

Computer Games Programming

Programming Assessment

Issa Haddad

H00278537

Feb 13, 2018

Table of Contents

[A particle Explosion 3](#_Toc506300650)

[A physically driven bouncing ball 4](#_Toc506300651)

[Flocking Boids in 3D 5](#_Toc506300652)

[A\* path planning search 6](#_Toc506300653)

[Screenshots 7](#_Toc506300654)

[Explosion 7](#_Toc506300655)

[Physically driven bouncing ball 8](#_Toc506300656)

[Flocking Boids in 3D 9](#_Toc506300657)

[A\* Path planning search 9](#_Toc506300658)

[Code listing 10](#_Toc506300659)

[Vector class 10](#_Toc506300660)

[Explosion, physically driven bouncing ball, and Flocking Boids in 3D 12](#_Toc506300661)

[A\* Path planning search 61](#_Toc506300662)

## A particle Explosion

In the main “source.cpp” file

* I initialized a vector of Spheres containing 100 instances of single sphere object
* I also initialize an instance of a functionality class called “Explode.cpp”
* I drew each instance of the sphere on the window
* For each sphere instance, I grab its movement from a class called “Explode.cpp”.
* At each iteration of the update function 🡪 I update movement physics for each particle undergoing the explosion

In the “Explode.cpp” file

* When this is initialized, it creates a 100 instances of a particle object
* All of those particles get the same initial position in the 3D space
* Each particle gets a random velocity in the 3D space
* At every iteration of the update function in “source.cpp”
  + I update the velocity and position of each particle with gravity taken into account
  + I decrease the alpha value of the fill color of each particle by 0.02
  + Check when alpha value < 0, and reset position of each particle to its initial position.

## A physically driven bouncing ball

In the main “source.cpp” file

* I initialized 1 cube object
* I also initialize an instance of a particle class called “Physics.cpp”
* I draw the cube on the window
* For the cube object, I grab its movement from the same physics class called “Physics.cpp”.
* At each iteration of the update function 🡪 I update movement physics of the cube undergoing the bouncing functionality

In the “Physics.cpp” file

* Ground is specified at y-axis position = -2
* When this is initialized, a specific position is attached to where the cube will be dropped
* A velocity of zero is also given to resemble free fall of an object
* At every iteration of the update function in “source.cpp”
  + When position of the cube reaches below ground level
    - Deflect velocity direction and multiply by “0.9” for object to reach lower level from where it was dropped and so on
    - Add velocity noise on the x-axis or friction
  + Update velocity and position of each particle with gravity taken into account
  + Reset position of cube to floor level if it went below

## Flocking Boids in 3D

In the main “source.cpp” file

* Clicking a button would draw a new Boid into the screen.
* Clicking the button would invoke a function in “flock.cpp” called “addBoid” that add a new particle instance associated with the Boid
* For each Boid instance, I grab its movement from a class called “Flock.cpp”.
* At each iteration of the update function 🡪 I update movement physics for each particle undergoing the flocking mechanism

In the “Flock.cpp” file

* When “addBoid” function is invoked, it creates and adds a new particle instance
  + It sets a specific position for each particle to where they would be drawn initially
  + Its sets a random velocity in the 3D vector of the space
  + Acceleration is initially set to zero
  + A maximum speed and maximum force is set to the particle that it shouldn’t exceed
* At every iteration of the update function in “source.cpp”
  + Separation forces are calculated and added to the particles acceleration
    - Desired separation distance is specified here
  + alignment forces are calculated and added to the particles acceleration
    - Desired neighborhood distance is specified here
  + cohesion forces are calculated and added to the particles acceleration
    - Desired neighborhood distance is specified here
  + velocity is updated for each particle and the total acceleration is added to it
  + velocity cannot exceed a specified limit
  + position is updated according to velocity only
  + acceleration is set to zero
  + wrap around function is applied for Boids to not escape the screen

## A\* path planning search

**Dijkstra algorithm vs A\* search**

In Dijkstra search algorithm when the node expands it works more like a brute force search in path finding, while A\* could be considered as a more improved version of Dijkstra search algorithm.

