**Full Screen Anti-Aliasing**

***Github Repo: https://github.com/30005209/CW\_FSAA***

**Technique One: Super-Sampling Anti-Aliasing**

Super Sampling is the technique in which a texture is first rendered at a larger size. When the scene is rendered (off screen) it can be done so natively at a desired size. Afterwards when transferred to the smaller screen – due to there being less pixels – multiple pixels ‘could’ be placed in the given location. For this reason, a sample is taken – the pixels are merged and then an average is placed in the given pixel (for one technique).

**Technique Two: Multi Sample Anti-Aliasing**

Similar to Super-Sampling a collection of sample points are chosen with the intention of combining for a desired look. For example, when an average is taken results could give a more blended look. Instead of an average, a weighting can be placed on off screen locations or on certain colours to distort the final result in a wanted fashion. For example, an extra weight could be given to dark colours (for example edges) this could allow for edges to be sharper and more clearly defined than they would be otherwise. These can either involve sub-pixels (as would be present within a Super-Sampled texture) or otherwise.

**Technique Three: Framebuffer Object (FBO)**

Instead of rending straight to screen a Framebuffer Object could be made instead. This will allow techniques such as super sampling to occur but also can be used to create things like mirrors within a scene. With the texture being created prior to being displayed – it can then be displayed in a distorted manner (as mentioned previously – a texture could be rendered within a FBO so that it can be cast onto a quad to give the illusion of a mirror).

**Implementation**

The program was built atop the file given to us for “Tutorial 7 – solution”. I will denote changes made to the file here.

First camera settings were set to {960, 540, 0.1, 100.0}. This resolution was chosen so that is smaller than the full resolution of any of the textures used – allowing the implementation of super-sampling. The default background of the window was also changed to solid white. For ease of testing and legibility two enums were created one that will handle the sample size and another that will handle the resolution the FBO will be rendered on (they were both named accordingly). Before the render loop GL\_MULTISAMPLE was enabled (or disabled – depending on preference of test) as was the sample size amount for glfwWindowHint. Then sampling amount as well as resolution were chosen. Though I later implemented a manner to change these at run time – it did not seem as reliable as hoped. The resolution seemed to be able to be changed at run-time but the enabling or disabling of the first two seems less predictable.

Before going into the FBO – via the render loop – note should be made of a control that was added via num-key presses. Numbers 0, 7 and 8 change the sample from none to simple (5 samples) to high (9 samples) within the shader whereas numbers 1,2,3 and 4 change the resolution to be either 1, 2, 3, or 4 times the size for the rendering size i.e., 3 would make the scene render (on the FBO) at the size 3\*960 by 3\*540 (2280 by 1620).

The FBO (found in the EarthScene .h and .cpp files) also has its own implementation of the sample size and resolution enums (with a ifndef function to stop multiple implementations). Textures, shaders, uniforms and a local sample size and resolution enum are added to the class. The constructor for the EarthScene was altered to instead of taking no parameters to take the two enums. This is to allow easy changing of the key variables in the Source.cpp file. Once sent via the constructor the two parameters are copied into the classes variables to allow access across the class. Camera settings are then adjusted using the resolution given and a default sphere has been made to be the basis for the objects within the scene. Three textures are then loaded from file, “Marble0.jpg”, “Marble1.jgp” and “Marble2.jpg”. These call all be found on the github repository. A custom shader error is then created from the “CustomShader.vert” and “CustomShader.frag” files with the program being placed into the classes “CustomShader” GLuint. The classes ‘shader’ variable is also set to this. The rationale being if at a later date a function is wanted to be added to allow the changing of shaders at run time the rest of the code will not need to be changed – instead only this shader variable would need to be. Then uniform locations are set for the shader. Following this the next change takes place with the Texture Paramater functions. Should the sample size not be set to none the Min Filter is changed to GL\_NEAREST to allow for smoother blending.

Most of the code is then the same until the scene is made a 40 by 40 row of spheres is created with the three textures alternating. This is to make the scene complicated enough for the techniques above to show their changes.

Text

Description automatically generatedCustomShader.frag after gaining information required from the .vert counterpart begins by working out the texel size. Following that it uses the preexisting code for light direction normalisation and dot product / lambertian value calculation. The texture colour is then worked out and should sampling take place the offsets (sample points) are worked out, the weight matrix is defined and the diffuse colour is created with this (and the previous information worked out). From there the specular intensity is calculated as well as the gamma correction performed.

