render photons into photons texture --------------------------------------------------------------------------------

glViewport(0, 0, PCMR \* 2, PCMR \* 3);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, FBO);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_COLOR\_ATTACHMENT0\_EXT, GL\_TEXTURE\_2D, PhotonsTexture, 0);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_DEPTH\_ATTACHMENT\_EXT, GL\_TEXTURE\_2D, 0, 0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glActiveTexture(GL\_TEXTURE0); glBindTexture(GL\_TEXTURE\_2D, WaterHeightMaps[WHMID]);

glActiveTexture(GL\_TEXTURE1); glBindTexture(GL\_TEXTURE\_2D, WaterNormalMap);

glActiveTexture(GL\_TEXTURE2); glBindTexture(GL\_TEXTURE\_2D, PhotonsTexture);

glUseProgram(PhotonProgram);

glBindBuffer(GL\_ARRAY\_BUFFER, PhotonsVBO);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 12, (void\*)0);

glDrawArrays(GL\_POINTS, 0, PhotonsCount);

glDisableClientState(GL\_VERTEX\_ARRAY);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glUseProgram(0);

glActiveTexture(GL\_TEXTURE2); glBindTexture(GL\_TEXTURE\_2D, 0);

glActiveTexture(GL\_TEXTURE1); glBindTexture(GL\_TEXTURE\_2D, 0);

glActiveTexture(GL\_TEXTURE0); glBindTexture(GL\_TEXTURE\_2D, 0);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, 0);

// --------------------------------------------------------------------------------------------------------------------

static vec2 PhotonsTextureCoords[] = {

vec2(0.0f / 2.0f, 0.0f / 3.0f), vec2(1.0f / 2.0f, 0.0f / 3.0f), vec2(1.0f / 2.0f, 1.0f / 3.0f), vec2(0.0f / 2.0f, 1.0f / 3.0f),

vec2(1.0f / 2.0f, 0.0f / 3.0f), vec2(2.0f / 2.0f, 0.0f / 3.0f), vec2(2.0f / 2.0f, 1.0f / 3.0f), vec2(1.0f / 2.0f, 1.0f / 3.0f),

vec2(1.0f / 2.0f, 1.0f / 3.0f), vec2(2.0f / 2.0f, 1.0f / 3.0f), vec2(2.0f / 2.0f, 2.0f / 3.0f), vec2(1.0f / 2.0f, 2.0f / 3.0f),

vec2(0.0f / 2.0f, 1.0f / 3.0f), vec2(1.0f / 2.0f, 1.0f / 3.0f), vec2(1.0f / 2.0f, 2.0f / 3.0f), vec2(0.0f / 2.0f, 2.0f / 3.0f),

vec2(1.0f / 2.0f, 2.0f / 3.0f), vec2(2.0f / 2.0f, 2.0f / 3.0f), vec2(2.0f / 2.0f, 3.0f / 3.0f), vec2(1.0f / 2.0f, 3.0f / 3.0f),

vec2(0.0f / 2.0f, 2.0f / 3.0f), vec2(1.0f / 2.0f, 2.0f / 3.0f), vec2(1.0f / 2.0f, 3.0f / 3.0f), vec2(0.0f / 2.0f, 3.0f / 3.0f),

};

// copy photons texture to photons temp cube map 1 --------------------------------------------------------------------

glViewport(0, 0, PCMR, PCMR);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

for(int i = 0; i < 6; i++)

{

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, FBO);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_COLOR\_ATTACHMENT0\_EXT, GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, PhotonsTempCubeMaps[0], 0);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_DEPTH\_ATTACHMENT\_EXT, GL\_TEXTURE\_2D, 0, 0);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, PhotonsTexture);

glBegin(GL\_QUADS);

glTexCoord2fv(&PhotonsTextureCoords[i \* 4 + 0]); glVertex2f(-1.0f, -1.0f);

glTexCoord2fv(&PhotonsTextureCoords[i \* 4 + 1]); glVertex2f(1.0f, -1.0f);

glTexCoord2fv(&PhotonsTextureCoords[i \* 4 + 2]); glVertex2f(1.0f, 1.0f);

glTexCoord2fv(&PhotonsTextureCoords[i \* 4 + 3]); glVertex2f(-1.0f, 1.0f);

glEnd();

glBindTexture(GL\_TEXTURE\_2D, 0);

glDisable(GL\_TEXTURE\_2D);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, 0);

}

// --------------------------------------------------------------------------------------------------------------------

static vec3 PhotonsCubeMapCoords[] = {

vec3( 1.0f, 1.0f, 1.0f), vec3( 1.0f, 1.0f, -1.0f), vec3( 1.0f, -1.0f, -1.0f), vec3( 1.0f, -1.0f, 1.0f),

vec3(-1.0f, 1.0f, -1.0f), vec3(-1.0f, 1.0f, 1.0f), vec3(-1.0f, -1.0f, 1.0f), vec3(-1.0f, -1.0f, -1.0f),

vec3(-1.0f, -1.0f, 1.0f), vec3( 1.0f, -1.0f, 1.0f), vec3( 1.0f, -1.0f, -1.0f), vec3(-1.0f, -1.0f, -1.0f),

vec3(-1.0f, 1.0f, -1.0f), vec3( 1.0f, 1.0f, -1.0f), vec3( 1.0f, 1.0f, 1.0f), vec3(-1.0f, 1.0f, 1.0f),

vec3( 1.0f, 1.0f, -1.0f), vec3(-1.0f, 1.0f, -1.0f), vec3(-1.0f, -1.0f, -1.0f), vec3( 1.0f, -1.0f, -1.0f),

vec3(-1.0f, 1.0f, 1.0f), vec3( 1.0f, 1.0f, 1.0f), vec3( 1.0f, -1.0f, 1.0f), vec3(-1.0f, -1.0f, 1.0f)

};

static float ODPCMR = 1.0f / (float)PCMR;

static vec3 PhotonsCubeMapHorizontalOffsets[] = {

vec3(0.0f, 0.0f, -ODPCMR), vec3(0.0f, 0.0f, ODPCMR),

vec3(ODPCMR, 0.0f, 0.0f), vec3(ODPCMR, 0.0f, 0.0f),

vec3(-ODPCMR, 0.0f, 0.0f), vec3(ODPCMR, 0.0f, 0.0f)

};

static vec3 PhotonsCubeMapVerticalOffsets[] = {

vec3(0.0f, -ODPCMR, 0.0f), vec3(0.0f, -ODPCMR, 0.0f),

vec3(0.0f, 0.0f, -ODPCMR), vec3(0.0f, 0.0f, ODPCMR),

vec3(0.0f, -ODPCMR, 0.0f), vec3(0.0f, -ODPCMR, 0.0f)

};

// blur photons temp cube map 1 horizontally --------------------------------------------------------------------------

for(int i = 0; i < 6; i++)

{

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, FBO);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_COLOR\_ATTACHMENT0\_EXT, GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, PhotonsTempCubeMaps[1], 0);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_DEPTH\_ATTACHMENT\_EXT, GL\_TEXTURE\_2D, 0, 0);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PhotonsTempCubeMaps[0]);

glUseProgram(CubeMapHBlurProgram);

glUniform3fv(CubeMapHBlurProgram.UniformLocations[0], 1, &PhotonsCubeMapHorizontalOffsets[i]);

glBegin(GL\_QUADS);

glTexCoord3fv(&PhotonsCubeMapCoords[i \* 4 + 0]); glVertex2f(-1.0f, -1.0f);

glTexCoord3fv(&PhotonsCubeMapCoords[i \* 4 + 1]); glVertex2f(1.0f, -1.0f);

glTexCoord3fv(&PhotonsCubeMapCoords[i \* 4 + 2]); glVertex2f(1.0f, 1.0f);

glTexCoord3fv(&PhotonsCubeMapCoords[i \* 4 + 3]); glVertex2f(-1.0f, 1.0f);

glEnd();

glUseProgram(0);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, 0);

}

// blur photons temp cube map 2 vertically ----------------------------------------------------------------------------

for(int i = 0; i < 6; i++)

{

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, FBO);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_COLOR\_ATTACHMENT0\_EXT, GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, PhotonsCubeMap, 0);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_DEPTH\_ATTACHMENT\_EXT, GL\_TEXTURE\_2D, 0, 0);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PhotonsTempCubeMaps[1]);

glUseProgram(CubeMapVBlurProgram);

glUniform3fv(CubeMapVBlurProgram.UniformLocations[0], 1, &PhotonsCubeMapVerticalOffsets[i]);

glBegin(GL\_QUADS);

glTexCoord3fv(&PhotonsCubeMapCoords[i \* 4 + 0]); glVertex2f(-1.0f, -1.0f);

glTexCoord3fv(&PhotonsCubeMapCoords[i \* 4 + 1]); glVertex2f(1.0f, -1.0f);

glTexCoord3fv(&PhotonsCubeMapCoords[i \* 4 + 2]); glVertex2f(1.0f, 1.0f);

glTexCoord3fv(&PhotonsCubeMapCoords[i \* 4 + 3]); glVertex2f(-1.0f, 1.0f);

glEnd();

glUseProgram(0);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, 0);

}

// generate mipmaps ---------------------------------------------------------------------------------------------------

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PhotonsCubeMap);

glGenerateMipmapEXT(GL\_TEXTURE\_CUBE\_MAP);

glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

}

// render pool sky mesh ---------------------------------------------------------------------------------------------------

glViewport(0, 0, Width, Height);

glMatrixMode(GL\_PROJECTION);

glLoadMatrixf(&ProjectionMatrix);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_CULL\_FACE);

glMatrixMode(GL\_MODELVIEW);

glLoadMatrixf(&ViewMatrix);

if(WireFrame)

{

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

}

glActiveTexture(GL\_TEXTURE0); glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PoolSkyCubeMap);

glActiveTexture(GL\_TEXTURE1); glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PhotonsCubeMap);

glUseProgram(PoolSkyProgram);

glBindBuffer(GL\_ARRAY\_BUFFER, PoolSkyVBO);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 12, (void\*)0);

glDrawArrays(GL\_QUADS, 0, 24);

glDisableClientState(GL\_VERTEX\_ARRAY);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glUseProgram(0);

glActiveTexture(GL\_TEXTURE1); glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

glActiveTexture(GL\_TEXTURE0); glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

glDisable(GL\_CULL\_FACE);

// render water surface ---------------------------------------------------------------------------------------------------

glActiveTexture(GL\_TEXTURE0); glBindTexture(GL\_TEXTURE\_2D, WaterHeightMaps[WHMID]);

glActiveTexture(GL\_TEXTURE1); glBindTexture(GL\_TEXTURE\_2D, WaterNormalMap);

glActiveTexture(GL\_TEXTURE2); glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PoolSkyCubeMap);

glActiveTexture(GL\_TEXTURE3); glBindTexture(GL\_TEXTURE\_CUBE\_MAP, PhotonsCubeMap);

glUseProgram(WaterProgram);

glUniform3fv(WaterProgram.UniformLocations[0], 1, &Camera.Position);

glBindBuffer(GL\_ARRAY\_BUFFER, WaterVBO);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 12, (void\*)0);

glDrawArrays(GL\_QUADS, 0, QuadsVerticesCount);

glDisableClientState(GL\_VERTEX\_ARRAY);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glUseProgram(0);

glActiveTexture(GL\_TEXTURE3); glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

glActiveTexture(GL\_TEXTURE2); glBindTexture(GL\_TEXTURE\_CUBE\_MAP, 0);

glActiveTexture(GL\_TEXTURE1); glBindTexture(GL\_TEXTURE\_2D, 0);

glActiveTexture(GL\_TEXTURE0); glBindTexture(GL\_TEXTURE\_2D, 0);

if(WireFrame)

{

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

}

glDisable(GL\_DEPTH\_TEST);

}

void COpenGLRenderer::Resize(int Width, int Height)

{

this->Width = Width;

this->Height = Height;

ProjectionMatrix = perspective(45.0f, (float)Width / (float)Height, 0.125f, 512.0f);

ProjectionBiasMatrixInverse = inverse(ProjectionMatrix) \* BiasMatrixInverse;

}

void COpenGLRenderer::Destroy()

{

PoolSkyCubeMap.Destroy();

WaterAddDropProgram.Destroy();

WaterHeightMapProgram.Destroy();

WaterNormalMapProgram.Destroy();

PhotonProgram.Destroy();

CubeMapHBlurProgram.Destroy();

CubeMapVBlurProgram.Destroy();

PoolSkyProgram.Destroy();

WaterProgram.Destroy();

glDeleteTextures(2, WaterHeightMaps);

glDeleteTextures(1, &WaterNormalMap);

glDeleteTextures(1, &PhotonsTexture);

glDeleteTextures(2, PhotonsTempCubeMaps);

glDeleteTextures(1, &PhotonsCubeMap);

glDeleteBuffers(1, &PhotonsVBO);

glDeleteBuffers(1, &PoolSkyVBO);

glDeleteBuffers(1, &WaterVBO);

if(GLEW\_EXT\_framebuffer\_object)

{

glDeleteFramebuffersEXT(1, &FBO);

}

}

void COpenGLRenderer::AddDrop(float x, float y, float DropRadius)

{

if(x >= -1.0f && x <= 1.0f && y >= -1.0f && y <= 1.0f)

{

glViewport(0, 0, WMR, WMR);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, FBO);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_COLOR\_ATTACHMENT0\_EXT, GL\_TEXTURE\_2D, WaterHeightMaps[(WHMID + 1) % 2], 0);

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_DEPTH\_ATTACHMENT\_EXT, GL\_TEXTURE\_2D, 0, 0);

glBindTexture(GL\_TEXTURE\_2D, WaterHeightMaps[WHMID]);

glUseProgram(WaterAddDropProgram);

glUniform1f(WaterAddDropProgram.UniformLocations[0], DropRadius);

glUniform2fv(WaterAddDropProgram.UniformLocations[1], 1, &vec2(x \* 0.5f + 0.5f, 0.5f - y \* 0.5f));

glBegin(GL\_QUADS);

glVertex2f(0.0f, 0.0f);

glVertex2f(1.0f, 0.0f);

glVertex2f(1.0f, 1.0f);

glVertex2f(0.0f, 1.0f);

glEnd();

glUseProgram(0);

glBindTexture(GL\_TEXTURE\_2D, 0);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, 0);

++WHMID %= 2;

}

}

void COpenGLRenderer::AddDropByMouseClick(int x, int y)

{

float s = (float)x / (float)(Width - 1);

float t = 1.0f - (float)y / (float)(Height - 1);

vec4 Position = ViewMatrixInverse \* (ProjectionBiasMatrixInverse \* vec4(s, t, 0.5f, 1.0f));

Position /= Position.w;

vec3 Ray = normalize(\*(vec3\*)&Position - Camera.Position);

vec3 Normal = vec3(0.0f, 1.0f, 0.0f);

float D = -dot(Normal, vec3(0.0f, 0.0f, 0.0f));

float NdotR = -dot(Normal, Ray);

if(NdotR != 0.0f)

{

float Distance = (dot(Normal, Camera.Position) + D) / NdotR;

if(Distance > 0.0f)

{

vec3 Position = Ray \* Distance + Camera.Position;

AddDrop(Position.x, Position.z, DropRadius);

}

}

}

// ----------------------------------------------------------------------------------------------------------------------------

COpenGLRenderer OpenGLRenderer;

// ----------------------------------------------------------------------------------------------------------------------------

CString ModuleDirectory, ErrorLog;

// ----------------------------------------------------------------------------------------------------------------------------

void GetModuleDirectory()

{

char \*moduledirectory = new char[256];

GetModuleFileName(GetModuleHandle(NULL), moduledirectory, 256);

\*(strrchr(moduledirectory, '\\') + 1) = 0;

ModuleDirectory = moduledirectory;

delete [] moduledirectory;

}

// ----------------------------------------------------------------------------------------------------------------------------

COpenGLView::COpenGLView()

{

}

COpenGLView::~COpenGLView()

{

}

bool COpenGLView::Init(HINSTANCE hInstance, char \*Title, int Width, int Height, int Samples)

{

this->Title = Title;

this->Width = Width;

this->Height = Height;

WNDCLASSEX WndClassEx;

memset(&WndClassEx, 0, sizeof(WNDCLASSEX));

WndClassEx.cbSize = sizeof(WNDCLASSEX);

WndClassEx.style = CS\_OWNDC | CS\_HREDRAW | CS\_VREDRAW;

WndClassEx.lpfnWndProc = WndProc;

WndClassEx.hInstance = hInstance;

WndClassEx.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

WndClassEx.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

WndClassEx.hCursor = LoadCursor(NULL, IDC\_ARROW);

WndClassEx.lpszClassName = "Win32OpenGLWindowClass";

if(RegisterClassEx(&WndClassEx) == 0)

{

ErrorLog.Set("RegisterClassEx failed!");

return false;

}

DWORD Style = WS\_OVERLAPPEDWINDOW | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN;

hWnd = CreateWindowEx(WS\_EX\_APPWINDOW, WndClassEx.lpszClassName, Title, Style, 0, 0, Width, Height, NULL, NULL, hInstance, NULL);

if(hWnd == NULL)

{

ErrorLog.Set("CreateWindowEx failed!");

return false;

}

HDC hDC = GetDC(hWnd);

if(hDC == NULL)

{

ErrorLog.Set("GetDC failed!");

return false;

}

PIXELFORMATDESCRIPTOR pfd;

memset(&pfd, 0, sizeof(PIXELFORMATDESCRIPTOR));

pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);

pfd.nVersion = 1;

pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW | PFD\_SUPPORT\_OPENGL | PFD\_DOUBLEBUFFER;

pfd.iPixelType = PFD\_TYPE\_RGBA;

pfd.cColorBits = 32;

pfd.cDepthBits = 24;

pfd.iLayerType = PFD\_MAIN\_PLANE;

int PixelFormat = ChoosePixelFormat(hDC, &pfd);

if(PixelFormat == 0)

{

ErrorLog.Set("ChoosePixelFormat failed!");

return false;

}

static int MSAAPixelFormat = 0;

if(SetPixelFormat(hDC, MSAAPixelFormat == 0 ? PixelFormat : MSAAPixelFormat, &pfd) == FALSE)

{

ErrorLog.Set("SetPixelFormat failed!");

return false;

}

hGLRC = wglCreateContext(hDC);

if(hGLRC == NULL)

{

ErrorLog.Set("wglCreateContext failed!");

return false;

}

if(wglMakeCurrent(hDC, hGLRC) == FALSE)

{

ErrorLog.Set("wglMakeCurrent failed!");

return false;

}

if(glewInit() != GLEW\_OK)

{

ErrorLog.Set("glewInit failed!");

return false;

}

if(!GLEW\_VERSION\_2\_1)

{

ErrorLog.Set("OpenGL 2.1 not supported!");

return false;

}

if(MSAAPixelFormat == 0 && Samples > 0)

{

if(GLEW\_ARB\_multisample && WGLEW\_ARB\_pixel\_format)

{

while(Samples > 0)

{

UINT NumFormats = 0;

int PFAttribs[] =

{

WGL\_DRAW\_TO\_WINDOW\_ARB, GL\_TRUE,

WGL\_SUPPORT\_OPENGL\_ARB, GL\_TRUE,

WGL\_DOUBLE\_BUFFER\_ARB, GL\_TRUE,

WGL\_PIXEL\_TYPE\_ARB, WGL\_TYPE\_RGBA\_ARB,

WGL\_COLOR\_BITS\_ARB, 32,

WGL\_DEPTH\_BITS\_ARB, 24,

WGL\_ACCELERATION\_ARB, WGL\_FULL\_ACCELERATION\_ARB,

WGL\_SAMPLE\_BUFFERS\_ARB, GL\_TRUE,

WGL\_SAMPLES\_ARB, Samples,

0

};

if(wglChoosePixelFormatARB(hDC, PFAttribs, NULL, 1, &MSAAPixelFormat, &NumFormats) == TRUE && NumFormats > 0) break;

Samples--;

}

wglDeleteContext(hGLRC);

DestroyWindow(hWnd);

UnregisterClass(WndClassEx.lpszClassName, hInstance);

return Init(hInstance, Title, Width, Height, Samples);

}

else

{

Samples = 0;

}

}

this->Samples = Samples;

GetModuleDirectory();

glGetIntegerv(GL\_MAX\_TEXTURE\_SIZE, &gl\_max\_texture\_size);

if(GLEW\_EXT\_texture\_filter\_anisotropic)

{

glGetIntegerv(GL\_MAX\_TEXTURE\_MAX\_ANISOTROPY\_EXT, &gl\_max\_texture\_max\_anisotropy\_ext);

}

if(WGLEW\_EXT\_swap\_control)

{

wglSwapIntervalEXT(0);

}

return OpenGLRenderer.Init();

}

void COpenGLView::Show(bool Maximized)

{

RECT dRect, wRect, cRect;

GetWindowRect(GetDesktopWindow(), &dRect);

GetWindowRect(hWnd, &wRect);

GetClientRect(hWnd, &cRect);

wRect.right += Width - cRect.right;

wRect.bottom += Height - cRect.bottom;

wRect.right -= wRect.left;

wRect.bottom -= wRect.top;

wRect.left = dRect.right / 2 - wRect.right / 2;

wRect.top = dRect.bottom / 2 - wRect.bottom / 2;

MoveWindow(hWnd, wRect.left, wRect.top, wRect.right, wRect.bottom, FALSE);

ShowWindow(hWnd, Maximized ? SW\_SHOWMAXIMIZED : SW\_SHOWNORMAL);

}

void COpenGLView::MessageLoop()

{

MSG Msg;

while(GetMessage(&Msg, NULL, 0, 0) > 0)

{

TranslateMessage(&Msg);

DispatchMessage(&Msg);

}

}

void COpenGLView::Destroy()

{

if(GLEW\_VERSION\_2\_1)

{

OpenGLRenderer.Destroy();

}

wglDeleteContext(hGLRC);

DestroyWindow(hWnd);

}

void COpenGLView::OnKeyDown(UINT Key)

{

switch(Key)

{

case VK\_F1:

OpenGLRenderer.WireFrame = !OpenGLRenderer.WireFrame;

break;

case '1':

OpenGLRenderer.DropRadius = 4.0f / 256.0f;

break;

case '2':

OpenGLRenderer.DropRadius = 4.0f / 128.0f;

break;

case '3':

OpenGLRenderer.DropRadius = 4.0f / 64.0f;

break;

case '4':

OpenGLRenderer.DropRadius = 4.0f / 32.0f;

break;

case '5':

OpenGLRenderer.DropRadius = 4.0f / 16.0f;

break;

case VK\_SPACE:

OpenGLRenderer.Pause = !OpenGLRenderer.Pause;

break;

}

}

void COpenGLView::OnLButtonDown(int X, int Y)

{

OpenGLRenderer.AddDropByMouseClick(X, Y);

}

void COpenGLView::OnMouseMove(int X, int Y)

{

if(GetKeyState(VK\_RBUTTON) & 0x80)

{

Camera.OnMouseMove(LastX - X, LastY - Y);

LastX = X;

LastY = Y;

}

}

void COpenGLView::OnMouseWheel(short zDelta)

{

Camera.OnMouseWheel(zDelta);

}

void COpenGLView::OnPaint()

{

static DWORD LastFPSTime = GetTickCount(), LastFrameTime = LastFPSTime, FPS = 0;

PAINTSTRUCT ps;

HDC hDC = BeginPaint(hWnd, &ps);

DWORD Time = GetTickCount();

float FrameTime = (Time - LastFrameTime) \* 0.001f;

LastFrameTime = Time;

if(Time - LastFPSTime > 1000)

{

CString Text = Title;

if(OpenGLRenderer.Text[0] != 0)

{

Text.Append(" - " + OpenGLRenderer.Text);

}

Text.Append(" - %dx%d", Width, Height);

Text.Append(", ATF %dx", gl\_max\_texture\_max\_anisotropy\_ext);

Text.Append(", MSAA %dx", Samples);

Text.Append(", FPS: %d", FPS);

Text.Append(" - %s", glGetString(GL\_RENDERER));

SetWindowText(hWnd, Text);

LastFPSTime = Time;

FPS = 0;

}

else

{

FPS++;

}

BYTE Keys = 0x00;

if(GetKeyState('W') & 0x80) Keys |= 0x01;

if(GetKeyState('S') & 0x80) Keys |= 0x02;

if(GetKeyState('A') & 0x80) Keys |= 0x04;

if(GetKeyState('D') & 0x80) Keys |= 0x08;

if(GetKeyState('R') & 0x80) Keys |= 0x10;

if(GetKeyState('F') & 0x80) Keys |= 0x20;

if(GetKeyState(VK\_SHIFT) & 0x80) Keys |= 0x40;

if(GetKeyState(VK\_CONTROL) & 0x80) Keys |= 0x80;

if(Keys & 0x3F)

{

Camera.Move(Camera.OnKeys(Keys, FrameTime));

}

OpenGLRenderer.Render(FrameTime);

SwapBuffers(hDC);

EndPaint(hWnd, &ps);

InvalidateRect(hWnd, NULL, FALSE);

}

void COpenGLView::OnRButtonDown(int X, int Y)

{

LastX = X;

LastY = Y;

}

void COpenGLView::OnSize(int Width, int Height)

{

this->Width = Width;

this->Height = Height;

OpenGLRenderer.Resize(Width, Height);

}

// ----------------------------------------------------------------------------------------------------------------------------

COpenGLView OpenGLView;

// ----------------------------------------------------------------------------------------------------------------------------

LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)

{

switch(uiMsg)

{

case WM\_CLOSE:

PostQuitMessage(0);

break;

case WM\_KEYDOWN:

OpenGLView.OnKeyDown((UINT)wParam);

break;

case WM\_LBUTTONDOWN:

OpenGLView.OnLButtonDown(LOWORD(lParam), HIWORD(lParam));

break;

case WM\_MOUSEMOVE:

OpenGLView.OnMouseMove(LOWORD(lParam), HIWORD(lParam));

break;

case 0x020A: // WM\_MOUSWHEEL

OpenGLView.OnMouseWheel(HIWORD(wParam));

break;

case WM\_PAINT:

OpenGLView.OnPaint();

break;

case WM\_RBUTTONDOWN:

OpenGLView.OnRButtonDown(LOWORD(lParam), HIWORD(lParam));

break;

case WM\_SIZE:

OpenGLView.OnSize(LOWORD(lParam), HIWORD(lParam));

break;

default:

return DefWindowProc(hWnd, uiMsg, wParam, lParam);

}

return 0;

}

// ----------------------------------------------------------------------------------------------------------------------------

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR sCmdLine, int iShow)

{

char \*AppName = "Water waves caustic GPU algorithm";

if(OpenGLView.Init(hInstance, AppName, 800, 600, 0))

{

OpenGLView.Show();

OpenGLView.MessageLoop();

}

else

{

MessageBox(NULL, ErrorLog, AppName, MB\_OK | MB\_ICONERROR);

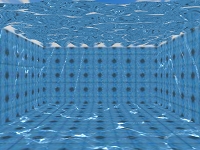
}

OpenGLView.Destroy();

return 0;

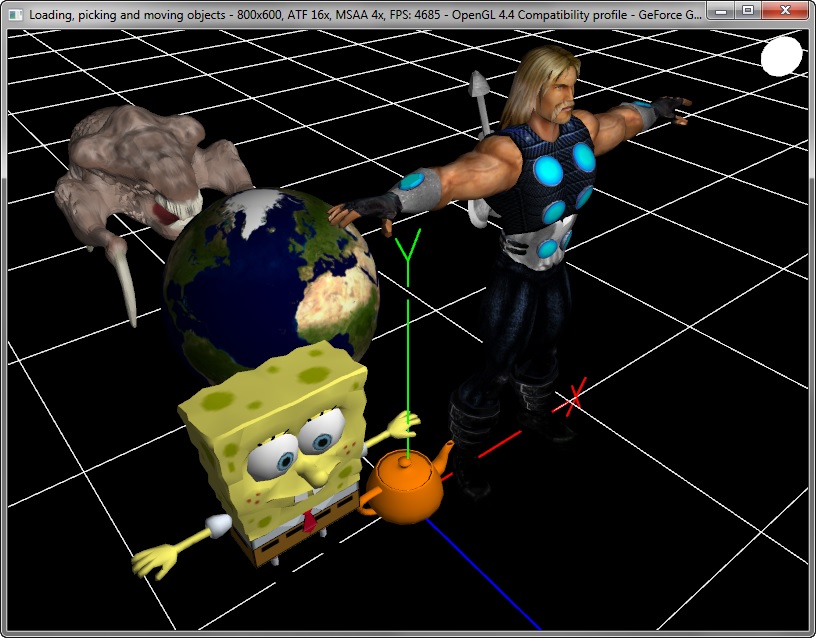
}

/////////////////////////////////插图///////////////////////////////////////



//////////////////////////////////////////////////////**Loading, picking and moving objects////////////////////////**

The CObject class is implemented with couple of useful methods to parse a Wavefront OBJ file, load data, create a 3D model, adjust and render it.  
  