**Node Expand**

**A\* search**

n is the last node expand

g(n) is the cost so far

h(n) is the heuristic value

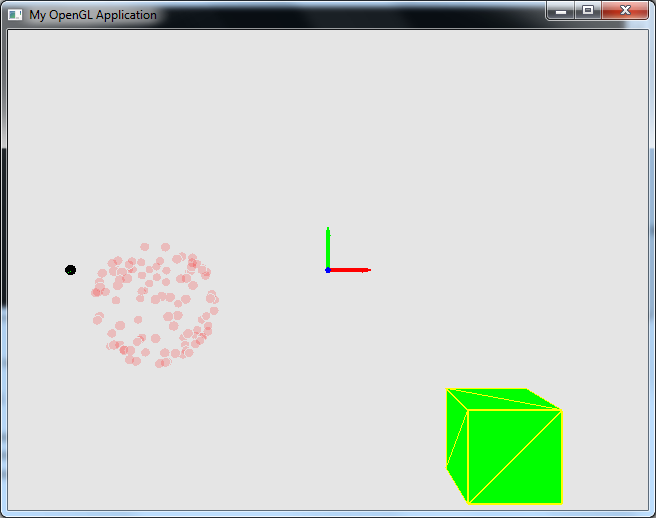
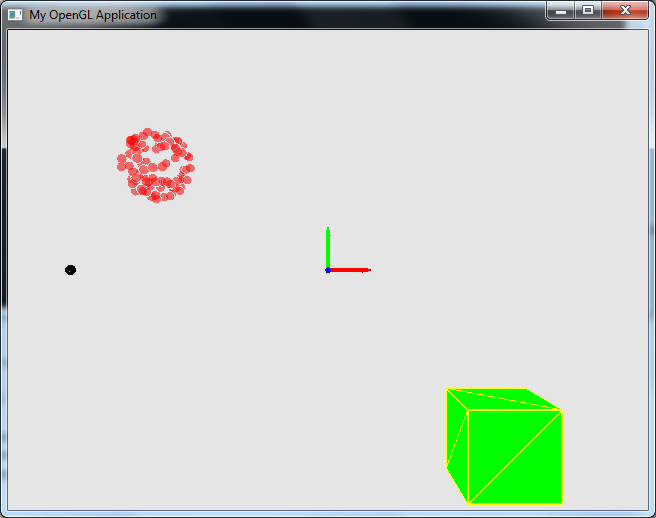
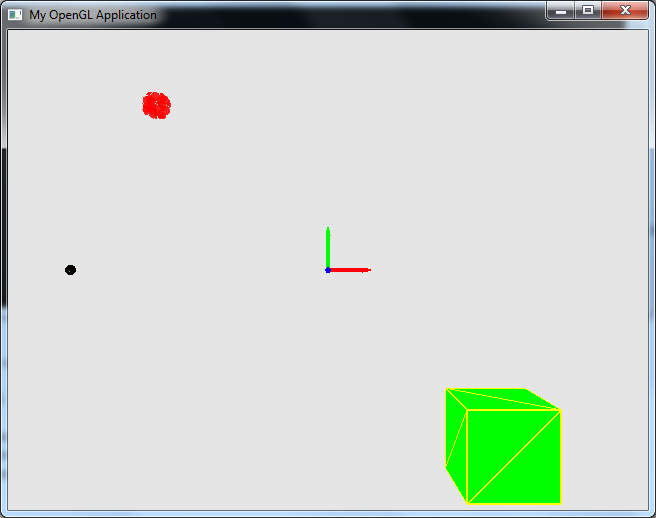
A\* will use the f(n) as a function to select the node the expand where it minimizes the cost in term of navigate to the goal. A\* search performance will hugely depend on the choice of heuristic whether it is admissible (optimizes) or not. So when the node expand it will choose more optimize node to expand (according to the heuristic) so it won’t check through every node until the current node have higher cost.

**Dijkstra search**

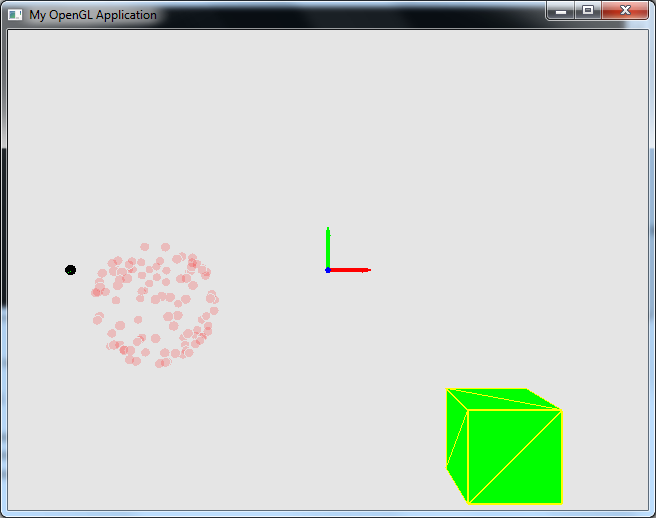
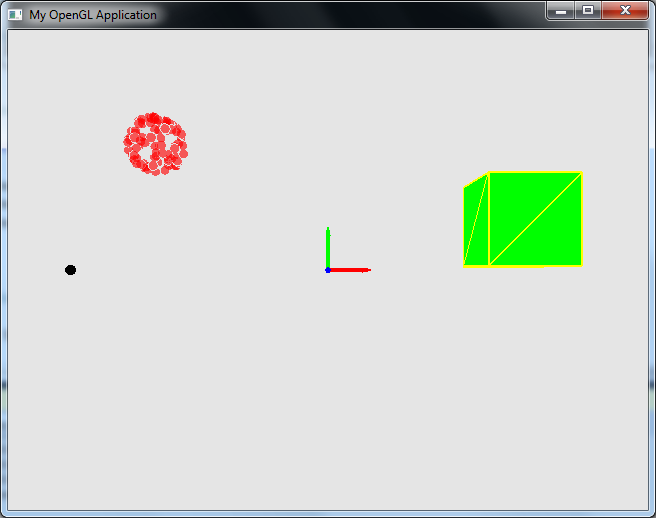
In term of node expand for Dijkstra algorithm, all node are considered as unvisited neighbors the algorithm then calculate the distance (cost) from the current node and select the one with shortest path (lowest cost) when compare to A\* Dijkstra will expand into unnecessary node before converge on the shortest path this will lead to longer computational time in general case.