**Testing**

There are two different variables that can be adjusted. Resolution on the FBO relative to the display screen and number of sample points used for colour creation in the shader. With regards to the FPS and SPF counts I waited for one minute denoting the highest and lowest recorded values. Care was taken to make sure images were taken in the same location though there is a slight variance of positioning. When taking the photos, I also made sure to stop the movements of the lights so that a fairer comparison could be made between the images.

*Default Texture is 960x540*

*High Sample points is 9 (surrounding box with a weight towards the centre)*

*Simple Sample points is 5 (corners with a weight towards the centre)*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Texture | Sample Points | FPS Low | FPS High | SPF Low | SPF High |
| Test One | Default | None | 25 | 28 | 0.033 | 0.040 |
| Test Two | Default | High | 27 | 30 | 0.034 | 0.042 |
| Test Three | X4 Size | High | 28 | 31 | 0.033 | 0.034 |
| Test Four | X4 Size | None | 27 | 29 | 0.033 | 0.049 |
| Test Five | Default | Simple | 25 | 30 | 0.033 | 0.038 |
| Test Six | X4 Size | Simple | 25 | 29 | 0.033 | 0.039 |
| Test Seven | X2 Size | None | 24 | 28 | 0.033 | 0.038 |
| Test Eight | X3 Size | None | 27 | 29 | 0.034 | 0.044 |
| Test Nine | X2 Size | Simple | 27 | 30 | 0.035 | 0.041 |
| Test Ten | X3 Size | Simple | 27 | 30 | 0.035 | 0.046 |
| Test Eleven | X2 Size | High | 25 | 29 | 0.035 | 0.037 |
| Test Twelve | X3 Size | High | 26 | 29 | 0.035 | 0.041 |

**A picture containing text, stationary, indoor, writing implement

Description automatically generated** *For sake of clarity I provided the images of the extreme values in the testing.*

**Top Left – Test One**

**Top Right – Test Two**

**Bottom Left – Test Three**

**Bottom Right – Test Four**

**Test One vs Test Two**

Though in theory there should be a visible difference between the two the fact that the rendering texture is so low the sampling simply cannot make meaningful enough distinctions to blend appropriately.

There was also a small decrease in speed for the implementation that had a higher number of sampling performed.

**Test One vs Test Three**

Jagged edges are notably seen on test one as well as a much worse detail quality. Though there was a drop in FPS for test three the SPF seems

**Test One vs Test Four**

There is a much higher clash in quality of both smoothness and jaggedness of curved surfaces though with no sampling there is still visible artefacts on the one with higher texture size.

Speed wise the four did perform slower in both FPS and SPF.

**Test Two vs Test Three**

There is a much smoother gradient along the curve of test three despite the same sampling rate. This is due to, as stated earlier the limitations of the smaller resolution that the default size possessed. There is also once again a much slower speed of the smaller textured test – though I think this may be an outlier.

**Test Three vs Test Four**

As expected, when there are enough pixels to denote a meaningful amount of colour differentiation the contrast between the two becomes more visible.

Though both did have a higher FPS compared to the low texture equivalents and the one with no sample points had a much larger read of SPF

**Test Two vs Test Three**

There is a much smoother gradient along the curve of test three despite the same sampling rate. This is due to, as stated earlier the limitations of the smaller resolution that the default size possessed. There is also once again a much slower speed of the smaller textured test – though I think this may be an outlier.

**Conclusion of Testing**

A correlation can be seen between not only the increase in quality when rendering at higher textures (super-sampling) but also an increase in quality with multi-sampling. Thought there is a performance hit for both of these techniques the jump in quality is, in these examples more than worth the dropping of a couple of frames.

**Default Texture Size**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Texture | Sample Points | FPS Low | FPS High | SPF Low | SPF High |
| Test One | Default | None | 25 | 28 | 0.033 | 0.040 |
| Test Five | Default | Simple | 25 | 30 | 0.033 | 0.038 |
| Test Two | Default | High | 27 | 30 | 0.034 | 0.042 |

**X2 Texture Size**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Texture | Sample Points | FPS Low | FPS High | SPF Low | SPF High |
| Test Seven | X2 Size | None | 24 | 28 | 0.033 | 0.038 |
| Test Nine | X2 Size | Simple | 27 | 30 | 0.035 | 0.041 |
| Test Eleven | X2 Size | High | 25 | 29 | 0.035 | 0.037 |

**X3 Texture Size**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Texture | Sample Points | FPS Low | FPS High | SPF Low | SPF High |
| Test Eight | X3 Size | None | 27 | 29 | 0.034 | 0.044 |
| Test Ten | X3 Size | Simple | 27 | 30 | 0.035 | 0.046 |
| Test Twelve | X3 Size | High | 26 | 29 | 0.035 | 0.041 |

