Image Processing Techniques

COMP3015 Lab 5

# Appendix

## Bloom Effect

### Composite Pass Function (5)

// (Read from BlurTex1 and HdrTex, write to default buffer).

vec4 pass5() {

/////////////// Tone mapping ///////////////

// Retrieve high-res color from texture

vec4 color = texture( HdrTex, TexCoord );

// Convert to XYZ

vec3 xyzCol = rgb2xyz \* vec3(color);

// Convert to xyY

float xyzSum = xyzCol.x + xyzCol.y + xyzCol.z;

vec3 xyYCol = vec3( xyzCol.x / xyzSum, xyzCol.y / xyzSum, xyzCol.y);

// Apply the tone mapping operation to the luminance (xyYCol.z or xyzCol.y)

float L = (Exposure \* xyYCol.z) / AveLum;

L = (L \* ( 1 + L / (White \* White) )) / ( 1 + L );

// Using the new luminance, convert back to XYZ

xyzCol.x = (L \* xyYCol.x) / (xyYCol.y);

xyzCol.y = L;

xyzCol.z = (L \* (1 - xyYCol.x - xyYCol.y))/xyYCol.y;

// Convert back to RGB

vec4 toneMapColor = vec4( xyz2rgb \* xyzCol, 1.0);

///////////// Combine with blurred texture /////////////

// We want linear filtering on this texture access so that

// we get additional blurring.

vec4 blurTex = texture(BlurTex1, TexCoord);

return toneMapColor + blurTex;

}

### Uniforms, Quad & Buffers

void SceneBasic\_Uniform::initScene()

{

compile();

glClearColor(0.5f, 0.5f, 0.5f, 1.0f);

glEnable(GL\_DEPTH\_TEST);

vec3 intense = vec3(0.6f);

prog.setUniform("Lights[0].L", intense);

prog.setUniform("Lights[1].L", intense);

prog.setUniform("Lights[2].L", intense);

intense = vec3(0.2f);

prog.setUniform("Lights[0].La", intense);

prog.setUniform("Lights[1].La", intense);

prog.setUniform("Lights[2].La", intense);

projection = mat4(1.0f);

angle = glm::pi<float>() / 2.0f;

setupFBO();

// Array for full-screen quad

GLfloat verts[] = {

-1.0f, -1.0f, 0.0f, 1.0f, -1.0f, 0.0f, 1.0f, 1.0f, 0.0f,

-1.0f, -1.0f, 0.0f, 1.0f, 1.0f, 0.0f, -1.0f, 1.0f, 0.0f

};

GLfloat tc[] = {

0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,

0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f

};

// Set up the buffers

unsigned int handle[2];

glGenBuffers(2, handle);

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

glBufferData(GL\_ARRAY\_BUFFER, 6 \* 3 \* sizeof(float), verts, GL\_STATIC\_DRAW);

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

glBufferData(GL\_ARRAY\_BUFFER, 6 \* 2 \* sizeof(float), tc, GL\_STATIC\_DRAW);

// Set up the vertex array object

glGenVertexArrays(1, &fsQuad);

glBindVertexArray(fsQuad);

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

glVertexAttribPointer((GLuint)0, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(0); // Vertex position

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

glVertexAttribPointer((GLuint)2, 2, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(2); // Texture coordinates

glBindVertexArray(0);

prog.setUniform("LumThresh", 1.7f);

float weights[10], sum, sigma2 = 25.0f;

// Compute and sum the weights

weights[0] = gauss(0, sigma2);

sum = weights[0];

for (int i = 1; i < 10; i++) {

weights[i] = gauss(float(i), sigma2);

sum += 2 \* weights[i];

}

// Normalize the weights and set the uniform

for (int i = 0; i < 10; i++) {

std::stringstream uniName;

uniName << "Weight[" << i << "]";

float val = weights[i] / sum;

prog.setUniform(uniName.str().c\_str(), val);