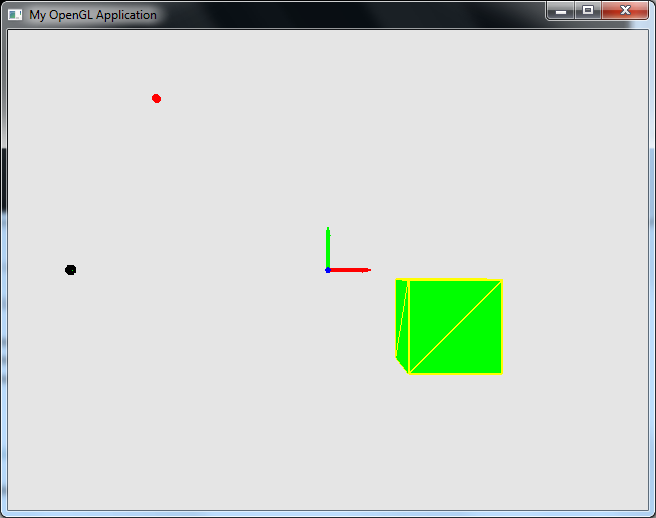
## Screenshots

### Explosion

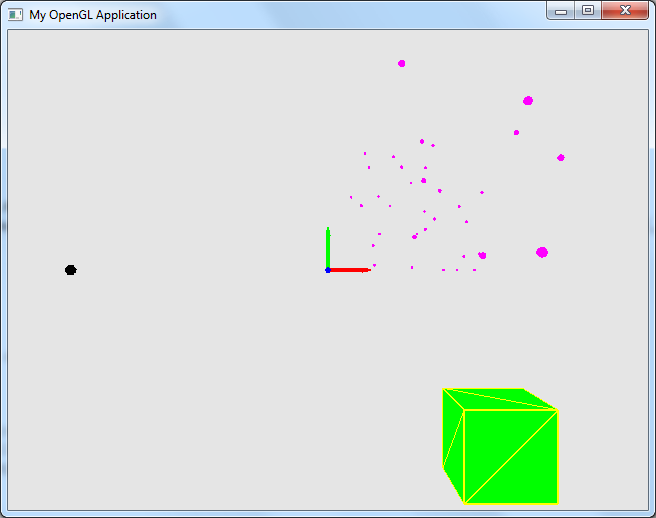
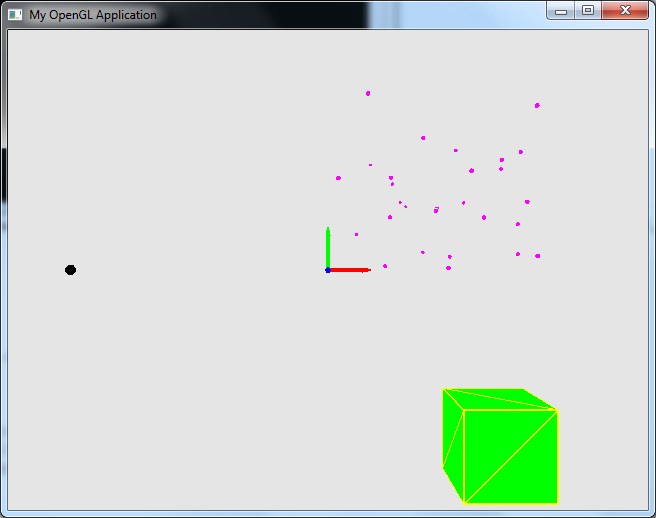