**X4 Texture Size**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Texture | Sample Points | FPS Low | FPS High | SPF Low | SPF High |
| Test Three | X4 Size | High | 28 | 31 | 0.033 | 0.034 |
| Test Four | X4 Size | None | 27 | 29 | 0.033 | 0.049 |
| Test Six | X4 Size | Simple | 25 | 29 | 0.033 | 0.039 |

**Appendix**

**Source.cpp**

#include "Includes.h"

#include "EarthScene.h"

// Function prototypes

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height);

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos);

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset);

void processInput(GLFWwindow \*window);

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mods);

// Camera settings

// width, height, near plane, far plane

Camera\_settings camera\_settings{ 960, 540, 0.1, 100.0 };

//Timer

Timer timer;

// Instantiate the camera object with basic data

Camera camera(camera\_settings, glm::vec3(0.0f, 0.5f, 4.0f));

double lastX = camera\_settings.screenWidth / 2.0f;

double lastY = camera\_settings.screenHeight / 2.0f;

bool showEarthQuad = false;

EarthScene \*earthScene = nullptr;

int currentSetting = 0;

int maxSetting = 3;

TexturedQuad\* earthQuad = nullptr;

#ifndef DETAIL\_STRUCTS

#define DETAIL\_STRUCTS

enum SampleSize

{

none = 0,

simple = 5,

high = 9,

};

enum Resolution

{

x1 = 1,

x2 = 2,

x3 = 3,

x4 = 4

};

#endif DETAIL\_STRUCTS

SampleSize sample;

Resolution resolution;

int main()

{

sample = none;

resolution = x1;

// glfw: initialize and configure

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

// glfw window creation

GLFWwindow\* window = glfwCreateWindow(camera\_settings.screenWidth, camera\_settings.screenHeight, "Real-Time Rendering: DEMO 3", NULL, NULL);

if (window == NULL)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

// Set the callback functions

glfwMakeContextCurrent(window);

glfwSetFramebufferSizeCallback(window, framebuffer\_size\_callback);

glfwSetCursorPosCallback(window, mouse\_callback);

glfwSetScrollCallback(window, scroll\_callback);

glfwSetKeyCallback(window, key\_callback);

// tell GLFW to capture our mouse

glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_HIDDEN);

// glad: load all OpenGL function pointers

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

//Rendering settings

glfwSwapInterval(1); // glfw enable swap interval to match screen v-sync

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_CULL\_FACE); //Enables face culling

glFrontFace(GL\_CCW);//Specifies which winding order if front facing

//glEnable(GL\_FRAMEBUFFER\_SRGB);

PrincipleAxes \*principleAxes = new PrincipleAxes();

//// Shaders - Textures - Models ////

GLuint phongShader;

GLSL\_ERROR glsl\_err = ShaderLoader::createShaderProgram(

string("Resources\\Shaders\\Basic\_shader.vert"),

string("Resources\\Shaders\\Basic\_shader.frag"),

&phongShader);

//

// Earth scene

//

bool leftCtrlPressed = false;

glEnable(GL\_MULTISAMPLE); // Comment out for non sampling

glfwWindowHint(GLFW\_SAMPLES, 10000); // Comment out for non sampling

sample = high; // Change for sampling amount

resolution = x2; // change for resolution

earthScene = new EarthScene(sample, resolution);

earthQuad = new TexturedQuad(earthScene->getEarthSceneTexture(), true);

// render loop

while (!glfwWindowShouldClose(window))

{

// input

processInput(window);

timer.tick();

// This demo performs 2 rendering passes...

// Pass 1) Render the Earth scene to a texture

// Pass 2) Render the basic demo scene with the principle axes and textured quad, where the texture on the quad is the Earth scene rendered in pass 1.

//

// Pass 1. Render the Earth scene

//

earthScene->render();

//

// Pass 2. Render the basic demo scene to the screen

//

// Clear the screen

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

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

//Reset the viewport

int width, height;

glfwGetFramebufferSize(window, &width, &height);

glViewport(0, 0, width, height);

glm::mat4 model = glm::scale(glm::mat4(1.0), glm::vec3(0.28, 0.28, 0.28));

glm::mat4 view = camera.getViewMatrix();

glm::mat4 projection = camera.getProjectionMatrix();

// Update earthScene state

if (earthScene)

earthScene->update(timer.getDeltaTimeSeconds());

if (earthQuad)

earthQuad->render(glm::mat4(1.0));

glUseProgram(0);

string title = "FPS: " + std::to\_string(timer.averageFPS()) + " SPF: " + std::to\_string(timer.currentSPF());

glfwSetWindowTitle(window, title.c\_str());

// glfw: swap buffers and poll events

glfwSwapBuffers(window);

glfwPollEvents();

}

// glfw: terminate, clearing all previously allocated GLFW resources.