The color picking method is used to determine which object was selected with the left mouse button click. If an object is selected, the pixel under the cursor is unprojected to world space and a plane equation is calculated.  
  
When the cursor is moved and the left mouse button is held down and an object is selected, the second unprojection is performed to get a ray. This ray is then intersected with the plane calculated earlier to derive the offset to shift the selected object.



VS:

1. glsl120shader.vs

#version 120

varying vec3 Position, Normal;

void main()

{

Position = (gl\_ModelViewMatrix \* gl\_Vertex).xyz;

Normal = gl\_NormalMatrix \* gl\_Normal;

gl\_FrontColor = gl\_Color;

gl\_TexCoord[0] = gl\_MultiTexCoord0;

gl\_Position = gl\_ModelViewProjectionMatrix \* gl\_Vertex;

}

2>glsl120shader.fs

#version 120

uniform sampler2D Texture;

uniform int Texturing;

varying vec3 Position, Normal;

void main()

{

vec3 LightDirection = gl\_LightSource[0].position.xyz - Position;

float LightDistance2 = dot(LightDirection, LightDirection);

float LightDistance = sqrt(LightDistance2);

LightDirection /= LightDistance;

float Attenuation = gl\_LightSource[0].constantAttenuation;

Attenuation += gl\_LightSource[0].linearAttenuation \* LightDistance;

Attenuation += gl\_LightSource[0].quadraticAttenuation \* LightDistance2;

float NdotLD = max(dot(normalize(Normal), LightDirection), 0.0);

vec3 Light = (gl\_LightSource[0].ambient.rgb + gl\_LightSource[0].diffuse.rgb \* NdotLD) / Attenuation;

gl\_FragColor.rgb = gl\_Color.rgb;

if(Texturing == 1) gl\_FragColor.rgb \*= texture2D(Texture, gl\_TexCoord[0].st).rgb;

gl\_FragColor.rgb \*= Light;

}

///////////////////////////Source/////////////////////////

**// ----------------------------------------------------------------------------------------------------------------------------**

**class COpenGLRenderer**

**{**

**protected:**

**int Width, Height;**

**mat4x4 Model, View, Projection, ProjectionBiasInverse;**

**CShaderProgram Shader;**

**int ObjectsCount, LightObjectID;**

**CObject \*Objects;**

**int SelectedObject;**

**float PlaneD;**

**vec3 SelectedPoint, PlaneNormal;**

**public:**

**bool ShowAxisGrid, WireFrame, Texturing;**

**public:**

**COpenGLRenderer();**

**~COpenGLRenderer();**

**bool Init();**

**void Render(float FrameTime);**

**void Resize(int Width, int Height);**

**void Destroy();**

**void MoveSelectedObject(int x, int y);**

**void SelectObject(int x, int y);**

**};**

**#include "opengl\_tutorials\_win32\_framework.h"**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CString ModuleDirectory, ErrorLog;**

**bool wgl\_context\_forward\_compatible = false;**

**int gl\_version = 0, use\_gl\_version = 21, gl\_max\_texture\_size = 0, gl\_max\_texture\_max\_anisotropy\_ext = 0;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CTexture::CTexture()**

**{**

**TextureID = 0;**

**}**

**CTexture::~CTexture()**

**{**

**}**

**CTexture::operator GLuint ()**

**{**

**return TextureID;**

**}**

**void CTexture::Delete()**

**{**

**glDeleteTextures(1, &TextureID);**

**TextureID = 0;**

**}**

**bool CTexture::LoadTexture2D(char \*Texture2DFileName)**

**{**

**CString FileName = ModuleDirectory + Texture2DFileName;**

**CString ErrorText = "Error loading file " + FileName + "! -> ";**

**FREE\_IMAGE\_FORMAT fif = FreeImage\_GetFileType(FileName);**

**if(fif == FIF\_UNKNOWN)**

**{**

**fif = FreeImage\_GetFIFFromFilename(FileName);**

**}**

**if(fif == FIF\_UNKNOWN)**

**{**

**ErrorLog.Append(ErrorText + "fif is FIF\_UNKNOWN" + "\r\n");**

**return false;**

**}**

**FIBITMAP \*dib = NULL;**

**if(FreeImage\_FIFSupportsReading(fif))**

**{**

**dib = FreeImage\_Load(fif, FileName);**

**}**

**if(dib == NULL)**

**{**

**ErrorLog.Append(ErrorText + "dib is NULL" + "\r\n");**

**return false;**

**}**

**int Width = FreeImage\_GetWidth(dib), oWidth = Width;**

**int Height = FreeImage\_GetHeight(dib), oHeight = Height;**

**int Pitch = FreeImage\_GetPitch(dib);**

**int BPP = FreeImage\_GetBPP(dib);**

**if(Width == 0 || Height == 0)**

**{**

**ErrorLog.Append(ErrorText + "Width or Height is 0" + "\r\n");**

**return false;**

**}**

**if(Width > gl\_max\_texture\_size) Width = gl\_max\_texture\_size;**

**if(Height > gl\_max\_texture\_size) Height = gl\_max\_texture\_size;**

**if(!GLEW\_ARB\_texture\_non\_power\_of\_two)**

**{**

**Width = 1 << (int)floor((log((float)Width) / log(2.0f)) + 0.5f);**

**Height = 1 << (int)floor((log((float)Height) / log(2.0f)) + 0.5f);**

**}**

**if(Width != oWidth || Height != oHeight)**

**{**

**FIBITMAP \*rdib = FreeImage\_Rescale(dib, Width, Height, FILTER\_BICUBIC);**

**FreeImage\_Unload(dib);**

**if((dib = rdib) == NULL)**

**{**

**ErrorLog.Append(ErrorText + "rdib is NULL" + "\r\n");**

**return false;**

**}**

**Pitch = FreeImage\_GetPitch(dib);**

**}**

**BYTE \*Data = FreeImage\_GetBits(dib);**

**if(Data == NULL)**

**{**

**ErrorLog.Append(ErrorText + "Data is NULL" + "\r\n");**

**return false;**

**}**

**GLenum Format = 0;**

**if(BPP == 32) Format = GL\_BGRA;**

**if(BPP == 24) Format = GL\_BGR;**

**if(Format == 0)**

**{**

**FreeImage\_Unload(dib);**

**ErrorLog.Append(ErrorText + "Format is 0" + "\r\n");**

**return false;**

**}**

**if(gl\_version < 12)**

**{**

**if(Format == GL\_BGRA) Format = GL\_RGBA;**

**if(Format == GL\_BGR) Format = GL\_RGB;**

**int bpp = BPP / 8;**

**BYTE \*line = Data;**

**for(int y = 0; y < Height; y++)**

**{**

**BYTE \*pixel = line;**

**for(int x = 0; x < Width; x++)**

**{**

**BYTE Temp = pixel[0];**

**pixel[0] = pixel[2];**

**pixel[2] = Temp;**

**pixel += bpp;**

**}**

**line += Pitch;**

**}**

**}**

**glDeleteTextures(1, &TextureID);**

**glGenTextures(1, &TextureID);**

**glBindTexture(GL\_TEXTURE\_2D, TextureID);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, gl\_version >= 14 ? GL\_LINEAR\_MIPMAP\_LINEAR : GL\_LINEAR);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);**

**if(GLEW\_EXT\_texture\_filter\_anisotropic)**

**{**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT, gl\_max\_texture\_max\_anisotropy\_ext);**

**}**

**if(gl\_version >= 14 && gl\_version <= 21)**

**{**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_GENERATE\_MIPMAP, GL\_TRUE);**

**}**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, Width, Height, 0, Format, GL\_UNSIGNED\_BYTE, Data);**

**if(gl\_version >= 30)**

**{**

**glGenerateMipmap(GL\_TEXTURE\_2D);**

**}**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**FreeImage\_Unload(dib);**

**return true;**

**}**

**// ----------------------------------------------------------------------------------------------------------------------------**

**#pragma warning(disable : 4996)**

**CObject::CObject()**

**{**

**SetDefaults();**

**}**

**CObject::~CObject()**

**{**

**}**

**void CObject::CreateSphere(float Radius, int Resolution, bool InvertNormals)**

**{**

**if(Resolution < 16) Resolution = 16;**

**if(Resolution % 4) Resolution += 4 - Resolution % 4;**

**TrianglesCount = Resolution \* (Resolution / 4 - 1) \* 4 + Resolution \* 2;**

**AllocateMemory();**

**float angle = (float)M\_PI \* 2.0f / Resolution;**

**vec3 a, b, c, d;**

**vec2 tca, tcb, tcc, tcd;**

**int i = 0;**

**float r = (float)Resolution, r4 = (float)Resolution / 2.0f;**

**for(int y = 0; y < Resolution / 4; y++)**

**{**

**for(int xz = 0; xz < Resolution; xz++)**

**{**

**if(y < Resolution / 4 - 1)**

**{**

**a = vec3(- sin(angle \* (xz + 0)) \* cos(angle \* (y + 0)), sin(angle \* (y + 0)), - cos(angle \* (xz + 0)) \* cos(angle \* (y + 0)));**

**b = vec3(- sin(angle \* (xz + 1)) \* cos(angle \* (y + 0)), sin(angle \* (y + 0)), - cos(angle \* (xz + 1)) \* cos(angle \* (y + 0)));**

**c = vec3(- sin(angle \* (xz + 1)) \* cos(angle \* (y + 1)), sin(angle \* (y + 1)), - cos(angle \* (xz + 1)) \* cos(angle \* (y + 1)));**

**d = vec3(- sin(angle \* (xz + 0)) \* cos(angle \* (y + 1)), sin(angle \* (y + 1)), - cos(angle \* (xz + 0)) \* cos(angle \* (y + 1)));**

**tca = TexCoords[i] = vec2((xz + 0) / r, 0.5f + (y + 0) / r4);**

**tcb = TexCoords[i] = vec2((xz + 1) / r, 0.5f + (y + 0) / r4);**

**tcc = TexCoords[i] = vec2((xz + 1) / r, 0.5f + (y + 1) / r4);**

**tcd = TexCoords[i] = vec2((xz + 0) / r, 0.5f + (y + 1) / r4);**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**TexCoords[i] = tcd; Normals[i] = d; Vertices[i++] = d \* Radius;**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**a.y = -a.y;**

**b.y = -b.y;**

**c.y = -c.y;**

**d.y = -d.y;**

**tca.y = 1.0f - tca.y;**

**tcb.y = 1.0f - tcb.y;**

**tcc.y = 1.0f - tcc.y;**

**tcd.y = 1.0f - tcd.y;**

**TexCoords[i] = tcd; Normals[i] = d; Vertices[i++] = d \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**TexCoords[i] = tcd; Normals[i] = d; Vertices[i++] = d \* Radius;**

**}**

**else**

**{**

**a = vec3(- sin(angle \* (xz + 0)) \* cos(angle \* (y + 0)), sin(angle \* (y + 0)), - cos(angle \* (xz + 0)) \* cos(angle \* (y + 0)));**

**b = vec3(- sin(angle \* (xz + 1)) \* cos(angle \* (y + 0)), sin(angle \* (y + 0)), - cos(angle \* (xz + 1)) \* cos(angle \* (y + 0)));**

**c = vec3(0.0f, 1.0f, 0.0f);**

**tca = TexCoords[i] = vec2((xz + 0) / r, 0.5f + (y + 0) / r4);**

**tcb = TexCoords[i] = vec2((xz + 1) / r, 0.5f + (y + 0) / r4);**

**tcc = TexCoords[i] = vec2((xz + 0.5f) / r, 1.0f);**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**a.y = -a.y;**

**b.y = -b.y;**

**c.y = -c.y;**

**tca.y = 1.0f - tca.y;**

**tcb.y = 1.0f - tcb.y;**

**tcc.y = 1.0f - tcc.y;**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**}**

**}**

**}**

**if(InvertNormals)**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**Normals[i] = -Normals[i];**

**}**

**}**

**Min = vec3(-Radius, -Radius, -Radius);**

**Max = vec3(Radius, Radius, Radius);**

**}**

**void CObject::Destroy()**

**{**

**if(gl\_version >= 15)**

**{**

**glDeleteBuffers(3, VBO);**

**}**

**Texture.Delete();**

**delete [] TexCoords;**

**delete [] Normals;**

**delete [] Vertices;**

**SetDefaults();**

**}**

**void CObject::InitVertexBuffers()**

**{**

**if(gl\_version >= 15)**

**{**

**glDeleteBuffers(3, VBO);**

**glGenBuffers(3, VBO);**

**glBindBuffer(GL\_ARRAY\_BUFFER, VBO[0]);**

**glBufferData(GL\_ARRAY\_BUFFER, TrianglesCount \* 3 \* 2 \* 4, TexCoords, GL\_STATIC\_DRAW);**

**glBindBuffer(GL\_ARRAY\_BUFFER, VBO[1]);**

**glBufferData(GL\_ARRAY\_BUFFER, TrianglesCount \* 3 \* 3 \* 4, Normals, GL\_STATIC\_DRAW);**

**glBindBuffer(GL\_ARRAY\_BUFFER, VBO[2]);**

**glBufferData(GL\_ARRAY\_BUFFER, TrianglesCount \* 3 \* 3 \* 4, Vertices, GL\_STATIC\_DRAW);**

**glBindBuffer(GL\_ARRAY\_BUFFER, 0);**

**}**

**}**

**bool CObject::Load(char \*Directory, char \*ObjFileName)**

**{**

**char \*ObjSource;**

**long ObjLength;**

**if(!ReadSource(Directory, ObjFileName, &ObjSource, ObjLength)) return false;**

**char \*Line, \*End = ObjSource + ObjLength;**

**float x, y, z;**

**int texcoordscount = 0, normalscount = 0, verticescount = 0;**

**int i1, i2, i3, i4, i5, i6, i7, i8, i9;**

**int v = 0, n = 0, t = 0, T = 0;**

**Line = ObjSource;**

**while(Line < End)**

**{**

**while(Line < End && (\*Line == ' ' || \*Line == '\t')) Line++;**

**if(Line[0] == 'm' && Line[1] == 't' && Line[2] == 'l' && Line[3] == 'l' && Line[4] == 'i' && Line[5] == 'b' && (Line[6] == ' ' || Line[6] == '\t'))**

**{**

**char \*MtlFileName = Line + 6;**

**while(MtlFileName < End && (\*MtlFileName == ' ' || \*MtlFileName == '\t')) MtlFileName++;**

**if(!ParseMtl(Directory, MtlFileName))**

**{**

**delete [] ObjSource;**

**return false;**

**}**

**}**

**else if(sscanf(Line, "vt %f %f", &x, &y) == 2)**

**{**

**texcoordscount++;**

**}**

**else if(sscanf(Line, "vn %f %f %f", &x, &y, &z) == 3)**

**{**

**normalscount++;**

**}**

**else if(sscanf(Line, "v %f %f %f", &x, &y, &z) == 3)**

**{**

**verticescount++;**

**}**

**else if(sscanf(Line, "f %d/%d/%d %d/%d/%d %d/%d/%d", &i1, &i2, &i3, &i4, &i5, &i6, &i7, &i8, &i9) == 9)**

**{**

**TrianglesCount++;**

**}**

**else if(sscanf(Line, "f %d//%d %d//%d %d//%d", &i1, &i2, &i3, &i4, &i5, &i6) == 6)**

**{**

**TrianglesCount++;**

**}**

**else if(sscanf(Line, "f %d/%d %d/%d %d/%d", &i1, &i2, &i3, &i4, &i5, &i6) == 6)**

**{**

**TrianglesCount++;**

**}**

**else if(sscanf(Line, "f %d %d %d", &i1, &i2, &i3) == 3)**

**{**

**TrianglesCount++;**

**}**

**while(Line < End && \*Line != 0) Line++;**

**while(Line < End && \*Line == 0) Line++;**

**}**

**vec2 \*texcoords = NULL;**

**vec3 \*normals = NULL;**

**vec3 \*vertices = NULL;**

**if(texcoordscount > 0) texcoords = new vec2[texcoordscount];**

**if(normalscount > 0) normals = new vec3[normalscount];**

**if(verticescount > 0) vertices = new vec3[verticescount];**

**AllocateMemory();**

**Line = ObjSource;**

**while(Line < End)**

**{**

**while(Line < End && (\*Line == ' ' || \*Line == '\t')) Line++;**

**if(sscanf(Line, "vt %f %f", &x, &y) == 2)**

**{**

**texcoords[t++] = vec2(x, y);**

**}**

**else if(sscanf(Line, "vn %f %f %f", &x, &y, &z) == 3)**

**{**

**normals[n++] = vec3(x, y ,z);**

**}**

**else if(sscanf(Line, "v %f %f %f", &x, &y, &z) == 3)**

**{**

**vertices[v++] = vec3(x, y ,z);**

**}**

**else if(sscanf(Line, "f %d/%d/%d %d/%d/%d %d/%d/%d", &i1, &i2, &i3, &i4, &i5, &i6, &i7, &i8, &i9) == 9)**

**{**

**TexCoords[T] = texcoords[i2 - 1];**

**Normals[T] = normals[i3 - 1];**

**Vertices[T++] = vertices[i1 - 1];**

**TexCoords[T] = texcoords[i5 - 1];**

**Normals[T] = normals[i6 - 1];**

**Vertices[T++] = vertices[i4 - 1];**

**TexCoords[T] = texcoords[i8 - 1];**

**Normals[T] = normals[i9 - 1];**

**Vertices[T++] = vertices[i7 - 1];**

**}**

**else if(sscanf(Line, "f %d//%d %d//%d %d//%d", &i1, &i2, &i3, &i4, &i5, &i6) == 6)**

**{**

**Normals[T] = normals[i2 - 1];**

**Vertices[T++] = vertices[i1 - 1];**

**Normals[T] = normals[i4 - 1];**

**Vertices[T++] = vertices[i3 - 1];**

**Normals[T] = normals[i6 - 1];**

**Vertices[T++] = vertices[i5 - 1];**

**}**

**else if(sscanf(Line, "f %d/%d %d/%d %d/%d", &i1, &i2, &i3, &i4, &i5, &i6) == 6)**

**{**

**if(texcoords != NULL && i1 - 1 < texcoordscount) TexCoords[T] = texcoords[i1 - 1];**

**if(texcoords != NULL && i2 - 1 < texcoordscount) TexCoords[T] = texcoords[i2 - 1];**

**if(normals != NULL && i1 - 1 < normalscount) Normals[T] = normals[i1 - 1];**

**if(normals != NULL && i2 - 1 < normalscount) Normals[T] = normals[i2 - 1];**

**Vertices[T++] = vertices[i1 - 1];**

**if(texcoords != NULL && i3 - 1 < texcoordscount) TexCoords[T] = texcoords[i3 - 1];**

**if(texcoords != NULL && i4 - 1 < texcoordscount) TexCoords[T] = texcoords[i4 - 1];**

**if(normals != NULL && i3 - 1 < normalscount) Normals[T] = normals[i3 - 1];**

**if(normals != NULL && i4 - 1 < normalscount) Normals[T] = normals[i4 - 1];**

**Vertices[T++] = vertices[i3 - 1];**

**if(texcoords != NULL && i5 - 1 < texcoordscount) TexCoords[T] = texcoords[i5 - 1];**

**if(texcoords != NULL && i6 - 1 < texcoordscount) TexCoords[T] = texcoords[i6 - 1];**

**if(normals != NULL && i5 - 1 < normalscount) Normals[T] = normals[i5 - 1];**

**if(normals != NULL && i6 - 1 < normalscount) Normals[T] = normals[i6 - 1];**

**Vertices[T++] = vertices[i5 - 1];**

**}**

**else if(sscanf(Line, "f %d %d %d", &i1, &i2, &i3) == 3)**

**{**

**if(texcoords != NULL && i1 - 1 < texcoordscount) TexCoords[T] = texcoords[i1 - 1];**

**if(normals != NULL && i1 - 1 < normalscount) Normals[T] = normals[i1 - 1];**

**Vertices[T++] = vertices[i1 - 1];**

**if(texcoords != NULL && i2 - 1 < texcoordscount) TexCoords[T] = texcoords[i2 - 1];**

**if(normals != NULL && i2 - 1 < normalscount) Normals[T] = normals[i2 - 1];**

**Vertices[T++] = vertices[i2 - 1];**

**if(texcoords != NULL && i3 - 1 < texcoordscount) TexCoords[T] = texcoords[i3 - 1];**

**if(normals != NULL && i3 - 1 < normalscount) Normals[T] = normals[i3 - 1];**

**Vertices[T++] = vertices[i3 - 1];**

**}**

**while(Line < End && \*Line != 0) Line++;**

**while(Line < End && \*Line == 0) Line++;**

**}**

**delete [] texcoords;**

**delete [] normals;**

**delete [] vertices;**

**delete [] ObjSource;**

**if(normalscount == 0)**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i += 3)**

**{**

**vec3 a = Vertices[i + 1] - Vertices[i];**

**vec3 b = Vertices[i + 2] - Vertices[i];**

**vec3 normal = normalize(cross(a, b));**

**Normals[i + 0] = normal;**

**Normals[i + 1] = normal;**

**Normals[i + 2] = normal;**

**}**

**}**

**GetMinMax();**

**return true;**

**}**

**void CObject::Rotate(float Angle, const vec3 &Axis)**

**{**

**mat4x4 Rotation = RotationMatrix(Angle, Axis);**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**Normals[i] = Rotation \* Normals[i];**

**Vertices[i] = Rotation \* Vertices[i];**

**}**

**GetMinMax();**

**}**

**void CObject::Scale(float ScaleFactor)**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**Vertices[i] \*= ScaleFactor;**

**}**

**Min \*= ScaleFactor;**

**Max \*= ScaleFactor;**

**}**

**void CObject::Translate(const vec3 &Translation)**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**Vertices[i] += Translation;**

**}**

**Min += Translation;**

**Max += Translation;**

**}**

**void CObject::AllocateMemory()**

**{**

**delete [] TexCoords;**

**delete [] Normals;**

**delete [] Vertices;**

**TexCoords = new vec2[TrianglesCount \* 3];**

**Normals = new vec3[TrianglesCount \* 3];**

**Vertices = new vec3[TrianglesCount \* 3];**

**}**

**void CObject::GetMinMax()**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**if(i == 0)**

**{**

**Min = Max = Vertices[i];**

**}**

**else**

**{**

**if(Min.x > Vertices[i].x) Min.x = Vertices[i].x;**

**if(Min.y > Vertices[i].y) Min.y = Vertices[i].y;**

**if(Min.z > Vertices[i].z) Min.z = Vertices[i].z;**

**if(Max.x < Vertices[i].x) Max.x = Vertices[i].x;**

**if(Max.y < Vertices[i].y) Max.y = Vertices[i].y;**

**if(Max.z < Vertices[i].z) Max.z = Vertices[i].z;**

**}**

**}**

**}**

**bool CObject::ParseMtl(char \*Directory, char \*MtlFileName)**

**{**

**char \*MtlSource;**

**long MtlLength;**

**if(!ReadSource(Directory, MtlFileName, &MtlSource, MtlLength)) return false;**

**char \*Line = MtlSource, \*End = MtlSource + MtlLength;**

**bool Error = false;**

**while(Line < End)**

**{**

**while(Line < End && (\*Line == ' ' || \*Line == '\t')) Line++;**

**if(Line[0] == 'm' && Line[1] == 'a' && Line[2] == 'p' && Line[3] == '\_' && Line[4] == 'K' && Line[5] == 'a' && (Line[6] == ' ' || Line[6] == '\t'))**