### Physically driven bouncing ball

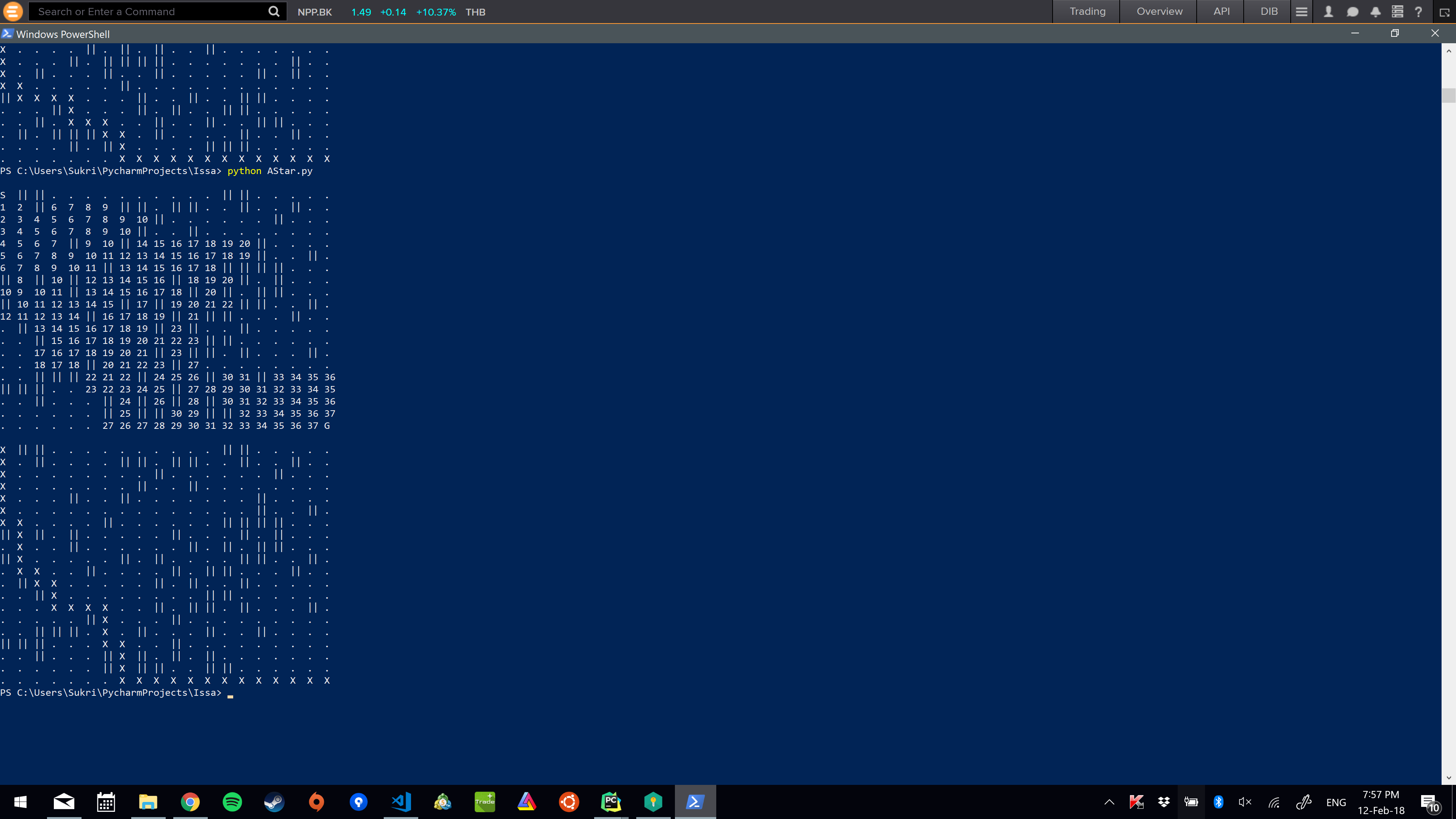




### Flocking Boids in 3D



### A\* Path planning search



## Code listing

### Vector class

class vec3 {

public:

vec3();

vec3(float, float, float);

int 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;

};

#include"vec3.h"

#include <math.h>

vec3::vec3()

{

}

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

{

this->x = x;

this->y = y;

this->z = z;

}

int 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);

}

float vec3::dot(vec3 rhs) {

return (x \* rhs.x +

y \* rhs.y +

z \* rhs.z);

}

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::length() {

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

}

### Explosion, physically driven bouncing ball, and Flocking Boids in 3D

#pragma once

#include <iostream>

#include <vector>

using namespace std;

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm/glm.hpp>

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();

};

#include <iostream>

#include <vector>

using namespace std;

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm/glm.hpp>

#define GLM\_ENABLE\_EXPERIMENTAL

#include <glm/gtx/transform.hpp>

#include "graphics.h"

#include "shapes.h"

class Physics {

public:

Physics(int t);

~Physics();

Physics(glm::vec3 p,glm::vec3 v, int l);

void Physics::update(double delta, float &f);

void Physics::updateEx(double delta, float &f);

void Physics::updateBoids(double delta, std::vector<Physics> &Boids);

void Physics::LessColor(float &f);

void Physics::Regen(float &f, int d);

void Physics::flocking(std::vector<Physics> Boids);

void Physics::updateflock(double delta);

void Physics::borders();

void Physics::render();

glm::vec3 Physics::GetPos();

glm::vec3 Physics::GetPos1();

glm::vec3 Physics::separate(std::vector<Physics> Boids);

glm::vec3 Physics::align(std::vector<Physics> Boids);

glm::vec3 Physics::cohesion(std::vector<Physics> Boids);

glm::vec3 Physics::seek(glm::vec3 t);

void Physics::applyForce(glm::vec3 a);

glm::vec3 position;

glm::vec3 velocity;

glm::vec3 acceleration;

float r;

float maxforce;

float maxspeed;

const glm::vec3 gravity = glm::vec3(0.0f, -9.8f, 0.0f);

glm::vec3 friction = glm::vec3(0.9f, 1.0f, 0.9f);

int Lifespan;

};

#pragma once

#include <iostream>

#include <vector>

using namespace std;

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm/glm.hpp>

void ErrorCallbackGLFW(int error, const char\* description);

class Graphics {

public:

Graphics();

