|  |
| --- |
|  |
| Games Programming |
| Coursework 1 |
|  |
| **Calum Timpany** |
| **H00150974** |

|  |
| --- |
|  |

**Vector Class**

**vec3.h**

#ifndef vec3\_H

#define vec3\_H

class vec3 {

public:

vec3();

vec3(float, float, float);

bool operator==(vec3 rhs);

vec3 operator+(vec3 rhs);

vec3 operator-(vec3 rhs);

vec3 operator\*(vec3 rhs);

vec3 operator/(vec3 rhs);

vec3 operator+(float scalar);

vec3 operator-(float scalar);

vec3 operator\*(float scalar);

vec3 operator/(float scalar);

vec3 cross(vec3 rhs);

float dot(vec3 rhs);

float length();

float x;

float y;

float z;

};

#endif

**vec3.cpp**

#include <math.h>

#include "vec3.h"

vec3::vec3() {

x = 0;

y = 0;

z = 0;

}

vec3::vec3(float x, float y, float z) : x(x), y(y), z(z){}

bool vec3::operator==(vec3 rhs) {

return(x == rhs.x && y == rhs.y && z == rhs.z);

}

vec3 vec3::operator+(vec3 rhs) {

return vec3(x + rhs.x,

y + rhs.y,

z + rhs.z);

}

vec3 vec3::operator-(vec3 rhs) {

return vec3(x - rhs.x,

y - rhs.y,

z - rhs.z);

}

vec3 vec3::operator\*(vec3 rhs) {

return vec3(x \* rhs.x,

y \* rhs.y,

z \* rhs.z);

}

vec3 vec3::operator/(vec3 rhs) {

return vec3(x / rhs.x,

y / rhs.y,

z / rhs.z);

}

vec3 vec3::operator+(float scalar) {

return vec3(x + scalar,

y + scalar,

z + scalar);

}

vec3 vec3::operator-(float scalar) {

return vec3(x - scalar,

y - scalar,

z - scalar);

}

vec3 vec3::operator\*(float scalar) {

return vec3(x \* scalar,

y \* scalar,

z \* scalar);

}

vec3 vec3::operator/(float scalar) {

return vec3(x / scalar,

y / scalar,

z / scalar);

}

vec3 vec3::cross(vec3 rhs) {

return vec3(y \* rhs.z - z \* rhs.y,

z \* rhs.x - x \* rhs.z,

x \* rhs.y - y \* rhs.x);

}

float vec3::dot(vec3 rhs) {

return (x \* rhs.x +

y \* rhs.y +

z \* rhs.z);

}

float vec3::length() {

return float(sqrt(x\*x + y\*y + z\*z));

}

**Bouncing Ball**

**Source.cpp**

// Simplified Renderer application for GP course

// Code is similar to the one in lab 1 but all the graphics sections were refactored into the Graphics Class.

// Extra improvements:

// Reduced OpenGL version from 4.5 to 3.3 to allow it to render in older laptops.

// Added Shapes library for rendering cubes, spheres and vectors.

// Added examples of matrix multiplication on Update.

// Added resize screen and keyboard callbacks.

//

// Suggestions or extra help please do email me S.Padilla@hw.ac.uk

//

// Note: Do not forget to link the libraries correctly and add the GLEW DLL in your debug/release folder.

#include <iostream>

#include <vector>

using namespace std;

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <GLM/glm.hpp>

#include <GLM/gtx/transform.hpp>

#include "graphics.h"

#include "shapes.h"

#include "PhysicsSphere.h"

// FUNCTIONS

void render(double currentTime);

void update(double currentTime);

void startup();

void onResizeCallback(GLFWwindow\* window, int w, int h);

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

// VARIABLES

bool running = true;

Graphics myGraphics; // Runing all the graphics in this object

Sphere mySphere;

PhysicsSphere phys;

float t = 0.001f; // Global variable for animation

float angleX = 0.0f;

float angleY = 0.0f;

float disp = -5.0;

float fovy = 45.0f;

float posX = 0.0f, posY = -10.0f, posZ = 50.0f;

glm::vec3 cameraPosition = glm::vec3(posX, posY, posZ);

glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);

glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

GLfloat yaw = -90.0f; // init pointing to inside

GLfloat pitch = 0.0f; // start centered

int main()

{

int errorGraphics = myGraphics.Init(); // Launch window and graphics context

if (errorGraphics) return 0; //Close if something went wrong...

startup(); // Setup all necessary information for startup (aka. load texture, shaders, models, etc).

// Mixed graphics and update functions - declared in main for simplicity.

glfwSetWindowSizeCallback(myGraphics.window, onResizeCallback); // Set callback for resize

glfwSetKeyCallback(myGraphics.window, onKeyCallback); // Set Callback for keys

// MAIN LOOP run until the window is closed

do {

double currentTime = glfwGetTime(); // retrieve timelapse

glfwPollEvents(); // poll callbacks

update(currentTime); // update (physics, animation, structures, etc)

render(currentTime); // call render function.

glfwSwapBuffers(myGraphics.window); // swap buffers (avoid flickering and tearing)

running &= (glfwGetKey(myGraphics.window, GLFW\_KEY\_ESCAPE) == GLFW\_RELEASE); // exit if escape key pressed

running &= (glfwWindowShouldClose(myGraphics.window) != GL\_TRUE);