**{**

**char \*Texture2DFileName = Line + 6;**

**while(Texture2DFileName < End && (\*Texture2DFileName == ' ' || \*Texture2DFileName == '\t')) Texture2DFileName++;**

**Error |= !Texture.LoadTexture2D(CString(Directory) + Texture2DFileName);**

**}**

**while(Line < End && \*Line != 0) Line++;**

**while(Line < End && \*Line == 0) Line++;**

**}**

**delete [] MtlSource;**

**return !Error;**

**}**

**bool CObject::ReadSource(char \*Directory, char \*FileName, char \*\*Source, long &Length)**

**{**

**CString PathFileName = ModuleDirectory + Directory + FileName;**

**FILE \*File;**

**if(fopen\_s(&File, PathFileName, "rb") != 0)**

**{**

**ErrorLog.Append("Error loading file " + PathFileName + "!\r\n");**

**return false;**

**}**

**fseek(File, 0, SEEK\_END);**

**Length = ftell(File);**

**fseek(File, 0, SEEK\_SET);**

**\*Source = new char[Length + 1];**

**fread(\*Source, 1, Length, File);**

**(\*Source)[Length] = 0;**

**fclose(File);**

**for(long i = 0; i < Length; i++)**

**{**

**if((\*Source)[i] == '\r' || (\*Source)[i] == '\n') (\*Source)[i] = 0;**

**}**

**return true;**

**}**

**void CObject::SetDefaults()**

**{**

**Movable = true;**

**CullFace = true;**

**FrontFace = GL\_CCW;**

**TrianglesCount = 0;**

**for(int i = 0; i < 3; i++)**

**{**

**VBO[i] = 0;**

**}**

**TexCoords = NULL;**

**Normals = NULL;**

**Vertices = NULL;**

**Color = vec3(1.0f, 1.0f, 1.0f);**

**Position = vec3(0.0f, 0.0f, 0.0f);**

**}**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CShaderProgram::CShaderProgram()**

**{**

**SetDefaults();**

**}**

**CShaderProgram::~CShaderProgram()**

**{**

**}**

**CShaderProgram::operator GLuint ()**

**{**

**return Program;**

**}**

**void CShaderProgram::Delete()**

**{**

**delete [] UniformLocations;**

**glDetachShader(Program, VertexShader);**

**glDetachShader(Program, FragmentShader);**

**glDeleteShader(VertexShader);**

**glDeleteShader(FragmentShader);**

**glDeleteProgram(Program);**

**SetDefaults();**

**}**

**bool CShaderProgram::Load(char \*VertexShaderFileName, char \*FragmentShaderFileName)**

**{**

**if(UniformLocations || VertexShader || FragmentShader || Program)**

**{**

**Delete();**

**}**

**bool Error = false;**

**Error |= ((VertexShader = LoadShader(GL\_VERTEX\_SHADER, VertexShaderFileName)) == 0);**

**Error |= ((FragmentShader = LoadShader(GL\_FRAGMENT\_SHADER, FragmentShaderFileName)) == 0);**

**if(Error)**

**{**

**Delete();**

**return false;**

**}**

**Program = glCreateProgram();**

**glAttachShader(Program, VertexShader);**

**glAttachShader(Program, FragmentShader);**

**glLinkProgram(Program);**

**int Param = 0;**

**glGetProgramiv(Program, GL\_LINK\_STATUS, &Param);**

**if(Param == GL\_FALSE)**

**{**

**ErrorLog.Append("Error linking program (%s, %s)!\r\n", VertexShaderFileName, FragmentShaderFileName);**

**int InfoLogLength = 0;**

**glGetProgramiv(Program, GL\_INFO\_LOG\_LENGTH, &InfoLogLength);**

**if(InfoLogLength > 0)**

**{**

**char \*InfoLog = new char[InfoLogLength];**

**int CharsWritten = 0;**

**glGetProgramInfoLog(Program, InfoLogLength, &CharsWritten, InfoLog);**

**ErrorLog.Append(InfoLog);**

**delete [] InfoLog;**

**}**

**Delete();**

**return false;**

**}**

**return true;**

**}**

**GLuint CShaderProgram::LoadShader(GLenum Type, char \*ShaderFileName)**

**{**

**CString FileName = ModuleDirectory + ShaderFileName;**

**FILE \*File;**

**if(fopen\_s(&File, FileName, "rb") != 0)**

**{**

**ErrorLog.Append("Error loading file " + FileName + "!\r\n");**

**return 0;**

**}**

**fseek(File, 0, SEEK\_END);**

**long Size = ftell(File);**

**fseek(File, 0, SEEK\_SET);**

**char \*Source = new char[Size + 1];**

**fread(Source, 1, Size, File);**

**fclose(File);**

**Source[Size] = 0;**

**GLuint Shader;**

**Shader = glCreateShader(Type);**

**glShaderSource(Shader, 1, (const char\*\*)&Source, NULL);**

**delete [] Source;**

**glCompileShader(Shader);**

**int Param = 0;**

**glGetShaderiv(Shader, GL\_COMPILE\_STATUS, &Param);**

**if(Param == GL\_FALSE)**

**{**

**ErrorLog.Append("Error compiling shader %s!\r\n", ShaderFileName);**

**int InfoLogLength = 0;**

**glGetShaderiv(Shader, GL\_INFO\_LOG\_LENGTH, &InfoLogLength);**

**if(InfoLogLength > 0)**

**{**

**char \*InfoLog = new char[InfoLogLength];**

**int CharsWritten = 0;**

**glGetShaderInfoLog(Shader, InfoLogLength, &CharsWritten, InfoLog);**

**ErrorLog.Append(InfoLog);**

**delete [] InfoLog;**

**}**

**glDeleteShader(Shader);**

**return 0;**

**}**

**return Shader;**

**}**

**void CShaderProgram::SetDefaults()**

**{**

**UniformLocations = NULL;**

**VertexShader = 0;**

**FragmentShader = 0;**

**Program = 0;**

**}**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CCamera::CCamera()**

**{**

**View = NULL;**

**Reference = vec3(0.0f, 0.0f, 0.0f);**

**Position = vec3(0.0f, 0.0f, 5.0f);**

**X = vec3(1.0f, 0.0f, 0.0f);**

**Y = vec3(0.0f, 1.0f, 0.0f);**

**Z = vec3(0.0f, 0.0f, 1.0f);**

**}**

**CCamera::~CCamera()**

**{**

**}**

**void CCamera::CalculateViewMatrix()**

**{**

**if(View)**

**{**

**\*View = ViewMatrix(X, Y, Z, Position);**

**}**

**}**

**void CCamera::LookAt(vec3 Reference, vec3 Position, bool RotateAroundReference)**

**{**

**this->Reference = Reference;**

**this->Position = Position;**

**Z = normalize(Position - Reference);**

**X = normalize(cross(vec3(0.0f, 1.0f, 0.0f), Z));**

**Y = cross(Z, X);**

**if(!RotateAroundReference)**

**{**

**this->Reference = this->Position;**

**this->Position += Z \* 0.05f;**

**}**

**CalculateViewMatrix();**

**}**

**void CCamera::Move(vec3 Movement)**

**{**

**Reference += Movement;**

**Position += Movement;**

**CalculateViewMatrix();**

**}**

**vec3 CCamera::OnKeys(BYTE Keys, float FrameTime)**

**{**

**float Speed = 5.0f;**

**if(Keys & 0x40) // SHIFT**

**{**

**Speed \*= 2.0f;**

**}**

**float Distance = Speed \* FrameTime;**

**vec3 Up(0.0f, 1.0f, 0.0f);**

**vec3 Right = X;**

**vec3 Forward = cross(Up, Right);**

**Up \*= Distance;**

**Right \*= Distance;**

**Forward \*= Distance;**

**vec3 Movement;**

**if(Keys & 0x01) // W**

**{**

**Movement += Forward;**

**}**

**if(Keys & 0x02) // S**

**{**

**Movement -= Forward;**

**}**

**if(Keys & 0x04) // A**

**{**

**Movement -= Right;**

**}**

**if(Keys & 0x08) // D**

**{**

**Movement += Right;**

**}**

**if(Keys & 0x10) // R**

**{**

**Movement += Up;**

**}**

**if(Keys & 0x20) // F**

**{**

**Movement -= Up;**

**}**

**return Movement;**

**}**

**void CCamera::OnMouseMove(int dx, int dy)**

**{**

**float sensitivity = 0.25f;**

**float hangle = (float)dx \* sensitivity;**

**float vangle = (float)dy \* sensitivity;**

**Position -= Reference;**

**Y = rotate(Y, vangle, X);**

**Z = rotate(Z, vangle, X);**

**if(Y.y < 0.0f)**

**{**

**Z = vec3(0.0f, Z.y > 0.0f ? 1.0f : -1.0f, 0.0f);**

**Y = cross(Z, X);**

**}**

**X = rotate(X, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Y = rotate(Y, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Z = rotate(Z, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Position = Reference + Z \* length(Position);**

**CalculateViewMatrix();**

**}**

**void CCamera::OnMouseWheel(float zDelta)**

**{**

**Position -= Reference;**

**if(zDelta < 0 && length(Position) < 500.0f)**

**{**

**Position += Position \* 0.1f;**

**}**

**if(zDelta > 0 && length(Position) > 0.05f)**

**{**

**Position -= Position \* 0.1f;**

**}**

**Position += Reference;**

**CalculateViewMatrix();**

**}**

**void CCamera::SetViewMatrixPointer(float \*View)**

**{**

**this->View = (mat4x4\*)View;**

**CalculateViewMatrix();**

**}**

**CCamera Camera;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**COpenGLRenderer::COpenGLRenderer()**

**{**

**ShowAxisGrid = true;**

**WireFrame = false;**

**Texturing = true;**

**SelectedObject = -1;**

**Camera.SetViewMatrixPointer(&View);**

**}**

**COpenGLRenderer::~COpenGLRenderer()**

**{**

**}**

**bool COpenGLRenderer::Init()**

**{**

**/\*if(gl\_version < 21)**

**{**

**ErrorLog.Set("OpenGL 2.1 not supported!");**

**return false;**

**}\*/**

**bool Error = false;**

**ObjectsCount = 6;**

**Objects = new CObject[ObjectsCount];**

**Error |= !Objects[0].Load("Models\\Thor\\", "thor.obj");**

**Error |= !Objects[1].Load("Models\\Spongebob\\", "spongebob\_bind.obj");**

**Error |= !Objects[2].Load("Models\\Alien2\\", "alien2.obj");**

**Error |= !Objects[3].Load("Models\\Teapot\\", "teapot.obj");**

**Error |= !Objects[4].Texture.LoadTexture2D("earthmap.jpg");**

**if(gl\_version >= 21)**

**{**

**Error |= !Shader.Load("glsl120shader.vs", "glsl120shader.fs");**

**}**

**if(Error)**

**{**

**return false;**

**}**

**Objects[0].Rotate(-90.0f, vec3(0.0f, 1.0f, 0.0f));**

**Objects[0].Scale(1.75f / (Objects[0].Max.y - Objects[0].Min.y));**

**Objects[0].Translate(vec3(-(Objects[0].Min.x + Objects[0].Max.x) / 2.0f, -Objects[0].Min.y, -(Objects[0].Min.z + Objects[0].Max.z) / 2.0f));**

**Objects[0].Position = vec3(0.5f, 0.0f, 0.0f);**

**Objects[1].Scale(0.875f / (Objects[1].Max.y - Objects[1].Min.y));**

**Objects[1].Translate(vec3(-(Objects[1].Min.x + Objects[1].Max.x) / 2.0f, -Objects[1].Min.y, -(Objects[1].Min.z + Objects[1].Max.z) / 2.0f));**

**Objects[1].Position = vec3(-0.5f, 0.0f, 0.0f);**

**Objects[2].Scale(1.0f / (Objects[2].Max.y - Objects[2].Min.y));**

**Objects[2].Translate(vec3(-(Objects[2].Min.x + Objects[2].Max.x) / 2.0f, -Objects[2].Min.y, -(Objects[2].Min.z + Objects[2].Max.z) / 2.0f));**

**Objects[2].Position = vec3(0.0f, 0.0f, -2.5f);**

**Objects[3].Color = vec3(1.0f, 0.5f, 0.0f);**

**Objects[3].CullFace = false;**

**Objects[3].FrontFace = GL\_CW;**

**Objects[3].Scale(0.25f / (Objects[3].Max.y - Objects[3].Min.y));**

**Objects[3].Translate(vec3(-(Objects[3].Min.x + Objects[3].Max.x) / 2.0f, -Objects[3].Min.y, -(Objects[3].Min.z + Objects[3].Max.z) / 2.0f));**

**Objects[3].Position = vec3(0.0f, 0.0f, 0.0f);**

**Objects[4].CreateSphere(0.5f, 32);**

**Objects[4].Position = vec3(0.0f, 0.5f, -1.0f);**

**Objects[5].CreateSphere(0.03125f, 16, true);**

**Objects[5].Position = vec3(0.0f, 2.0f, 1.0f);**

**LightObjectID = 5;**

**for(int i = 0; i < ObjectsCount; i++)**

**{**

**Objects[i].InitVertexBuffers();**

**}**

**if(gl\_version >= 21)**

**{**

**Shader.UniformLocations = new GLuint[1];**

**Shader.UniformLocations[0] = glGetUniformLocation(Shader, "Texturing");**

**}**

**GLfloat LightModelAmbient[] = {0.0f, 0.0f, 0.0f, 1.0f};**

**glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, LightModelAmbient);**

**GLfloat LightAmbient[] = {0.25f, 0.25f, 0.25f, 1.0f};**

**glLightfv(GL\_LIGHT0, GL\_AMBIENT, LightAmbient);**

**GLfloat LightDiffuse[] = {0.75f, 0.75f, 0.75f, 1.0f};**

**glLightfv(GL\_LIGHT0, GL\_DIFFUSE, LightDiffuse);**

**GLfloat MaterialAmbient[] = {0.25f, 0.25f, 0.25f, 1.0f};**

**glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, MaterialAmbient);**

**GLfloat MaterialDiffuse[] = {0.75f, 0.75f, 0.75f, 1.0f};**

**glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, MaterialDiffuse);**

**glEnable(GL\_LIGHT0);**

**glEnable(GL\_COLOR\_MATERIAL);**

**Camera.LookAt(vec3(0.0f, 0.875f, 0.0f), vec3(0.0f, 0.875f, 2.5f), true);**

**return true;**

**}**

**void COpenGLRenderer::Render(float FrameTime)**

**{**

**int gl\_version = use\_gl\_version < ::gl\_version ? use\_gl\_version : ::gl\_version;**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadMatrixf(&View);**

**glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);**

**glEnable(GL\_DEPTH\_TEST);**

**// render axises and grid -------------------------------------------------------------------------------------------------**

**if(ShowAxisGrid)**

**{**

**glLineWidth(2.0f);**

**glBegin(GL\_LINES);**

**glColor3f(1.0f, 0.0f, 0.0f);**

**glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(1.0f, 0.0f, 0.0f);**

**glVertex3f(1.0f, 0.1f, 0.0f); glVertex3f(1.1f, -0.1f, 0.0f);**

**glVertex3f(1.1f, 0.1f, 0.0f); glVertex3f(1.0f, -0.1f, 0.0f);**

**glColor3f(0.0f, 1.0f, 0.0f);**

**glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(0.0f, 1.0f, 0.0f);**

**glVertex3f(-0.05f, 1.25f, 0.0f); glVertex3f(0.0f, 1.15f, 0.0f);**

**glVertex3f(0.05f,1.25f, 0.0f); glVertex3f(0.0f, 1.15f, 0.0f);**

**glVertex3f(0.0f,1.15f, 0.0f); glVertex3f(0.0f, 1.05f, 0.0f);**

**glColor3f(0.0f, 0.0f, 1.0f);**

**glVertex3f(0.0f,0.0f,0.0f); glVertex3f(0.0f, 0.0f, 1.0f);**

**glVertex3f(-0.05f,0.1f,1.05f); glVertex3f(0.05f, 0.1f, 1.05f);**

**glVertex3f(0.05f,0.1f,1.05f); glVertex3f(-0.05f, -0.1f, 1.05f);**

**glVertex3f(-0.05f,-0.1f,1.05f); glVertex3f(0.05f, -0.1f, 1.05f);**

**glEnd();**

**glLineWidth(1.0f);**

**glColor3f(1.0f, 1.0f, 1.0f);**

**glBegin(GL\_LINES);**

**float d = 50.0f;**

**for(float i = -d; i <= d; i += 1.0f)**

**{**

**glVertex3f(i, 0.0f, -d);**

**glVertex3f(i, 0.0f, d);**

**glVertex3f(-d, 0.0f, i);**

**glVertex3f(d, 0.0f, i);**

**}**

**glEnd();**

**}**

**// render objects ---------------------------------------------------------------------------------------------------------**

**glEnable(GL\_LIGHTING);**

**glLightfv(GL\_LIGHT0, GL\_POSITION, &vec4(Objects[LightObjectID].Position, 1.0f));**

**if(WireFrame)**

**{**

**glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);**

**}**

**if(gl\_version >= 21)**

**{**

**glUseProgram(Shader);**

**}**

**glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY);**

**glEnableClientState(GL\_NORMAL\_ARRAY);**

**glEnableClientState(GL\_VERTEX\_ARRAY);**

**for(int i = 0; i < ObjectsCount; i++)**

**{**

**if(Objects[i].TrianglesCount <= 0) continue;**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadMatrixf(&View);**

**glTranslatef(Objects[i].Position.x, Objects[i].Position.y, Objects[i].Position.z);**

**if(!WireFrame && Objects[i].CullFace)**

**{**

**glEnable(GL\_CULL\_FACE);**

**}**

**glColor3fv(&Objects[i].Color);**

**bool Texturing = this->Texturing && Objects[i].Texture;**

**if(Texturing)**

**{**

**glEnable(GL\_TEXTURE\_2D);**

**glBindTexture(GL\_TEXTURE\_2D, Objects[i].Texture);**

**}**

**if(gl\_version >= 21)**

**{**

**glUniform1i(Shader.UniformLocations[0], Texturing ? 1 : 0);**

**}**

**if(gl\_version >= 15)**

**{**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[i].VBO[0]);**

**glTexCoordPointer(2, GL\_FLOAT, 0, NULL);**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[i].VBO[1]);**

**glNormalPointer(GL\_FLOAT, 0, NULL);**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[i].VBO[2]);**

**glVertexPointer(3, GL\_FLOAT, 0, NULL);**

**}**

**else**

**{**

**glTexCoordPointer(2, GL\_FLOAT, 0, Objects[i].TexCoords);**

**glNormalPointer(GL\_FLOAT, 0, Objects[i].Normals);**

**glVertexPointer(3, GL\_FLOAT, 0, Objects[i].Vertices);**

**}**

**glFrontFace(Objects[i].FrontFace);**

**glDrawArrays(GL\_TRIANGLES, 0, Objects[i].TrianglesCount \* 3);**

**if(Texturing)**

**{**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**glDisable(GL\_TEXTURE\_2D);**

**}**

**if(!WireFrame && Objects[i].CullFace)**

**{**

**glDisable(GL\_CULL\_FACE);**

**}**

**}**

**if(gl\_version >= 15)**

**{**

**glBindBuffer(GL\_ARRAY\_BUFFER, 0);**

**}**

**glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY);**

**glDisableClientState(GL\_NORMAL\_ARRAY);**

**glDisableClientState(GL\_VERTEX\_ARRAY);**

**if(gl\_version >= 21)**

**{**

**glUseProgram(0);**

**}**

**if(WireFrame)**

**{**

**glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);**

**}**

**glDisable(GL\_LIGHTING);**

**glDisable(GL\_DEPTH\_TEST);**

**}**

**void COpenGLRenderer::Resize(int Width, int Height)**

**{**

**this->Width = Width;**

**this->Height = Height;**

**glViewport(0, 0, Width, Height);**

**Projection = PerspectiveProjectionMatrix(45.0f, (float)Width, (float)Height, 0.125f, 512.0f);**

**ProjectionBiasInverse = PerspectiveProjectionMatrixInverse(Projection) \* BiasMatrixInverse();**

**glMatrixMode(GL\_PROJECTION);**

**glLoadMatrixf(&Projection);**

**}**

**void COpenGLRenderer::Destroy()**

**{**

**for(int i = 0; i < ObjectsCount; i++)**

**{**

**Objects[i].Destroy();**

**}**

**delete [] Objects;**

**if(gl\_version >= 21)**

**{**

**Shader.Delete();**

**}**

**}**

**void COpenGLRenderer::MoveSelectedObject(int x, int y)**

**{**

**if(SelectedObject < 0 || SelectedObject >= ObjectsCount || !Objects[SelectedObject].Movable) return;**

**y = Height - 1 - y;**

**float s = (float)x / (float)(Width - 1);**

**float t = (float)y / (float)(Height - 1);**

**vec4 Point = ViewMatrixInverse(View) \* (ProjectionBiasInverse \* vec4(s, t, 0.5f, 1.0f));**

**Point /= Point.w;**

**vec3 Ray = normalize(Point - Camera.Position);**

**float NdotR = -dot(PlaneNormal, Ray);**

**if(NdotR != 0.0f)**

**{**

**float Distance = (dot(PlaneNormal, Camera.Position) + PlaneD) / NdotR;**

**vec3 Point = Ray \* Distance + Camera.Position;**

**vec3 Offset = Point - SelectedPoint;**

**SelectedPoint = Point;**

**Objects[SelectedObject].Position += Offset;**

**}**

**}**

**void COpenGLRenderer::SelectObject(int x, int y)**

**{**

**int gl\_version = use\_gl\_version < ::gl\_version ? use\_gl\_version : ::gl\_version;**

**y = Height - 1 - y;**

**glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);**

**glEnable(GL\_DEPTH\_TEST);**

**glEnableClientState(GL\_VERTEX\_ARRAY);**

**for(int i = 0; i < ObjectsCount; i++)**

**{**

**if(Objects[i].TrianglesCount <= 0) continue;**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadMatrixf(&View);**

**glTranslatef(Objects[i].Position.x, Objects[i].Position.y, Objects[i].Position.z);**

**if(Objects[i].CullFace)**

**{**

**glEnable(GL\_CULL\_FACE);**

**}**

**int ID = i + 1;**

**glColor3ub(ID & 0xFF, (ID >> 8) & 0xFF, (ID >> 16) & 0xFF);**

**if(gl\_version >= 15)**

**{**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[i].VBO[2]);**

**glVertexPointer(3, GL\_FLOAT, 0, NULL);**

**}**

**else**

**{**

**glVertexPointer(3, GL\_FLOAT, 0, Objects[i].Vertices);**

**}**

**glFrontFace(Objects[i].FrontFace);**

**glDrawArrays(GL\_TRIANGLES, 0, Objects[i].TrianglesCount \* 3);**

**if(Objects[i].CullFace)**

**{**

**glDisable(GL\_CULL\_FACE);**

**}**

**}**

**if(gl\_version >= 15)**

**{**

**glBindBuffer(GL\_ARRAY\_BUFFER, 0);**

**}**

**glDisableClientState(GL\_VERTEX\_ARRAY);**

**glDisable(GL\_DEPTH\_TEST);**

**BYTE Pixel[4];**

**glReadPixels(x, y, 1, 1, GL\_RGB, GL\_UNSIGNED\_BYTE, &Pixel);**

**SelectedObject = (Pixel[0] | (Pixel[1] << 8) | (Pixel[2] << 16)) - 1;**

**if(SelectedObject >= 0 && SelectedObject < ObjectsCount)**

**{**

**float s = (float)x / (float)(Width - 1);**

**float t = (float)y / (float)(Height - 1);**

**float Depth;**

**glReadPixels(x, y, 1, 1, GL\_DEPTH\_COMPONENT, GL\_FLOAT, &Depth);**

**vec4 Point = ViewMatrixInverse(View) \* (ProjectionBiasInverse \* vec4(s, t, Depth, 1.0f));**

**Point /= Point.w;**

**SelectedPoint = Point;**

**float omcospi4 = 1.0f - cos((float)M\_PI / 4.0f);**

**if(Camera.Z.y > -omcospi4 && Camera.Z.y < omcospi4)**

**{**

**PlaneNormal = normalize(vec3(Camera.Z.x, 0.0f, Camera.Z.z));**

**}**

**else**

**{**

**PlaneNormal = vec3(0.0f, 1.0f, 0.0f);**

**}**

**PlaneD = -dot(PlaneNormal, SelectedPoint);**

**}**

**}**

**COpenGLRenderer OpenGLRenderer;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CWnd::CWnd()**

**{**

**char \*moduledirectory = new char[256];**

**GetModuleFileName(GetModuleHandle(NULL), moduledirectory, 256);**

**\*(strrchr(moduledirectory, '\\') + 1) = 0;**

**ModuleDirectory = moduledirectory;**

**delete [] moduledirectory;**

**}**

**CWnd::~CWnd()**

**{**

**}**

**bool CWnd::Create(HINSTANCE hInstance, char \*WindowName, int Width, int Height, int Samples, bool CreateForwardCompatibleContext, bool DisableVerticalSynchronization)**

**{**

**WNDCLASSEX WndClassEx;**

**memset(&WndClassEx, 0, sizeof(WNDCLASSEX));**

**WndClassEx.cbSize = sizeof(WNDCLASSEX);**

**WndClassEx.style = CS\_OWNDC | CS\_HREDRAW | CS\_VREDRAW;**

**WndClassEx.lpfnWndProc = WndProc;**

**WndClassEx.hInstance = hInstance;**

**WndClassEx.hIcon = LoadIcon(NULL, IDI\_APPLICATION);**

**WndClassEx.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);**

**WndClassEx.hCursor = LoadCursor(NULL, IDC\_ARROW);**

**WndClassEx.lpszClassName = "Win32OpenGLWindowClass";**

**if(RegisterClassEx(&WndClassEx) == 0)**

**{**

**ErrorLog.Set("RegisterClassEx failed!");**

**return false;**

**}**

**this->WindowName = WindowName;**

**this->Width = Width;**

**this->Height = Height;**

**DWORD Style = WS\_OVERLAPPEDWINDOW | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN;**

**if((hWnd = CreateWindowEx(WS\_EX\_APPWINDOW, WndClassEx.lpszClassName, WindowName, Style, 0, 0, Width, Height, NULL, NULL, hInstance, NULL)) == NULL)**