~Graphics();

int Init();

void hintsGLFW();

void SetupRender();

void endProgram();

void SetOptimisations();

void ClearViewport();

GLFWwindow\* window;

int windowWidth = 640;

int windowHeight = 480;

float aspect;

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

};

#include <iostream>

#include <vector>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm/glm.hpp>

#define GLM\_ENABLE\_EXPERIMENTAL

#include <glm/gtx/transform.hpp>

class Physics;

class Flock {

public:

Flock();

void run(double delta);

void addBoid();

glm::vec3 Flock::GetPos(int index);

std::vector<Physics> Boids;

float test = 0.0f;

int count = 0;

};

#include <iostream>

#include <vector>

using namespace std;

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm/glm.hpp>

#define GLM\_ENABLE\_EXPERIMENTAL

#include <glm/gtx/transform.hpp>

#include "graphics.h"

#include "shapes.h"

class Explode{

public:

Explode(int num);

void update\_balls(double delta, int ballsN, std::vector <Sphere> & all);

glm::vec3 Explode::GetPos(int index);

std::vector<Physics> Spheres;

};

// 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.

//

// Update 2018/01 updated libraries and created project for VS2015.

// 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>

#define GLM\_ENABLE\_EXPERIMENTAL

#include <glm/gtx/transform.hpp>

#include <ctime>

#include <cstdlib>

#include "graphics.h"

#include "shapes.h"

#include "Physics.h"

#include "Explode.h"

#include "Flock.h"

// FUNCTIONS

void render(double currentTime);

void update(double currentTime);

void startup();

void CreateBoid();

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

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

// VARIABLES

bool running = true;

int draw = 0;

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

Cube myCube;

std::vector<Sphere> mySpheres;

std::vector<Sphere> myBoids;

Sphere mySphere;

Arrow arrowX;

Arrow arrowY;

Arrow arrowZ;

Physics sp1(2);

Physics cube(1);

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

float angleY = 0.000f;

double deltatime;

int ballsN = 100;

Explode ex(ballsN);

Flock flock;

int main()

{

srand(static\_cast <unsigned> (time(0)));

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

double starttime = glfwGetTime(); // MAIN LOOP run until the window is closed

do {

double currentTime = glfwGetTime(); // retrieve timelapse

deltatime = currentTime - starttime;

starttime = currentTime;

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 CreateBoid()

{

flock.addBoid();

draw++;

Sphere b;

b.Load();

b.fillColor = glm::vec4(1.0f, 0.0f, 1.0f, 1.0f);

b.lineColor = glm::vec4(1.0f, 1.0f, 0.0f, 0.0f);

myBoids.push\_back(b);

}

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

myCube.Load();

myCube.fillColor = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f);

myCube.lineColor = glm::vec4(1.0f, 1.0f, 0.0f, 1.0f);

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

{

Sphere s;

s.Load();

s.fillColor = glm::vec4(1.0f, 0.0f, 0.0f, 1.0f);

s.lineColor = glm::vec4(1.0f, 1.0f, 0.0f, 0.0f);

mySpheres.push\_back(s);

}

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

arrowX.Load(); arrowY.Load(); arrowZ.Load();

arrowX.fillColor = glm::vec4(1.0f, 0.0f, 0.0f, 1.0f); arrowX.lineColor = glm::vec4(1.0f, 0.0f, 0.0f, 1.0f);

arrowY.fillColor = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f); arrowY.lineColor = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f);

arrowZ.fillColor = glm::vec4(0.0f, 0.0f, 1.0f, 1.0f); arrowZ.lineColor = glm::vec4(0.0f, 0.0f, 1.0f, 1.0f);

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

}

void update(double currentTime) {

//cube.LessColor(myCube.fillColor.a);

if (draw > 0)

{

flock.run(deltatime);

//cout << "draw=" << draw;

}

cube.update(deltatime, myCube.fillColor.a);

ex.update\_balls(deltatime, ballsN, mySpheres);

//sp1.updateEx(deltatime, mySpheres.at(0).fillColor.a);

//sp1.LessColor(mySpheres.at(0).fillColor.a);

//sp1.Regen(mySpheres.at(0).fillColor.a);

// Calculate Cube movement ( T \* R \* S ) http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

glm::mat4 mv\_matrix\_cube =

glm::translate(cube.GetPos()) \*

glm::rotate(angleY, glm::vec3(0.0f, 1.0f, 0.0f)) \*

glm::rotate(angleY, glm::vec3(1.0f, 0.0f, 0.0f)) \*

glm::mat4(1.0f);

myCube.mv\_matrix = mv\_matrix\_cube;

myCube.proj\_matrix = myGraphics.proj\_matrix;

if (draw > 0)

{

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

{

//cout << "in:"<<i<<" "<<flock.GetPos(i).x<<" "<< flock.GetPos(i).y << " " << flock.GetPos(i).z<<endl;

glm::mat4 mv\_matrix\_sphere1 =

glm::translate(flock.GetPos(i)) \*

glm::rotate(-t, glm::vec3(0.0f, 1.0f, 0.0f)) \*

glm::rotate(-t, glm::vec3(1.0f, 0.0f, 0.0f)) \*

glm::scale(glm::vec3(0.1f, 0.1f, 0.1f)) \*

glm::mat4(1.0f);

myBoids.at(i).mv\_matrix = mv\_matrix\_sphere1;

myBoids.at(i).proj\_matrix = myGraphics.proj\_matrix;

