Contents

[Which version Visual Studio and Operating System you used to test/write the code locally? 2](#_Toc71651972)

[How does it work? What makes your shader program special and how does it compare to similar things? (Where did you get the idea from? What did you start with? How did you make yours unique? Did you start with a given shader/project?) 2](#_Toc71651973)

[A link to the Gitub repo you created for public use 33](#_Toc71651974)

[A Link to the unlisted YouTube Video 33](#_Toc71651975)

[References 33](#_Toc71651976)

## Which version Visual Studio and Operating System you used to test/write the code locally?

Microsoft Visual Studio Community 2019

Version 16.7.6

## How does it work? What makes your shader program special and how does it compare to similar things? (Where did you get the idea from? What did you start with? How did you make yours unique? Did you start with a given shader/project?)

For the project I wanted to create a surreal room which showed off as many techniques as possible. In a game sense the scene was envisioned as the climactic boss in a video game however the prototype has no interactivity but the premise of a final level is there. The idea is that the final boss has been attacked and is shooting out particles, the floor and walls are electrified so you need to fly around to avoid it, the orb is powering the machine in the middle. The two coins are collectables and there’s a sky in the middle which the machine is creating. This is a prototype of the finale of a game. The scene also spins so you can view it from different angles.

The project uses two frag vert shader pairs, the basic\_uniform shaders which allow for surface animation, where the objects spins. The Noise\_Shader is misnamed as it originally was for noise but now it just works as a different type of surface animation where the texture moves up and down. Three other techniques are used. Transparency has been implemented in the walls and the orb in the middle, particles are being created near the robot to look like it has been attacked, and the robots face is using noise to look like a sky and clouds.

This was built on top of the template provided by the university and sections of the code are from the lectures and labs however I have modified them to work with one another and added my own content such as all of the objects, a lot of the textures, the Noise\_Shader and a lot of the code. Below is the code.

**scenebasic\_uniform.cpp**

#include "scenebasic\_uniform.h"

#include <iostream>

#include <sstream>

#include <glm/gtc/matrix\_transform.hpp>

#include <cstdio>

#include <cstdlib>

#include <string>

#include "helper/glutils.h"

#include "helper/noisetex.h"

#include <ctime>

using std::cerr;

using std::endl;

using glm::vec3;

using glm::vec4;

using glm::mat4;

using std::string;

SceneBasic\_Uniform::SceneBasic\_Uniform() : plane(13.0f, 10.0f, 200, 2), angle(0.0f), drawBuf(1), time(0), deltaT(0), nParticles(4000), particleLifetime(6.0f), emitterPos(1, 0, 0), emitterDir(-1, 2, 0)

{

// Fountain

mesh = ObjMesh::load("../Project\_Template/media/Fountain.obj", true);

// Building

mesh2 = ObjMesh::load("../Project\_Template/media/BrokenBuilding.obj", true);

// Ball

mesh3 = ObjMesh::load("../Project\_Template/media/Ball.obj", true);

// Coin

mesh4 = ObjMesh::load("../Project\_Template/media/Coin.obj", true);

// Machine

mesh5 = ObjMesh::load("../Project\_Template/media/Sign.obj", true);

}

// Initialises scene

void SceneBasic\_Uniform::initScene()

{

compile();

// Void And Sky In Cloud Colour

glClearColor(0.2f, 0.4f, 1.7f, 0.9f);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glEnable(GL\_DEPTH\_TEST);

projection = mat4(1.0f);

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

};

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);

glGenVertexArrays(1, &quad);

glBindVertexArray(quad);

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

glVertexAttribPointer((GLuint)0, 3, GL\_FLOAT, GL\_FALSE, 0, ((GLubyte\*)NULL + (0)));

glEnableVertexAttribArray(0);

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

glVertexAttribPointer((GLuint)2, 2, GL\_FLOAT, GL\_FALSE, 0, ((GLubyte\*)NULL + (0)));

glEnableVertexAttribArray(2);

glBindVertexArray(0);

model = mat4(1.0f);

glActiveTexture(GL\_TEXTURE0);

Texture::loadTexture("../Project\_Template/media/texture/Disco.jpg");

glActiveTexture(GL\_TEXTURE1);