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly

void processInput(GLFWwindow\* window)

{

timer.updateDeltaTime();

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)

camera.processKeyboard(FORWARD, timer.getDeltaTimeSeconds());

if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)

camera.processKeyboard(BACKWARD, timer.getDeltaTimeSeconds());

if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)

camera.processKeyboard(LEFT, timer.getDeltaTimeSeconds());

if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)

camera.processKeyboard(RIGHT, timer.getDeltaTimeSeconds());

int setting0 = sample;

if (glfwGetKey(window, GLFW\_KEY\_KP\_0) == GLFW\_PRESS)

sample = none;

if (glfwGetKey(window, GLFW\_KEY\_KP\_7) == GLFW\_PRESS)

sample = simple;

if (glfwGetKey(window, GLFW\_KEY\_KP\_8) == GLFW\_PRESS)

sample = high;

int setting1 = resolution;

if (glfwGetKey(window, GLFW\_KEY\_KP\_1) == GLFW\_PRESS)

resolution = x1;

if (glfwGetKey(window, GLFW\_KEY\_KP\_2) == GLFW\_PRESS)

resolution = x2;

if (glfwGetKey(window, GLFW\_KEY\_KP\_3) == GLFW\_PRESS)

resolution = x3;

if (glfwGetKey(window, GLFW\_KEY\_KP\_4) == GLFW\_PRESS)

resolution = x4;

if (setting0 != sample || setting1 != resolution)

{

cout << "--New Settings--\nSample Size: " << sample << "\nResolution: " << resolution << "\n\n";

}

if (glfwGetKey(window, GLFW\_KEY\_SPACE) == GLFW\_PRESS)

{

earthScene = new EarthScene(sample, resolution);

earthQuad = new TexturedQuad(earthScene->getEarthSceneTexture(), true);

// Comment / uncomment if changes are desired to be made at runtime

// if (sample == none)

// {

// glDisable(GL\_MULTISAMPLE);

// }

// else

// {

// glEnable(GL\_MULTISAMPLE);

// glfwWindowHint(GLFW\_SAMPLES, 10000);

// }

}

}

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mods)

{

if (glfwGetKey(window, GLFW\_KEY\_SPACE) == GLFW\_PRESS)

showEarthQuad = !showEarthQuad;

}

// glfw: whenever the window size changed (by OS or user resize) this callback function executes

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height)

{

// make sure the viewport matches the new window dimensions; note that width and

glViewport(0, 0, width, height);

camera.updateScreenSize(width, height);

}

// glfw: whenever the mouse moves, this callback is called

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos)

{

double xoffset = xpos - lastX;

double yoffset = lastY - ypos; // reversed since y-coordinates go from bottom to top

lastX = xpos;

lastY = ypos;

if (glfwGetKey(window, GLFW\_KEY\_LEFT\_CONTROL) == GLFW\_PRESS && glfwGetMouseButton(window, GLFW\_MOUSE\_BUTTON\_LEFT) == GLFW\_PRESS)

{

Camera \*ecam = earthScene->getEarthSceneCamera();

ecam->processMouseMovement(xoffset, yoffset);

}

else if (glfwGetMouseButton(window, GLFW\_MOUSE\_BUTTON\_LEFT) == GLFW\_PRESS)

{

camera.processMouseMovement(xoffset, yoffset);

}

}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)

{

if (glfwGetKey(window, GLFW\_KEY\_LEFT\_CONTROL) == GLFW\_PRESS)

{

Camera \*ecam = earthScene->getEarthSceneCamera();

ecam->processMouseScroll(yoffset);

}

else

{

camera.processMouseScroll(yoffset);

}

}

**Custom Shader.vert**

#version 330

//

// model-view-projection matrices

// note: seperate out model transform matrix so we can move vertices into world coords for lighting

//

uniform mat4 modelMatrix; // to calc world coords of vertex

uniform mat4 invTransposeModelMatrix; // inverse transpose of model matrix to transform normal vector into world coords

uniform mat4 viewProjectionMatrix; // to calc clip coords once lighting done in world space

//

// input vertex packet

//

layout (location = 0) in vec4 vertexPos;

layout (location = 1) in vec4 vertexColour;

layout (location = 2) in vec3 vertexNormal;

layout (location = 3) in vec2 vertexTexCoord;