**{**

**ErrorLog.Set("CreateWindowEx failed!");**

**return false;**

**}**

**if((hDC = GetDC(hWnd)) == NULL)**

**{**

**ErrorLog.Set("GetDC failed!");**

**return false;**

**}**

**PIXELFORMATDESCRIPTOR pfd;**

**memset(&pfd, 0, sizeof(PIXELFORMATDESCRIPTOR));**

**pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);**

**pfd.nVersion = 1;**

**pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW | PFD\_SUPPORT\_OPENGL | PFD\_DOUBLEBUFFER;**

**pfd.iPixelType = PFD\_TYPE\_RGBA;**

**pfd.cColorBits = 32;**

**pfd.cDepthBits = 24;**

**pfd.iLayerType = PFD\_MAIN\_PLANE;**

**int PixelFormat;**

**if((PixelFormat = ChoosePixelFormat(hDC, &pfd)) == 0)**

**{**

**ErrorLog.Set("ChoosePixelFormat failed!");**

**return false;**

**}**

**static int MSAAPixelFormat = 0;**

**if(SetPixelFormat(hDC, MSAAPixelFormat == 0 ? PixelFormat : MSAAPixelFormat, &pfd) == FALSE)**

**{**

**ErrorLog.Set("SetPixelFormat failed!");**

**return false;**

**}**

**if((hGLRC = wglCreateContext(hDC)) == NULL)**

**{**

**ErrorLog.Set("wglCreateContext failed!");**

**return false;**

**}**

**if(wglMakeCurrent(hDC, hGLRC) == FALSE)**

**{**

**ErrorLog.Set("wglMakeCurrent failed!");**

**return false;**

**}**

**if(glewInit() != GLEW\_OK)**

**{**

**ErrorLog.Set("glewInit failed!");**

**return false;**

**}**

**if(MSAAPixelFormat == 0 && Samples > 0)**

**{**

**if(GLEW\_ARB\_multisample && WGLEW\_ARB\_pixel\_format)**

**{**

**while(Samples > 0)**

**{**

**UINT NumFormats = 0;**

**int iAttributes[] =**

**{**

**WGL\_DRAW\_TO\_WINDOW\_ARB, GL\_TRUE,**

**WGL\_SUPPORT\_OPENGL\_ARB, GL\_TRUE,**

**WGL\_DOUBLE\_BUFFER\_ARB, GL\_TRUE,**

**WGL\_PIXEL\_TYPE\_ARB, WGL\_TYPE\_RGBA\_ARB,**

**WGL\_COLOR\_BITS\_ARB, 32,**

**WGL\_DEPTH\_BITS\_ARB, 24,**

**WGL\_ACCELERATION\_ARB, WGL\_FULL\_ACCELERATION\_ARB,**

**WGL\_SAMPLE\_BUFFERS\_ARB, GL\_TRUE,**

**WGL\_SAMPLES\_ARB, Samples,**

**0**

**};**

**if(wglChoosePixelFormatARB(hDC, iAttributes, NULL, 1, &MSAAPixelFormat, &NumFormats) == TRUE && NumFormats > 0) break;**

**Samples--;**

**}**

**wglDeleteContext(hGLRC);**

**DestroyWindow(hWnd);**

**UnregisterClass(WndClassEx.lpszClassName, hInstance);**

**return Create(hInstance, WindowName, Width, Height, Samples, CreateForwardCompatibleContext, DisableVerticalSynchronization);**

**}**

**else**

**{**

**Samples = 0;**

**}**

**}**

**this->Samples = Samples;**

**int major, minor;**

**sscanf\_s((char\*)glGetString(GL\_VERSION), "%d.%d", &major, &minor);**

**gl\_version = major \* 10 + minor;**

**if(CreateForwardCompatibleContext && gl\_version >= 30 && WGLEW\_ARB\_create\_context)**

**{**

**wglDeleteContext(hGLRC);**

**int GLFCRCAttribs[] =**

**{**

**WGL\_CONTEXT\_MAJOR\_VERSION\_ARB, major,**

**WGL\_CONTEXT\_MINOR\_VERSION\_ARB, minor,**

**WGL\_CONTEXT\_FLAGS\_ARB, WGL\_CONTEXT\_FORWARD\_COMPATIBLE\_BIT\_ARB,**

**0**

**};**

**if((hGLRC = wglCreateContextAttribsARB(hDC, 0, GLFCRCAttribs)) == NULL)**

**{**

**ErrorLog.Set("wglCreateContextAttribsARB failed!");**

**return false;**

**}**

**if(wglMakeCurrent(hDC, hGLRC) == FALSE)**

**{**

**ErrorLog.Set("wglMakeCurrent failed!");**

**return false;**

**}**

**wgl\_context\_forward\_compatible = true;**

**}**

**else**

**{**

**wgl\_context\_forward\_compatible = false;**

**}**

**glGetIntegerv(GL\_MAX\_TEXTURE\_SIZE, &gl\_max\_texture\_size);**

**if(GLEW\_EXT\_texture\_filter\_anisotropic)**

**{**

**glGetIntegerv(GL\_MAX\_TEXTURE\_MAX\_ANISOTROPY\_EXT, &gl\_max\_texture\_max\_anisotropy\_ext);**

**}**

**if(DisableVerticalSynchronization && WGLEW\_EXT\_swap\_control)**

**{**

**wglSwapIntervalEXT(0);**

**}**

**return OpenGLRenderer.Init();**

**}**

**void CWnd::Show(bool Maximized)**

**{**

**RECT dRect, wRect, cRect;**

**GetWindowRect(GetDesktopWindow(), &dRect);**

**GetWindowRect(hWnd, &wRect);**

**GetClientRect(hWnd, &cRect);**

**wRect.right += Width - cRect.right;**

**wRect.bottom += Height - cRect.bottom;**

**wRect.right -= wRect.left;**

**wRect.bottom -= wRect.top;**

**wRect.left = dRect.right / 2 - wRect.right / 2;**

**wRect.top = dRect.bottom / 2 - wRect.bottom / 2;**

**MoveWindow(hWnd, wRect.left, wRect.top, wRect.right, wRect.bottom, FALSE);**

**ShowWindow(hWnd, Maximized ? SW\_SHOWMAXIMIZED : SW\_SHOWNORMAL);**

**}**

**void CWnd::MessageLoop()**

**{**

**MSG Msg;**

**while(GetMessage(&Msg, NULL, 0, 0) > 0)**

**{**

**TranslateMessage(&Msg);**

**DispatchMessage(&Msg);**

**}**

**}**

**void CWnd::Destroy()**

**{**

**OpenGLRenderer.Destroy();**

**wglDeleteContext(hGLRC);**

**DestroyWindow(hWnd);**

**}**

**void CWnd::OnKeyDown(UINT Key)**

**{**

**switch(Key)**

**{**

**case VK\_F1:**

**OpenGLRenderer.ShowAxisGrid = !OpenGLRenderer.ShowAxisGrid;**

**break;**

**case VK\_F2:**

**OpenGLRenderer.WireFrame = !OpenGLRenderer.WireFrame;**

**break;**

**case VK\_F3:**

**OpenGLRenderer.Texturing = !OpenGLRenderer.Texturing;**

**break;**

**case '1':**

**use\_gl\_version = 11;**

**break;**

**case '2':**

**use\_gl\_version = 15;**

**break;**

**case '3':**

**use\_gl\_version = 21;**

**break;**

**}**

**}**

**void CWnd::OnLButtonDown(int cx, int cy)**

**{**

**OpenGLRenderer.SelectObject(cx, cy);**

**}**

**void CWnd::OnMouseMove(int cx, int cy)**

**{**

**if(GetKeyState(VK\_LBUTTON) & 0x80)**

**{**

**OpenGLRenderer.MoveSelectedObject(cx, cy);**

**}**

**if(GetKeyState(VK\_RBUTTON) & 0x80)**

**{**

**Camera.OnMouseMove(LastCurPos.x - cx, LastCurPos.y - cy);**

**LastCurPos.x = cx;**

**LastCurPos.y = cy;**

**}**

**}**

**void CWnd::OnMouseWheel(short zDelta)**

**{**

**Camera.OnMouseWheel(zDelta);**

**}**

**void CWnd::OnPaint()**

**{**

**PAINTSTRUCT ps;**

**BeginPaint(hWnd, &ps);**

**static DWORD LastFPSTime = GetTickCount(), LastFrameTime = LastFPSTime;**

**static int FPS = 0;**

**DWORD Time = GetTickCount();**

**float FrameTime = (Time - LastFrameTime) \* 0.001f;**

**LastFrameTime = Time;**

**if(Time - LastFPSTime > 1000)**

**{**

**CString Text = WindowName;**

**Text.Append(" - %dx%d", Width, Height);**

**Text.Append(", ATF %dx", gl\_max\_texture\_max\_anisotropy\_ext);**

**Text.Append(", MSAA %dx", Samples);**

**Text.Append(", FPS: %d", FPS);**

**Text.Append(" - OpenGL %d.%d", gl\_version / 10, gl\_version % 10);**

**if(gl\_version >= 30) if(wgl\_context\_forward\_compatible) Text.Append(" Forward compatible"); else Text.Append(" Compatibility profile");**

**Text.Append(" - %s", (char\*)glGetString(GL\_RENDERER));**

**SetWindowText(hWnd, Text);**

**LastFPSTime = Time;**

**FPS = 0;**

**}**

**else**

**{**

**FPS++;**

**}**

**BYTE Keys = 0x00;**

**if(GetKeyState('W') & 0x80) Keys |= 0x01;**

**if(GetKeyState('S') & 0x80) Keys |= 0x02;**

**if(GetKeyState('A') & 0x80) Keys |= 0x04;**

**if(GetKeyState('D') & 0x80) Keys |= 0x08;**

**if(GetKeyState('R') & 0x80) Keys |= 0x10;**

**if(GetKeyState('F') & 0x80) Keys |= 0x20;**

**if(GetKeyState(VK\_SHIFT) & 0x80) Keys |= 0x40;**

**if(Keys & 0x3F)**

**{**

**vec3 Movement = Camera.OnKeys(Keys, FrameTime);**

**Camera.Move(Movement);**

**}**

**OpenGLRenderer.Render(FrameTime);**

**SwapBuffers(hDC);**

**EndPaint(hWnd, &ps);**

**InvalidateRect(hWnd, NULL, FALSE);**

**}**

**void CWnd::OnRButtonDown(int cx, int cy)**

**{**

**LastCurPos.x = cx;**

**LastCurPos.y = cy;**

**}**

**void CWnd::OnSize(int Width, int Height)**

**{**

**this->Width = Width;**

**this->Height = Height;**

**OpenGLRenderer.Resize(Width, Height);**

**}**

**CWnd Wnd;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)**

**{**

**switch(uiMsg)**

**{**

**case WM\_CLOSE:**

**PostQuitMessage(0);**

**break;**

**case WM\_KEYDOWN:**

**Wnd.OnKeyDown((UINT)wParam);**

**break;**

**case WM\_LBUTTONDOWN:**

**Wnd.OnLButtonDown(LOWORD(lParam), HIWORD(lParam));**

**break;**

**case WM\_MOUSEMOVE:**

**Wnd.OnMouseMove(LOWORD(lParam), HIWORD(lParam));**

**break;**

**case 0x020A: // WM\_MOUSWHEEL**

**Wnd.OnMouseWheel(HIWORD(wParam));**

**break;**

**case WM\_PAINT:**

**Wnd.OnPaint();**

**break;**

**case WM\_RBUTTONDOWN:**

**Wnd.OnRButtonDown(LOWORD(lParam), HIWORD(lParam));**

**break;**

**case WM\_SIZE:**

**Wnd.OnSize(LOWORD(lParam), HIWORD(lParam));**

**break;**

**default:**

**return DefWindowProc(hWnd, uiMsg, wParam, lParam);**

**}**

**return 0;**

**}**

**int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR sCmdLine, int iShow)**

**{**

**if(Wnd.Create(hInstance, "Loading, picking and moving objects", 800, 600))**

**{**

**Wnd.Show();**

**Wnd.MessageLoop();**

**}**

**else**

**{**

**MessageBox(NULL, ErrorLog, "Error", MB\_OK | MB\_ICONERROR);**

**}**

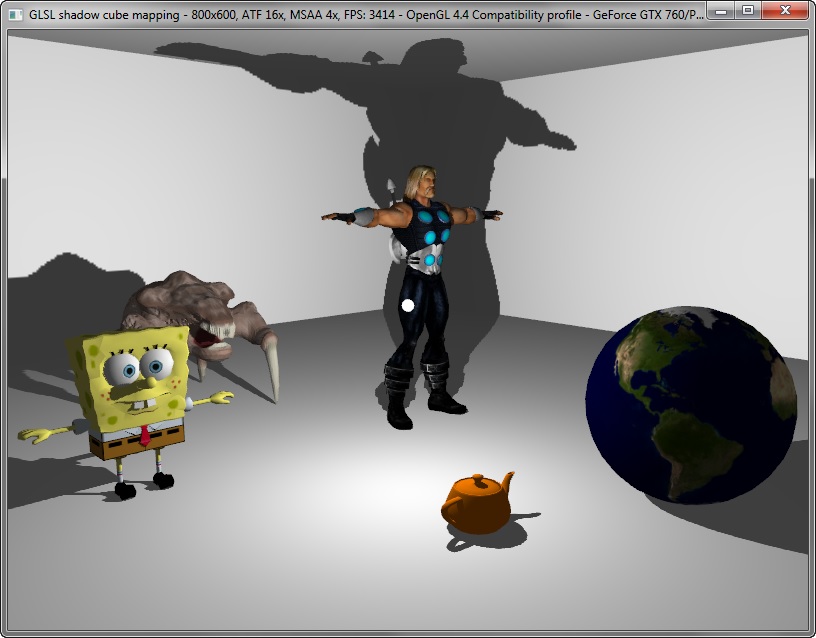
**Wnd.Destroy();**

**return 0;**

**}**

**///////////////////////////////////////GLSL shadow cube mapping/////////////////////////////**

Another extended GLSL shadow mapping. 6 light matrices are calculated and 6 corresponding shadow maps (6 sides of a cube: right, left, top, bottom, back and front) around the point light are rendered to one bilinearly filtered array depth texture. In the fragment shader the largest magnitude coordinate (the major axis) of the negative light direction vector is used to select one light matrix and one corresponding layer of the array depth texture. The selected light matrix is multiplied by the world space position of a fragment and a depth comparison lookup on the selected layer of the array depth texture is done by the shadow2DArray function.

****

**VS:**

**1>shadow\_cube\_mapping.vs**

**#version 120**

**uniform mat4x4 Model;**

**varying vec3 Position, Normal;**

**void main()**

**{**

**Position = (Model \* gl\_Vertex).xyz;**

**Normal = gl\_Normal;**

**gl\_FrontColor = gl\_Color;**

**gl\_TexCoord[0] = gl\_MultiTexCoord0;**

**gl\_Position = gl\_ModelViewProjectionMatrix \* vec4(Position, 1.0);**

**}**

**2>shadow\_cube\_mapping.fs**

**#version 120**

**#extension GL\_EXT\_texture\_array : enable**

**uniform sampler2D Texture;**

**uniform sampler2DArrayShadow ShadowCubeMap;**

**uniform int Texturing;**

**uniform mat4x4 LightTexture[6];**

**varying vec3 Position, Normal;**

**void main()**

**{**

**vec3 LightDirection = gl\_LightSource[0].position.xyz - Position;**

**float LightDistance2 = dot(LightDirection, LightDirection);**

**float LightDistance = sqrt(LightDistance2);**

**LightDirection /= LightDistance;**

**float Attenuation = gl\_LightSource[0].constantAttenuation;**

**Attenuation += gl\_LightSource[0].linearAttenuation \* LightDistance;**

**Attenuation += gl\_LightSource[0].quadraticAttenuation \* LightDistance2;**

**float Axis[6];**

**Axis[0] = -LightDirection.x;**

**Axis[1] = LightDirection.x;**

**Axis[2] = -LightDirection.y;**

**Axis[3] = LightDirection.y;**

**Axis[4] = -LightDirection.z;**

**Axis[5] = LightDirection.z;**

**int MaxAxisID = 0;**

**for(int i = 1; i < 6; i++)**

**{**

**if(Axis[i] > Axis[MaxAxisID])**

**{**

**MaxAxisID = i;**

**}**

**}**

**vec4 ShadowTexCoord = LightTexture[MaxAxisID] \* vec4(Position, 1.0);**

**ShadowTexCoord.xyz /= ShadowTexCoord.w;**

**ShadowTexCoord.w = ShadowTexCoord.z;**

**ShadowTexCoord.z = float(MaxAxisID);**

**float Shadow = shadow2DArray(ShadowCubeMap, ShadowTexCoord).r;**

**float NdotLD = max(dot(normalize(Normal), LightDirection), 0.0) \* Shadow;**

**vec3 Light = (gl\_LightSource[0].ambient.rgb + gl\_LightSource[0].diffuse.rgb \* NdotLD) / Attenuation;**

**gl\_FragColor.rgb = gl\_Color.rgb;**

**if(Texturing == 1) gl\_FragColor.rgb \*= texture2D(Texture, gl\_TexCoord[0].st).rgb;**

**gl\_FragColor.rgb \*= Light;**

**}**

**/////////////////////////////Source////////////////////////**

**#define SHADOW\_CUBE\_MAP\_SIZE 512**

**class COpenGLRenderer**

**{**

**protected:**

**int Width, Height;**

**mat4x4 Model, View, Projection, ProjectionBiasInverse;**

**int ObjectsCount, LightObjectID;**

**CObject \*Objects;**

**CShaderProgram ShadowCubeMapping;**

**GLuint ShadowCubeMap, FBO;**

**mat4x4 LightView[6], LightProjection, LightTexture[6];**

**int SelectedObject;**

**float PlaneD;**

**vec3 SelectedPoint, PlaneNormal;**

**public:**

**COpenGLRenderer();**

**~COpenGLRenderer();**

**bool Init();**

**void Render(float FrameTime);**

**void Resize(int Width, int Height);**

**void Destroy();**

**void MoveSelectedObject(int x, int y);**

**void SelectObject(int x, int y);**

**protected:**

**void RenderShadowCubeMap();**

**};**

**#include "opengl\_tutorials\_win32\_framework.h"**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CString ModuleDirectory, ErrorLog;**

**bool wgl\_context\_forward\_compatible = false;**

**int gl\_version = 0, gl\_max\_texture\_size = 0, gl\_max\_texture\_max\_anisotropy\_ext = 0;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CTexture::CTexture()**

**{**

**TextureID = 0;**

**}**

**CTexture::~CTexture()**

**{**

**}**

**CTexture::operator GLuint ()**

**{**

**return TextureID;**

**}**

**void CTexture::Delete()**

**{**

**glDeleteTextures(1, &TextureID);**

**TextureID = 0;**

**}**

**bool CTexture::LoadTexture2D(char \*Texture2DFileName)**

**{**

**CString FileName = ModuleDirectory + Texture2DFileName;**

**CString ErrorText = "Error loading file " + FileName + "! -> ";**

**FREE\_IMAGE\_FORMAT fif = FreeImage\_GetFileType(FileName);**

**if(fif == FIF\_UNKNOWN)**

**{**

**fif = FreeImage\_GetFIFFromFilename(FileName);**

**}**

**if(fif == FIF\_UNKNOWN)**

**{**

**ErrorLog.Append(ErrorText + "fif is FIF\_UNKNOWN" + "\r\n");**

**return false;**

**}**

**FIBITMAP \*dib = NULL;**

**if(FreeImage\_FIFSupportsReading(fif))**

**{**

**dib = FreeImage\_Load(fif, FileName);**

**}**

**if(dib == NULL)**

**{**

**ErrorLog.Append(ErrorText + "dib is NULL" + "\r\n");**

**return false;**

**}**

**int Width = FreeImage\_GetWidth(dib), oWidth = Width;**

**int Height = FreeImage\_GetHeight(dib), oHeight = Height;**

**int Pitch = FreeImage\_GetPitch(dib);**

**int BPP = FreeImage\_GetBPP(dib);**

**if(Width == 0 || Height == 0)**

**{**

**ErrorLog.Append(ErrorText + "Width or Height is 0" + "\r\n");**

**return false;**

**}**

**if(Width > gl\_max\_texture\_size) Width = gl\_max\_texture\_size;**

**if(Height > gl\_max\_texture\_size) Height = gl\_max\_texture\_size;**

**if(!GLEW\_ARB\_texture\_non\_power\_of\_two)**

**{**

**Width = 1 << (int)floor((log((float)Width) / log(2.0f)) + 0.5f);**

**Height = 1 << (int)floor((log((float)Height) / log(2.0f)) + 0.5f);**

**}**

**if(Width != oWidth || Height != oHeight)**

**{**

**FIBITMAP \*rdib = FreeImage\_Rescale(dib, Width, Height, FILTER\_BICUBIC);**

**FreeImage\_Unload(dib);**

**if((dib = rdib) == NULL)**

**{**

**ErrorLog.Append(ErrorText + "rdib is NULL" + "\r\n");**

**return false;**

**}**

**Pitch = FreeImage\_GetPitch(dib);**

**}**

**BYTE \*Data = FreeImage\_GetBits(dib);**

**if(Data == NULL)**

**{**

**ErrorLog.Append(ErrorText + "Data is NULL" + "\r\n");**

**return false;**

**}**

**GLenum Format = 0;**

**if(BPP == 32) Format = GL\_BGRA;**

**if(BPP == 24) Format = GL\_BGR;**

**if(Format == 0)**

**{**

**FreeImage\_Unload(dib);**

**ErrorLog.Append(ErrorText + "Format is 0" + "\r\n");**

**return false;**

**}**

**if(gl\_version < 12)**

**{**

**if(Format == GL\_BGRA) Format = GL\_RGBA;**

**if(Format == GL\_BGR) Format = GL\_RGB;**

**int bpp = BPP / 8;**

**BYTE \*line = Data;**

**for(int y = 0; y < Height; y++)**

**{**

**BYTE \*pixel = line;**

**for(int x = 0; x < Width; x++)**

**{**

**BYTE Temp = pixel[0];**

**pixel[0] = pixel[2];**

**pixel[2] = Temp;**

**pixel += bpp;**

**}**

**line += Pitch;**

**}**

**}**

**glDeleteTextures(1, &TextureID);**

**glGenTextures(1, &TextureID);**

**glBindTexture(GL\_TEXTURE\_2D, TextureID);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, gl\_version >= 14 ? GL\_LINEAR\_MIPMAP\_LINEAR : GL\_LINEAR);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);**

**if(GLEW\_EXT\_texture\_filter\_anisotropic)**

**{**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT, gl\_max\_texture\_max\_anisotropy\_ext);**

**}**

**if(gl\_version >= 14 && gl\_version <= 21)**

**{**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_GENERATE\_MIPMAP, GL\_TRUE);**

**}**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, Width, Height, 0, Format, GL\_UNSIGNED\_BYTE, Data);**

**if(gl\_version >= 30)**

**{**

**glGenerateMipmap(GL\_TEXTURE\_2D);**

**}**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**FreeImage\_Unload(dib);**

**return true;**

**}**

**// ----------------------------------------------------------------------------------------------------------------------------**

**#pragma warning(disable : 4996)**

**CObject::CObject()**

**{**

**SetDefaults();**

**}**

**CObject::~CObject()**

**{**

**}**

**void CObject::CreateSphere(float Radius, int Resolution, bool InvertNormals)**

**{**

**if(Resolution < 16) Resolution = 16;**

**if(Resolution % 4) Resolution += 4 - Resolution % 4;**

**TrianglesCount = Resolution \* (Resolution / 4 - 1) \* 4 + Resolution \* 2;**

**AllocateMemory();**

**float angle = (float)M\_PI \* 2.0f / Resolution;**

**vec3 a, b, c, d;**

**vec2 tca, tcb, tcc, tcd;**

**int i = 0;**

**float r = (float)Resolution, r4 = (float)Resolution / 2.0f;**

**for(int y = 0; y < Resolution / 4; y++)**

**{**

**for(int xz = 0; xz < Resolution; xz++)**

**{**

**if(y < Resolution / 4 - 1)**

**{**

**a = vec3(- sin(angle \* (xz + 0)) \* cos(angle \* (y + 0)), sin(angle \* (y + 0)), - cos(angle \* (xz + 0)) \* cos(angle \* (y + 0)));**

**b = vec3(- sin(angle \* (xz + 1)) \* cos(angle \* (y + 0)), sin(angle \* (y + 0)), - cos(angle \* (xz + 1)) \* cos(angle \* (y + 0)));**

**c = vec3(- sin(angle \* (xz + 1)) \* cos(angle \* (y + 1)), sin(angle \* (y + 1)), - cos(angle \* (xz + 1)) \* cos(angle \* (y + 1)));**

**d = vec3(- sin(angle \* (xz + 0)) \* cos(angle \* (y + 1)), sin(angle \* (y + 1)), - cos(angle \* (xz + 0)) \* cos(angle \* (y + 1)));**

**tca = TexCoords[i] = vec2((xz + 0) / r, 0.5f + (y + 0) / r4);**

**tcb = TexCoords[i] = vec2((xz + 1) / r, 0.5f + (y + 0) / r4);**

**tcc = TexCoords[i] = vec2((xz + 1) / r, 0.5f + (y + 1) / r4);**

**tcd = TexCoords[i] = vec2((xz + 0) / r, 0.5f + (y + 1) / r4);**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**TexCoords[i] = tcd; Normals[i] = d; Vertices[i++] = d \* Radius;**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**a.y = -a.y;**

**b.y = -b.y;**

**c.y = -c.y;**

**d.y = -d.y;**

**tca.y = 1.0f - tca.y;**

**tcb.y = 1.0f - tcb.y;**

**tcc.y = 1.0f - tcc.y;**

**tcd.y = 1.0f - tcd.y;**

**TexCoords[i] = tcd; Normals[i] = d; Vertices[i++] = d \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**TexCoords[i] = tcd; Normals[i] = d; Vertices[i++] = d \* Radius;**

**}**

**else**

**{**

**a = vec3(- sin(angle \* (xz + 0)) \* cos(angle \* (y + 0)), sin(angle \* (y + 0)), - cos(angle \* (xz + 0)) \* cos(angle \* (y + 0)));**

**b = vec3(- sin(angle \* (xz + 1)) \* cos(angle \* (y + 0)), sin(angle \* (y + 0)), - cos(angle \* (xz + 1)) \* cos(angle \* (y + 0)));**

**c = vec3(0.0f, 1.0f, 0.0f);**

**tca = TexCoords[i] = vec2((xz + 0) / r, 0.5f + (y + 0) / r4);**

**tcb = TexCoords[i] = vec2((xz + 1) / r, 0.5f + (y + 0) / r4);**

**tcc = TexCoords[i] = vec2((xz + 0.5f) / r, 1.0f);**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**a.y = -a.y;**