}

}

// calculate Spheres movement

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

{

glm::mat4 mv\_matrix\_sphere1 =

glm::translate(ex.GetPos(i)) \*

glm::rotate(-t, glm::vec3(0.0f, 1.0f, 0.0f)) \*

glm::rotate(-t, glm::vec3(1.0f, 0.0f, 0.0f)) \*

glm::scale(glm::vec3(0.1f, 0.1f, 0.1f)) \*

glm::mat4(1.0f);

mySpheres.at(i).mv\_matrix = mv\_matrix\_sphere1;

mySpheres.at(i).proj\_matrix = myGraphics.proj\_matrix;

}

// calculate Sphere movement

glm::mat4 mv\_matrix\_sphere =

glm::translate(glm::vec3(-3.0f, 0.0f, -6.0f)) \*

glm::rotate(-t, glm::vec3(0.0f, 1.0f, 0.0f)) \*

glm::rotate(-t, glm::vec3(1.0f, 0.0f, 0.0f)) \*

glm::scale(glm::vec3(0.1f, 0.1f, 0.1f)) \*

glm::mat4(1.0f);

mySphere.mv\_matrix = mv\_matrix\_sphere;

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

//Calculate Arrows translations (note: arrow model points up)

glm::mat4 mv\_matrix\_x =

glm::translate(glm::vec3(0.0f, 0.0f, -6.0f)) \*

glm::rotate(glm::radians(-90.0f), glm::vec3(0.0f, 0.0f, 1.0f)) \*

glm::scale(glm::vec3(0.2f, 0.5f, 0.2f)) \*

glm::mat4(1.0f);

arrowX.mv\_matrix = mv\_matrix\_x;

arrowX.proj\_matrix = myGraphics.proj\_matrix;

glm::mat4 mv\_matrix\_y =

glm::translate(glm::vec3(0.0f, 0.0f, -6.0f)) \*

//glm::rotate(glm::radians(-90.0f), glm::vec3(0.0f, 0.0f, 1.0f)) \* // already model pointing up

glm::scale(glm::vec3(0.2f, 0.5f, 0.2f)) \*

glm::mat4(1.0f);

arrowY.mv\_matrix = mv\_matrix\_y;

arrowY.proj\_matrix = myGraphics.proj\_matrix;

glm::mat4 mv\_matrix\_z =

glm::translate(glm::vec3(0.0f, 0.0f, -6.0f)) \*

glm::rotate(glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f)) \*

glm::scale(glm::vec3(0.2f, 0.5f, 0.2f)) \*

glm::mat4(1.0f);

arrowZ.mv\_matrix = mv\_matrix\_z;

arrowZ.proj\_matrix = myGraphics.proj\_matrix;

t += 0.01f; // increment mzvement variable

}

void render(double currentTime) {

// Clear viewport - start a new frame.

myGraphics.ClearViewport();

if (draw > 0)

{

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

{

myBoids.at(i).Draw();

}

}

// Draw

myCube.Draw();

mySphere.Draw();

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

{

mySpheres.at(i).Draw();

}

arrowX.Draw();

arrowY.Draw();

arrowZ.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\_A && action == GLFW\_PRESS)

CreateBoid();

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

}

#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);

glEnable(GL\_BLEND);

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

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() {

}

#include "Physics.h"

#include "Shapes.h"

#include <ctime>

#include <cstdlib>

#include <iostream>

Physics::Physics(int t)

{

if (t == 1)

{

position = glm::vec3(2.0f, 0.0f, -6.0f);

velocity = glm::vec3(0.0f, 0.0f, 0.0f);

Lifespan = 5;

}

else if(t==2)

{

position = glm::vec3(-2.0f, 2.0f, -6.0f);

//srand(static\_cast <unsigned> (time(0)));

glm::vec3 speed1(glm::normalize(glm::vec3((rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f)));

velocity = speed1;

Lifespan = 5;

}

else

{

position = glm::vec3(2.5f, 2.0f, -6.0f);

glm::vec3 speed2 = glm::vec3(0.0f, 0.0f, 0.0f);

//glm::vec3 speed1(glm::normalize(glm::vec3((rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f)));;

velocity = speed2;

}

}

Physics::~Physics() {};

Physics::Physics(glm::vec3 p, glm::vec3 v, int l)

{

position = p;

velocity = v;

Lifespan = l;

}

void Physics::update(double delta, float &f)

{

if (position.y <= -2)

{

velocity = -velocity\* (float)0.9;

//Add noise

if (velocity.y > 0.3)

{

if ((rand() % 2) == 1)

{

velocity.x += 0.2;

}

else

{

velocity.x -= 0.2;

}

}

else {

velocity.x \*= friction.x;

velocity.y \*= friction.y;

velocity.z \*= friction.z;