ParticleUtils::createRandomTex1D(nParticles \* 3);

initBuffers();

//Spin Shader Spins Textures

prog.use();

prog.setUniform("RandomTex", 1);

prog.setUniform("ParticleTex", 0);

// Particle lifetime

prog.setUniform("ParticleLifetime", particleLifetime);

// Particle accelleration

prog.setUniform("Accel", vec3(0.0f, -0.5, 0.0f));

// Size of particles, also messes with all that uses then spin shader

prog.setUniform("ParticleSize", 0.05f);

// Position of emitter

prog.setUniform("EmitterPos", emitterPos);

prog.setUniform("EmitterBasis", ParticleUtils::makeArbitraryBasis(emitterDir));

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

angle = glm::root\_half\_pi<float>();

//Jutter Shader Jitters Textures

flatProg.use();

flatProg.setUniform("RandomTex", 2);

flatProg.setUniform("ParticleTex", 0);

flatProg.setUniform("ParticleLifetime", particleLifetime);

flatProg.setUniform("Accel", vec3(0.0f, -0.5, 0.0f));

flatProg.setUniform("ParticleSize", 0.05f);

flatProg.setUniform("EmitterPos", emitterPos);

flatProg.setUniform("EmitterBasis", ParticleUtils::makeArbitraryBasis(emitterDir));

flatProg.setUniform("Light.Intensity", vec3(1.0f, 1.0f, 1.0f));

}

// Initialises buffers

void SceneBasic\_Uniform::initBuffers()

{

glGenBuffers(2, posBuf);

glGenBuffers(2, velBuf);

glGenBuffers(2, age);

// Establishes and binds buffers

int size = nParticles \* 3 \* sizeof(GLfloat);

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

glBufferData(GL\_ARRAY\_BUFFER, size, 0, GL\_DYNAMIC\_COPY);

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

glBufferData(GL\_ARRAY\_BUFFER, size, 0, GL\_DYNAMIC\_COPY);

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

glBufferData(GL\_ARRAY\_BUFFER, size, 0, GL\_DYNAMIC\_COPY);

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

glBufferData(GL\_ARRAY\_BUFFER, size, 0, GL\_DYNAMIC\_COPY);

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

glBufferData(GL\_ARRAY\_BUFFER, nParticles \* sizeof(float), 0, GL\_DYNAMIC\_COPY);

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

glBufferData(GL\_ARRAY\_BUFFER, nParticles \* sizeof(float), 0, GL\_DYNAMIC\_COPY);

std::vector<GLfloat>tempData(nParticles);

float rate = particleLifetime / nParticles;

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

tempData[i] = rate \* (i - nParticles);

}

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

glBufferSubData(GL\_ARRAY\_BUFFER, 0, nParticles \* sizeof(float), tempData.data());

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glGenVertexArrays(2, particleArray);

glBindVertexArray(particleArray[0]);

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

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

glEnableVertexAttribArray(0);

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

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(1);

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

glVertexAttribPointer(2, 1, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(2);

glBindVertexArray(particleArray[1]);

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

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

glEnableVertexAttribArray(0);

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

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(1);

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

glVertexAttribPointer(2, 1, GL\_FLOAT, GL\_FALSE, 0, 0);

glEnableVertexAttribArray(2);

glBindVertexArray(0);

glGenTransformFeedbacks(2, feedback);

glBindTransformFeedback(GL\_TRANSFORM\_FEEDBACK, feedback[0]);

glBindBufferBase(GL\_TRANSFORM\_FEEDBACK\_BUFFER, 0, posBuf[0]);

glBindBufferBase(GL\_TRANSFORM\_FEEDBACK\_BUFFER, 1, velBuf[0]);

glBindBufferBase(GL\_TRANSFORM\_FEEDBACK\_BUFFER, 2, age[0]);

glBindTransformFeedback(GL\_TRANSFORM\_FEEDBACK, feedback[1]);

glBindBufferBase(GL\_TRANSFORM\_FEEDBACK\_BUFFER, 0, posBuf[1]);

glBindBufferBase(GL\_TRANSFORM\_FEEDBACK\_BUFFER, 1, velBuf[1]);

glBindBufferBase(GL\_TRANSFORM\_FEEDBACK\_BUFFER, 2, age[1]);

glBindTransformFeedback(GL\_TRANSFORM\_FEEDBACK, 0);