} while (running);

myGraphics.endProgram(); // Close and clean everything up...

cout << "\nPress any key to continue...\n";

cin.ignore(); cin.get(); // delay closing console to read debugging errors.

return 0;

}

void startup() {

// Calculate proj\_matrix for the first time.

myGraphics.aspect = (float)myGraphics.windowWidth / (float)myGraphics.windowHeight;

myGraphics.proj\_matrix = glm::perspective(glm::radians(50.0f), myGraphics.aspect, 0.1f, 1000.0f);

// Load Geometry

mySphere.Load();

mySphere.fillColor = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f); // You can change the shape fill colour, line colour or linewidth

glm::vec3 front;

front.x = cos(glm::radians(yaw)) \* cos(glm::radians(pitch));

front.y = sin(glm::radians(pitch));

front.z = sin(glm::radians(yaw)) \* cos(glm::radians(pitch));

cameraFront = glm::normalize(front);

myGraphics.SetOptimisations(); // Cull and depth testing

}

void update(double currentTime) {

glm::mat4 viewMatrix = glm::lookAt(cameraPosition, // eye

cameraPosition + cameraFront, // centre

cameraUp); // up

// calculate Sphere movement

phys.Update();

glm::mat4 mv\_matrix\_sphere2 =

glm::translate(phys.getPosition()) \*

glm::scale(glm::vec3(1.0f, 1.0f, 1.0f)) \*

glm::mat4(1.0f);

mySphere.mv\_matrix =viewMatrix \* mv\_matrix\_sphere2;

mySphere.proj\_matrix = myGraphics.proj\_matrix;

}

void render(double currentTime) {

// Clear viewport - start a new frame.

myGraphics.ClearViewport();

mySphere.Draw();

}

void onResizeCallback(GLFWwindow\* window, int w, int h) { // call everytime the window is resized

myGraphics.windowWidth = w;

myGraphics.windowHeight = h;

myGraphics.aspect = (float)w / (float)h;

myGraphics.proj\_matrix = glm::perspective(glm::radians(50.0f), myGraphics.aspect, 0.1f, 1000.0f);

}

void onKeyCallback(GLFWwindow\* window, int key, int scancode, int action, int mods) { // called everytime a key is pressed

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, GLFW\_TRUE);

//if (key == GLFW\_KEY\_RIGHT) posX += 1.0f;

//if (key == GLFW\_KEY\_LEFT) angleY += 0.05f;

}

**PhysicsSphere.h**

#include <iostream>

#include <vector>

using namespace std;

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <GLM/glm.hpp>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <ctime>

#include "vec3.h"

class PhysicsSphere {

public:

PhysicsSphere();

void Update();

glm::vec3 getPosition();

glm::vec3 getDisplacement();

glm::vec3 getVelocity();

protected:

const float gravity = -9.81;

const float resistance = 0.1;

glm::vec3 m\_Position, m\_Displacement, m\_Velocity, m\_Force, m\_gravity, m\_resistance;

float oldTime, newTime;

void Init();

};

**PhysicsSphere.cpp**

#include <iostream>

#include <sstream>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <GLM/glm.hpp>

#include "PhysicsSphere.h"

PhysicsSphere::PhysicsSphere(){

Init();

}

void PhysicsSphere::Init(){

oldTime = NULL;

m\_Position = glm::vec3(0.0, 0.0, 0.0);

m\_Velocity = glm::vec3(0.0, 0.0, 0.0);

m\_gravity = glm::vec3(0.0, gravity, 0.0);

m\_resistance = glm::vec3(resistance, 0.0, 0.0);

m\_Force = glm::vec3(5.0, 100.0, 0.0);

m\_Displacement = glm::vec3(0.0, 0.0, 0.0);

}

void PhysicsSphere::Update() {

float timeDif;

if (oldTime == NULL) {

oldTime = clock();

}

newTime = clock();

timeDif = (newTime - oldTime)/CLOCKS\_PER\_SEC;

if (m\_Force.x > 0.0) {

m\_Force -= m\_resistance;

if (m\_Force.x < 0.0)

m\_Force.x = 0.0;

}

else if (m\_Force.x < 0.0) {

m\_Force += m\_resistance;

if (m\_Force.x > 0.0)

m\_Force.x = 0.0;

}

if (m\_Force.y != gravity) {

m\_Force += m\_gravity;

if (m\_Force.y < gravity)

m\_Force.y = gravity;

}

m\_Displacement = (m\_Velocity\*timeDif) + (0.5f \* (m\_Force \*(timeDif\*timeDif)));

m\_Position += m\_Displacement;

m\_Velocity = m\_Velocity + (m\_Force \* timeDif);

if (m\_Position.y <= -20.0) {

if (m\_Velocity.y > 1.0 || m\_Velocity.y < -1.0) {

m\_Velocity.y = (-m\_Velocity.y)\*0.9;

m\_Velocity.x \*= 0.9;

}else {

m\_Velocity.y = 0.0;

m\_Force.y = 0.0;

m\_Position.y = -20.0;

}

}

if (m\_Position.x <= -20.0 || m\_Position.x >= 20.0) {

m\_Velocity.x = (-m\_Velocity.x)\*0.9;

m\_Velocity.y \*= 0.9;

}

oldTime = newTime;