}

// Set up two sampler objects for linear and nearest filtering

GLuint samplers[2];

glGenSamplers(2, samplers);

linearSampler = samplers[0];

nearestSampler = samplers[1];

GLfloat border[] = { 0.0f,0.0f,0.0f,0.0f };

// Set up the nearest sampler

glSamplerParameteri(nearestSampler, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

glSamplerParameteri(nearestSampler, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glSamplerParameteri(nearestSampler, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_BORDER);

glSamplerParameteri(nearestSampler, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_BORDER);

glSamplerParameterfv(nearestSampler, GL\_TEXTURE\_BORDER\_COLOR, border);

// Set up the linear sampler

glSamplerParameteri(linearSampler, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glSamplerParameteri(linearSampler, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glSamplerParameteri(linearSampler, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_BORDER);

glSamplerParameteri(linearSampler, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_BORDER);

glSamplerParameterfv(linearSampler, GL\_TEXTURE\_BORDER\_COLOR, border);

// We want nearest sampling except for the last pass.

glBindSampler(0, nearestSampler);

glBindSampler(1, nearestSampler);

glBindSampler(2, nearestSampler);

}

### FBO Setup Function

//sets up the fbo for rendering to a texture

void SceneBasic\_Uniform::setupFBO() {

// Generate and bind the framebuffer

glGenFramebuffers(1, &hdrFbo);

glBindFramebuffer(GL\_FRAMEBUFFER, hdrFbo);

// Create the texture object

glGenTextures(1, &hdrTex);

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, hdrTex);

glTexStorage2D(GL\_TEXTURE\_2D, 1, GL\_RGB32F, width, height);

// Bind the texture to the FBO

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, hdrTex, 0);

// Create the depth buffer

GLuint depthBuf;

glGenRenderbuffers(1, &depthBuf);

glBindRenderbuffer(GL\_RENDERBUFFER, depthBuf);

glRenderbufferStorage(GL\_RENDERBUFFER, GL\_DEPTH\_COMPONENT, width, height);

// Bind the depth buffer to the FBO

glFramebufferRenderbuffer(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT,

GL\_RENDERBUFFER, depthBuf);

// Set the targets for the fragment output variables

GLenum drawBuffers[] = { GL\_COLOR\_ATTACHMENT0 };

glDrawBuffers(1, drawBuffers);

// Create an FBO for the bright-pass filter and blur

glGenFramebuffers(1, &blurFbo);

glBindFramebuffer(GL\_FRAMEBUFFER, blurFbo);

// Create two texture objects to ping-pong for the bright-pass filter

// and the two-pass blur

bloomBufWidth = width / 8;

bloomBufHeight = height / 8;

glGenTextures(1, &tex1);

glActiveTexture(GL\_TEXTURE1);

glBindTexture(GL\_TEXTURE\_2D, tex1);

glTexStorage2D(GL\_TEXTURE\_2D, 1, GL\_RGB32F, bloomBufWidth, bloomBufHeight);

glActiveTexture(GL\_TEXTURE2);

glGenTextures(1, &tex2);

glBindTexture(GL\_TEXTURE\_2D, tex2);

glTexStorage2D(GL\_TEXTURE\_2D, 1, GL\_RGB32F, bloomBufWidth, bloomBufHeight);

// Bind tex1 to the FBO

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, tex1, 0);

glDrawBuffers(1, drawBuffers);

// Unbind the framebuffer, and revert to default framebuffer

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

}

## Deferred Shading

### Init Scene Function

void SceneBasic\_Uniform::initScene()

{

compile();

glEnable(GL\_DEPTH\_TEST);

float c = 1.5f;

angle = glm::pi<float>() / 2.0f;

// Array for quad

GLfloat verts[] = {

-1.0f, -1.0f, 0.0f, 1.0f, -1.0f, 0.0f, 1.0f, 1.0f, 0.0f,

-1.0f, -1.0f, 0.0f, 1.0f, 1.0f, 0.0f, -1.0f, 1.0f, 0.0f

};