}

}

Lifespan--;

float deltafloat = (float)delta;

velocity.x = velocity.x + gravity.x\*(deltafloat);

velocity.y = velocity.y + gravity.y\*(deltafloat);

velocity.z = velocity.z + gravity.z\*(deltafloat);

position = position + (velocity\*(deltafloat)+(float)0.5\*gravity\*deltafloat\*deltafloat);

if (position.y < -2)

{

position.y = -2;

}

//f = f - 0.01;

//cout << f.r << f.g << f.b << f.a << "\n";

}

void Physics::updateEx(double delta, float &f)

{

Lifespan--;

float deltafloat = (float)delta;

velocity.x = velocity.x + gravity.x\*(deltafloat);

//velocity.x = velocity.x + (deltafloat);

velocity.y = velocity.y + gravity.y\*(deltafloat);

//velocity.y = velocity.y + (deltafloat);

velocity.z = velocity.z + gravity.z\*(deltafloat);

//velocity.z = velocity.z + (deltafloat);

position = position + (velocity\*(deltafloat)+(float)0.5\*gravity\*deltafloat\*deltafloat);

//position = position + (velocity\*(deltafloat)+(float)deltafloat\*deltafloat);

}

void Physics::updateBoids(double delta, std::vector<Physics> &Boids)

{

//cout << "in boids: "<<position.x<<" "<< position.y << " " << position.z << " " << "\n";

flocking(Boids);

updateflock(delta);

borders();

render();

}

void Physics::updateflock(double delta)

{

//cout << "in boids position: " << position.x << " " << position.y << " " << position.z << " " << "\n";

//cout << "in boids velocity: " << velocity.x << " " << velocity.y << " " << velocity.z << " " << "\n";

cout << "add: " << position.x<<"\n";

//cout << "add: " << velocity.x << "\n";

//cout << "add: " << velocity.x + position.x << "\n";

velocity = velocity + acceleration;

if (velocity.x > maxspeed)

{

velocity.x = maxspeed;

}

if (velocity.x < -maxspeed)

{

velocity.x = -maxspeed;

}

if (velocity.y > maxspeed)

{

velocity.y = maxspeed;

}

if (velocity.y < -maxspeed)

{

velocity.y = -maxspeed;

}

if (velocity.z > maxspeed)

{

velocity.z = maxspeed;

}

if (velocity.z < -maxspeed)

{

velocity.z = -maxspeed;

}

//cout << "addbefore: " << position.x << "\n";

position = position + velocity;

//cout << "addafter: " << position.x << "\n";

acceleration = acceleration \* (float) 0;

}

void Physics::applyForce(glm::vec3 a)

{

acceleration = acceleration + a;

}

void Physics::flocking(std::vector<Physics> Boids)

{

//cout << "we in flocking";

glm::vec3 sep = separate(Boids);

glm::vec3 ali = align(Boids);

glm::vec3 coh = cohesion(Boids);

sep = sep \* (float)1.5;

ali = ali \* (float)1.0;

coh = coh \* (float)1.0;

applyForce(sep);

applyForce(ali);

applyForce(coh);

}

void Physics::borders()

{

}

void Physics::render()

{

if (position.x < 0)

{

position.x = 5;

}

if (position.y < 0)

{

position.y = 4;

}

if (position.x > 5)

{

position.x = 0;

}

if (position.y > 4)

{

position.y = 0;

}

}

void Physics::LessColor(float &f)

{

f = f - 0.02;

}

void Physics::Regen(float &f, int d)

{

if (f<0)

{

position = glm::vec3(-2.0f, 2.0f, -6.0f);

//srand(time(0));

glm::vec3 speed1(glm::normalize(glm::vec3((rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f)));

velocity = speed1;

Lifespan = 5;

f = 1.0;

}

}

glm::vec3 Physics::GetPos()

{

return position;

}

glm::vec3 Physics::GetPos1()

{

cout << "getpos: " << position.x << endl;

return position;

}

glm::vec3 Physics::separate(std::vector<Physics> Boids)

{

float desiredseparartion = 2.0f;

glm::vec3 steer = glm::vec3(0.0f, 0.0f, 0.0f);

int count = 0;

for (Physics b : Boids)

{

float d = glm::distance(position, b.position);

if ((d > 0) && (d < desiredseparartion))

{

glm::vec3 diff = position - b.position;

glm::normalize(diff);

diff = diff / d;

steer = steer + diff;

count++;

}

}

if (count > 0)

{

steer = steer / (float)count;

}

float steerMag = sqrt(pow(steer.x, 2) + pow(steer.y, 2) + pow(steer.z, 2));

if (steerMag > 0)

{

steer = glm::normalize(steer);

steer = steer \* maxspeed;

steer = steer - velocity;

if (steer.x > maxforce)

{

steer.x = maxforce;

}

if (steer.x < -maxforce)

{

steer.x = -maxforce;

}

if (steer.y > maxforce)

{

steer.y = maxforce;

}

if (steer.y < -maxforce)

{

steer.y = -maxforce;

}

if (steer.z > maxforce)

{

steer.z = maxforce;

}

if (steer.z < -maxforce)