}

glm::vec3 PhysicsSphere::getPosition() {

return m\_Position;

}

glm::vec3 PhysicsSphere::getDisplacement() {

return m\_Displacement;

}

glm::vec3 PhysicsSphere::getVelocity() {

return m\_Velocity;

}

**Shapes.h**

#pragma once

#include <iostream>

#include <vector>

using namespace std;

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <GLM/glm.hpp>

#define \_USE\_MATH\_DEFINES

#include <math.h>

class Shapes {

public:

Shapes();

~Shapes();

void Load();

void Draw();

void checkErrorShader(GLuint shader);

vector<GLfloat> vertexPositions;

GLuint program;

GLuint vao;

GLuint buffer;

GLint mv\_location;

GLint proj\_location;

GLint color\_location;

glm::mat4 proj\_matrix = glm::mat4(1.0f);

glm::mat4 mv\_matrix = glm::mat4(1.0f);

glm::vec4 fillColor = glm::vec4(1.0, 0.0, 0.0, 1.0);

glm::vec4 lineColor = glm::vec4(0.0, 0.0, 0.0, 1.0);

float lineWidth = 2.0f;

protected:

string rawData; // Import obj file from Blender (note: no textures or UVs).

void LoadObj();

};

class Cube : public Shapes {

public:

Cube();

~Cube();

};

class Sphere : public Shapes {

public:

Sphere();

~Sphere();

};

class Arrow : public Shapes {

public:

Arrow();

~Arrow();

};

**Shapes.cpp**

#include "shapes.h"

#include <iostream>

#include <sstream>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <GLM/glm.hpp>

Shapes::Shapes() {

};

Shapes::~Shapes() {

}

void Shapes::LoadObj() {

std::vector< glm::vec3 > obj\_vertices;

std::vector< unsigned int > vertexIndices;

istringstream rawDataStream(rawData);

string dataLine; int linesDone = 0;

while (std::getline(rawDataStream, dataLine)) {

if (dataLine.find("v ") != string::npos) { // does this line have a vector?

glm::vec3 vertex;

int foundStart = dataLine.find(" "); int foundEnd = dataLine.find(" ", foundStart + 1);

vertex.x = stof(dataLine.substr(foundStart, foundEnd - foundStart));

foundStart = foundEnd; foundEnd = dataLine.find(" ", foundStart + 1);

vertex.y = stof(dataLine.substr(foundStart, foundEnd - foundStart));

foundStart = foundEnd; foundEnd = dataLine.find(" ", foundStart + 1);

vertex.z = stof(dataLine.substr(foundStart, foundEnd - foundStart));

obj\_vertices.push\_back(vertex);

}

else if (dataLine.find("f ") != string::npos) { // does this line defines a triangle face?

string parts[3];

int foundStart = dataLine.find(" "); int foundEnd = dataLine.find(" ", foundStart + 1);

parts[0] = dataLine.substr(foundStart + 1, foundEnd - foundStart - 1);

foundStart = foundEnd; foundEnd = dataLine.find(" ", foundStart + 1);

parts[1] = dataLine.substr(foundStart + 1, foundEnd - foundStart - 1);

foundStart = foundEnd; foundEnd = dataLine.find(" ", foundStart + 1);

parts[2] = dataLine.substr(foundStart + 1, foundEnd - foundStart - 1);

for (int i = 0; i < 3; i++) { // for each part

vertexIndices.push\_back(stoul(parts[i].substr(0, parts[i].find("/"))));

int firstSlash = parts[i].find("/"); int secondSlash = parts[i].find("/", firstSlash + 1);

if (firstSlash != (secondSlash + 1)) { // there is texture coordinates.

// add code for my texture coordintes here.

}

}

}

linesDone++;

}

for (unsigned int i = 0; i < vertexIndices.size(); i += 3) {

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 0] - 1].x);

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 0] - 1].y);

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 0] - 1].z);

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 1] - 1].x);

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 1] - 1].y);

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 1] - 1].z);

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 2] - 1].x);

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 2] - 1].y);

vertexPositions.push\_back(obj\_vertices[vertexIndices[i + 2] - 1].z);

}

}

void Shapes::Load() {

static const char \* vs\_source[] = { R"(

#version 330 core

in vec4 position;

uniform mat4 mv\_matrix;

uniform mat4 proj\_matrix;

void main(void){

gl\_Position = proj\_matrix \* mv\_matrix \* position;

}

)" };

static const char \* fs\_source[] = { R"(

#version 330 core

uniform vec4 inColor;

out vec4 color;

void main(void){

color = inColor;

}

)" };

program = glCreateProgram();

GLuint fs = glCreateShader(GL\_FRAGMENT\_SHADER);

glShaderSource(fs, 1, fs\_source, NULL);

glCompileShader(fs);

checkErrorShader(fs);

GLuint vs = glCreateShader(GL\_VERTEX\_SHADER);

glShaderSource(vs, 1, vs\_source, NULL);

glCompileShader(vs);

checkErrorShader(vs);

glAttachShader(program, vs);

glAttachShader(program, fs);

glLinkProgram(program);

mv\_location = glGetUniformLocation(program, "mv\_matrix");

proj\_location = glGetUniformLocation(program, "proj\_matrix");

color\_location = glGetUniformLocation(program, "inColor");

glGenVertexArrays(1, &vao);

glBindVertexArray(vao);