**b.y = -b.y;**

**c.y = -c.y;**

**tca.y = 1.0f - tca.y;**

**tcb.y = 1.0f - tcb.y;**

**tcc.y = 1.0f - tcc.y;**

**TexCoords[i] = tca; Normals[i] = a; Vertices[i++] = a \* Radius;**

**TexCoords[i] = tcc; Normals[i] = c; Vertices[i++] = c \* Radius;**

**TexCoords[i] = tcb; Normals[i] = b; Vertices[i++] = b \* Radius;**

**}**

**}**

**}**

**if(InvertNormals)**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**Normals[i] = -Normals[i];**

**}**

**}**

**Min = vec3(-Radius, -Radius, -Radius);**

**Max = vec3(Radius, Radius, Radius);**

**}**

**void CObject::Destroy()**

**{**

**if(gl\_version >= 15)**

**{**

**glDeleteBuffers(3, VBO);**

**}**

**Texture.Delete();**

**delete [] TexCoords;**

**delete [] Normals;**

**delete [] Vertices;**

**SetDefaults();**

**}**

**void CObject::InitVertexBuffers()**

**{**

**if(gl\_version >= 15)**

**{**

**glDeleteBuffers(3, VBO);**

**glGenBuffers(3, VBO);**

**glBindBuffer(GL\_ARRAY\_BUFFER, VBO[0]);**

**glBufferData(GL\_ARRAY\_BUFFER, TrianglesCount \* 3 \* 2 \* 4, TexCoords, GL\_STATIC\_DRAW);**

**glBindBuffer(GL\_ARRAY\_BUFFER, VBO[1]);**

**glBufferData(GL\_ARRAY\_BUFFER, TrianglesCount \* 3 \* 3 \* 4, Normals, GL\_STATIC\_DRAW);**

**glBindBuffer(GL\_ARRAY\_BUFFER, VBO[2]);**

**glBufferData(GL\_ARRAY\_BUFFER, TrianglesCount \* 3 \* 3 \* 4, Vertices, GL\_STATIC\_DRAW);**

**glBindBuffer(GL\_ARRAY\_BUFFER, 0);**

**}**

**}**

**bool CObject::Load(char \*Directory, char \*ObjFileName)**

**{**

**char \*ObjSource;**

**long ObjLength;**

**if(!ReadSource(Directory, ObjFileName, &ObjSource, ObjLength)) return false;**

**char \*Line, \*End = ObjSource + ObjLength;**

**float x, y, z;**

**int texcoordscount = 0, normalscount = 0, verticescount = 0;**

**int i1, i2, i3, i4, i5, i6, i7, i8, i9;**

**int v = 0, n = 0, t = 0, T = 0;**

**Line = ObjSource;**

**while(Line < End)**

**{**

**while(Line < End && (\*Line == ' ' || \*Line == '\t')) Line++;**

**if(Line[0] == 'm' && Line[1] == 't' && Line[2] == 'l' && Line[3] == 'l' && Line[4] == 'i' && Line[5] == 'b' && (Line[6] == ' ' || Line[6] == '\t'))**

**{**

**char \*MtlFileName = Line + 6;**

**while(MtlFileName < End && (\*MtlFileName == ' ' || \*MtlFileName == '\t')) MtlFileName++;**

**if(!ParseMtl(Directory, MtlFileName))**

**{**

**delete [] ObjSource;**

**return false;**

**}**

**}**

**else if(sscanf(Line, "vt %f %f", &x, &y) == 2)**

**{**

**texcoordscount++;**

**}**

**else if(sscanf(Line, "vn %f %f %f", &x, &y, &z) == 3)**

**{**

**normalscount++;**

**}**

**else if(sscanf(Line, "v %f %f %f", &x, &y, &z) == 3)**

**{**

**verticescount++;**

**}**

**else if(sscanf(Line, "f %d/%d/%d %d/%d/%d %d/%d/%d", &i1, &i2, &i3, &i4, &i5, &i6, &i7, &i8, &i9) == 9)**

**{**

**TrianglesCount++;**

**}**

**else if(sscanf(Line, "f %d//%d %d//%d %d//%d", &i1, &i2, &i3, &i4, &i5, &i6) == 6)**

**{**

**TrianglesCount++;**

**}**

**else if(sscanf(Line, "f %d/%d %d/%d %d/%d", &i1, &i2, &i3, &i4, &i5, &i6) == 6)**

**{**

**TrianglesCount++;**

**}**

**else if(sscanf(Line, "f %d %d %d", &i1, &i2, &i3) == 3)**

**{**

**TrianglesCount++;**

**}**

**while(Line < End && \*Line != 0) Line++;**

**while(Line < End && \*Line == 0) Line++;**

**}**

**vec2 \*texcoords = NULL;**

**vec3 \*normals = NULL;**

**vec3 \*vertices = NULL;**

**if(texcoordscount > 0) texcoords = new vec2[texcoordscount];**

**if(normalscount > 0) normals = new vec3[normalscount];**

**if(verticescount > 0) vertices = new vec3[verticescount];**

**AllocateMemory();**

**Line = ObjSource;**

**while(Line < End)**

**{**

**while(Line < End && (\*Line == ' ' || \*Line == '\t')) Line++;**

**if(sscanf(Line, "vt %f %f", &x, &y) == 2)**

**{**

**texcoords[t++] = vec2(x, y);**

**}**

**else if(sscanf(Line, "vn %f %f %f", &x, &y, &z) == 3)**

**{**

**normals[n++] = vec3(x, y ,z);**

**}**

**else if(sscanf(Line, "v %f %f %f", &x, &y, &z) == 3)**

**{**

**vertices[v++] = vec3(x, y ,z);**

**}**

**else if(sscanf(Line, "f %d/%d/%d %d/%d/%d %d/%d/%d", &i1, &i2, &i3, &i4, &i5, &i6, &i7, &i8, &i9) == 9)**

**{**

**TexCoords[T] = texcoords[i2 - 1];**

**Normals[T] = normals[i3 - 1];**

**Vertices[T++] = vertices[i1 - 1];**

**TexCoords[T] = texcoords[i5 - 1];**

**Normals[T] = normals[i6 - 1];**

**Vertices[T++] = vertices[i4 - 1];**

**TexCoords[T] = texcoords[i8 - 1];**

**Normals[T] = normals[i9 - 1];**

**Vertices[T++] = vertices[i7 - 1];**

**}**

**else if(sscanf(Line, "f %d//%d %d//%d %d//%d", &i1, &i2, &i3, &i4, &i5, &i6) == 6)**

**{**

**Normals[T] = normals[i2 - 1];**

**Vertices[T++] = vertices[i1 - 1];**

**Normals[T] = normals[i4 - 1];**

**Vertices[T++] = vertices[i3 - 1];**

**Normals[T] = normals[i6 - 1];**

**Vertices[T++] = vertices[i5 - 1];**

**}**

**else if(sscanf(Line, "f %d/%d %d/%d %d/%d", &i1, &i2, &i3, &i4, &i5, &i6) == 6)**

**{**

**if(texcoords != NULL && i1 - 1 < texcoordscount) TexCoords[T] = texcoords[i1 - 1];**

**if(texcoords != NULL && i2 - 1 < texcoordscount) TexCoords[T] = texcoords[i2 - 1];**

**if(normals != NULL && i1 - 1 < normalscount) Normals[T] = normals[i1 - 1];**

**if(normals != NULL && i2 - 1 < normalscount) Normals[T] = normals[i2 - 1];**

**Vertices[T++] = vertices[i1 - 1];**

**if(texcoords != NULL && i3 - 1 < texcoordscount) TexCoords[T] = texcoords[i3 - 1];**

**if(texcoords != NULL && i4 - 1 < texcoordscount) TexCoords[T] = texcoords[i4 - 1];**

**if(normals != NULL && i3 - 1 < normalscount) Normals[T] = normals[i3 - 1];**

**if(normals != NULL && i4 - 1 < normalscount) Normals[T] = normals[i4 - 1];**

**Vertices[T++] = vertices[i3 - 1];**

**if(texcoords != NULL && i5 - 1 < texcoordscount) TexCoords[T] = texcoords[i5 - 1];**

**if(texcoords != NULL && i6 - 1 < texcoordscount) TexCoords[T] = texcoords[i6 - 1];**

**if(normals != NULL && i5 - 1 < normalscount) Normals[T] = normals[i5 - 1];**

**if(normals != NULL && i6 - 1 < normalscount) Normals[T] = normals[i6 - 1];**

**Vertices[T++] = vertices[i5 - 1];**

**}**

**else if(sscanf(Line, "f %d %d %d", &i1, &i2, &i3) == 3)**

**{**

**if(texcoords != NULL && i1 - 1 < texcoordscount) TexCoords[T] = texcoords[i1 - 1];**

**if(normals != NULL && i1 - 1 < normalscount) Normals[T] = normals[i1 - 1];**

**Vertices[T++] = vertices[i1 - 1];**

**if(texcoords != NULL && i2 - 1 < texcoordscount) TexCoords[T] = texcoords[i2 - 1];**

**if(normals != NULL && i2 - 1 < normalscount) Normals[T] = normals[i2 - 1];**

**Vertices[T++] = vertices[i2 - 1];**

**if(texcoords != NULL && i3 - 1 < texcoordscount) TexCoords[T] = texcoords[i3 - 1];**

**if(normals != NULL && i3 - 1 < normalscount) Normals[T] = normals[i3 - 1];**

**Vertices[T++] = vertices[i3 - 1];**

**}**

**while(Line < End && \*Line != 0) Line++;**

**while(Line < End && \*Line == 0) Line++;**

**}**

**delete [] texcoords;**

**delete [] normals;**

**delete [] vertices;**

**delete [] ObjSource;**

**if(normalscount == 0)**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i += 3)**

**{**

**vec3 a = Vertices[i + 1] - Vertices[i];**

**vec3 b = Vertices[i + 2] - Vertices[i];**

**vec3 normal = normalize(cross(a, b));**

**Normals[i + 0] = normal;**

**Normals[i + 1] = normal;**

**Normals[i + 2] = normal;**

**}**

**}**

**GetMinMax();**

**return true;**

**}**

**void CObject::Rotate(float Angle, const vec3 &Axis)**

**{**

**mat4x4 Rotation = RotationMatrix(Angle, Axis);**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**Normals[i] = Rotation \* Normals[i];**

**Vertices[i] = Rotation \* Vertices[i];**

**}**

**GetMinMax();**

**}**

**void CObject::Scale(float ScaleFactor)**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**Vertices[i] \*= ScaleFactor;**

**}**

**Min \*= ScaleFactor;**

**Max \*= ScaleFactor;**

**}**

**void CObject::Translate(const vec3 &Translation)**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**Vertices[i] += Translation;**

**}**

**Min += Translation;**

**Max += Translation;**

**}**

**void CObject::AllocateMemory()**

**{**

**delete [] TexCoords;**

**delete [] Normals;**

**delete [] Vertices;**

**TexCoords = new vec2[TrianglesCount \* 3];**

**Normals = new vec3[TrianglesCount \* 3];**

**Vertices = new vec3[TrianglesCount \* 3];**

**}**

**void CObject::GetMinMax()**

**{**

**for(int i = 0; i < TrianglesCount \* 3; i++)**

**{**

**if(i == 0)**

**{**

**Min = Max = Vertices[i];**

**}**

**else**

**{**

**if(Min.x > Vertices[i].x) Min.x = Vertices[i].x;**

**if(Min.y > Vertices[i].y) Min.y = Vertices[i].y;**

**if(Min.z > Vertices[i].z) Min.z = Vertices[i].z;**

**if(Max.x < Vertices[i].x) Max.x = Vertices[i].x;**

**if(Max.y < Vertices[i].y) Max.y = Vertices[i].y;**

**if(Max.z < Vertices[i].z) Max.z = Vertices[i].z;**

**}**

**}**

**}**

**bool CObject::ParseMtl(char \*Directory, char \*MtlFileName)**

**{**

**char \*MtlSource;**

**long MtlLength;**

**if(!ReadSource(Directory, MtlFileName, &MtlSource, MtlLength)) return false;**

**char \*Line = MtlSource, \*End = MtlSource + MtlLength;**

**bool Error = false;**

**while(Line < End)**

**{**

**while(Line < End && (\*Line == ' ' || \*Line == '\t')) Line++;**

**if(Line[0] == 'm' && Line[1] == 'a' && Line[2] == 'p' && Line[3] == '\_' && Line[4] == 'K' && Line[5] == 'a' && (Line[6] == ' ' || Line[6] == '\t'))**

**{**

**char \*Texture2DFileName = Line + 6;**

**while(Texture2DFileName < End && (\*Texture2DFileName == ' ' || \*Texture2DFileName == '\t')) Texture2DFileName++;**

**Error |= !Texture.LoadTexture2D(CString(Directory) + Texture2DFileName);**

**}**

**while(Line < End && \*Line != 0) Line++;**

**while(Line < End && \*Line == 0) Line++;**

**}**

**delete [] MtlSource;**

**return !Error;**

**}**

**bool CObject::ReadSource(char \*Directory, char \*FileName, char \*\*Source, long &Length)**

**{**

**CString PathFileName = ModuleDirectory + Directory + FileName;**

**FILE \*File;**

**if(fopen\_s(&File, PathFileName, "rb") != 0)**

**{**

**ErrorLog.Append("Error loading file " + PathFileName + "!\r\n");**

**return false;**

**}**

**fseek(File, 0, SEEK\_END);**

**Length = ftell(File);**

**fseek(File, 0, SEEK\_SET);**

**\*Source = new char[Length + 1];**

**fread(\*Source, 1, Length, File);**

**(\*Source)[Length] = 0;**

**fclose(File);**

**for(long i = 0; i < Length; i++)**

**{**

**if((\*Source)[i] == '\r' || (\*Source)[i] == '\n') (\*Source)[i] = 0;**

**}**

**return true;**

**}**

**void CObject::SetDefaults()**

**{**

**Movable = true;**

**CullFace = true;**

**FrontFace = GL\_CCW;**

**TrianglesCount = 0;**

**for(int i = 0; i < 3; i++)**

**{**

**VBO[i] = 0;**

**}**

**TexCoords = NULL;**

**Normals = NULL;**

**Vertices = NULL;**

**Color = vec3(1.0f, 1.0f, 1.0f);**

**Position = vec3(0.0f, 0.0f, 0.0f);**

**}**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CShaderProgram::CShaderProgram()**

**{**

**SetDefaults();**

**}**

**CShaderProgram::~CShaderProgram()**

**{**

**}**

**CShaderProgram::operator GLuint ()**

**{**

**return Program;**

**}**

**void CShaderProgram::Delete()**

**{**

**delete [] UniformLocations;**

**glDetachShader(Program, VertexShader);**

**glDetachShader(Program, FragmentShader);**

**glDeleteShader(VertexShader);**

**glDeleteShader(FragmentShader);**

**glDeleteProgram(Program);**

**SetDefaults();**

**}**

**bool CShaderProgram::Load(char \*VertexShaderFileName, char \*FragmentShaderFileName)**

**{**

**if(UniformLocations || VertexShader || FragmentShader || Program)**

**{**

**Delete();**

**}**

**bool Error = false;**

**Error |= ((VertexShader = LoadShader(GL\_VERTEX\_SHADER, VertexShaderFileName)) == 0);**

**Error |= ((FragmentShader = LoadShader(GL\_FRAGMENT\_SHADER, FragmentShaderFileName)) == 0);**

**if(Error)**

**{**

**Delete();**

**return false;**

**}**

**Program = glCreateProgram();**

**glAttachShader(Program, VertexShader);**

**glAttachShader(Program, FragmentShader);**

**glLinkProgram(Program);**

**int Param = 0;**

**glGetProgramiv(Program, GL\_LINK\_STATUS, &Param);**

**if(Param == GL\_FALSE)**

**{**

**ErrorLog.Append("Error linking program (%s, %s)!\r\n", VertexShaderFileName, FragmentShaderFileName);**

**int InfoLogLength = 0;**

**glGetProgramiv(Program, GL\_INFO\_LOG\_LENGTH, &InfoLogLength);**

**if(InfoLogLength > 0)**

**{**

**char \*InfoLog = new char[InfoLogLength];**

**int CharsWritten = 0;**

**glGetProgramInfoLog(Program, InfoLogLength, &CharsWritten, InfoLog);**

**ErrorLog.Append(InfoLog);**

**delete [] InfoLog;**

**}**

**Delete();**

**return false;**

**}**

**return true;**

**}**

**GLuint CShaderProgram::LoadShader(GLenum Type, char \*ShaderFileName)**

**{**

**CString FileName = ModuleDirectory + ShaderFileName;**

**FILE \*File;**

**if(fopen\_s(&File, FileName, "rb") != 0)**

**{**

**ErrorLog.Append("Error loading file " + FileName + "!\r\n");**

**return 0;**

**}**

**fseek(File, 0, SEEK\_END);**

**long Size = ftell(File);**

**fseek(File, 0, SEEK\_SET);**

**char \*Source = new char[Size + 1];**

**fread(Source, 1, Size, File);**

**fclose(File);**

**Source[Size] = 0;**

**GLuint Shader;**

**Shader = glCreateShader(Type);**

**glShaderSource(Shader, 1, (const char\*\*)&Source, NULL);**

**delete [] Source;**

**glCompileShader(Shader);**

**int Param = 0;**

**glGetShaderiv(Shader, GL\_COMPILE\_STATUS, &Param);**

**if(Param == GL\_FALSE)**

**{**

**ErrorLog.Append("Error compiling shader %s!\r\n", ShaderFileName);**

**int InfoLogLength = 0;**

**glGetShaderiv(Shader, GL\_INFO\_LOG\_LENGTH, &InfoLogLength);**

**if(InfoLogLength > 0)**

**{**

**char \*InfoLog = new char[InfoLogLength];**

**int CharsWritten = 0;**

**glGetShaderInfoLog(Shader, InfoLogLength, &CharsWritten, InfoLog);**

**ErrorLog.Append(InfoLog);**

**delete [] InfoLog;**

**}**

**glDeleteShader(Shader);**

**return 0;**

**}**

**return Shader;**

**}**

**void CShaderProgram::SetDefaults()**

**{**

**UniformLocations = NULL;**

**VertexShader = 0;**

**FragmentShader = 0;**

**Program = 0;**

**}**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CCamera::CCamera()**

**{**

**View = NULL;**

**Reference = vec3(0.0f, 0.0f, 0.0f);**

**Position = vec3(0.0f, 0.0f, 5.0f);**

**X = vec3(1.0f, 0.0f, 0.0f);**

**Y = vec3(0.0f, 1.0f, 0.0f);**

**Z = vec3(0.0f, 0.0f, 1.0f);**

**}**

**CCamera::~CCamera()**

**{**

**}**

**void CCamera::CalculateViewMatrix()**

**{**

**if(View)**

**{**

**\*View = ViewMatrix(X, Y, Z, Position);**

**}**

**}**

**void CCamera::LookAt(vec3 Reference, vec3 Position, bool RotateAroundReference)**

**{**

**this->Reference = Reference;**

**this->Position = Position;**

**Z = normalize(Position - Reference);**

**X = normalize(cross(vec3(0.0f, 1.0f, 0.0f), Z));**

**Y = cross(Z, X);**

**if(!RotateAroundReference)**

**{**

**this->Reference = this->Position;**

**this->Position += Z \* 0.05f;**

**}**

**CalculateViewMatrix();**

**}**

**void CCamera::Move(vec3 Movement)**

**{**

**Reference += Movement;**

**Position += Movement;**

**CalculateViewMatrix();**

**}**

**vec3 CCamera::OnKeys(BYTE Keys, float FrameTime)**

**{**

**float Speed = 5.0f;**

**if(Keys & 0x40) // SHIFT**

**{**

**Speed \*= 2.0f;**

**}**

**float Distance = Speed \* FrameTime;**

**vec3 Up(0.0f, 1.0f, 0.0f);**

**vec3 Right = X;**

**vec3 Forward = cross(Up, Right);**

**Up \*= Distance;**

**Right \*= Distance;**

**Forward \*= Distance;**

**vec3 Movement;**

**if(Keys & 0x01) // W**

**{**

**Movement += Forward;**

**}**

**if(Keys & 0x02) // S**

**{**

**Movement -= Forward;**

**}**

**if(Keys & 0x04) // A**

**{**

**Movement -= Right;**

**}**

**if(Keys & 0x08) // D**

**{**

**Movement += Right;**

**}**

**if(Keys & 0x10) // R**

**{**

**Movement += Up;**

**}**

**if(Keys & 0x20) // F**

**{**

**Movement -= Up;**

**}**

**return Movement;**

**}**

**void CCamera::OnMouseMove(int dx, int dy)**

**{**

**float sensitivity = 0.25f;**

**float hangle = (float)dx \* sensitivity;**

**float vangle = (float)dy \* sensitivity;**

**Position -= Reference;**

**Y = rotate(Y, vangle, X);**

**Z = rotate(Z, vangle, X);**

**if(Y.y < 0.0f)**

**{**

**Z = vec3(0.0f, Z.y > 0.0f ? 1.0f : -1.0f, 0.0f);**

**Y = cross(Z, X);**

**}**

**X = rotate(X, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Y = rotate(Y, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Z = rotate(Z, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Position = Reference + Z \* length(Position);**

**CalculateViewMatrix();**

**}**

**void CCamera::OnMouseWheel(float zDelta)**

**{**

**Position -= Reference;**

**if(zDelta < 0 && length(Position) < 500.0f)**

**{**

**Position += Position \* 0.1f;**

**}**

**if(zDelta > 0 && length(Position) > 0.05f)**

**{**

**Position -= Position \* 0.1f;**

**}**

**Position += Reference;**

**CalculateViewMatrix();**

**}**

**void CCamera::SetViewMatrixPointer(float \*View)**

**{**

**this->View = (mat4x4\*)View;**

**CalculateViewMatrix();**

**}**

**CCamera Camera;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**COpenGLRenderer::COpenGLRenderer()**

**{**

**SelectedObject = -1;**

**Camera.SetViewMatrixPointer(&View);**

**}**

**COpenGLRenderer::~COpenGLRenderer()**

**{**

**}**

**bool COpenGLRenderer::Init()**

**{**

**if(gl\_version < 21)**

**{**

**ErrorLog.Set("OpenGL 2.1 not supported!");**

**return false;**

**}**

**bool Error = false;**

**if(!GLEW\_ARB\_depth\_texture)**

**{**

**ErrorLog.Append("GL\_ARB\_depth\_texture not supported!\r\n");**

**Error = true;**

**}**

**if(!GLEW\_EXT\_texture\_array)**

**{**

**ErrorLog.Append("GL\_EXT\_texture\_array not supported!\r\n");**

**Error = true;**

**}**

**if(!GLEW\_ARB\_framebuffer\_object)**

**{**

**ErrorLog.Append("GL\_ARB\_framebuffer\_object not supported!\r\n");**

**Error = true;**

**}**

**ObjectsCount = 7;**

**Objects = new CObject[ObjectsCount];**

**Error |= !Objects[0].Load("Models\\", "room.obj");**

**Error |= !Objects[1].Load("Models\\Thor\\", "thor.obj");**

**Error |= !Objects[2].Load("Models\\Spongebob\\", "spongebob\_bind.obj");**

**Error |= !Objects[3].Load("Models\\Alien2\\", "alien2.obj");**

**Error |= !Objects[4].Load("Models\\Teapot\\", "teapot.obj");**

**Error |= !Objects[5].Texture.LoadTexture2D("earthmap.jpg");**

**Error |= !ShadowCubeMapping.Load("shadow\_cube\_mapping.vs", "shadow\_cube\_mapping.fs");**

**if(Error)**

**{**

**return false;**

**}**

**Objects[0].Movable = false;**

**Objects[1].Rotate(-90.0f, vec3(0.0f, 1.0f, 0.0f));**

**Objects[1].Scale(1.75f / (Objects[1].Max.y - Objects[1].Min.y));**

**Objects[1].Translate(vec3(-(Objects[1].Min.x + Objects[1].Max.x) / 2.0f, -Objects[1].Min.y, -(Objects[1].Min.z + Objects[1].Max.z) / 2.0f));**

**Objects[1].Position = vec3(1.0f, 0.0f, -1.0f);**

**Objects[2].Scale(0.875f / (Objects[2].Max.y - Objects[2].Min.y));**

**Objects[2].Translate(vec3(-(Objects[2].Min.x + Objects[2].Max.x) / 2.0f, -Objects[2].Min.y, -(Objects[2].Min.z + Objects[2].Max.z) / 2.0f));**

**Objects[2].Position = vec3(-1.0f, 0.0f, -1.0f);**

**Objects[3].Scale(1.0f / (Objects[3].Max.y - Objects[3].Min.y));**

**Objects[3].Translate(vec3(-(Objects[3].Min.x + Objects[3].Max.x) / 2.0f, -Objects[3].Min.y, -(Objects[3].Min.z + Objects[3].Max.z) / 2.0f));**

**Objects[3].Position = vec3(0.0f, 0.0f, -2.5f);**

**Objects[4].Color = vec3(1.0f, 0.5f, 0.0f);**

**Objects[4].CullFace = false;**

**Objects[4].FrontFace = GL\_CW;**

**Objects[4].Scale(0.25f / (Objects[4].Max.y - Objects[4].Min.y));**

**Objects[4].Translate(vec3(-(Objects[4].Min.x + Objects[4].Max.x) / 2.0f, -Objects[4].Min.y, -(Objects[4].Min.z + Objects[4].Max.z) / 2.0f));**

**Objects[4].Position = vec3(0.0f, 0.0f, 0.5f);**

**Objects[5].CreateSphere(0.5f, 32);**

**Objects[5].Position = vec3(1.0f, 0.5f, 1.0f);**

**Objects[6].CreateSphere(0.03125f, 16, true);**

**Objects[6].Position = vec3(0.0f, 1.0f, 0.0f);**

**LightObjectID = 6;**

**for(int i = 0; i < ObjectsCount; i++)**

**{**

**Objects[i].InitVertexBuffers();**

**}**

**ShadowCubeMapping.UniformLocations = new GLuint[3];**

**ShadowCubeMapping.UniformLocations[0] = glGetUniformLocation(ShadowCubeMapping, "Model");**

**ShadowCubeMapping.UniformLocations[1] = glGetUniformLocation(ShadowCubeMapping, "Texturing");**

**ShadowCubeMapping.UniformLocations[2] = glGetUniformLocation(ShadowCubeMapping, "LightTexture");**

**glUseProgram(ShadowCubeMapping);**

**glUniform1i(glGetUniformLocation(ShadowCubeMapping, "Texture"), 0);**

**glUniform1i(glGetUniformLocation(ShadowCubeMapping, "ShadowCubeMap"), 1);**

**glUseProgram(0);**

**glGenTextures(1, &ShadowCubeMap);**

**glBindTexture(GL\_TEXTURE\_2D\_ARRAY, ShadowCubeMap);**

**glTexParameteri(GL\_TEXTURE\_2D\_ARRAY, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);**

**glTexParameteri(GL\_TEXTURE\_2D\_ARRAY, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);**

**glTexParameteri(GL\_TEXTURE\_2D\_ARRAY, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);**

**glTexParameteri(GL\_TEXTURE\_2D\_ARRAY, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);**

**glTexParameteri(GL\_TEXTURE\_2D\_ARRAY, GL\_TEXTURE\_COMPARE\_MODE, GL\_COMPARE\_R\_TO\_TEXTURE);**

**glTexParameteri(GL\_TEXTURE\_2D\_ARRAY, GL\_TEXTURE\_COMPARE\_FUNC, GL\_LEQUAL);**

**glTexParameteri(GL\_TEXTURE\_2D\_ARRAY, GL\_DEPTH\_TEXTURE\_MODE, GL\_INTENSITY);**

**glTexImage3D(GL\_TEXTURE\_2D\_ARRAY, 0, GL\_DEPTH\_COMPONENT32, SHADOW\_CUBE\_MAP\_SIZE, SHADOW\_CUBE\_MAP\_SIZE, 6, 0, GL\_DEPTH\_COMPONENT, GL\_FLOAT, NULL);**