//

// output vertex packet

//

out vec4 posWorldCoord;

out vec4 colour;

out vec3 normalWorldCoord;

out vec2 texCoord;

void main(void) {

// vertex position in world coords - for fragment shader

posWorldCoord = modelMatrix \* vertexPos;

// setup output packet (fragment shader gets packet with interpolated values)

colour = vertexColour;

normalWorldCoord = (invTransposeModelMatrix \* vec4(vertexNormal, 0.0)).xyz; // normal transformed to world coordinate space

//normalWorldCoord = normalize(normalWorldCoord); // can renormalise normal due to scaling (but done in fragment shader anyway!)

texCoord = vertexTexCoord;

// vertex position in clip coords - necessary for pipeline

gl\_Position = viewProjectionMatrix \* modelMatrix \* vertexPos;

}

**CustomShader.frag**

#version 330

//

// basic directional light model

//

uniform vec4 lightDirection; // direction light comes FROM (specified in World Coordinates)

uniform vec4 lightDiffuseColour;

uniform vec4 lightSpecularColour;

uniform float lightSpecularExponent;

uniform vec3 cameraPos; // to calculate specular lighting in world coordinate space, we need the location of the camera since the specular light

// term is viewer dependent

uniform sampler2D modelTexture;

uniform float screenWidth;

uniform float screenHeight;

uniform int sampleSize;

//

// input fragment packet (contains interpolated values for the fragment calculated by the rasteriser)

//

in vec4 posWorldCoord;

in vec4 colour;

in vec3 normalWorldCoord;

in vec2 texCoord;

//

// output fragment colour

//

layout (location = 0) out vec4 fragColour;

void main(void) {

float tx = 1/screenWidth;

float ty = 1/screenHeight;

// make sure light direction vector is unit length (store in L)

vec4 L = normalize(lightDirection);

// important to normalise length of normal otherwise shading artefacts occur

vec3 N = normalize(normalWorldCoord);

// calculate lambertian term

float dp = dot(L.xyz, N);

float lambertian = clamp(dp, 0.0, 1.0);

//

// calculate diffuse light colour

//vec4 texColour = texture(modelTexture, texCoord);

vec4 texColour = texture(modelTexture, vec2(texCoord.x, texCoord.y));

vec3 diffuseColour;

if(sampleSize == 0)

{

diffuseColour = texColour.rgb \* lightDiffuseColour.rgb \* lambertian; // input colour actually diffuse colour

}

else

{

vec3 col = vec3(1.0);

if( sampleSize == 5)

{

// work out offsets

vec2 offsets[5] = vec2[]

(

vec2(-tx, ty), // TL

vec2( tx, ty), // TR

vec2( 0.0f, 0.0f), // CM

vec2(-tx, -ty), // BL

vec2( tx, -ty) // BR

);

//make sure to weight things appropriately

float weightMatrix[5] = float[](

-1, -1,

5,

-1, -1

);

// get textures using offsets

vec3 sampleTex[5];

for(int i = 0; i < 5; i++)

{

sampleTex[i] = vec3(texture(modelTexture, texCoord.st + offsets[i]));

}

vec3 col = vec3(0.0);

// obtain colour (with weightMatrix)

for(int i = 0; i < 5; i++)

col += sampleTex[i] \* weightMatrix[i];

vec3 diffuseColour = col.rgb \* lightDiffuseColour.rgb \* lambertian; // input colour actually diffuse colour

}

if (sampleSize == 9)

{

vec2 offsets[9] = vec2[]

(

vec2(-tx, ty), // TL

vec2( 0.0f, ty), // CR

vec2( tx, ty), // TR

vec2(-tx, 0.0f), // CL

vec2( 0.0f, 0.0f), // CM

vec2( tx, 0.0f), // CR

vec2(-tx, -ty), // BL

vec2( 0.0f, -ty), // BC

vec2( tx, -ty) // BR

);

float weightMatrix[9] = float[](

-1, -1, -1,

-1, 9, -1,

-1, -1, -1

);

vec3 sampleTex[9];

for(int i = 0; i < 9; i++)

{

sampleTex[i] = vec3(texture(modelTexture, texCoord.st + offsets[i]));

}

vec3 col = vec3(0.0);

for(int i = 0; i < 9; i++)

col += sampleTex[i] \* weightMatrix[i];

vec3 diffuseColour = col.rgb \* lightDiffuseColour.rgb \* lambertian; // input colour actually diffuse colour

}

}

//

// calculate specular light colour

//

// vectors needed for specular light calculation...

vec3 E = cameraPos - posWorldCoord.xyz; // vector from point on object surface in world coords to camera