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);

glBufferData(GL\_ARRAY\_BUFFER,

vertexPositions.size() \* sizeof(GLfloat),

&vertexPositions[0],

GL\_STATIC\_DRAW);

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

glEnableVertexAttribArray(0);

glLinkProgram(0); // unlink

glDisableVertexAttribArray(0); // Disable

glBindVertexArray(0); // Unbind

}

void Shapes::Draw() {

glUseProgram(program);

glBindVertexArray(vao);

glEnableVertexAttribArray(0);

glUniformMatrix4fv(proj\_location, 1, GL\_FALSE, &proj\_matrix[0][0]);

glUniformMatrix4fv(mv\_location, 1, GL\_FALSE, &mv\_matrix[0][0]);

glUniform4f(color\_location, fillColor.r, fillColor.g, fillColor.b, fillColor.a);

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

glDrawArrays(GL\_TRIANGLES, 0, vertexPositions.size() / 3);

glUniform4f(color\_location, lineColor.r, lineColor.g, lineColor.b, lineColor.a);

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

glDrawArrays(GL\_TRIANGLES, 0, vertexPositions.size() / 3);

}

void Shapes::checkErrorShader(GLuint shader) {

// Get log length

GLint maxLength;

glGetShaderiv(shader, GL\_INFO\_LOG\_LENGTH, &maxLength);

// Init a string for it

std::vector<GLchar> errorLog(maxLength);

if (maxLength > 1) {

// Get the log file

glGetShaderInfoLog(shader, maxLength, &maxLength, &errorLog[0]);

cout << "--------------Shader compilation error-------------\n";

cout << errorLog.data();

}

}

Cube::Cube() {

// Exported from Blender a cube by default (OBJ File)

rawData = R"(

v 0.500000 -0.500000 -0.500000

v 0.500000 -0.500000 0.500000

v -0.500000 -0.500000 0.500000

v -0.500000 -0.500000 -0.500000

v 0.500000 0.500000 -0.499999

v 0.499999 0.500000 0.500000

v -0.500000 0.500000 0.500000

v -0.500000 0.500000 -0.500000

f 1 3 4

f 8 6 5

f 5 2 1

f 6 3 2

f 7 4 3

f 1 8 5

f 1 2 3

f 8 7 6

f 5 6 2

f 6 7 3

f 7 8 4

f 1 4 8)";

LoadObj();