**glBindTexture(GL\_TEXTURE\_2D\_ARRAY, 0);**

**glGenFramebuffers(1, &FBO);**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glDrawBuffers(0, NULL);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**LightProjection = PerspectiveProjectionMatrix(90.0f, SHADOW\_CUBE\_MAP\_SIZE, SHADOW\_CUBE\_MAP\_SIZE, 0.125f, 512.0f);**

**vec3 LightColor = vec3(1.0f, 1.0f, 1.0f);**

**glLightfv(GL\_LIGHT0, GL\_AMBIENT, &vec4(LightColor \* 0.25f, 1.0f));**

**glLightfv(GL\_LIGHT0, GL\_DIFFUSE, &vec4(LightColor \* 0.75f, 1.0f));**

**glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, 1.0f / 128.0f);**

**glLightf(GL\_LIGHT0, GL\_QUADRATIC\_ATTENUATION, 1.0f / 256.0f);**

**Camera.LookAt(vec3(0.0f, 0.875f, 0.0f), vec3(0.0f, 0.875f, 2.5f), true);**

**RenderShadowCubeMap();**

**return true;**

**}**

**void COpenGLRenderer::Render(float FrameTime)**

**{**

**glViewport(0, 0, Width, Height);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadMatrixf(&Projection);**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadIdentity();**

**glLightfv(GL\_LIGHT0, GL\_POSITION, &vec4(Objects[LightObjectID].Position, 1.0f));**

**glLoadMatrixf(&View);**

**glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);**

**glEnable(GL\_DEPTH\_TEST);**

**glActiveTexture(GL\_TEXTURE1); glBindTexture(GL\_TEXTURE\_2D\_ARRAY, ShadowCubeMap);**

**glActiveTexture(GL\_TEXTURE0);**

**glUseProgram(ShadowCubeMapping);**

**glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY);**

**glEnableClientState(GL\_NORMAL\_ARRAY);**

**glEnableClientState(GL\_VERTEX\_ARRAY);**

**for(int i = 0; i < ObjectsCount; i++)**

**{**

**if(Objects[i].TrianglesCount <= 0) continue;**

**Model = TranslationMatrix(Objects[i].Position.x, Objects[i].Position.y, Objects[i].Position.z);**

**glUniformMatrix4fv(ShadowCubeMapping.UniformLocations[0], 1, GL\_FALSE, &Model);**

**if(Objects[i].CullFace)**

**{**

**glEnable(GL\_CULL\_FACE);**

**}**

**glColor3fv(&Objects[i].Color);**

**if(Objects[i].Texture)**

**{**

**glBindTexture(GL\_TEXTURE\_2D, Objects[i].Texture);**

**}**

**glUniform1i(ShadowCubeMapping.UniformLocations[1], Objects[i].Texture ? 1 : 0);**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[i].VBO[0]);**

**glTexCoordPointer(2, GL\_FLOAT, 0, NULL);**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[i].VBO[1]);**

**glNormalPointer(GL\_FLOAT, 0, NULL);**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[i].VBO[2]);**

**glVertexPointer(3, GL\_FLOAT, 0, NULL);**

**glFrontFace(Objects[i].FrontFace);**

**glDrawArrays(GL\_TRIANGLES, 0, Objects[i].TrianglesCount \* 3);**

**if(Objects[i].Texture)**

**{**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**}**

**if(Objects[i].CullFace)**

**{**

**glDisable(GL\_CULL\_FACE);**

**}**

**}**

**glBindBuffer(GL\_ARRAY\_BUFFER, 0);**

**glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY);**

**glDisableClientState(GL\_NORMAL\_ARRAY);**

**glDisableClientState(GL\_VERTEX\_ARRAY);**

**glUseProgram(0);**

**glActiveTexture(GL\_TEXTURE1); glBindTexture(GL\_TEXTURE\_2D\_ARRAY, 0);**

**glActiveTexture(GL\_TEXTURE0);**

**glDisable(GL\_DEPTH\_TEST);**

**}**

**void COpenGLRenderer::Resize(int Width, int Height)**

**{**

**this->Width = Width;**

**this->Height = Height;**

**Projection = PerspectiveProjectionMatrix(45.0f, (float)Width, (float)Height, 0.125f, 512.0f);**

**ProjectionBiasInverse = PerspectiveProjectionMatrixInverse(Projection) \* BiasMatrixInverse();**

**}**

**void COpenGLRenderer::Destroy()**

**{**

**for(int i = 0; i < ObjectsCount; i++)**

**{**

**Objects[i].Destroy();**

**}**

**delete [] Objects;**

**if(gl\_version >= 21)**

**{**

**ShadowCubeMapping.Delete();**

**}**

**glDeleteTextures(1, &ShadowCubeMap);**

**if(GLEW\_ARB\_framebuffer\_object)**

**{**

**glDeleteFramebuffers(1, &FBO);**

**}**

**}**

**void COpenGLRenderer::MoveSelectedObject(int x, int y)**

**{**

**if(SelectedObject < 0 || SelectedObject >= ObjectsCount || !Objects[SelectedObject].Movable) return;**

**y = Height - 1 - y;**

**float s = (float)x / (float)(Width - 1);**

**float t = (float)y / (float)(Height - 1);**

**vec4 Point = ViewMatrixInverse(View) \* (ProjectionBiasInverse \* vec4(s, t, 0.5f, 1.0f));**

**Point /= Point.w;**

**vec3 Ray = normalize(Point - Camera.Position);**

**float NdotR = -dot(PlaneNormal, Ray);**

**if(NdotR != 0.0f)**

**{**

**float Distance = (dot(PlaneNormal, Camera.Position) + PlaneD) / NdotR;**

**vec3 Point = Ray \* Distance + Camera.Position;**

**vec3 Offset = Point - SelectedPoint;**

**SelectedPoint = Point;**

**Objects[SelectedObject].Position += Offset;**

**RenderShadowCubeMap();**

**}**

**}**

**void COpenGLRenderer::SelectObject(int x, int y)**

**{**

**y = Height - 1 - y;**

**glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);**

**glEnable(GL\_DEPTH\_TEST);**

**glEnableClientState(GL\_VERTEX\_ARRAY);**

**for(int i = 0; i < ObjectsCount; i++)**

**{**

**if(Objects[i].TrianglesCount <= 0) continue;**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadMatrixf(&View);**

**glTranslatef(Objects[i].Position.x, Objects[i].Position.y, Objects[i].Position.z);**

**if(Objects[i].CullFace)**

**{**

**glEnable(GL\_CULL\_FACE);**

**}**

**int ID = i + 1;**

**glColor3ub(ID & 0xFF, (ID >> 8) & 0xFF, (ID >> 16) & 0xFF);**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[i].VBO[2]);**

**glVertexPointer(3, GL\_FLOAT, 0, NULL);**

**glFrontFace(Objects[i].FrontFace);**

**glDrawArrays(GL\_TRIANGLES, 0, Objects[i].TrianglesCount \* 3);**

**if(Objects[i].CullFace)**

**{**

**glDisable(GL\_CULL\_FACE);**

**}**

**}**

**glBindBuffer(GL\_ARRAY\_BUFFER, 0);**

**glDisableClientState(GL\_VERTEX\_ARRAY);**

**glDisable(GL\_DEPTH\_TEST);**

**BYTE Pixel[4];**

**glReadPixels(x, y, 1, 1, GL\_RGB, GL\_UNSIGNED\_BYTE, &Pixel);**

**SelectedObject = (Pixel[0] | (Pixel[1] << 8) | (Pixel[2] << 16)) - 1;**

**if(SelectedObject >= 0 && SelectedObject < ObjectsCount)**

**{**

**float s = (float)x / (float)(Width - 1);**

**float t = (float)y / (float)(Height - 1);**

**float Depth;**

**glReadPixels(x, y, 1, 1, GL\_DEPTH\_COMPONENT, GL\_FLOAT, &Depth);**

**vec4 Point = ViewMatrixInverse(View) \* (ProjectionBiasInverse \* vec4(s, t, Depth, 1.0f));**

**Point /= Point.w;**

**SelectedPoint = Point;**

**float omcospi4 = 1.0f - cos((float)M\_PI / 4.0f);**

**if(Camera.Z.y > -omcospi4 && Camera.Z.y < omcospi4)**

**{**

**PlaneNormal = normalize(vec3(Camera.Z.x, 0.0f, Camera.Z.z));**

**}**

**else**

**{**

**PlaneNormal = vec3(0.0f, 1.0f, 0.0f);**

**}**

**PlaneD = -dot(PlaneNormal, SelectedPoint);**

**}**

**}**

**void COpenGLRenderer::RenderShadowCubeMap()**

**{**

**// calculate light matrices -----------------------------------------------------------------------------------------------**

**LightView[0] = ViewMatrix(vec3( 0.0f, 0.0f, 1.0f), vec3(0.0f, 1.0f, 0.0f), vec3(-1.0f, 0.0f, 0.0f), Objects[LightObjectID].Position);**

**LightView[1] = ViewMatrix(vec3( 0.0f, 0.0f,-1.0f), vec3(0.0f, 1.0f, 0.0f), vec3( 1.0f, 0.0f, 0.0f), Objects[LightObjectID].Position);**

**LightView[2] = ViewMatrix(vec3( 1.0f, 0.0f, 0.0f), vec3(0.0f, 0.0f, 1.0f), vec3( 0.0f,-1.0f, 0.0f), Objects[LightObjectID].Position);**

**LightView[3] = ViewMatrix(vec3( 1.0f, 0.0f, 0.0f), vec3(0.0f, 0.0f,-1.0f), vec3( 0.0f, 1.0f, 0.0f), Objects[LightObjectID].Position);**

**LightView[4] = ViewMatrix(vec3(-1.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f), vec3( 0.0f, 0.0f,-1.0f), Objects[LightObjectID].Position);**

**LightView[5] = ViewMatrix(vec3( 1.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f), vec3( 0.0f, 0.0f, 1.0f), Objects[LightObjectID].Position);**

**LightTexture[0] = BiasMatrix() \* LightProjection \* LightView[0];**

**LightTexture[1] = BiasMatrix() \* LightProjection \* LightView[1];**

**LightTexture[2] = BiasMatrix() \* LightProjection \* LightView[2];**

**LightTexture[3] = BiasMatrix() \* LightProjection \* LightView[3];**

**LightTexture[4] = BiasMatrix() \* LightProjection \* LightView[4];**

**LightTexture[5] = BiasMatrix() \* LightProjection \* LightView[5];**

**glUseProgram(ShadowCubeMapping);**

**glUniformMatrix4fv(ShadowCubeMapping.UniformLocations[2], 6, GL\_FALSE, (GLfloat\*)LightTexture);**

**glUseProgram(0);**

**// render shadow cube map -------------------------------------------------------------------------------------------------**

**glViewport(0, 0, SHADOW\_CUBE\_MAP\_SIZE, SHADOW\_CUBE\_MAP\_SIZE);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadMatrixf(&LightProjection);**

**glEnable(GL\_DEPTH\_TEST);**

**glEnable(GL\_CULL\_FACE);**

**glCullFace(GL\_FRONT);**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glEnableClientState(GL\_VERTEX\_ARRAY);**

**for(int i = 0; i < 6; i++)**

**{**

**glFramebufferTextureLayer(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, ShadowCubeMap, 0, i);**

**glClear(GL\_DEPTH\_BUFFER\_BIT);**

**for(int ii = 0; ii < ObjectsCount; ii++)**

**{**

**if(Objects[ii].TrianglesCount <= 0) continue;**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadMatrixf(&LightView[i]);**

**glTranslatef(Objects[ii].Position.x, Objects[ii].Position.y, Objects[ii].Position.z);**

**glBindBuffer(GL\_ARRAY\_BUFFER, Objects[ii].VBO[2]);**

**glVertexPointer(3, GL\_FLOAT, 0, NULL);**

**glFrontFace(Objects[ii].FrontFace);**

**glDrawArrays(GL\_TRIANGLES, 0, Objects[ii].TrianglesCount \* 3);**

**}**

**}**

**glBindBuffer(GL\_ARRAY\_BUFFER, 0);**

**glDisableClientState(GL\_VERTEX\_ARRAY);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**glCullFace(GL\_BACK);**

**glDisable(GL\_CULL\_FACE);**

**glDisable(GL\_DEPTH\_TEST);**

**}**

**COpenGLRenderer OpenGLRenderer;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CWnd::CWnd()**

**{**

**char \*moduledirectory = new char[256];**

**GetModuleFileName(GetModuleHandle(NULL), moduledirectory, 256);**

**\*(strrchr(moduledirectory, '\\') + 1) = 0;**

**ModuleDirectory = moduledirectory;**

**delete [] moduledirectory;**

**}**

**CWnd::~CWnd()**

**{**

**}**

**bool CWnd::Create(HINSTANCE hInstance, char \*WindowName, int Width, int Height, int Samples, bool CreateForwardCompatibleContext, bool DisableVerticalSynchronization)**

**{**

**WNDCLASSEX WndClassEx;**

**memset(&WndClassEx, 0, sizeof(WNDCLASSEX));**

**WndClassEx.cbSize = sizeof(WNDCLASSEX);**

**WndClassEx.style = CS\_OWNDC | CS\_HREDRAW | CS\_VREDRAW;**

**WndClassEx.lpfnWndProc = WndProc;**

**WndClassEx.hInstance = hInstance;**

**WndClassEx.hIcon = LoadIcon(NULL, IDI\_APPLICATION);**

**WndClassEx.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);**

**WndClassEx.hCursor = LoadCursor(NULL, IDC\_ARROW);**

**WndClassEx.lpszClassName = "Win32OpenGLWindowClass";**

**if(RegisterClassEx(&WndClassEx) == 0)**

**{**

**ErrorLog.Set("RegisterClassEx failed!");**

**return false;**

**}**

**this->WindowName = WindowName;**

**this->Width = Width;**

**this->Height = Height;**

**DWORD Style = WS\_OVERLAPPEDWINDOW | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN;**

**if((hWnd = CreateWindowEx(WS\_EX\_APPWINDOW, WndClassEx.lpszClassName, WindowName, Style, 0, 0, Width, Height, NULL, NULL, hInstance, NULL)) == NULL)**

**{**

**ErrorLog.Set("CreateWindowEx failed!");**

**return false;**

**}**

**if((hDC = GetDC(hWnd)) == NULL)**

**{**

**ErrorLog.Set("GetDC failed!");**

**return false;**

**}**

**PIXELFORMATDESCRIPTOR pfd;**

**memset(&pfd, 0, sizeof(PIXELFORMATDESCRIPTOR));**

**pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);**

**pfd.nVersion = 1;**

**pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW | PFD\_SUPPORT\_OPENGL | PFD\_DOUBLEBUFFER;**

**pfd.iPixelType = PFD\_TYPE\_RGBA;**

**pfd.cColorBits = 32;**

**pfd.cDepthBits = 24;**

**pfd.iLayerType = PFD\_MAIN\_PLANE;**

**int PixelFormat;**

**if((PixelFormat = ChoosePixelFormat(hDC, &pfd)) == 0)**

**{**

**ErrorLog.Set("ChoosePixelFormat failed!");**

**return false;**

**}**

**static int MSAAPixelFormat = 0;**

**if(SetPixelFormat(hDC, MSAAPixelFormat == 0 ? PixelFormat : MSAAPixelFormat, &pfd) == FALSE)**

**{**

**ErrorLog.Set("SetPixelFormat failed!");**

**return false;**

**}**

**if((hGLRC = wglCreateContext(hDC)) == NULL)**

**{**

**ErrorLog.Set("wglCreateContext failed!");**

**return false;**

**}**

**if(wglMakeCurrent(hDC, hGLRC) == FALSE)**

**{**

**ErrorLog.Set("wglMakeCurrent failed!");**

**return false;**

**}**

**if(glewInit() != GLEW\_OK)**

**{**

**ErrorLog.Set("glewInit failed!");**

**return false;**

**}**

**if(MSAAPixelFormat == 0 && Samples > 0)**

**{**

**if(GLEW\_ARB\_multisample && WGLEW\_ARB\_pixel\_format)**

**{**

**while(Samples > 0)**

**{**

**UINT NumFormats = 0;**

**int iAttributes[] =**

**{**

**WGL\_DRAW\_TO\_WINDOW\_ARB, GL\_TRUE,**

**WGL\_SUPPORT\_OPENGL\_ARB, GL\_TRUE,**

**WGL\_DOUBLE\_BUFFER\_ARB, GL\_TRUE,**

**WGL\_PIXEL\_TYPE\_ARB, WGL\_TYPE\_RGBA\_ARB,**

**WGL\_COLOR\_BITS\_ARB, 32,**

**WGL\_DEPTH\_BITS\_ARB, 24,**

**WGL\_ACCELERATION\_ARB, WGL\_FULL\_ACCELERATION\_ARB,**

**WGL\_SAMPLE\_BUFFERS\_ARB, GL\_TRUE,**

**WGL\_SAMPLES\_ARB, Samples,**

**0**

**};**

**if(wglChoosePixelFormatARB(hDC, iAttributes, NULL, 1, &MSAAPixelFormat, &NumFormats) == TRUE && NumFormats > 0) break;**

**Samples--;**

**}**

**wglDeleteContext(hGLRC);**

**DestroyWindow(hWnd);**

**UnregisterClass(WndClassEx.lpszClassName, hInstance);**

**return Create(hInstance, WindowName, Width, Height, Samples, CreateForwardCompatibleContext, DisableVerticalSynchronization);**

**}**

**else**

**{**

**Samples = 0;**

**}**

**}**

**this->Samples = Samples;**

**int major, minor;**

**sscanf\_s((char\*)glGetString(GL\_VERSION), "%d.%d", &major, &minor);**

**gl\_version = major \* 10 + minor;**

**if(CreateForwardCompatibleContext && gl\_version >= 30 && WGLEW\_ARB\_create\_context)**

**{**

**wglDeleteContext(hGLRC);**

**int GLFCRCAttribs[] =**

**{**

**WGL\_CONTEXT\_MAJOR\_VERSION\_ARB, major,**

**WGL\_CONTEXT\_MINOR\_VERSION\_ARB, minor,**

**WGL\_CONTEXT\_FLAGS\_ARB, WGL\_CONTEXT\_FORWARD\_COMPATIBLE\_BIT\_ARB,**

**0**

**};**

**if((hGLRC = wglCreateContextAttribsARB(hDC, 0, GLFCRCAttribs)) == NULL)**

**{**

**ErrorLog.Set("wglCreateContextAttribsARB failed!");**

**return false;**

**}**

**if(wglMakeCurrent(hDC, hGLRC) == FALSE)**

**{**

**ErrorLog.Set("wglMakeCurrent failed!");**

**return false;**

**}**

**wgl\_context\_forward\_compatible = true;**

**}**

**else**

**{**

**wgl\_context\_forward\_compatible = false;**

**}**

**glGetIntegerv(GL\_MAX\_TEXTURE\_SIZE, &gl\_max\_texture\_size);**

**if(GLEW\_EXT\_texture\_filter\_anisotropic)**

**{**

**glGetIntegerv(GL\_MAX\_TEXTURE\_MAX\_ANISOTROPY\_EXT, &gl\_max\_texture\_max\_anisotropy\_ext);**

**}**

**if(DisableVerticalSynchronization && WGLEW\_EXT\_swap\_control)**

**{**

**wglSwapIntervalEXT(0);**

**}**

**return OpenGLRenderer.Init();**

**}**

**void CWnd::Show(bool Maximized)**

**{**

**RECT dRect, wRect, cRect;**

**GetWindowRect(GetDesktopWindow(), &dRect);**

**GetWindowRect(hWnd, &wRect);**

**GetClientRect(hWnd, &cRect);**

**wRect.right += Width - cRect.right;**

**wRect.bottom += Height - cRect.bottom;**

**wRect.right -= wRect.left;**

**wRect.bottom -= wRect.top;**

**wRect.left = dRect.right / 2 - wRect.right / 2;**

**wRect.top = dRect.bottom / 2 - wRect.bottom / 2;**

**MoveWindow(hWnd, wRect.left, wRect.top, wRect.right, wRect.bottom, FALSE);**

**ShowWindow(hWnd, Maximized ? SW\_SHOWMAXIMIZED : SW\_SHOWNORMAL);**

**}**

**void CWnd::MessageLoop()**

**{**

**MSG Msg;**

**while(GetMessage(&Msg, NULL, 0, 0) > 0)**

**{**

**TranslateMessage(&Msg);**

**DispatchMessage(&Msg);**

**}**

**}**

**void CWnd::Destroy()**

**{**

**OpenGLRenderer.Destroy();**

**wglDeleteContext(hGLRC);**

**DestroyWindow(hWnd);**

**}**

**void CWnd::OnKeyDown(UINT Key)**

**{**

**/\*switch(Key)**

**{**

**}\*/**

**}**

**void CWnd::OnLButtonDown(int cx, int cy)**

**{**

**OpenGLRenderer.SelectObject(cx, cy);**

**}**

**void CWnd::OnMouseMove(int cx, int cy)**

**{**

**if(GetKeyState(VK\_LBUTTON) & 0x80)**

**{**

**OpenGLRenderer.MoveSelectedObject(cx, cy);**

**}**

**if(GetKeyState(VK\_RBUTTON) & 0x80)**

**{**

**Camera.OnMouseMove(LastCurPos.x - cx, LastCurPos.y - cy);**

**LastCurPos.x = cx;**

**LastCurPos.y = cy;**

**}**

**}**

**void CWnd::OnMouseWheel(short zDelta)**

**{**

**Camera.OnMouseWheel(zDelta);**

**}**

**void CWnd::OnPaint()**

**{**

**PAINTSTRUCT ps;**

**BeginPaint(hWnd, &ps);**

**static DWORD LastFPSTime = GetTickCount(), LastFrameTime = LastFPSTime;**

**static int FPS = 0;**

**DWORD Time = GetTickCount();**

**float FrameTime = (Time - LastFrameTime) \* 0.001f;**

**LastFrameTime = Time;**

**if(Time - LastFPSTime > 1000)**

**{**

**CString Text = WindowName;**

**Text.Append(" - %dx%d", Width, Height);**

**Text.Append(", ATF %dx", gl\_max\_texture\_max\_anisotropy\_ext);**

**Text.Append(", MSAA %dx", Samples);**

**Text.Append(", FPS: %d", FPS);**

**Text.Append(" - OpenGL %d.%d", gl\_version / 10, gl\_version % 10);**

**if(gl\_version >= 30) if(wgl\_context\_forward\_compatible) Text.Append(" Forward compatible"); else Text.Append(" Compatibility profile");**

**Text.Append(" - %s", (char\*)glGetString(GL\_RENDERER));**

**SetWindowText(hWnd, Text);**

**LastFPSTime = Time;**

**FPS = 0;**

**}**

**else**

**{**

**FPS++;**

**}**

**BYTE Keys = 0x00;**

**if(GetKeyState('W') & 0x80) Keys |= 0x01;**

**if(GetKeyState('S') & 0x80) Keys |= 0x02;**

**if(GetKeyState('A') & 0x80) Keys |= 0x04;**

**if(GetKeyState('D') & 0x80) Keys |= 0x08;**

**if(GetKeyState('R') & 0x80) Keys |= 0x10;**

**if(GetKeyState('F') & 0x80) Keys |= 0x20;**

**if(GetKeyState(VK\_SHIFT) & 0x80) Keys |= 0x40;**

**if(Keys & 0x3F)**

**{**

**vec3 Movement = Camera.OnKeys(Keys, FrameTime);**

**Camera.Move(Movement);**

**}**

**OpenGLRenderer.Render(FrameTime);**

**SwapBuffers(hDC);**

**EndPaint(hWnd, &ps);**

**InvalidateRect(hWnd, NULL, FALSE);**

**}**

**void CWnd::OnRButtonDown(int cx, int cy)**

**{**

**LastCurPos.x = cx;**

**LastCurPos.y = cy;**

**}**

**void CWnd::OnSize(int Width, int Height)**

**{**

**this->Width = Width;**

**this->Height = Height;**

**OpenGLRenderer.Resize(Width, Height);**

**}**

**CWnd Wnd;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)**

**{**

**switch(uiMsg)**

**{**

**case WM\_CLOSE:**

**PostQuitMessage(0);**

**break;**

**case WM\_KEYDOWN:**

**Wnd.OnKeyDown((UINT)wParam);**

**break;**

**case WM\_LBUTTONDOWN:**

**Wnd.OnLButtonDown(LOWORD(lParam), HIWORD(lParam));**

**break;**

**case WM\_MOUSEMOVE:**

**Wnd.OnMouseMove(LOWORD(lParam), HIWORD(lParam));**

**break;**

**case 0x020A: // WM\_MOUSWHEEL**

**Wnd.OnMouseWheel(HIWORD(wParam));**

**break;**

**case WM\_PAINT:**

**Wnd.OnPaint();**

**break;**

**case WM\_RBUTTONDOWN:**

**Wnd.OnRButtonDown(LOWORD(lParam), HIWORD(lParam));**

**break;**

**case WM\_SIZE:**

**Wnd.OnSize(LOWORD(lParam), HIWORD(lParam));**

**break;**

**default:**

**return DefWindowProc(hWnd, uiMsg, wParam, lParam);**

**}**

**return 0;**

**}**

**int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR sCmdLine, int iShow)**

**{**

**if(Wnd.Create(hInstance, "GLSL shadow cube mapping", 800, 600))**

**{**

**Wnd.Show();**

**Wnd.MessageLoop();**

**}**

**else**

**{**

**MessageBox(NULL, ErrorLog, "Error", MB\_OK | MB\_ICONERROR);**

**}**

**Wnd.Destroy();**

**return 0;**

**}**

|  |
| --- |
| **Jeremy**, June 13, 2013, 10:15 PM |
| Hi I'm trying to add this affect to my project but I was reading on one of the opengl functions you are using and it says that its only can be used on opengl 3.2 or greater.   glFramebufferTextureLayer(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, ShadowCubeMap, 0, i);  What puzzles me is I am able to run this program of yours and it works fine on my system. And in your program its says that it is using opengl 3.1 in the title bar which is the highest version that can be used on my system. Are there some kind of settings they you have implemented some where to allow the use of this function on ealeir versions of opengl. I ask because when i try to use the function in my project it does not work. I can create normal 2d depth textures just find but when i try to use this texture array with that function it will not capture any information.  You can find what i read here under notes.  http://www.opengl.org/sdk/docs/man3/xhtml/glFramebufferTextureLayer.xml |

|  |
| --- |
| **Jeremy**, June 13, 2013, 10:16 PM |
| By the way very clean code. very nice, well done. |

|  |
| --- |
| **Admin**, June 15, 2013, 11:19 AM |
| The glFramebufferTextureLayer function is part of the GL\_ARB\_framebuffer\_object extension. Look it up in gew.h. If my program runs fine on your system, then your graphics card supports all needed extensions: GL\_ARB\_depth\_texture, GL\_EXT\_texture\_array and GL\_ARB\_framebuffer\_object. You must be doing something wrong in your project ... |

**High dynamic range, bloom**  
  
The exposed scene is rendered to a texture with 16-bit color components (GL\_RGBA16F texture format used). First, the maximal value of the RGB channels of the HDR image is found using a 2x2-supersampling-like shader. This calculations are done only 4 times per second for better performance. Next, the HDR scene texture is rendered to the LDR scene texture with the tone mapping shader applied. Finally, the brightest pixels of the LDR image are efficiently blurred and blended over the screen.