}

// Compiles Shaders

void SceneBasic\_Uniform::compile()

{

try {

// Compile spin shader

prog.compileShader("shader/basic\_uniform.vert");

prog.compileShader("shader/basic\_uniform.frag");

GLuint progHandle = prog.getHandle();

const char\* outputNames[] = { "Position", "Velocity", "Age" };

glTransformFeedbackVaryings(progHandle, 3, outputNames, GL\_SEPARATE\_ATTRIBS);

prog.link();

// Compile jitter shader

flatProg.compileShader("shader/Noise\_Shader.frag");

flatProg.compileShader("shader/Noise\_Shader.vert");

flatProg.link();

}

catch (GLSLProgramException& e) {

cerr << e.what() << endl;

exit(EXIT\_FAILURE);

}

}

// Updates

void SceneBasic\_Uniform::update(float t)

{

deltaT = t - time;

time = t;

angle = std::fmod(angle + 0.01f, glm::two\_pi<float>());

}

// Renders everything

void SceneBasic\_Uniform::render()

{

model = mat4(1.0f);

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

flatProg.use();

setMatrices(flatProg);

prog.use();

prog.setUniform("Time", time);

prog.setUniform("DeltaT", deltaT);

prog.setUniform("Pass", 1);

glEnable(GL\_RASTERIZER\_DISCARD);

glBindTransformFeedback(GL\_TRANSFORM\_FEEDBACK, feedback[drawBuf]);

glBeginTransformFeedback(GL\_POINTS);

glBindVertexArray(particleArray[1 - drawBuf]);

glVertexAttribDivisor(0, 0);

glVertexAttribDivisor(1, 0);

glVertexAttribDivisor(2, 0);

glDrawArrays(GL\_POINTS, 0, nParticles);

glBindVertexArray(0);

glEndTransformFeedback();

glDisable(GL\_RASTERIZER\_DISCARD);

prog.setUniform("Pass", 2);

view = glm::lookAt(vec3(4.0f \* cos(angle), 1.5f, 4.0f \* sin(angle)), vec3(0.0f, 1.5f, 0.0f), vec3(0.0f, 1.0f, 0.0f));

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

// Sets uniform for both shaders

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

prog.setUniform("Material.Ks", 0.8f, 0.8f, 0.8f);

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

prog.setUniform("Material.Shininess", 100.0f);

flatProg.setUniform("Material.kd", 0.2f, 0.5f, 0.9f);

flatProg.setUniform("Material.Ks", 0.8f, 0.8f, 0.8f);

flatProg.setUniform("Material.Ka", 0.2f, 0.5f, 0.9f);

flatProg.setUniform("Material.Shininess", 100.0f);

view = glm::translate(view, glm::vec3(0.0f, 3.0f, 0.0f));

// Amount of Clouds

GLuint noiseTex = NoiseTex::generate2DTex(5.0f);

glBindTexture(GL\_TEXTURE\_2D, noiseTex);

glActiveTexture(GL\_TEXTURE0);

drawScene();

prog.link();

prog.use();

setMatrices(prog);

model = mat4(1.0f);

// Mesh 1

view = glm::translate(view, glm::vec3(0.0f, -3.0f, 0.0f));

glActiveTexture(GL\_TEXTURE0);

Texture::loadTexture("../Project\_Template/media/texture/Disco.jpg");

prog.link();

prog.use();

model = mat4(1.0f);

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

setMatrices(prog);

mesh->render();

// Plane

glActiveTexture(GL\_TEXTURE0);

Texture::loadTexture("../Project\_Template/media/texture/Laser.jpg");

flatProg.link();

flatProg.use();

model = mat4(1.0f);

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

setMatrices(flatProg);

plane.render();

// Mesh 2 + 3

GLuint texture2 = Texture::loadTexture("../Project\_Template/media/texture/Energy.png");

prog.link();

prog.use();

glBindTexture(GL\_TEXTURE\_2D, texture2);

glActiveTexture(GL\_TEXTURE1);

model = mat4(1.0f);

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

setMatrices(prog);

mesh2->render();

mesh3->render();

// Mesh 4

glActiveTexture(GL\_TEXTURE0);