}

Cube::~Cube() {

}

Sphere::Sphere() {

rawData = R"(

o Sphere

v -0.097545 0.490393 0.000000

v -0.277785 0.415735 0.000000

v -0.415735 0.277785 0.000000

v -0.490393 0.097545 0.000000

v -0.490393 -0.097545 0.000000

v -0.415735 -0.277785 0.000000

v -0.277785 -0.415735 0.000000

v -0.097545 -0.490393 0.000000

v -0.090120 0.490393 -0.037329

v -0.256640 0.415735 -0.106304

v -0.384089 0.277785 -0.159095

v -0.453064 0.097545 -0.187665

v -0.453064 -0.097545 -0.187665

v -0.384089 -0.277785 -0.159095

v -0.256640 -0.415735 -0.106304

v -0.090120 -0.490393 -0.037329

v -0.068975 0.490393 -0.068975

v -0.196424 0.415735 -0.196424

v -0.293969 0.277785 -0.293969

v -0.346760 0.097545 -0.346760

v -0.346760 -0.097545 -0.346760

v -0.293969 -0.277785 -0.293969

v -0.196424 -0.415735 -0.196424

v -0.068975 -0.490393 -0.068975

v -0.037329 0.490393 -0.090120

v -0.106304 0.415735 -0.256640

v -0.159095 0.277785 -0.384089

v -0.187665 0.097545 -0.453064

v -0.187665 -0.097545 -0.453064

v -0.159095 -0.277785 -0.384089

v -0.106304 -0.415735 -0.256640

v -0.037329 -0.490393 -0.090120

v 0.000000 0.490393 -0.097545

v 0.000000 0.415735 -0.277785

v 0.000000 0.277785 -0.415735

v 0.000000 0.097545 -0.490393

v 0.000000 -0.097545 -0.490393

v 0.000000 -0.277785 -0.415735

v 0.000000 -0.415735 -0.277785

v 0.000000 -0.490393 -0.097545

v 0.037329 0.490393 -0.090120

v 0.106304 0.415735 -0.256640

v 0.159095 0.277785 -0.384089

v 0.187665 0.097545 -0.453064

v 0.187665 -0.097545 -0.453064

v 0.159095 -0.277785 -0.384089

v 0.106304 -0.415735 -0.256640

v 0.037329 -0.490393 -0.090120

v 0.068975 0.490393 -0.068975

v 0.196424 0.415735 -0.196424

v 0.293969 0.277785 -0.293969

v 0.346760 0.097545 -0.346760

v 0.346760 -0.097545 -0.346760

v 0.293969 -0.277785 -0.293969

v 0.196424 -0.415735 -0.196424

v 0.068975 -0.490393 -0.068975

v 0.090120 0.490393 -0.037329

v 0.256640 0.415735 -0.106304

v 0.384089 0.277785 -0.159095

v 0.453064 0.097545 -0.187665

v 0.453064 -0.097545 -0.187665

v 0.384089 -0.277785 -0.159095

v 0.256640 -0.415735 -0.106304

v 0.090120 -0.490393 -0.037329

v 0.097545 0.490393 0.000000

v 0.277785 0.415735 -0.000000

v 0.415735 0.277785 0.000000

v 0.490393 0.097545 0.000000

v 0.490393 -0.097545 0.000000

v 0.415735 -0.277785 0.000000

v 0.277785 -0.415735 0.000000

v 0.097545 -0.490393 -0.000000

v 0.090120 0.490393 0.037329

v 0.256640 0.415735 0.106304

v 0.384089 0.277785 0.159095

v 0.453064 0.097545 0.187665

v 0.453064 -0.097545 0.187665

v 0.384089 -0.277785 0.159095

v 0.256640 -0.415735 0.106304

v 0.090120 -0.490393 0.037329

v 0.068975 0.490393 0.068975

v 0.196424 0.415735 0.196424

v 0.293969 0.277785 0.293969

v 0.346760 0.097545 0.346760

v 0.346760 -0.097545 0.346760

v 0.293969 -0.277785 0.293969

v 0.196424 -0.415735 0.196424

v 0.068975 -0.490393 0.068975

v 0.000000 -0.500000 0.000000

v 0.037329 0.490393 0.090120

v 0.106304 0.415735 0.256640

v 0.159095 0.277785 0.384089

v 0.187665 0.097545 0.453064

v 0.187665 -0.097545 0.453064

v 0.159095 -0.277785 0.384089

v 0.106304 -0.415735 0.256640

v 0.037329 -0.490393 0.090120

v 0.000000 0.490393 0.097545

v 0.000000 0.415735 0.277785

v 0.000000 0.277785 0.415735

v 0.000000 0.097545 0.490392

v 0.000000 -0.097545 0.490392

v 0.000000 -0.277785 0.415735

v 0.000000 -0.415735 0.277785

v 0.000000 -0.490393 0.097545

v -0.037329 0.490393 0.090120

v -0.106304 0.415735 0.256640

v -0.159095 0.277785 0.384089

v -0.187665 0.097545 0.453063

v -0.187665 -0.097545 0.453063

v -0.159095 -0.277785 0.384089

v -0.106304 -0.415735 0.256640

v -0.037329 -0.490393 0.090120

v -0.068975 0.490393 0.068975

v -0.196424 0.415735 0.196424