VS: blur.vs

#version 120

void main()

{

gl\_TexCoord[0] = gl\_Vertex;

gl\_Position = gl\_Vertex \* 2.0 - 1.0;

}  
2>blurh.fs

#version 120

uniform sampler2D Texture;

uniform int Width;

uniform float odw;

void main()

{

vec3 Color = vec3(0.0);

int wp1 = Width + 1;

float Sum = 0.0;

for(int x = -Width; x <= Width; x++)

{

float width = (wp1 - abs(float(x)));

Color += texture2D(Texture, gl\_TexCoord[0].st + vec2(odw \* x, 0.0)).rgb \* width;

Sum += width;

}

gl\_FragColor = vec4(Color / Sum, 1.0);

}

1. blurv.fs

#version 120

uniform sampler2D Texture;

uniform int Width;

uniform float odh;

void main()

{

vec3 Color = vec3(0.0);

int wp1 = Width + 1;

float Sum = 0.0;

for(int y = -Width; y <= Width; y++)

{

float width = (wp1 - abs(float(y)));

Color += texture2D(Texture, gl\_TexCoord[0].st + vec2(0.0, odh \* y)).rgb \* width;

Sum += width;

}

gl\_FragColor = vec4(Color / Sum, 1.0);

}

4>bright\_pixels.fs

#version 120

uniform sampler2D LDRColorBuffer;

void main()

{

vec3 Color = texture2D(LDRColorBuffer, gl\_TexCoord[0].st).rgb;

float MaxRGBValue = max(Color.r, max(Color.g, Color.b));

if(MaxRGBValue > 0.90)

{

gl\_FragColor = vec4(Color, 1.0);

}

}

**5>bright\_pixels.vs**

**#version 120**

**void main()**

**{**

**gl\_TexCoord[0] = gl\_Vertex;**

**gl\_Position = gl\_Vertex \* 2.0 - 1.0;**

**}**

**6>luminance.vs**

**#version 120**

**void main()**

**{**

**gl\_TexCoord[0] = gl\_Vertex;**

**gl\_Position = gl\_Vertex \* 2.0 - 1.0;**

**}**

**7>luminance.fs**

**#version 120**

**uniform sampler2D HDRColorBuffer;**

**void main()**

**{**

**vec3 Color = texture2D(HDRColorBuffer, gl\_TexCoord[0].st).rgb;**

**float MaxRGBValue = max(Color.r, max(Color.g, Color.b));**

**gl\_FragColor = vec4(MaxRGBValue, 0.0, 0.0, 0.0);**

**}**

**8>minification.vs**

**#version 120**

**void main()**

**{**

**gl\_TexCoord[0] = gl\_MultiTexCoord0;**

**gl\_Position = gl\_Vertex \* 2.0 - 1.0;**

**}**

**9>minification.fs**

**#version 120**

**uniform sampler2D LuminanceBuffer;**

**uniform float odx, ody;**

**void main()**

**{**

**float a = texture2D(LuminanceBuffer, gl\_TexCoord[0].st + vec2(0.0, 0.0)).r;**

**float b = texture2D(LuminanceBuffer, gl\_TexCoord[0].st + vec2(odx, 0.0)).r;**

**float c = texture2D(LuminanceBuffer, gl\_TexCoord[0].st + vec2(odx, ody)).r;**

**float d = texture2D(LuminanceBuffer, gl\_TexCoord[0].st + vec2(0.0, ody)).r;**

**float maxlum = max(max(a, b), max(c, d));**

**gl\_FragColor = vec4(maxlum, 0.0, 0.0, 0.0);**

**}**

**10>per\_pixel\_lighting.vs**

**#version 120**

**uniform vec3 CameraPosition;**

**varying vec3 LD, Normal, LDR, EV;**

**void main()**

**{**

**LD = gl\_LightSource[0].position.xyz - gl\_Vertex.xyz;**

**Normal = gl\_Normal;**

**LDR = reflect(-LD, Normal);**

**EV = CameraPosition - gl\_Vertex.xyz;**

**gl\_TexCoord[0] = gl\_MultiTexCoord0;**

**gl\_Position = gl\_ModelViewProjectionMatrix \* gl\_Vertex;**

**}**

**11>per\_pixel\_lighting.fs**

**#version 120**

**uniform sampler2D Texture;**

**varying vec3 LD, Normal, EV, LDR;**

**void main()**

**{**

**vec3 LDN = normalize(LD);**

**vec3 NormalN = normalize(Normal);**

**vec3 LDRN = normalize(LDR);**

**vec3 EVN = normalize(EV);**

**float NdotLD = max(dot(NormalN, LDN), 0.0);**

**float EVdotLDR = pow(max(dot(EVN, LDRN), 0.0), 32.0);**

**vec3 Color = texture2D(Texture, gl\_TexCoord[0].st).rgb \* (gl\_LightSource[0].ambient.rgb + gl\_LightSource[0].diffuse.rgb \* (NdotLD + EVdotLDR));**

**gl\_FragColor = vec4(Color, 1.0);**

**}**

**12>tone\_mapping.vs**

**#version 120**

**void main()**

**{**

**gl\_TexCoord[0] = gl\_Vertex;**

**gl\_Position = gl\_Vertex \* 2.0 - 1.0;**

**}  
13>tone\_mapping.fs**

**#version 120**

**uniform sampler2D HDRColorBuffer;**

**uniform float MaxRGBValue;**

**void main()**

**{**

**vec3 Color = texture2D(HDRColorBuffer, gl\_TexCoord[0].st).rgb;**

**gl\_FragColor = vec4(Color / MaxRGBValue, 1.0);**

**}**

**//////////Source**

**class COpenGLRenderer**

**{**

**protected:**

**int Width, Height;**

**mat4x4 Model, View, Projection;**

**CTexture Kocka, Podlaha;**

**GLuint HDRColorBuffer, DepthBuffer, LuminanceBuffer, MinificationBuffer[8], LDRColorBuffer, BrightPixelsBuffer, BloomBuffer[12];**

**GLuint FBO;**

**CShaderProgram PerPixelLighting, Luminance, Minification, ToneMapping, BrightPixels, BlurH, BlurV;**

**float \*data;**

**public:**

**CString Text;**

**float Intensity;**

**public:**

**COpenGLRenderer();**

**~COpenGLRenderer();**

**bool Init();**

**void Render(float FrameTime);**

**void Resize(int Width, int Height);**

**void Destroy();**

**};**

**#include "opengl\_tutorials\_win32\_framework.h"**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CString ModuleDirectory, ErrorLog;**

**bool wgl\_context\_forward\_compatible = false;**

**int gl\_version = 0, gl\_max\_texture\_size = 0, gl\_max\_texture\_max\_anisotropy\_ext = 0;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CTexture::CTexture()**

**{**

**TextureID = 0;**

**}**

**CTexture::~CTexture()**

**{**

**}**

**CTexture::operator GLuint ()**

**{**

**return TextureID;**

**}**

**void CTexture::Delete()**

**{**

**glDeleteTextures(1, &TextureID);**

**TextureID = 0;**

**}**

**bool CTexture::LoadTexture2D(char \*Texture2DFileName)**

**{**

**CString FileName = ModuleDirectory + Texture2DFileName;**

**CString ErrorText = "Error loading file " + FileName + "! -> ";**

**FREE\_IMAGE\_FORMAT fif = FreeImage\_GetFileType(FileName);**

**if(fif == FIF\_UNKNOWN)**

**{**

**fif = FreeImage\_GetFIFFromFilename(FileName);**

**}**

**if(fif == FIF\_UNKNOWN)**

**{**

**ErrorLog.Append(ErrorText + "fif is FIF\_UNKNOWN" + "\r\n");**

**return false;**

**}**

**FIBITMAP \*dib = NULL;**

**if(FreeImage\_FIFSupportsReading(fif))**

**{**

**dib = FreeImage\_Load(fif, FileName);**

**}**

**if(dib == NULL)**

**{**

**ErrorLog.Append(ErrorText + "dib is NULL" + "\r\n");**

**return false;**

**}**

**int Width = FreeImage\_GetWidth(dib), oWidth = Width;**

**int Height = FreeImage\_GetHeight(dib), oHeight = Height;**

**int Pitch = FreeImage\_GetPitch(dib);**

**int BPP = FreeImage\_GetBPP(dib);**

**if(Width == 0 || Height == 0)**

**{**

**ErrorLog.Append(ErrorText + "Width or Height is 0" + "\r\n");**

**return false;**

**}**

**if(Width > gl\_max\_texture\_size) Width = gl\_max\_texture\_size;**

**if(Height > gl\_max\_texture\_size) Height = gl\_max\_texture\_size;**

**if(!GLEW\_ARB\_texture\_non\_power\_of\_two)**

**{**

**Width = 1 << (int)floor((log((float)Width) / log(2.0f)) + 0.5f);**

**Height = 1 << (int)floor((log((float)Height) / log(2.0f)) + 0.5f);**

**}**

**if(Width != oWidth || Height != oHeight)**

**{**

**FIBITMAP \*rdib = FreeImage\_Rescale(dib, Width, Height, FILTER\_BICUBIC);**

**FreeImage\_Unload(dib);**

**if((dib = rdib) == NULL)**

**{**

**ErrorLog.Append(ErrorText + "rdib is NULL" + "\r\n");**

**return false;**

**}**

**Pitch = FreeImage\_GetPitch(dib);**

**}**

**BYTE \*Data = FreeImage\_GetBits(dib);**

**if(Data == NULL)**

**{**

**ErrorLog.Append(ErrorText + "Data is NULL" + "\r\n");**

**return false;**

**}**

**GLenum Format = 0;**

**if(BPP == 32) Format = GL\_BGRA;**

**if(BPP == 24) Format = GL\_BGR;**

**if(Format == 0)**

**{**

**FreeImage\_Unload(dib);**

**ErrorLog.Append(ErrorText + "Format is 0" + "\r\n");**

**return false;**

**}**

**if(gl\_version < 12)**

**{**

**if(Format == GL\_BGRA) Format = GL\_RGBA;**

**if(Format == GL\_BGR) Format = GL\_RGB;**

**int bpp = BPP / 8;**

**BYTE \*line = Data;**

**for(int y = 0; y < Height; y++)**

**{**

**BYTE \*pixel = line;**

**for(int x = 0; x < Width; x++)**

**{**

**BYTE Temp = pixel[0];**

**pixel[0] = pixel[2];**

**pixel[2] = Temp;**

**pixel += bpp;**

**}**

**line += Pitch;**

**}**

**}**

**glGenTextures(1, &TextureID);**

**glBindTexture(GL\_TEXTURE\_2D, TextureID);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, gl\_version >= 14 ? GL\_LINEAR\_MIPMAP\_LINEAR : GL\_LINEAR);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);**

**if(GLEW\_EXT\_texture\_filter\_anisotropic)**

**{**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAX\_ANISOTROPY\_EXT, gl\_max\_texture\_max\_anisotropy\_ext);**

**}**

**if(gl\_version >= 14 && gl\_version <= 21)**

**{**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_GENERATE\_MIPMAP, GL\_TRUE);**

**}**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, Width, Height, 0, Format, GL\_UNSIGNED\_BYTE, Data);**

**if(gl\_version >= 30)**

**{**

**glGenerateMipmap(GL\_TEXTURE\_2D);**

**}**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**FreeImage\_Unload(dib);**

**return true;**

**}**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CShaderProgram::CShaderProgram()**

**{**

**SetDefaults();**

**}**

**CShaderProgram::~CShaderProgram()**

**{**

**}**

**CShaderProgram::operator GLuint ()**

**{**

**return Program;**

**}**

**void CShaderProgram::Delete()**

**{**

**delete [] UniformLocations;**

**glDetachShader(Program, VertexShader);**

**glDetachShader(Program, FragmentShader);**

**glDeleteShader(VertexShader);**

**glDeleteShader(FragmentShader);**

**glDeleteProgram(Program);**

**SetDefaults();**

**}**

**bool CShaderProgram::Load(char \*VertexShaderFileName, char \*FragmentShaderFileName)**

**{**

**if(UniformLocations || VertexShader || FragmentShader || Program)**

**{**

**Delete();**

**}**

**bool Error = false;**

**Error |= ((VertexShader = LoadShader(GL\_VERTEX\_SHADER, VertexShaderFileName)) == 0);**

**Error |= ((FragmentShader = LoadShader(GL\_FRAGMENT\_SHADER, FragmentShaderFileName)) == 0);**

**if(Error)**

**{**

**Delete();**

**return false;**

**}**

**Program = glCreateProgram();**

**glAttachShader(Program, VertexShader);**

**glAttachShader(Program, FragmentShader);**

**glLinkProgram(Program);**

**int Param = 0;**

**glGetProgramiv(Program, GL\_LINK\_STATUS, &Param);**

**if(Param == GL\_FALSE)**

**{**

**ErrorLog.Append("Error linking program (%s, %s)!\r\n", VertexShaderFileName, FragmentShaderFileName);**

**int InfoLogLength = 0;**

**glGetProgramiv(Program, GL\_INFO\_LOG\_LENGTH, &InfoLogLength);**

**if(InfoLogLength > 0)**

**{**

**char \*InfoLog = new char[InfoLogLength];**

**int CharsWritten = 0;**

**glGetProgramInfoLog(Program, InfoLogLength, &CharsWritten, InfoLog);**

**ErrorLog.Append(InfoLog);**

**delete [] InfoLog;**

**}**

**Delete();**

**return false;**

**}**

**return true;**

**}**

**GLuint CShaderProgram::LoadShader(GLenum Type, char \*ShaderFileName)**

**{**

**CString FileName = ModuleDirectory + ShaderFileName;**

**FILE \*File;**

**if(fopen\_s(&File, FileName, "rb") != 0)**

**{**

**ErrorLog.Append("Error loading file " + FileName + "!\r\n");**

**return 0;**

**}**

**fseek(File, 0, SEEK\_END);**

**long Size = ftell(File);**

**fseek(File, 0, SEEK\_SET);**

**char \*Source = new char[Size + 1];**

**fread(Source, 1, Size, File);**

**fclose(File);**

**Source[Size] = 0;**

**GLuint Shader;**

**Shader = glCreateShader(Type);**

**glShaderSource(Shader, 1, (const char\*\*)&Source, NULL);**

**delete [] Source;**

**glCompileShader(Shader);**

**int Param = 0;**

**glGetShaderiv(Shader, GL\_COMPILE\_STATUS, &Param);**

**if(Param == GL\_FALSE)**

**{**

**ErrorLog.Append("Error compiling shader %s!\r\n", ShaderFileName);**

**int InfoLogLength = 0;**

**glGetShaderiv(Shader, GL\_INFO\_LOG\_LENGTH, &InfoLogLength);**

**if(InfoLogLength > 0)**

**{**

**char \*InfoLog = new char[InfoLogLength];**

**int CharsWritten = 0;**

**glGetShaderInfoLog(Shader, InfoLogLength, &CharsWritten, InfoLog);**

**ErrorLog.Append(InfoLog);**

**delete [] InfoLog;**

**}**

**glDeleteShader(Shader);**

**return 0;**

**}**

**return Shader;**

**}**

**void CShaderProgram::SetDefaults()**

**{**

**UniformLocations = NULL;**

**VertexShader = 0;**

**FragmentShader = 0;**

**Program = 0;**

**}**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CCamera::CCamera()**

**{**

**View = NULL;**

**Reference = vec3(0.0f, 0.0f, 0.0f);**

**Position = vec3(0.0f, 0.0f, 5.0f);**

**X = vec3(1.0f, 0.0f, 0.0f);**

**Y = vec3(0.0f, 1.0f, 0.0f);**

**Z = vec3(0.0f, 0.0f, 1.0f);**

**}**

**CCamera::~CCamera()**

**{**

**}**

**void CCamera::CalculateViewMatrix()**

**{**

**if(View)**

**{**

**\*View = ViewMatrix(X, Y, Z, Position);**

**}**

**}**

**void CCamera::LookAt(vec3 Reference, vec3 Position, bool RotateAroundReference)**

**{**

**this->Reference = Reference;**

**this->Position = Position;**

**Z = normalize(Position - Reference);**

**X = normalize(cross(vec3(0.0f, 1.0f, 0.0f), Z));**

**Y = cross(Z, X);**

**if(!RotateAroundReference)**

**{**

**this->Reference = this->Position;**

**this->Position += Z \* 0.05f;**

**}**

**CalculateViewMatrix();**

**}**

**void CCamera::Move(vec3 Movement)**

**{**

**Reference += Movement;**

**Position += Movement;**

**CalculateViewMatrix();**

**}**

**vec3 CCamera::OnKeys(BYTE Keys, float FrameTime)**

**{**

**float Speed = 5.0f;**

**if(Keys & 0x40) // SHIFT**

**{**

**Speed \*= 2.0f;**

**}**

**float Distance = Speed \* FrameTime;**

**vec3 Up(0.0f, 1.0f, 0.0f);**

**vec3 Right = X;**

**vec3 Forward = cross(Up, Right);**

**Up \*= Distance;**

**Right \*= Distance;**

**Forward \*= Distance;**

**vec3 Movement;**

**if(Keys & 0x01) // W**

**{**

**Movement += Forward;**

**}**

**if(Keys & 0x02) // S**

**{**

**Movement -= Forward;**

**}**

**if(Keys & 0x04) // A**

**{**

**Movement -= Right;**

**}**

**if(Keys & 0x08) // D**

**{**

**Movement += Right;**

**}**

**if(Keys & 0x10) // R**

**{**

**Movement += Up;**

**}**

**if(Keys & 0x20) // F**

**{**

**Movement -= Up;**

**}**

**return Movement;**

**}**

**void CCamera::OnMouseMove(int dx, int dy)**

**{**

**float sensitivity = 0.25f;**

**float hangle = (float)dx \* sensitivity;**

**float vangle = (float)dy \* sensitivity;**

**Position -= Reference;**

**Y = rotate(Y, vangle, X);**

**Z = rotate(Z, vangle, X);**

**if(Y.y < 0.0f)**

**{**

**Z = vec3(0.0f, Z.y > 0.0f ? 1.0f : -1.0f, 0.0f);**

**Y = cross(Z, X);**

**}**

**X = rotate(X, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Y = rotate(Y, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Z = rotate(Z, hangle, vec3(0.0f, 1.0f, 0.0f));**

**Position = Reference + Z \* length(Position);**

**CalculateViewMatrix();**

**}**

**void CCamera::OnMouseWheel(float zDelta)**

**{**

**Position -= Reference;**

**if(zDelta < 0 && length(Position) < 500.0f)**

**{**

**Position += Position \* 0.1f;**

**}**

**if(zDelta > 0 && length(Position) > 0.05f)**

**{**

**Position -= Position \* 0.1f;**

**}**

**Position += Reference;**

**CalculateViewMatrix();**

**}**

**void CCamera::SetViewMatrixPointer(float \*View)**

**{**

**this->View = (mat4x4\*)View;**

**CalculateViewMatrix();**

**}**

**CCamera Camera;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**COpenGLRenderer::COpenGLRenderer()**

**{**

**Camera.SetViewMatrixPointer(&View);**

**}**

**COpenGLRenderer::~COpenGLRenderer()**

**{**

**}**

**bool COpenGLRenderer::Init()**

**{**

**if(gl\_version < 21)**

**{**

**ErrorLog.Set("OpenGL 2.1 not supported!");**

**return false;**

**}**

**bool Error = false;**

**if(!GLEW\_ARB\_texture\_non\_power\_of\_two)**

**{**

**ErrorLog.Append("GL\_ARB\_texture\_non\_power\_of\_two not supported!\r\n");**

**Error = true;**

**}**

**if(!GLEW\_ARB\_depth\_texture)**

**{**

**ErrorLog.Append("GL\_ARB\_depth\_texture not supported!\r\n");**

**Error = true;**

**}**

**if(!GLEW\_ARB\_texture\_float)**

**{**

**ErrorLog.Append("GL\_ARB\_texture\_float not supported!\r\n");**

**Error = true;**

**}**

**if(!GLEW\_ARB\_framebuffer\_object)**

**{**

**ErrorLog.Append("GL\_ARB\_framebuffer\_object not supported!\r\n");**

**Error = true;**

**}**

**Error |= !Kocka.LoadTexture2D("kocka.jpg");**

**Error |= !Podlaha.LoadTexture2D("podlaha.jpg");**

**Error |= !PerPixelLighting.Load("per\_pixel\_lighting.vs", "per\_pixel\_lighting.fs");**

**Error |= !Luminance.Load("luminance.vs", "luminance.fs");**

**Error |= !Minification.Load("minification.vs", "minification.fs");**

**Error |= !ToneMapping.Load("tone\_mapping.vs", "tone\_mapping.fs");**

**Error |= !BrightPixels.Load("bright\_pixels.vs", "bright\_pixels.fs");**

**Error |= !BlurH.Load("blur.vs", "blurh.fs");**

**Error |= !BlurV.Load("blur.vs", "blurv.fs");**

**if(Error)**

**{**

**return false;**

**}**

**PerPixelLighting.UniformLocations = new GLuint[1];**

**PerPixelLighting.UniformLocations[0] = glGetUniformLocation(PerPixelLighting, "CameraPosition");**

**Luminance.UniformLocations = new GLuint[2];**

**Luminance.UniformLocations[0] = glGetUniformLocation(Luminance, "odx");**

**Luminance.UniformLocations[1] = glGetUniformLocation(Luminance, "ody");**

**ToneMapping.UniformLocations = new GLuint[1];**

**ToneMapping.UniformLocations[0] = glGetUniformLocation(ToneMapping, "MaxRGBValue");**

**BlurH.UniformLocations = new GLuint[2];**

**BlurH.UniformLocations[0] = glGetUniformLocation(BlurH, "Width");**

**BlurH.UniformLocations[1] = glGetUniformLocation(BlurH, "odw");**

**BlurV.UniformLocations = new GLuint[2];**

**BlurV.UniformLocations[0] = glGetUniformLocation(BlurV, "Width");**

**BlurV.UniformLocations[1] = glGetUniformLocation(BlurV, "odh");**

**glUseProgram(BlurH);**

**glUniform1i(BlurH.UniformLocations[0], 5);**

**glUseProgram(BlurV);**

**glUniform1i(BlurV.UniformLocations[0], 5);**

**glUseProgram(0);**

**glGenTextures(1, &HDRColorBuffer);**

**glGenTextures(1, &DepthBuffer);**

**glGenTextures(1, &LuminanceBuffer);**

**glGenTextures(8, MinificationBuffer);**

**glGenTextures(1, &LDRColorBuffer);**

**glGenTextures(1, &BrightPixelsBuffer);**

**glGenTextures(12, BloomBuffer);**

**glGenFramebuffers(1, &FBO);**

**data = new float[4096];**

**Intensity = 2.0f;**

**glLightfv(GL\_LIGHT0, GL\_AMBIENT, &vec4(vec3(0.25f), 1.0f));**

**glLightfv(GL\_LIGHT0, GL\_POSITION, &vec4(-2.0f, 2.0f, -2.0f, 1.0f));**

**return true;**

**}**

**void COpenGLRenderer::Render(float FrameTime)**

**{**

**// set viewport, perspective projection and modelview matrix --------------------------------------------------------------**

**glViewport(0, 0, Width, Height);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadMatrixf(&Projection);**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadMatrixf(&View);**

**// set light diffuse color ------------------------------------------------------------------------------------------------**

**glLightfv(GL\_LIGHT0, GL\_DIFFUSE, &vec4(vec3(Intensity), 1.0f));**

**// render scene to HDRColorBuffer texture ---------------------------------------------------------------------------------**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, HDRColorBuffer, 0);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, DepthBuffer, 0);**

**glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);**

**glEnable(GL\_DEPTH\_TEST);**

**glEnable(GL\_CULL\_FACE);**

**glUseProgram(PerPixelLighting);**

**glUniform3fv(PerPixelLighting.UniformLocations[0], 1, &Camera.Position);**

**glBindTexture(GL\_TEXTURE\_2D, Kocka);**

**glBegin(GL\_QUADS);**

**glNormal3f( 0.0f, 0.0f, 1.0f);**

**glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, -0.5f, 0.5f);**

**glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f, -0.5f, 0.5f);**

**glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f, 0.5f, 0.5f);**

**glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f, 0.5f, 0.5f);**

**glNormal3f( 0.0f, 0.0f, -1.0f);**

**glTexCoord2f(0.0f, 0.0f); glVertex3f( 0.5f, -0.5f, -0.5f);**

**glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -0.5f, -0.5f);**

**glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f, 0.5f, -0.5f);**

**glTexCoord2f(0.0f, 1.0f); glVertex3f( 0.5f, 0.5f, -0.5f);**

**glNormal3f( 1.0f, 0.0f, 0.0f);**

**glTexCoord2f(0.0f, 0.0f); glVertex3f( 0.5f, -0.5f, 0.5f);**

**glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f, -0.5f, -0.5f);**

**glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f, 0.5f, -0.5f);**

**glTexCoord2f(0.0f, 1.0f); glVertex3f( 0.5f, 0.5f, 0.5f);**

**glNormal3f(-1.0f, 0.0f, 0.0f);**

**glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, -0.5f, -0.5f);**

**glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -0.5f, 0.5f);**

**glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f, 0.5f, 0.5f);**

**glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f, 0.5f, -0.5f);**

**glNormal3f( 0.0f, 1.0f, 0.0f);**

**glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, 0.5f, 0.5f);**

**glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f, 0.5f, 0.5f);**

**glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f, 0.5f, -0.5f);**

**glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f, 0.5f, -0.5f);**

**glNormal3f( 0.0f, -1.0f, 0.0f);**

**glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, -0.5f, -0.5f);**

**glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f, -0.5f, -0.5f);**

**glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f, -0.5f, 0.5f);**

**glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f, -0.5f, 0.5f);**

**glEnd();**

**glBindTexture(GL\_TEXTURE\_2D, Podlaha);**

**glBegin(GL\_QUADS);**

**glNormal3f(0.0f, 1.0f, 0.0f);**

**glTexCoord2f( 0.0f, 0.0f); glVertex3f(-5.0f, -0.5f, 5.0f);**

**glTexCoord2f(10.0f, 0.0f); glVertex3f( 5.0f, -0.5f, 5.0f);**

**glTexCoord2f(10.0f, 10.0f); glVertex3f( 5.0f, -0.5f, -5.0f);**

**glTexCoord2f( 0.0f, 10.0f); glVertex3f(-5.0f, -0.5f, -5.0f);**

**glEnd();**

**glUseProgram(0);**

**glDisable(GL\_CULL\_FACE);**

**glDisable(GL\_DEPTH\_TEST);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**// set orthogonal projection and reset modelview matrix -------------------------------------------------------------------**