Texture::loadTexture("../Project\_Template/media/texture/Gold.jpg");

prog.link();

prog.use();

model = mat4(1.0f);

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

view = glm::translate(view, glm::vec3(3.0f, 0.0f, 0.0f));

setMatrices(prog);

mesh4->render();

model = mat4(1.0f);

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

view = glm::translate(view, glm::vec3(-6.0f, 0.0f, 0.0f));

setMatrices(prog);

mesh4->render();

// Mesh 5

view = glm::translate(view, glm::vec3(6.0f, 0.0f, 0.0f));

glActiveTexture(GL\_TEXTURE0);

Texture::loadTexture("../Project\_Template/media/texture/Level10.png");

prog.link();

prog.use();

model = mat4(1.0f);

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

view = glm::translate(view, glm::vec3(-3.0f, 0.0f, -3.0f));

setMatrices(prog);

mesh5->render();

//Particles

Texture::loadTexture("../Project\_Template/media/texture/star.png");

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

setMatrices(prog);

glDepthMask(GL\_FALSE);

glBindVertexArray(particleArray[drawBuf]);

glVertexAttribDivisor(0, 1);

glVertexAttribDivisor(1, 1);

glVertexAttribDivisor(2, 1);

glDrawArraysInstanced(GL\_TRIANGLES, 0, 6, nParticles);

glActiveTexture(GL\_TEXTURE0);

setMatrices(prog);

prog.link();

prog.use();

model = mat4(1.0f);

setMatrices(prog);

glBindVertexArray(0);

glDepthMask(GL\_TRUE);

glBindVertexArray(quad);

glDrawArrays(GL\_TRIANGLES, 0, 6);

drawBuf = 1 - drawBuf;

}

// Draw scene

void SceneBasic\_Uniform::drawScene()

{

model = mat4(1.0f);

setMatrices(prog);

glBindVertexArray(quad);

glDrawArrays(GL\_TRIANGLES, 0, 6);

}

// Set matrices

void SceneBasic\_Uniform::setMatrices(GLSLProgram& p)

{

mat4 mv = view \* model;

// View

p.setUniform("MV", mv);

p.setUniform("Proj", projection);

// Spin Shader

prog.setUniform("ModelViewMatrix", mv);

prog.setUniform("NormalMatrix", glm::mat3(vec3(mv[0]), vec3(mv[1]), vec3(mv[2])));

prog.setUniform("MVP", projection \* mv);

// Jitter Shader

flatProg.setUniform("ModelViewMatrix", mv);

flatProg.setUniform("NormalMatrix", glm::mat3(vec3(mv[0]), vec3(mv[1]), vec3(mv[2])));

flatProg.setUniform("MVP", projection \* mv);

}

void SceneBasic\_Uniform::resize(int w, int h)

{

glViewport(0, 0, w, h);

width = w;

height = h;

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

}

**scenebasic\_uniform.h**

#ifndef SCENEBASIC\_UNIFORM\_H

#define SCENEBASIC\_UNIFORM\_H

#include "helper/scene.h"

#include "helper/torus.h"

#include <glm/glm.hpp>

#include "helper/plane.h"

#include "helper/objmesh.h"

#include "helper/cube.h"

#include "helper/texture.h"

#include <glad/glad.h>

#include "helper/glslprogram.h"

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include "helper/random.h"

#include "helper/grid.h"

#include "helper/particleutils.h"

class SceneBasic\_Uniform : public Scene

{

private:

Plane plane;

std::unique\_ptr<ObjMesh>mesh;

std::unique\_ptr<ObjMesh>mesh2;

std::unique\_ptr<ObjMesh>mesh3;

std::unique\_ptr<ObjMesh>mesh4;

std::unique\_ptr<ObjMesh>mesh5;

GLSLProgram prog, flatProg;

GLuint quad;

glm::vec3 lightPos;

Random rand;

glm::vec3 emitterPos, emitterDir;

GLuint posBuf[2], velBuf[2], age[2];

GLuint particleArray[2];

GLuint feedback[2];

GLuint drawBuf;

Grid grid;

int nParticles;

float particleLifetime;

float angle;

float time, deltaT;

void initBuffers();

void setMatrices(GLSLProgram&);

void compile();

void drawScene();

public:

SceneBasic\_Uniform();

void initScene();

void update(float t);

void render();

void resize(int, int);

};

#endif // SCENEBASIC\_UNIFORM\_H

**basic\_uniform.frag**

#version 460

in vec2 TexCoord;

in vec3 Position;

in vec3 Normal;

// Light

uniform struct LightInfo {

vec4 LightPosition;

vec3 La;

vec3 L;

vec3 Intensity;

} Light;

// Material

uniform struct MaterialInfo {

vec3 Ka;

vec3 Kd;

vec3 Ks;

float Shininess;

} Material;

// New Frag colour (where most shader informaiton is)

layout (location = 1) out vec4 NewFragColor;

// Frag Color helps it look a little bit better when layered with NewFragColor

layout (location = 0) out vec4 FragColor;

//phongModelDiffAndSpec

vec3 phongModelDiffAndSpec()

{

vec3 n = Normal;

vec3 s = normalize(vec3(Light.LightPosition) - Position);

vec3 v = normalize(-Position.xyz);

vec3 r = reflect(-s,n);

float sDotN = max(dot(s,n),0.0);

vec3 diffuse = Light.Intensity \* Material.Kd \* sDotN;

vec3 spec = vec3(0.0);

if (sDotN > 0.0)

spec = Light.Intensity \* Material.Ks \* pow(max(dot(r,v),0.0),Material.Shininess);

return diffuse + spec;

}

//phongModelDiffAndSpec

subroutine void RenderPassType();

subroutine uniform RenderPassType RenderPass;

subroutine(RenderPassType)

void shadeWithShadow()

{

vec3 ambient = Light.Intensity \* Material.Ka;

vec3 diffAndSpec = phongModelDiffAndSpec();

float shadow = 1.0;

// NewFragColor

NewFragColor = vec4(diffAndSpec \* shadow + ambient, 1.0);

NewFragColor = pow(NewFragColor, vec4(1.0 / 2.2));

}

subroutine (RenderPassType)

void recordDepth()

{

}

uniform sampler2D ParticleTex;

void main()

{

// FragColor

FragColor = texture(ParticleTex, TexCoord);

// Allows both FragColor and NewFragColor to be passed

RenderPass();

}

**basic\_uniform.vert**

#version 460

const float NUM = 1.23456789;

layout (location = 0) in vec3 VertexPosition;

layout (location = 1) in vec3 VertexVelocity;

layout (location = 2) in float VertexAge;

layout (location = 0) in vec3 AnimVertexPosition;

layout (location = 1) in vec3 VertexNormal;

layout (location = 2) in vec2 VertexTexCoord;

out vec4 AnimPosition;

out vec3 Normal;

out vec3 ShadowNormal;

out vec4 ShadowCoord;

uniform float Freq = 2.5;

uniform float AnimVelocity = 2.5;

uniform float Amp = 0.6;

uniform mat4 ModelViewMatrix;

uniform mat3 NormalMatrix;

uniform mat4 MVP;

uniform mat4 ShadowMatrix;

uniform int Pass;

/\*layout( xfb\_buffer = 0, xfb\_offset = 0)\*/ out vec3 Position;

/\*layout( xfb\_buffer = 1, xfb\_offset = 0)\*/ out vec3 Velocity;

/\*layout( xfb\_buffer = 2, xfb\_offset = 0)\*/ out float Age;

out float Transp;

out vec2 TexCoord;

uniform float Time;

uniform float DeltaT;

uniform vec3 Accel;

uniform float ParticleLifetime;

uniform vec3 Emitter = vec3(0);

uniform mat3 EmitterBasis;

uniform float ParticleSize;

uniform mat4 MV;

uniform mat4 Proj;

uniform sampler1D RandomTex;

const vec3 offsets[] = vec3[](vec3(-0.5,-0.5,0), vec3(0.5,-0.5,0), vec3(0.5,0.5,0), vec3(-0.5,-0.5,0), vec3(0.5,0.5,0.),vec3(-0.5,0.5,0));

const vec2 texCoords[] = vec2[](vec2(0,0), vec2(1,0), vec2(1,1), vec2(0,0), vec2(1,1), vec2(0,1));

vec3 randomInitialVelocity(){

float theta = mix(0.0, NUM / 8.0, texelFetch(RandomTex, 3 \* gl\_VertexID, 0).r);

float phi = mix(0.0, 2.0 \* NUM, texelFetch(RandomTex, 3 \* gl\_VertexID + 1, 0).r);

float velocity = mix(1.25, 1.5, texelFetch(RandomTex, 3 \* gl\_VertexID + 2, 0).r);

vec3 v = vec3(sin(theta) \* cos(phi), cos(theta), sin(theta) \* sin(phi));

return normalize(EmitterBasis \* v) \* velocity;