v -0.293969 0.277785 0.293969

v -0.346760 0.097545 0.346760

v -0.346760 -0.097545 0.346760

v -0.293969 -0.277785 0.293969

v -0.196423 -0.415735 0.196424

v -0.068975 -0.490393 0.068975

v 0.000000 0.500000 0.000000

v -0.090120 0.490393 0.037329

v -0.256640 0.415735 0.106304

v -0.384088 0.277785 0.159095

v -0.453063 0.097545 0.187665

v -0.453063 -0.097545 0.187665

v -0.384088 -0.277785 0.159095

v -0.256640 -0.415735 0.106304

v -0.090120 -0.490393 0.037329

s off

f 7 14 15

f 3 10 11

f 12 3 11

f 8 15 16

f 5 12 13

f 2 125 124

f 2 9 10

f 6 13 14

f 89 8 16

f 122 17 9

f 7 128 6

f 20 27 28

f 8 129 7

f 22 29 30

f 19 26 27

f 29 36 37

f 31 22 30

f 89 16 24

f 26 33 34

f 24 31 32

f 28 35 36

f 122 25 17

f 27 34 35

f 37 44 45

f 38 29 37

f 89 24 32

f 42 33 41

f 32 39 40

f 36 43 44

f 31 38 39

f 122 33 25

f 43 34 42

f 45 52 53

f 46 37 45

f 89 32 40

f 43 50 51

f 48 39 47

f 52 43 51

f 39 46 47

f 50 41 49

f 122 41 33

f 53 60 61

f 47 54 55

f 46 53 54

f 48 55 56

f 60 51 59

f 58 49 57

f 122 49 41

f 89 40 48

f 61 68 69

f 55 62 63

f 54 61 62

f 51 58 59

f 58 65 66

f 68 59 67

f 122 57 49

f 56 63 64

f 89 48 56

f 63 70 71

f 62 69 70

f 59 66 67

f 69 76 77

f 66 73 74

f 122 65 57

f 64 71 72

f 76 67 75

f 89 56 64

f 79 70 78

f 70 77 78

f 67 74 75

f 77 84 85

f 72 79 80

f 122 73 65

f 76 83 84

f 89 64 72

f 74 81 82

f 87 78 86

f 86 77 85

f 75 82 83

f 85 93 94

f 80 87 88

f 84 92 93

f 122 81 73

f 89 72 80

f 91 81 90

f 87 95 96

f 86 94 95

f 83 91 92

f 94 101 102

f 93 100 101

f 89 80 88

f 122 90 81

f 91 98 99

f 88 96 97

f 95 102 103

f 92 99 100

f 102 109 110

f 96 103 104

f 122 98 90

f 89 88 97

f 99 106 107

f 105 96 104

f 109 100 108

f 108 99 107

f 110 117 118

f 104 111 112

f 122 106 98

f 89 97 105

f 107 114 115

f 103 110 111

f 117 108 116

f 113 104 112

f 108 115 116

f 118 126 127

f 120 111 119

f 122 114 106

f 115 123 124

f 111 118 119

f 89 105 113

f 113 120 121

f 126 116 125

f 119 127 128

f 116 124 125

f 120 128 129

f 89 113 121

f 121 129 130

f 122 123 114

f 89 121 130

f 122 1 123

f 89 130 8

f 3 126 125

f 5 126 4

f 15 22 23

f 10 17 18

f 24 15 23

f 13 20 21

f 18 25 26

f 14 21 22

f 21 28 29

f 12 19 20

f 11 18 19

f 1 124 123

f 122 9 1

f 6 127 5

f 7 6 14

f 3 2 10

f 12 4 3

f 8 7 15

f 5 4 12

f 2 3 125

f 2 1 9

f 6 5 13

f 7 129 128

f 20 19 27

f 8 130 129

f 22 21 29

f 19 18 26

f 29 28 36

f 31 23 22

f 26 25 33

f 24 23 31

f 28 27 35

f 27 26 34

f 37 36 44

f 38 30 29

f 42 34 33

f 32 31 39

f 36 35 43

f 31 30 38

f 43 35 34

f 45 44 52

f 46 38 37

f 43 42 50

f 48 40 39

f 52 44 43

f 39 38 46

f 50 42 41

f 53 52 60

f 47 46 54

f 46 45 53

f 48 47 55

f 60 52 51

f 58 50 49

f 61 60 68

f 55 54 62

f 54 53 61

f 51 50 58

f 58 57 65

f 68 60 59

f 56 55 63

f 63 62 70

f 62 61 69

f 59 58 66

f 69 68 76

f 66 65 73

f 64 63 71

f 76 68 67

f 79 71 70

f 70 69 77

f 67 66 74

f 77 76 84

f 72 71 79

f 76 75 83

f 74 73 81

f 87 79 78

f 86 78 77

f 75 74 82

f 85 84 93

f 80 79 87

f 84 83 92

f 91 82 81

f 87 86 95

f 86 85 94

f 83 82 91

f 94 93 101

f 93 92 100

f 91 90 98

f 88 87 96

f 95 94 102

f 92 91 99

f 102 101 109

f 96 95 103

f 99 98 106

f 105 97 96

f 109 101 100

f 108 100 99

f 110 109 117

f 104 103 111

f 107 106 114

f 103 102 110

f 117 109 108

f 113 105 104

f 108 107 115

f 118 117 126

f 120 112 111

f 115 114 123

f 111 110 118

f 113 112 120

f 126 117 116

f 119 118 127

f 116 115 124

f 120 119 128

f 121 120 129

f 3 4 126

f 5 127 126

f 15 14 22

f 10 9 17

f 24 16 15

f 13 12 20

f 18 17 25

f 14 13 21

f 21 20 28

f 12 11 19

f 11 10 18

f 1 2 124

f 6 128 127

)";

LoadObj();