**glMatrixMode(GL\_PROJECTION);**

**glLoadMatrixf(&OrthogonalProjectionMatrix(0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f));**

**glMatrixMode(GL\_MODELVIEW);**

**glLoadIdentity();**

**// get the maximal value of the RGB components of the HDR image -----------------------------------------------------------**

**static float MaxRGBValue = 1.0f, maxrgbvalue = 1.0f, mrgbvi, oldmaxrgbvalue = maxrgbvalue;**

**static DWORD LastTime = 0;**

**DWORD Time = GetTickCount();**

**if(Time - LastTime > 250) // 4 times per second only ----------------------------------------------------------------------**

**{**

**LastTime = Time;**

**// render LiminanceBuffer texture -------------------------------------------------------------------------------------**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, LuminanceBuffer, 0);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, 0, 0);**

**glBindTexture(GL\_TEXTURE\_2D, HDRColorBuffer);**

**glUseProgram(Luminance);**

**glBegin(GL\_QUADS);**

**glVertex2f(0.0f, 0.0f);**

**glVertex2f(1.0f, 0.0f);**

**glVertex2f(1.0f, 1.0f);**

**glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glUseProgram(0);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**int i = 0, x = Width, y = Height;**

**float odx = 1.0f / (float)x, ody = 1.0f / (float)y;**

**// downscale LuminanceBuffer texture to less than 32x32 pixels --------------------------------------------------------**

**do**

**{**

**x /= 2;**

**y /= 2;**

**glViewport(0, 0, x, y);**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, MinificationBuffer[i], 0);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, 0, 0);**

**glBindTexture(GL\_TEXTURE\_2D, i == 0 ? LuminanceBuffer : MinificationBuffer[i - 1]);**

**glUseProgram(Minification);**

**glUniform1f(Luminance.UniformLocations[0], odx);**

**glUniform1f(Luminance.UniformLocations[1], ody);**

**glBegin(GL\_QUADS);**

**glTexCoord2f(0.0f, 0.0f); glVertex2f(0.0f, 0.0f);**

**glTexCoord2f(1.0f - odx, 0.0f); glVertex2f(1.0f, 0.0f);**

**glTexCoord2f(1.0f - odx, 1.0f - ody); glVertex2f(1.0f, 1.0f);**

**glTexCoord2f(0.0f, 1.0f - ody); glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glUseProgram(0);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**odx = 1.0f / (float)x;**

**ody = 1.0f / (float)y;**

**i++;**

**}**

**while(x > 32 || y > 32);**

**glViewport(0, 0, Width, Height);**

**// read downscaled LuminanceBuffer texture data -----------------------------------------------------------------------**

**glBindTexture(GL\_TEXTURE\_2D, MinificationBuffer[i - 1]);**

**glGetTexImage(GL\_TEXTURE\_2D, 0, GL\_RGBA, GL\_FLOAT, data);**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**// get the maximal luminance value ------------------------------------------------------------------------------------**

**maxrgbvalue = 0.0f;**

**for(int p = 0; p < x \* y \* 4; p += 4)**

**{**

**maxrgbvalue = max(maxrgbvalue, data[p]);**

**}**

**if(maxrgbvalue < 1.0) maxrgbvalue = 1.0;**

**if(maxrgbvalue != oldmaxrgbvalue) mrgbvi = abs(maxrgbvalue - MaxRGBValue);**

**oldmaxrgbvalue = maxrgbvalue;**

**}**

**if(MaxRGBValue < maxrgbvalue) { MaxRGBValue += mrgbvi \* FrameTime; if(MaxRGBValue > maxrgbvalue) MaxRGBValue = maxrgbvalue; }**

**if(MaxRGBValue > maxrgbvalue) { MaxRGBValue -= mrgbvi \* FrameTime; if(MaxRGBValue < maxrgbvalue) MaxRGBValue = maxrgbvalue; }**

**// render HDRColorBuffer texture to LDRColorBuffer texture with tone mapping shader applied -------------------------------**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, LDRColorBuffer, 0);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, 0, 0);**

**glBindTexture(GL\_TEXTURE\_2D, HDRColorBuffer);**

**glUseProgram(ToneMapping);**

**glUniform1f(ToneMapping.UniformLocations[0], MaxRGBValue);**

**glBegin(GL\_QUADS);**

**glVertex2f(0.0f, 0.0f);**

**glVertex2f(1.0f, 0.0f);**

**glVertex2f(1.0f, 1.0f);**

**glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glUseProgram(0);**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**// render BrightPixelsBuffer texture --------------------------------------------------------------------------------------**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, BrightPixelsBuffer, 0);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, 0, 0);**

**glBindTexture(GL\_TEXTURE\_2D, LDRColorBuffer);**

**glUseProgram(BrightPixels);**

**glBegin(GL\_QUADS);**

**glVertex2f(0.0f, 0.0f);**

**glVertex2f(1.0f, 0.0f);**

**glVertex2f(1.0f, 1.0f);**

**glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glUseProgram(0);**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**// downscale and blur BrightPixelsBuffer texture 4x -----------------------------------------------------------------------**

**for(int i = 0; i < 4; i++)**

**{**

**int ds = 2 \* (i + 1);**

**// set viewport to 1/ds of the screen ----------------------------------------------------------------------------------**

**glViewport(0, 0, Width / ds, Height / ds);**

**// downscale ----------------------------------------------------------------------------------------------------------**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, BloomBuffer[i\*3+0], 0);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, 0, 0);**

**glEnable(GL\_TEXTURE\_2D);**

**glBindTexture(GL\_TEXTURE\_2D, BrightPixelsBuffer);**

**glColor3f(1.0f, 1.0f, 1.0f);**

**glBegin(GL\_QUADS);**

**glTexCoord2f(0.0f, 0.0f); glVertex2f(0.0f, 0.0f);**

**glTexCoord2f(1.0f, 0.0f); glVertex2f(1.0f, 0.0f);**

**glTexCoord2f(1.0f, 1.0f); glVertex2f(1.0f, 1.0f);**

**glTexCoord2f(0.0f, 1.0f); glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**glDisable(GL\_TEXTURE\_2D);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**// horizontal blur ----------------------------------------------------------------------------------------------------**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, BloomBuffer[i\*3+1], 0);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, 0, 0);**

**glBindTexture(GL\_TEXTURE\_2D, BloomBuffer[i\*3+0]);**

**glUseProgram(BlurH);**

**glUniform1f(BlurH.UniformLocations[1], 1.0f / (float)(Width / ds));**

**glBegin(GL\_QUADS);**

**glVertex2f(0.0f, 0.0f);**

**glVertex2f(1.0f, 0.0f);**

**glVertex2f(1.0f, 1.0f);**

**glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glUseProgram(0);**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**// vertical blur ------------------------------------------------------------------------------------------------------**

**glBindFramebuffer(GL\_FRAMEBUFFER, FBO);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, BloomBuffer[i\*3+2], 0);**

**glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_TEXTURE\_2D, 0, 0);**

**glBindTexture(GL\_TEXTURE\_2D, BloomBuffer[i\*3+1]);**

**glUseProgram(BlurV);**

**glUniform1f(BlurH.UniformLocations[1], 1.0f / (float)(Height / ds));**

**glBegin(GL\_QUADS);**

**glVertex2f(0.0f, 0.0f);**

**glVertex2f(1.0f, 0.0f);**

**glVertex2f(1.0f, 1.0f);**

**glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glUseProgram(0);**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**glBindFramebuffer(GL\_FRAMEBUFFER, 0);**

**}**

**// display LDRColorBuffer texture ----------------------------------------------------------------------------------------**

**glViewport(0, 0, Width, Height);**

**glEnable(GL\_TEXTURE\_2D);**

**glBindTexture(GL\_TEXTURE\_2D, LDRColorBuffer);**

**glColor3f(1.0f, 1.0f, 1.0f);**

**glBegin(GL\_QUADS);**

**glTexCoord2f(0.0f, 0.0f); glVertex2f(0.0f, 0.0f);**

**glTexCoord2f(1.0f, 0.0f); glVertex2f(1.0f, 0.0f);**

**glTexCoord2f(1.0f, 1.0f); glVertex2f(1.0f, 1.0f);**

**glTexCoord2f(0.0f, 1.0f); glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**glDisable(GL\_TEXTURE\_2D);**

**// blend 4 downscaled and blurred BrightPixelsBuffer textures over the screen ---------------------------------------------**

**for(int i = 0; i < 4; i++)**

**{**

**glEnable(GL\_TEXTURE\_2D);**

**glBindTexture(GL\_TEXTURE\_2D, BloomBuffer[(3-i)\*3+2]);**

**glBlendFunc(GL\_ONE, GL\_ONE\_MINUS\_SRC\_COLOR);**

**glEnable(GL\_BLEND);**

**glColor3f(1.0f, 1.0f, 1.0f);**

**glBegin(GL\_QUADS);**

**glTexCoord2f(0.0f, 0.0f); glVertex2f(0.0f, 0.0f);**

**glTexCoord2f(1.0f, 0.0f); glVertex2f(1.0f, 0.0f);**

**glTexCoord2f(1.0f, 1.0f); glVertex2f(1.0f, 1.0f);**

**glTexCoord2f(0.0f, 1.0f); glVertex2f(0.0f, 1.0f);**

**glEnd();**

**glDisable(GL\_BLEND);**

**glBindTexture(GL\_TEXTURE\_2D, 0);**

**glDisable(GL\_TEXTURE\_2D);**

**}**

**// end --------------------------------------------------------------------------------------------------------------------**

**Text.Set("I: %.02f, MRGBV: %.03f", Intensity, MaxRGBValue);**

**}**

**void COpenGLRenderer::Resize(int Width, int Height)**

**{**

**this->Width = Width;**

**this->Height = Height;**

**Projection = PerspectiveProjectionMatrix(45.0f, (float)Width, (float)Height, 0.125f, 512.0f);**

**glBindTexture(GL\_TEXTURE\_2D, HDRColorBuffer);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA16F, Width, Height, 0, GL\_RGBA, GL\_FLOAT, NULL);**

**glBindTexture(GL\_TEXTURE\_2D, DepthBuffer);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_DEPTH\_COMPONENT32, Width, Height, 0, GL\_DEPTH\_COMPONENT, GL\_FLOAT, NULL);**

**glBindTexture(GL\_TEXTURE\_2D, LuminanceBuffer);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA16F, Width, Height, 0, GL\_RGBA, GL\_FLOAT, NULL);**

**int i = 0, x = Width, y = Height;**

**do**

**{**

**x /= 2;**

**y /= 2;**

**glBindTexture(GL\_TEXTURE\_2D, MinificationBuffer[i]);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA16F, x, y, 0, GL\_RGBA, GL\_FLOAT, NULL);**

**i++;**

**}**

**while(x > 32 || y > 32);**

**glBindTexture(GL\_TEXTURE\_2D, LDRColorBuffer);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, Width, Height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, NULL);**

**glBindTexture(GL\_TEXTURE\_2D, BrightPixelsBuffer);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, Width, Height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, NULL);**

**for(int i = 0; i < 4; i++)**

**{**

**int ds = 2 \* (i + 1);**

**for(int ii = 0; ii < 3; ii++)**

**{**

**glBindTexture(GL\_TEXTURE\_2D, BloomBuffer[3\*i+ii]);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP);**

**glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP);**

**glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, Width / ds, Height / ds, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, NULL);**

**}**

**}**

**}**

**void COpenGLRenderer::Destroy()**

**{**

**delete [] data;**

**Kocka.Delete();**

**Podlaha.Delete();**

**glDeleteTextures(1, &HDRColorBuffer);**

**glDeleteTextures(1, &DepthBuffer);**

**glDeleteTextures(1, &LuminanceBuffer);**

**glDeleteTextures(8, MinificationBuffer);**

**glDeleteTextures(1, &LDRColorBuffer);**

**glDeleteTextures(1, &BrightPixelsBuffer);**

**glDeleteTextures(12, BloomBuffer);**

**if(gl\_version >= 21)**

**{**

**PerPixelLighting.Delete();**

**Luminance.Delete();**

**Minification.Delete();**

**ToneMapping.Delete();**

**BrightPixels.Delete();**

**BlurH.Delete();**

**BlurV.Delete();**

**}**

**if(GLEW\_ARB\_framebuffer\_object)**

**{**

**glDeleteFramebuffers(1, &FBO);**

**}**

**}**

**COpenGLRenderer OpenGLRenderer;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**CWnd::CWnd()**

**{**

**char \*moduledirectory = new char[256];**

**GetModuleFileName(GetModuleHandle(NULL), moduledirectory, 256);**

**\*(strrchr(moduledirectory, '\\') + 1) = 0;**

**ModuleDirectory = moduledirectory;**

**delete [] moduledirectory;**

**}**

**CWnd::~CWnd()**

**{**

**}**

**bool CWnd::Create(HINSTANCE hInstance, char \*WindowName, int Width, int Height, int Samples, bool CreateForwardCompatibleContext, bool DisableVerticalSynchronization)**

**{**

**WNDCLASSEX WndClassEx;**

**memset(&WndClassEx, 0, sizeof(WNDCLASSEX));**

**WndClassEx.cbSize = sizeof(WNDCLASSEX);**

**WndClassEx.style = CS\_OWNDC | CS\_HREDRAW | CS\_VREDRAW;**

**WndClassEx.lpfnWndProc = WndProc;**

**WndClassEx.hInstance = hInstance;**

**WndClassEx.hIcon = LoadIcon(NULL, IDI\_APPLICATION);**

**WndClassEx.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);**

**WndClassEx.hCursor = LoadCursor(NULL, IDC\_ARROW);**

**WndClassEx.lpszClassName = "Win32OpenGLWindowClass";**

**if(RegisterClassEx(&WndClassEx) == 0)**

**{**

**ErrorLog.Set("RegisterClassEx failed!");**

**return false;**

**}**

**this->WindowName = WindowName;**

**this->Width = Width;**

**this->Height = Height;**

**DWORD Style = WS\_OVERLAPPEDWINDOW | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN;**

**if((hWnd = CreateWindowEx(WS\_EX\_APPWINDOW, WndClassEx.lpszClassName, WindowName, Style, 0, 0, Width, Height, NULL, NULL, hInstance, NULL)) == NULL)**

**{**

**ErrorLog.Set("CreateWindowEx failed!");**

**return false;**

**}**

**if((hDC = GetDC(hWnd)) == NULL)**

**{**

**ErrorLog.Set("GetDC failed!");**

**return false;**

**}**

**PIXELFORMATDESCRIPTOR pfd;**

**memset(&pfd, 0, sizeof(PIXELFORMATDESCRIPTOR));**

**pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);**

**pfd.nVersion = 1;**

**pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW | PFD\_SUPPORT\_OPENGL | PFD\_DOUBLEBUFFER;**

**pfd.iPixelType = PFD\_TYPE\_RGBA;**

**pfd.cColorBits = 32;**

**pfd.cDepthBits = 24;**

**pfd.iLayerType = PFD\_MAIN\_PLANE;**

**int PixelFormat;**

**if((PixelFormat = ChoosePixelFormat(hDC, &pfd)) == 0)**

**{**

**ErrorLog.Set("ChoosePixelFormat failed!");**

**return false;**

**}**

**static int MSAAPixelFormat = 0;**

**if(SetPixelFormat(hDC, MSAAPixelFormat == 0 ? PixelFormat : MSAAPixelFormat, &pfd) == FALSE)**

**{**

**ErrorLog.Set("SetPixelFormat failed!");**

**return false;**

**}**

**if((hGLRC = wglCreateContext(hDC)) == NULL)**

**{**

**ErrorLog.Set("wglCreateContext failed!");**

**return false;**

**}**

**if(wglMakeCurrent(hDC, hGLRC) == FALSE)**

**{**

**ErrorLog.Set("wglMakeCurrent failed!");**

**return false;**

**}**

**if(glewInit() != GLEW\_OK)**

**{**

**ErrorLog.Set("glewInit failed!");**

**return false;**

**}**

**if(MSAAPixelFormat == 0 && Samples > 0)**

**{**

**if(GLEW\_ARB\_multisample && WGLEW\_ARB\_pixel\_format)**

**{**

**while(Samples > 0)**

**{**

**UINT NumFormats = 0;**

**int iAttributes[] =**

**{**

**WGL\_DRAW\_TO\_WINDOW\_ARB, GL\_TRUE,**

**WGL\_SUPPORT\_OPENGL\_ARB, GL\_TRUE,**

**WGL\_DOUBLE\_BUFFER\_ARB, GL\_TRUE,**

**WGL\_PIXEL\_TYPE\_ARB, WGL\_TYPE\_RGBA\_ARB,**

**WGL\_COLOR\_BITS\_ARB, 32,**

**WGL\_DEPTH\_BITS\_ARB, 24,**

**WGL\_ACCELERATION\_ARB, WGL\_FULL\_ACCELERATION\_ARB,**

**WGL\_SAMPLE\_BUFFERS\_ARB, GL\_TRUE,**

**WGL\_SAMPLES\_ARB, Samples,**

**0**

**};**

**if(wglChoosePixelFormatARB(hDC, iAttributes, NULL, 1, &MSAAPixelFormat, &NumFormats) == TRUE && NumFormats > 0) break;**

**Samples--;**

**}**

**wglDeleteContext(hGLRC);**

**DestroyWindow(hWnd);**

**UnregisterClass(WndClassEx.lpszClassName, hInstance);**

**return Create(hInstance, WindowName, Width, Height, Samples, CreateForwardCompatibleContext, DisableVerticalSynchronization);**

**}**

**else**

**{**

**Samples = 0;**

**}**

**}**

**this->Samples = Samples;**

**int major, minor;**

**sscanf\_s((char\*)glGetString(GL\_VERSION), "%d.%d", &major, &minor);**

**gl\_version = major \* 10 + minor;**

**if(CreateForwardCompatibleContext && gl\_version >= 30 && WGLEW\_ARB\_create\_context)**

**{**

**wglDeleteContext(hGLRC);**

**int GLFCRCAttribs[] =**

**{**

**WGL\_CONTEXT\_MAJOR\_VERSION\_ARB, major,**

**WGL\_CONTEXT\_MINOR\_VERSION\_ARB, minor,**

**WGL\_CONTEXT\_FLAGS\_ARB, WGL\_CONTEXT\_FORWARD\_COMPATIBLE\_BIT\_ARB,**

**0**

**};**

**if((hGLRC = wglCreateContextAttribsARB(hDC, 0, GLFCRCAttribs)) == NULL)**

**{**

**ErrorLog.Set("wglCreateContextAttribsARB failed!");**

**return false;**

**}**

**if(wglMakeCurrent(hDC, hGLRC) == FALSE)**

**{**

**ErrorLog.Set("wglMakeCurrent failed!");**

**return false;**

**}**

**wgl\_context\_forward\_compatible = true;**

**}**

**else**

**{**

**wgl\_context\_forward\_compatible = false;**

**}**

**glGetIntegerv(GL\_MAX\_TEXTURE\_SIZE, &gl\_max\_texture\_size);**

**if(GLEW\_EXT\_texture\_filter\_anisotropic)**

**{**

**glGetIntegerv(GL\_MAX\_TEXTURE\_MAX\_ANISOTROPY\_EXT, &gl\_max\_texture\_max\_anisotropy\_ext);**

**}**

**if(DisableVerticalSynchronization && WGLEW\_EXT\_swap\_control)**

**{**

**wglSwapIntervalEXT(0);**

**}**

**return OpenGLRenderer.Init();**

**}**

**void CWnd::Show(bool Maximized)**

**{**

**RECT dRect, wRect, cRect;**

**GetWindowRect(GetDesktopWindow(), &dRect);**

**GetWindowRect(hWnd, &wRect);**

**GetClientRect(hWnd, &cRect);**

**wRect.right += Width - cRect.right;**

**wRect.bottom += Height - cRect.bottom;**

**wRect.right -= wRect.left;**

**wRect.bottom -= wRect.top;**

**wRect.left = dRect.right / 2 - wRect.right / 2;**

**wRect.top = dRect.bottom / 2 - wRect.bottom / 2;**

**MoveWindow(hWnd, wRect.left, wRect.top, wRect.right, wRect.bottom, FALSE);**

**ShowWindow(hWnd, Maximized ? SW\_SHOWMAXIMIZED : SW\_SHOWNORMAL);**

**}**

**void CWnd::MessageLoop()**

**{**

**MSG Msg;**

**while(GetMessage(&Msg, NULL, 0, 0) > 0)**

**{**

**TranslateMessage(&Msg);**

**DispatchMessage(&Msg);**

**}**

**}**

**void CWnd::Destroy()**

**{**

**OpenGLRenderer.Destroy();**

**wglDeleteContext(hGLRC);**

**DestroyWindow(hWnd);**

**}**

**void CWnd::OnKeyDown(UINT Key)**

**{**

**switch(Key)**

**{**

**case VK\_ADD:**

**OpenGLRenderer.Intensity += 0.01f;**

**break;**

**case VK\_SUBTRACT:**

**OpenGLRenderer.Intensity -= 0.01f;**

**if(OpenGLRenderer.Intensity < 0.0f) OpenGLRenderer.Intensity = 0.0f;**

**break;**

**}**

**}**

**void CWnd::OnMouseMove(int cx, int cy)**

**{**

**if(GetKeyState(VK\_RBUTTON) & 0x80)**

**{**

**Camera.OnMouseMove(LastCurPos.x - cx, LastCurPos.y - cy);**

**LastCurPos.x = cx;**

**LastCurPos.y = cy;**

**}**

**}**

**void CWnd::OnMouseWheel(short zDelta)**

**{**

**Camera.OnMouseWheel(zDelta);**

**}**

**void CWnd::OnPaint()**

**{**

**PAINTSTRUCT ps;**

**BeginPaint(hWnd, &ps);**

**static DWORD LastFPSTime = GetTickCount(), LastFrameTime = LastFPSTime;**

**static int FPS = 0;**

**DWORD Time = GetTickCount();**

**float FrameTime = (Time - LastFrameTime) \* 0.001f;**

**LastFrameTime = Time;**

**if(Time - LastFPSTime > 250)**

**{**

**CString Text = WindowName;**

**if(OpenGLRenderer.Text[0])**

**{**

**Text.Append(" - ");**

**Text.Append(OpenGLRenderer.Text);**

**}**

**Text.Append(" - %dx%d", Width, Height);**

**Text.Append(", ATF %dx", gl\_max\_texture\_max\_anisotropy\_ext);**

**Text.Append(", MSAA %dx", Samples);**

**Text.Append(", FPS: %d", FPS\*4);**

**Text.Append(" - OpenGL %d.%d", gl\_version / 10, gl\_version % 10);**

**if(gl\_version >= 30) if(wgl\_context\_forward\_compatible) Text.Append(" Forward compatible"); else Text.Append(" Compatibility profile");**

**Text.Append(" - %s", (char\*)glGetString(GL\_RENDERER));**

**SetWindowText(hWnd, Text);**

**LastFPSTime = Time;**

**FPS = 0;**

**}**

**else**

**{**

**FPS++;**

**}**

**BYTE Keys = 0x00;**

**if(GetKeyState('W') & 0x80) Keys |= 0x01;**

**if(GetKeyState('S') & 0x80) Keys |= 0x02;**

**if(GetKeyState('A') & 0x80) Keys |= 0x04;**

**if(GetKeyState('D') & 0x80) Keys |= 0x08;**

**if(GetKeyState('R') & 0x80) Keys |= 0x10;**

**if(GetKeyState('F') & 0x80) Keys |= 0x20;**

**if(GetKeyState(VK\_SHIFT) & 0x80) Keys |= 0x40;**

**if(Keys & 0x3F)**

**{**

**vec3 Movement = Camera.OnKeys(Keys, FrameTime);**

**Camera.Move(Movement);**

**}**

**OpenGLRenderer.Render(FrameTime);**

**SwapBuffers(hDC);**

**EndPaint(hWnd, &ps);**

**InvalidateRect(hWnd, NULL, FALSE);**

**}**

**void CWnd::OnRButtonDown(int cx, int cy)**

**{**

**LastCurPos.x = cx;**

**LastCurPos.y = cy;**

**}**

**void CWnd::OnSize(int Width, int Height)**

**{**

**this->Width = Width;**

**this->Height = Height;**

**OpenGLRenderer.Resize(Width, Height);**

**}**

**CWnd Wnd;**

**// ----------------------------------------------------------------------------------------------------------------------------**

**LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)**

**{**

**switch(uiMsg)**

**{**

**case WM\_CLOSE:**

**PostQuitMessage(0);**

**break;**

**case WM\_MOUSEMOVE:**

**Wnd.OnMouseMove(LOWORD(lParam), HIWORD(lParam));**

**break;**

**case 0x020A: // WM\_MOUSWHEEL**

**Wnd.OnMouseWheel(HIWORD(wParam));**

**break;**

**case WM\_KEYDOWN:**

**Wnd.OnKeyDown((UINT)wParam);**

**break;**

**case WM\_PAINT:**

**Wnd.OnPaint();**

**break;**

**case WM\_RBUTTONDOWN:**

**Wnd.OnRButtonDown(LOWORD(lParam), HIWORD(lParam));**

**break;**

**case WM\_SIZE:**

**Wnd.OnSize(LOWORD(lParam), HIWORD(lParam));**

**break;**

**default:**

**return DefWindowProc(hWnd, uiMsg, wParam, lParam);**

**}**

**return 0;**

**}**

**int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR sCmdLine, int iShow)**

**{**

**if(Wnd.Create(hInstance, "High dynamic range, bloom", 800, 600, 0))**

**{**

**Wnd.Show();**

**Wnd.MessageLoop();**

**}**

**else**

**{**

**MessageBox(NULL, ErrorLog, "Error", MB\_OK | MB\_ICONERROR);**

**}**

**Wnd.Destroy();**

**return 0;**

**}  
////////////////////////**