}

void update(){

if(VertexAge < 0 || VertexAge > ParticleLifetime) {

Position = Emitter;

Velocity = randomInitialVelocity();

if(VertexAge < 0) Age = VertexAge + DeltaT;

else Age = (VertexAge - ParticleLifetime) + DeltaT;

}

else

{

Position = VertexPosition + VertexVelocity \* DeltaT;

Velocity = VertexVelocity + Accel \* DeltaT;

Age = VertexAge + DeltaT;

}

}

void render() {

Transp = 0.0;

vec3 posCam = vec3(0.0);

if(VertexAge >= 0.0) {

posCam = (MV \* vec4(VertexPosition,1)).xyz + offsets[gl\_VertexID]\* ParticleSize;

Transp = clamp(1.0 - VertexAge / ParticleLifetime, 0, 1);

}

TexCoord = texCoords[gl\_VertexID];

gl\_Position = Proj \* vec4(posCam,1);

}

void main(){

vec4 pos = vec4(AnimVertexPosition,1.0);

float u = Freq \* pos.x - AnimVelocity \* Time;

pos.y = Amp \* sin(u);

vec3 n = vec3(0.0);

n.xy = normalize(vec2(cos(u), 1.0));

AnimPosition = ModelViewMatrix \* pos;

Normal = NormalMatrix \* n;

TexCoord = VertexTexCoord;

gl\_Position = MVP \* pos;

Position = (ModelViewMatrix \* vec4(VertexPosition,1.0)).xyz;

ShadowNormal = normalize(NormalMatrix \* VertexNormal);

ShadowCoord = ShadowMatrix \* vec4(VertexPosition,1.0);

if(Pass == 1)

update();

else

render();

}

**Noise\_Shader.frag**

#version 460

in float Transp;

in vec2 TexCoord;

in vec3 Position;

in vec3 Normal;

layout(binding=0) uniform sampler2D Tex1;

layout(binding=1) uniform sampler2D Tex2;

uniform vec3 Ld;

uniform vec3 f;

out vec3 LightIntensity;

uniform struct LightInfo {

// Light position in the eye coordinates

vec4 LightPosition;

// Ambient light intensity

vec3 La;

// Diffuse and specular light intensity

vec3 L;

} lights[3];

uniform struct MaterialInfo {

// Ambient reflectivity

vec3 Ka;

// Diffuse reflectivity

vec3 Kd;

// Specular reflectivity

vec3 Ks;

// Specular shininess

float Shininess;

} Material;

//blinnPhong shader

vec3 blinnPhong(vec3 position, vec3 n ) {

vec4 texColor1 = texture(Tex1, TexCoord);

vec4 texColor2 = texture(Tex2, TexCoord);

vec3 col = mix(texColor1.rgb, texColor2.rgb, texColor2.a);

vec3 ambient = col \* lights[1].La;

vec3 s = lights[0].LightPosition.xyz - col;

float sDotN = (dot(s,n));

vec3 diffuse = col \* sDotN;

vec3 spec = vec3(1.0);

if( sDotN > 0.0 )

{

vec3 v = normalize(-position.xyz);

vec3 h = normalize( v + s );

spec = Material.Ks \* pow( max( dot(h,n), 0.0 ), Material.Shininess );

}

return ambient + lights[2].L \* (diffuse + spec);

}

uniform sampler2D ParticleTex;

layout (location = 0) out vec4 FragColor;

void main()

{

// FragColor

FragColor = texture(ParticleTex, TexCoord);

FragColor.a \*= Transp;