}

Sphere::~Sphere() {

}

Arrow::Arrow() {

rawData = R"(

o Cone

v 0.000000 0.800000 -0.100000

v 0.070711 0.800000 -0.070711

v 0.100000 0.800000 -0.000000

v 0.000000 1.000000 0.000000

v 0.070711 0.800000 0.070711

v -0.000000 0.800000 0.100000

v -0.070711 0.800000 0.070711

v -0.100000 0.800000 -0.000000

v -0.070711 0.800000 -0.070711

s off

f 4 7 6

f 5 7 2

f 4 8 7

f 3 4 5

f 5 4 6

f 4 9 8

f 4 1 9

f 2 1 4

f 2 4 3

f 9 1 2

f 2 3 5

f 5 6 7

f 7 8 9

f 9 2 7

o Cylinder

v 0.000000 0.000000 -0.050000

v 0.009755 0.900000 -0.049039

v 0.019134 0.000000 -0.046194

v 0.027779 0.900000 -0.041573

v 0.035355 0.000000 -0.035355

v 0.041573 0.900000 -0.027779

v 0.046194 0.000000 -0.019134

v 0.049039 0.900000 -0.009755

v 0.050000 0.000000 -0.000000

v 0.049039 0.900000 0.009755

v 0.046194 0.000000 0.019134

v 0.041573 0.900000 0.027779

v 0.035355 0.000000 0.035355

v 0.027779 0.900000 0.041573

v 0.019134 0.000000 0.046194

v 0.009755 0.900000 0.049039

v -0.000000 0.000000 0.050000

v -0.009755 0.900000 0.049039

v -0.019134 0.000000 0.046194

v -0.027779 0.900000 0.041573

v -0.035355 0.000000 0.035355

v -0.041574 0.900000 0.027778

v -0.046194 0.000000 0.019134

v -0.049039 0.900000 0.009754

v -0.050000 0.000000 -0.000000

v -0.049039 0.900000 -0.009755

v -0.046194 0.000000 -0.019134

v -0.041573 0.900000 -0.027779

v -0.035355 0.000000 -0.035355

v -0.027778 0.900000 -0.041574

v -0.019134 0.000000 -0.046194

v -0.009754 0.900000 -0.049039

s off

f 13 15 14

f 16 14 15

f 17 19 18

f 18 16 17

f 19 21 20

f 20 18 19

f 21 23 22

f 22 20 21

f 23 25 24

f 24 22 23

f 25 27 26

f 26 24 25

f 27 29 28

f 28 26 27

f 29 31 30

f 30 28 29

f 31 33 32

f 32 30 31

f 33 35 34

f 34 32 33

f 35 37 36

f 36 34 35

f 37 39 38

f 38 36 37

f 41 40 39

f 40 38 39

f 41 10 40

f 29 21 37

f 11 12 10

f 24 32 16

f 15 17 16

f 11 13 12

f 14 12 13

f 10 41 11

f 13 11 41

f 41 39 37

f 37 35 33

f 33 31 29

f 29 27 25

f 25 23 29

f 21 19 17

f 17 15 13

f 13 41 37

f 37 33 29

f 29 23 21

f 21 17 13

f 13 37 21

f 40 10 12

f 12 14 16

f 16 18 20

f 20 22 24

f 24 26 28

f 28 30 32

f 32 34 36

f 36 38 40

f 40 12 16

f 16 20 24

f 24 28 32

f 32 36 40

f 40 16 32

)";

LoadObj();

}

Arrow::~Arrow() {

}

**A\* Search**

**Source.cpp**

#include "Node.h"

#include <vector>

#include <random>

#include <iostream>

#include <ctime>

#include <list>

using namespace std;

const int h = 20;

const int w = 20;

string map[w][h];

vector<Node> closed = {};

vector<Node> open = {};

int xStart, yStart, xEnd, yEnd;

int dir = 8;

enum directions{N, NE, E, SE, S, SW, W, NW};

void InitMap(int numImpassable);

void PrintMap();

void AStarSearch(int xStart, int yStart, int xEnd, int yEnd);

void addToOpen(int x, int y, int g, int px, int py);

int diff(int a, int b);

int main() {

InitMap(80);

PrintMap();

cout << "Enter the X coord of the start position:" << endl;

cin >> xStart;

cout << "Enter the Y coord of the start position:" << endl;

cin >> yStart;

cout << "Enter the X coord of the end position:" << endl;

cin >> xEnd;

cout << "Enter the Y coord of the end position:" << endl;

cin >> yEnd;

AStarSearch(xStart, yStart, xEnd, yEnd);

PrintMap();

cout << "\nPress any key to continue...\n";

cin.ignore(); cin.get();

return 0;

}

void InitMap(int numImpassable) {

int impassable = numImpassable;

srand(time(NULL));

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

int temp = (rand() % (impassable+1)) % (w/2);

if (h-i == 1 && temp < impassable) {

temp = impassable;

}

impassable -= temp;

for (int j = 0; j < w; j++) {

if (h - j != temp) {

if (temp != 0) {

int r = rand() % 5;

switch (r) {

case 1:

map[i][j] = "[X]";

temp--;

break;

default:

map[i][j] = "[ ]";

break;

}

}

else {

map[i][j] = "[ ]";

}

}

else

{

map[i][j] = "[X]";

temp--;

}

}

}

}

void PrintMap() {

cout << " ";

for (int j = 0; j < w; j++) {

if(j<10)

cout << " "<< j << " ";

else

cout << j << " ";

}

cout << endl;

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

if(i<10)

cout << i << " ";

else

cout << i << " ";

for (int j = 0; j < w; j++) {

string tmp = map[i][j];

cout << tmp.c\_str();

}

cout << "\n";

}

}

void AStarSearch(int xStart, int yStart, int xEnd, int yEnd){

if (open.size() == 0) {

int h = diff(xStart, xEnd) + diff(yStart, yEnd);

Node n = Node(xStart, yStart, 0, h, -1, -1);

open.push\_back(n);

}

bool finished = false;

while (!finished) {

int f = NULL;

Node chosen = Node();

int xtmp = 0;

int ytmp = 0;

for each (Node n in open) {

if (n.xCoord == xEnd && n.yCoord == yEnd) {

chosen = n;

finished = true;

break;

}else if (n.f < f || f == NULL) {

f = n.f;

chosen = n;

}

}

if (!finished) {

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

switch (i) {

case N:

addToOpen(chosen.xCoord, chosen.yCoord + 1, chosen.g + 10, chosen.xCoord, chosen.yCoord);

break;

case NE:

addToOpen(chosen.xCoord + 1, chosen.yCoord + 1, chosen.g + 14, chosen.xCoord, chosen.yCoord);

break;

case E:

addToOpen(chosen.xCoord + 1, chosen.yCoord, chosen.g + 10, chosen.xCoord, chosen.yCoord);

break;

case SE:

addToOpen(chosen.xCoord + 1, chosen.yCoord - 1, chosen.g + 14, chosen.xCoord, chosen.yCoord);

break;

case S:

addToOpen(chosen.xCoord, chosen.yCoord - 1, chosen.g + 10, chosen.xCoord, chosen.yCoord);

break;

case SW:

addToOpen(chosen.xCoord - 1, chosen.yCoord - 1, chosen.g + 14, chosen.xCoord, chosen.yCoord);

break;

case W:

addToOpen(chosen.xCoord - 1, chosen.yCoord, chosen.g + 10, chosen.xCoord, chosen.yCoord);

break;

case NW:

addToOpen(chosen.xCoord - 1, chosen.yCoord + 1, chosen.g + 14, chosen.xCoord, chosen.yCoord);

break;