GLfloat tc[] = {

0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,

0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f

};

// Set up the buffers

unsigned int handle[2];

glGenBuffers(2, handle);

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

glBufferData(GL\_ARRAY\_BUFFER, 6 \* 3 \* sizeof(float), verts, GL\_STATIC\_DRAW);

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

glBufferData(GL\_ARRAY\_BUFFER, 6 \* 2 \* sizeof(float), tc, GL\_STATIC\_DRAW);

// Set up the vertex array object

glGenVertexArrays(1, &quad);

glBindVertexArray(quad);

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

glVertexAttribPointer((GLuint)0, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(0); // Vertex position

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

glVertexAttribPointer((GLuint)2, 2, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(2); // Texture coordinates

glBindVertexArray(0);

setupFBO();

prog.setUniform("Light.L", vec3(1.0f));

}

### FBO Setup Function

void SceneBasic\_Uniform::setupFBO()

{

GLuint depthBuf, posTex, normTex, colorTex;

// Create and bind the FBO

glGenFramebuffers(1, &deferredFBO);

glBindFramebuffer(GL\_FRAMEBUFFER, deferredFBO);

// The depth buffer

glGenRenderbuffers(1, &depthBuf);

glBindRenderbuffer(GL\_RENDERBUFFER, depthBuf);

glRenderbufferStorage(GL\_RENDERBUFFER, GL\_DEPTH\_COMPONENT, width, height);

// Create the textures for position, normal and color

createGBufTex(GL\_TEXTURE0, GL\_RGB32F, posTex); // Position

createGBufTex(GL\_TEXTURE1, GL\_RGB32F, normTex); // Normal

createGBufTex(GL\_TEXTURE2, GL\_RGB8, colorTex); // Color

// Attach the textures to the framebuffer

glFramebufferRenderbuffer(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT, GL\_RENDERBUFFER, depthBuf);

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, posTex, 0);

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT1, GL\_TEXTURE\_2D, normTex, 0);

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT2, GL\_TEXTURE\_2D, colorTex, 0);

GLenum drawBuffers[] = { GL\_NONE, GL\_COLOR\_ATTACHMENT0, GL\_COLOR\_ATTACHMENT1,

GL\_COLOR\_ATTACHMENT2 };

glDrawBuffers(4, drawBuffers);

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

}

### Pass 1 Function

void SceneBasic\_Uniform::pass1()

{

prog.setUniform("Pass", 1);

glBindFramebuffer(GL\_FRAMEBUFFER, deferredFBO);

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

glEnable(GL\_DEPTH\_TEST);

view = glm::lookAt(vec3(7.0f \* cos(angle), 4.0f, 7.0f \* sin(angle)), vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f));

projection = glm::perspective(glm::radians(60.0f), (float)width / height, 0.3f, 100.0f);

prog.setUniform("Light.Position", glm::vec4(0.0f, 0.0f, 0.0f, 1.0f));

prog.setUniform("Material.Kd", 0.9f, 0.9f, 0.9f);

model = mat4(1.0f);

model = glm::translate(model, vec3(0.0f, 0.0f, 0.0f));

model = glm::rotate(model, glm::radians(-90.0f), vec3(1.0f, 0.0f, 0.0f));

setMatrices();

teapot.render();

prog.setUniform("Material.Kd", 0.4f, 0.4f, 0.4f);

model = mat4(1.0f);

model = glm::translate(model, vec3(0.0f, -0.75f, 0.0f));

setMatrices();

plane.render();

prog.setUniform("Light.Position", glm::vec4(0.0f, 0.0f, 0.0f, 1.0f));

prog.setUniform("Material.Kd", 0.2f, 0.5f, 0.9f);

model = mat4(1.0f);

model = glm::translate(model, vec3(1.0f, 1.0f, 3.0f));

model = glm::rotate(model, glm::radians(90.0f), vec3(1.0f, 0.0f, 0.0f));

setMatrices();

torus.render();

glFinish();

}