}

**Noise\_Shader.vert**

#version 460

const float PI = 3.14159265359;

layout (location = 0) in vec3 VertexPosition;

layout (location = 1) in vec3 VertexVelocity;

layout (location = 2) in float VertexAge;

layout (location = 0) in vec3 AnimVertexPosition;

layout (location = 1) in vec3 VertexNormal;

layout (location = 2) in vec2 VertexTexCoord;

out vec4 AnimPosition;

out vec3 Normal;

uniform float Freq = 2.5;

uniform float AnimVelocity = 2.5;

uniform float Amp = 0.6;

uniform mat4 ModelViewMatrix;

uniform mat3 NormalMatrix;

uniform mat4 MVP;

uniform int Pass;

/\*layout( xfb\_buffer = 0, xfb\_offset = 0)\*/ out vec3 Position;

/\*layout( xfb\_buffer = 1, xfb\_offset = 0)\*/ out vec3 Velocity;

/\*layout( xfb\_buffer = 2, xfb\_offset = 0)\*/ out float Age;

out float Transp;

out vec2 TexCoord;

uniform float Time;

uniform float DeltaT;

uniform vec3 Accel;

uniform float ParticleLifetime;

uniform vec3 Emitter = vec3(0);

uniform mat3 EmitterBasis;

uniform float ParticleSize;

uniform mat4 MV;

uniform mat4 Proj;

uniform sampler1D RandomTex;

const vec3 offsets[] = vec3[](vec3(-0.5,-0.5,0), vec3(0.5,-0.5,0), vec3(0.5,0.5,0), vec3(-0.5,-0.5,0), vec3(0.5,0.5,0.),vec3(-0.5,0.5,0));

const vec2 texCoords[] = vec2[](vec2(0,0), vec2(1,0), vec2(1,1), vec2(0,0), vec2(1,1), vec2(0,1));

vec3 randomInitialVelocity(){

float theta = mix(0.0, PI / 8.0, texelFetch(RandomTex, 3 \* gl\_VertexID, 0).r);

float phi = mix(0.0, 2.0 \* PI, texelFetch(RandomTex, 3 \* gl\_VertexID + 1, 0).r);

float velocity = mix(1.25, 1.5, texelFetch(RandomTex, 3 \* gl\_VertexID + 2, 0).r);

vec3 v = vec3(sin(theta) \* cos(phi), cos(theta), sin(theta) \* sin(phi));

return normalize(EmitterBasis \* v) \* velocity;

}

void update(){

if(VertexAge < 0 || VertexAge > ParticleLifetime) {

Position = Emitter;

Velocity = randomInitialVelocity();

if(VertexAge < 0) Age = VertexAge + DeltaT;

else Age = (VertexAge - ParticleLifetime) + DeltaT;

}

else

{

Position = VertexPosition + VertexVelocity \* DeltaT;

Velocity = VertexVelocity + Accel \* DeltaT;

Age = VertexAge + DeltaT;

}

}

void render() {

Transp = 0.0;

vec3 posCam = vec3(0.0);

if(VertexAge >= 0.0) {

posCam = (MV \* vec4(VertexPosition,1)).xyz + offsets[gl\_VertexID]\* ParticleSize;

Transp = clamp(1.0 - VertexAge / ParticleLifetime, 0, 1);

}

TexCoord = texCoords[gl\_VertexID];

gl\_Position = Proj \* vec4(posCam,1);

}

void main(){

vec4 pos = vec4(AnimVertexPosition,1.0);

float u = Freq \* pos.x - AnimVelocity \* Time;

pos.y = Amp \* sin(u);

vec3 n = vec3(0.0);

n.xy = normalize(vec2(cos(u), 1.0));

AnimPosition = ModelViewMatrix \* pos;

Normal = NormalMatrix \* n;

TexCoord = VertexTexCoord;

gl\_Position = MVP \* pos;

if(Pass == 1)

update();

else

render();

}

A link to the Gitub repo you created for public usehttps://github.com/CameronDavies01/COMP3015-Games-Graphics-Pipelines-2

## A Link to the unlisted YouTube Video

<https://youtu.be/21JeiLrxtOw>

## References

Viking Signs, n.d. Exit Sign. [image] Available at: <https://img.safetysignsupplies.co.uk/highres/SB432-L32-V-xl.png?status=user-zoom> [Accessed 5 May 2021].