E = normalize(E);

vec3 R = reflect(-L.xyz, N); // reflected light vector about normal N

float specularIntensity = pow(max(dot(R, E), 0.0), lightSpecularExponent);

vec3 specularColour = vec3(1.0f, 1.0f, 1.0f) \* lightSpecularColour.rgb \* specularIntensity \* lambertian;

vec3 dayTime = (diffuseColour + (specularColour)) \* lambertian;

//

// combine colour components to get final pixel / fragment colour

//

vec3 finalColour = dayTime;

// Output final gamma corrected colour to framebuffer

vec3 P = vec3(1.0 / 2.2);

fragColour = vec4(pow(finalColour, P), 1.0);

}

**EarthScene.h**

#ifndef EARTH\_SCENE\_H

#define EARTH\_SCENE\_H

#include "Sphere.h"

#include "Camera.h"

#ifndef DETAIL\_STRUCTS

#define DETAIL\_STRUCTS

enum SampleSize

{

none = 0,

simple = 5,

high = 9,

};

enum Resolution

{

x1 = 1,

x2 = 2,

x3 = 3,

x4 = 4

};

#endif DETAIL\_STRUCTS

class Sphere;

class EarthScene{

private:

Sphere \*marbleSphere;

// Move around the earth with a seperate camera to the main scene camera

Camera \*earthCamera;

Camera\_settings camera\_settings;

// Textures for multi-texturing the earth model

GLuint marbleTexture0;

GLuint marbleTexture1;

GLuint marbleTexture2;

// Shader for multi-texturing the earth

GLuint earthShader;

GLuint CustomShader;

GLuint marbleShader1;

GLuint marbleShader2;

GLuint marbleShader3;

// Unifom locations for earthShader

// Texture uniforms

GLuint marbleTextureUniform0;

// Camera uniforms

GLint modelMatrixLocation;

GLint invTransposeMatrixLocation;

GLint viewProjectionMatrixLocation;

// Directional light uniforms

GLint lightDirectionLocation;

GLint lightDiffuseLocation;

GLint lightSpecularLocation;

GLint lightSpecExpLocation;

GLint cameraPosLocation;

GLint screenWidth;

GLint screenHeight;

GLint sampleSize;

//

// Animation state

//

float sunTheta; // Angle to the Sun in the orbital plane of the Earth (the xz plane in the demo)

float earthTheta;

//

// Framebuffer Object (FBO) variables

//

// Actual FBO

GLuint demoFBO;

// Colour texture to render into

GLuint fboColourTexture;

// Depth texture to render into

GLuint fboDepthTexture;

// Flag to indicate that the FBO is valid

bool fboOkay;

GLuint& shader = CustomShader;

int shaderType;

SampleSize sample;

Resolution resolution;

public:

EarthScene(SampleSize \_sample, Resolution \_resolution);

~EarthScene();

// Accessor methods

Camera\* getEarthSceneCamera();

GLuint getEarthSceneTexture();

float getSunTheta();

void updateSunTheta(float thetaDelta);

// Scene update

void update(const float timeDelta);

// Rendering methods

void render();

};

#endif

**EarthScene.cpp**

#include "EarthScene.h"

#include "TextureLoader.h"

#include "ShaderLoader.h"

#include <iostream>

using namespace std;

EarthScene::EarthScene(SampleSize \_sample, Resolution \_resolution)

{

sample = \_sample;

resolution = \_resolution;

// Camera settings

// width, heigh, near plane, far plane

camera\_settings = { 960 \* (unsigned int)\_resolution, 540 \* (unsigned int)\_resolution, 0.1, 100.0 };

marbleSphere = new Sphere(32, 16, 0.10f, glm::vec4(1.0f, 0.0f, 0.0f, 1.0f), CG\_RIGHTHANDED);

// Instanciate the camera object with basic data

earthCamera = new Camera(camera\_settings, glm::vec3(0.0, 0.0, 5.0));

//

// Setup textures for rendering the model

//

marbleTexture0 = TextureLoader::loadTexture(string("Resources\\Models\\Marbles\\Marble0.jpg"), TextureGenProperties(GL\_SRGB8\_ALPHA8));

marbleTexture1 = TextureLoader::loadTexture(string("Resources\\Models\\Marbles\\Marble1.jpg"), TextureGenProperties(GL\_SRGB8\_ALPHA8));

marbleTexture2 = TextureLoader::loadTexture(string("Resources\\Models\\Marbles\\Marble2.jpg"), TextureGenProperties(GL\_SRGB8\_ALPHA8));