}

}

cout << "Explored: " << chosen.xCoord << ", " << chosen.yCoord << endl;

closed.push\_back(chosen);

for (unsigned int i = 0; i < open.size(); i++) {

if (open[i] == chosen) {

open.erase(open.begin()+i);

break;

}

}

}

else

{

while (chosen.xParent != -1 && chosen.yParent != -1) {

if(chosen.xCoord == xEnd && chosen.yCoord == yEnd)

map[chosen.yCoord][chosen.xCoord] = "[F]";

else

map[chosen.yCoord][chosen.xCoord] = "[O]";

for each(Node n in closed) {

if (chosen.xParent == n.xCoord && chosen.yParent == n.yCoord)

chosen = n;

}

}

cout << chosen.xParent << " " << chosen.yParent << endl;

map[chosen.yCoord][chosen.xCoord] = "[S]";

}

}

}

void addToOpen(int x, int y, int g, int px, int py) {

if (!(x<0 || x>=w || y < 0 || y>=h)) {

if (!(map[y][x] == "[X]")) {

int h = diff(x, xEnd) + diff(y, yEnd);

Node tmp = Node(x, y, g, h, px, py);

bool inClosed = false;

for (unsigned int i = 0; i < closed.size(); i++) {

if (closed[i].xCoord == tmp.xCoord && closed[i].yCoord == tmp.yCoord) {

inClosed = true;

break;

}

}

if (!inClosed) {

vector<Node> tmpOpen;

bool added = false;

if (open.empty()) {

open.push\_back(tmp);

added = true;

}

else {

for (unsigned int i = 0; i < open.size(); i++) {

if (open[i].xCoord == tmp.xCoord && open[i].yCoord == tmp.yCoord) {

if (tmp.g < open[i].g) {

tmpOpen.push\_back(tmp);

added = true;

}

else {

tmpOpen.push\_back(open[i]);

added = true;

}

}

else {

tmpOpen.push\_back(open[i]);

}

}

if (!added)

tmpOpen.push\_back(tmp);

open = tmpOpen;

}

}

}

}

}

int diff(int a, int b) {

if (a > b)

return a - b;

else

return b - a;

}

**Node.h**

#pragma once

class Node {

public:

Node();

Node(int xCoord, int yCoord, int g, int h, int xParent, int yParent);

bool operator==(const Node& other) const;

int xCoord, yCoord, g, h, f, xParent, yParent;

};

**Node.cpp**

#include "Node.h"

Node::Node() {

}

Node::Node(int xCoord, int yCoord, int g, int h, int xParent, int yParent) {

this->xCoord = xCoord;

this->yCoord = yCoord;

this->g = g;

this->h = h;

f = g + h;

this->xParent = xParent;

this->yParent = yParent;

}

bool Node::operator==(const Node & other) const

{

if (this->xCoord == other.xCoord &&

this->yCoord == other.yCoord &&

this->xParent == other.xParent &&

this->yParent == other.yParent &&

this->f == other.f &&

this->g == other.g &&

this->h == other.h)

return true;

else

return false;

}

**Separation of behaviour in Bouncing Ball**

To separate the physical behaviour from visualisation in the bouncing ball lab I chose to use Stefanos “Shapes” class in order to create a sphere to represent the ball graphically, and created a “PhysicsSphere” class to handle the physics calculations.

During each update call of the main program the update method of the “PhysicsSphere” class would also be called which would then run all of the necessary physics calculations in order to calculate the new position of the ball as well as any other variables which would be used during the next loop.

The position calculated in the “PhysicsSphere” class would be used when creating the model view matrix for the graphical ball in the main program, which would allow for the ball to be rendered with the correct positioning.

The physics sphere class could be improved in the way of an additional method to allow for manual user input of forces to apply to the “PhysicsSphere” during runtime, as opposed to having everything hard coded as it is now.

**A\* Search Description**

The most notable difference between A\* and Dijkstra is that A\* is a more “informed” version of Dijkstra, as Dijkstra only uses the distance travelled so far, g(x), as opposed to A\* which combines g(x) with the estimated distance left to go, h(x), into f(x). This results in A\* greedily choosing which node to expand upon next. Removing h(x) from an A\* algorithm would cause it to default to a Dijkstra algorithm.

In an A\* search the first thing that will happen is that the starting node will be added to the **open** list, followed by all adjacent nodes being added to the **open** list (ignoring obstacle nodes), along with their respective f(x) values and a reference to their parent node (adjacent nodes could vary depending on implementation, e.g. if the algorithm can move diagonally or not). After this the starting node will be removed from the **open** list and added to the **closed** list, preventing it from being re-assessed.

Next, the algorithm will choose whichever node in the **open** list has the lowest f(x) value to assess next. It will then add all nodes adjacent to the one being assessed to the **open** list (ignoring obstacle nodes). If any adjacent nodes are already present in the **open** list the algorithm will compare the nodes previous f(x) value to the one it would receive if the node currently being assessed was its parent. If the new f(x) is smaller the current node will be set as its parent, otherwise the nodes parent will remain unchanged. After all adjacent nodes have been added the current node will be removed from the **open** list and added to the **closed** list.

This process will repeat until the goal node appears in the **open** list. At this point a complete path will have been formed from start to goal.