GLSL\_ERROR glsl\_err = ShaderLoader::createShaderProgram(

"Resources\\Shaders\\CustomShader.vert",

"Resources\\Shaders\\CustomShader.frag",

&CustomShader);

shader = CustomShader;

marbleTextureUniform0 = glGetUniformLocation(shader, "modelTexture");

modelMatrixLocation = glGetUniformLocation(shader, "modelMatrix");

invTransposeMatrixLocation = glGetUniformLocation(shader, "invTransposeModelMatrix");

viewProjectionMatrixLocation = glGetUniformLocation(shader, "viewProjectionMatrix");

lightDirectionLocation = glGetUniformLocation(shader, "lightDirection");

lightDiffuseLocation = glGetUniformLocation(shader, "lightDiffuseColour");

lightSpecularLocation = glGetUniformLocation(shader, "lightSpecularColour");

lightSpecExpLocation = glGetUniformLocation(shader, "lightSpecularExponent");

cameraPosLocation = glGetUniformLocation(shader, "cameraPos");

screenWidth = glGetUniformLocation(shader, "screenWidth");

screenHeight = glGetUniformLocation(shader, "screenHeight");

sampleSize = glGetUniformLocation(shader, "sampleSize");

// Set constant uniform data (uniforms that will not change while the application is running)

// Note: Remember we need to bind the shader before we can set uniform variables!

glUseProgram(shader);

glUniform1i(marbleTextureUniform0, 0);

glUseProgram(0);

//

// Setup FBO (which Earth rendering pass will draw into)

//

glGenFramebuffers(1, &demoFBO);

glBindFramebuffer(GL\_FRAMEBUFFER, demoFBO);

//

// Setup textures that will be drawn into through the FBO

//

// Setup colour buffer texture.

// Note: The texture is stored as linear RGB values (GL\_RGBA8).

//There is no need to pass a pointer to image data -

//we're going to fill in the image when we render the Earth scene at render time!

glGenTextures(1, &fboColourTexture);

glBindTexture(GL\_TEXTURE\_2D, fboColourTexture);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA8, camera\_settings.screenWidth, camera\_settings.screenHeight, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, NULL);

if (sample != none)

{

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

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

}

else

{

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\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

// Setup depth texture

glGenTextures(1, &fboDepthTexture);

glBindTexture(GL\_TEXTURE\_2D, fboDepthTexture);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_DEPTH\_COMPONENT24, camera\_settings.screenWidth, camera\_settings.screenHeight, 0, GL\_DEPTH\_COMPONENT, GL\_UNSIGNED\_INT, NULL);

if (sample != none)

{

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

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

}

else

{

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\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

// Attach textures to the FBO

//

// Attach the colour texture object to the framebuffer object's colour attachment point #0

glFramebufferTexture2D(

GL\_FRAMEBUFFER,

GL\_COLOR\_ATTACHMENT0,

GL\_TEXTURE\_2D,

fboColourTexture,

0);

// Attach the depth texture object to the framebuffer object's depth attachment point

glFramebufferTexture2D(

GL\_FRAMEBUFFER,

GL\_DEPTH\_ATTACHMENT,

GL\_TEXTURE\_2D,

fboDepthTexture,

0);

//

// Before proceeding make sure FBO can be used for rendering

//

GLenum demoFBOStatus = glCheckFramebufferStatus(GL\_FRAMEBUFFER);

if (demoFBOStatus != GL\_FRAMEBUFFER\_COMPLETE) {

fboOkay = false;

cout << "Could not successfully create framebuffer object to render texture!" << endl;

}

else {

fboOkay = true;

cout << "FBO successfully created" << endl;

}

// Unbind FBO for now! (Plug main framebuffer back in as rendering destination)

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

glBlitFramebuffer(0, 0, camera\_settings.screenWidth, camera\_settings.screenHeight, 0, 0, camera\_settings.screenWidth, camera\_settings.screenHeight, GL\_COLOR\_BUFFER\_BIT, GL\_NEAREST);

//

// Setup demo / animation variables

//

sunTheta = 0.0f;

earthTheta = 0.0f;

}

EarthScene::~EarthScene() {

}

// Accessor methods

Camera\* EarthScene::getEarthSceneCamera() {

return earthCamera;

}

GLuint EarthScene::getEarthSceneTexture() {

return fboColourTexture;

}

float EarthScene::getSunTheta() {

return sunTheta;

}

void EarthScene::updateSunTheta(float thetaDelta) {

sunTheta += thetaDelta;

}

// Scene update

void EarthScene::update(const float timeDelta) {

// Update rotation angle ready for next frame

//earthTheta += 5.0f \* float(timeDelta);

//updateSunTheta(timeDelta \* 5.0f);

marbleSphere->render();

}

// Rendering methods

void EarthScene::render() {

if (!fboOkay)

return; // Don't render anything if the FBO was not created successfully

// Bind framebuffer object so all rendering redirected to attached images (i.e. our texture)

glBindFramebuffer(GL\_FRAMEBUFFER, demoFBO);

// All rendering from this point goes to the bound textures (setup at initialisation time) and NOT the actual screen!!!!!

// Clear the screen (i.e. the texture)

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

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

//glEnable(GL\_DEPTH\_TEST);

// Set viewport to specified texture size (see above)

glViewport(0, 0, camera\_settings.screenWidth, camera\_settings.screenHeight);

// Get view-projection transform as a CGMatrix4

glm::mat4 T = earthCamera->getProjectionMatrix() \* earthCamera->getViewMatrix();

if (shader)

{

const int cubeAmount = 1600;

vector<glm::vec3> cubePositions;

cubePositions.reserve(cubeAmount);

for (int i = 0; i < 40; i++)

{

float y = 3 - (i \* 0.1f);

for (int j = 0; j < 40; j++)

{

float x = -4 + (j \* 0.1f);

cubePositions.emplace\_back(glm::vec3(x, y, 0.0f));

}

}

glUseProgram(shader);

int j = 0;

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

{

// Modelling transform

glm::mat4 modelTransform;

modelTransform = glm::rotate(glm::mat4(1.0), glm::radians(23.44f), glm::vec3(0.0, 0.0, 1.0)); //Earth tilt

modelTransform = glm::translate(glm::mat4(1.0), cubePositions[i]); // Move position

modelTransform = glm::rotate(modelTransform, glm::radians(earthTheta), glm::vec3(0.0, 1.0, 0.0)); //Earth rotation

// Calculate inverse transpose of the modelling transform for correct transformation of normal vectors

glm::mat4 inverseTranspose = glm::transpose(glm::inverse(modelTransform));;

// Get the location of the camera in world coords and set the corresponding uniform in the shader

glm::vec3 cameraPos = earthCamera->getCameraPosition();

glUniform3fv(cameraPosLocation, 1, (GLfloat\*)&cameraPos);

// Set the model, view and projection matrix uniforms (from the camera data obtained above)

glUniformMatrix4fv(modelMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(modelTransform));

glUniformMatrix4fv(invTransposeMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(inverseTranspose));

glUniformMatrix4fv(viewProjectionMatrixLocation, 1, GL\_FALSE, glm::value\_ptr(T));

// Set the light direction uniform vector in world coordinates based on the Sun's position

glUniform4f(lightDirectionLocation, cosf(glm::radians(sunTheta)), 0.0f, sinf(glm::radians(sunTheta)), 0.0f);

glUniform4f(lightDiffuseLocation, 1.0f, 1.0f, 1.0f, 1.0f); // white diffuse light

glUniform4f(lightSpecularLocation, 0.4f, 0.4f, 0.4f, 1.0f); // white specular light

glUniform1f(lightSpecExpLocation, 8.0f); // specular exponent / falloff

glUniform1f(screenWidth, (float)camera\_settings.screenWidth); // specular exponent / falloff

glUniform1f(screenHeight, (float)camera\_settings.screenHeight); // specular exponent / falloff

glUniform1f(sampleSize, (GLfloat)sample); // specular exponent / falloff

// Activate and Bind the textures to texture units

glActiveTexture(GL\_TEXTURE0);

if (j == 0)

{

glBindTexture(GL\_TEXTURE\_2D, marbleTexture0);

j++;

}

else if (j == 1)

{

glBindTexture(GL\_TEXTURE\_2D, marbleTexture1);

j++;

}

else if (j == 2)

{

glBindTexture(GL\_TEXTURE\_2D, marbleTexture2);

j = 0;

}

//Render the model

marbleSphere->render();

}

}

// Restore default OpenGL shaders (Fixed function operations)

glUseProgram(0);

// Set OpenGL to render to the MAIN framebuffer (ie. the screen itself!!)

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

}

**A full repository of code used can be found at the following github address. Should there be any errors or code be incorrectly displayed please confirm with code found at the repository. This will include images referenced within the code. All images were found on** [**https://www.publicdomainpictures.net**](https://www.publicdomainpictures.net) **with their specific links found under the issues tab of the repository. They are public domain images and do not require accreditation. A copy of this report can also be found under the issues tab.**

[**www.github.com/30005209/CW\_FSAA**](http://www.github.com/30005209/CW_FSAA)