{

steer.z = -maxforce;

}

}

return steer;

}

glm::vec3 Physics::align(std::vector<Physics> Boids)

{

float neighbordist = 50;

glm::vec3 sum = glm::vec3(0.0f, 0.0f, 0.0f);

int count = 0;

for (Physics b : Boids)

{

float d = glm::distance(position, b.position);

if ((d > 0) && (d < neighbordist))

{

sum = sum + b.velocity;

count++;

}

}

if (count > 0)

{

sum = sum / (float)count;

sum = glm::normalize(sum);

sum = sum \* maxspeed;

glm::vec3 steer = sum - velocity;

if (steer.x > maxforce)

{

steer.x = maxforce;

}

if (steer.x < -maxforce)

{

steer.x = -maxforce;

}

if (steer.y > maxforce)

{

steer.y = maxforce;

}

if (steer.y < -maxforce)

{

steer.y = -maxforce;

}

if (steer.z > maxforce)

{

steer.z = maxforce;

}

if (steer.z < -maxforce)

{

steer.z = -maxforce;

}

return steer;

}

else

{

return glm::vec3(0.0f, 0.0f, 0.0f);

}

}

glm::vec3 Physics::cohesion(std::vector<Physics> Boids)

{

float neighbordist = 50;

glm::vec3 sum = glm::vec3(0.0f, 0.0f, 0.0f);

int count = 0;

for (Physics b : Boids)

{

float d = glm::distance(position, b.position);

if ((d > 0) && (d < neighbordist))

{

sum = sum + b.position;

count++;

}

}

if (count > 0)

{

sum = sum / (float)count;

return seek(sum);

}

else

{

return glm::vec3(0.0f, 0.0f, 0.0f);

}

}

glm::vec3 Physics::seek(glm::vec3 t)

{

glm::vec3 desired = t - position;

desired = glm::normalize(desired);

desired = desired \* maxspeed;

glm::vec3 steer = desired - velocity;

if (steer.x > maxforce)

{

steer.x = maxforce;

}

if (steer.x < -maxforce)

{

steer.x = -maxforce;

}

if (steer.y > maxforce)

{

steer.y = maxforce;

}

if (steer.y < -maxforce)

{

steer.y = -maxforce;

}

if (steer.z > maxforce)

{

steer.z = maxforce;

}

if (steer.z < -maxforce)

{

steer.z = -maxforce;

}

return steer;

}

#include <iostream>

#include <vector>

using namespace std;

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm/glm.hpp>

#include "Graphics.h"

Graphics::Graphics() {

};

Graphics::~Graphics() {

};

int Graphics::Init() {

if (!glfwInit()) { // Checking for GLFW

cout << "Could not initialise GLFW...";

return 1;

}

glfwSetErrorCallback(ErrorCallbackGLFW); // Setup a function to catch and display all GLFW errors.

hintsGLFW(); // Setup glfw with various hints.

// Start a window using GLFW

string title = "My OpenGL Application";

window = glfwCreateWindow(windowWidth, windowHeight, title.c\_str(), NULL, NULL);

if (!window) { // Window or OpenGL context creation failed

cout << "Could not initialise GLFW...";

endProgram();

return 1;

}

glfwMakeContextCurrent(window); // making the OpenGL context current

// Start GLEW (note: always initialise GLEW after creating your window context.)

glewExperimental = GL\_TRUE; // hack: catching them all - forcing newest debug callback (glDebugMessageCallback)

GLenum errGLEW = glewInit();

if (GLEW\_OK != errGLEW) { // Problems starting GLEW?

cout << "Could not initialise GLEW...";

endProgram();

return 1;

}

SetupRender();

return 0;

}

void Graphics::hintsGLFW() {

glfwWindowHint(GLFW\_OPENGL\_DEBUG\_CONTEXT, GL\_TRUE); // Create context in debug mode - for debug message callback

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

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

}

void ErrorCallbackGLFW(int error, const char\* description) {

cout << "Error GLFW: " << description << "\n";

}

void Graphics::endProgram() {

glfwMakeContextCurrent(window); // destroys window handler

glfwTerminate(); // destroys all windows and releases resources.

}

void Graphics::SetupRender() {

glfwSwapInterval(1); // Ony render when synced (V SYNC)

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

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

glfwWindowHint(GLFW\_SAMPLES, 0);

glfwWindowHint(GLFW\_STEREO, GL\_FALSE);

}

void Graphics::SetOptimisations() {

glEnable(GL\_CULL\_FACE);

glFrontFace(GL\_CCW);

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LEQUAL);

}

void Graphics::ClearViewport() {

glViewport(0, 0, windowWidth, windowHeight);

static const GLfloat silver[] = { 0.9f, 0.9f, 0.9f, 1.0f };

glClearBufferfv(GL\_COLOR, 0, silver);

static const GLfloat one = 1.0f;

glClearBufferfv(GL\_DEPTH, 0, &one);

}

#include "Flock.h"

#include "Physics.h"

#include <ctime>

#include <cstdlib>

#include <iostream>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <cmath>

Flock::Flock()

{

}

void Flock::run(double delta)

{

//int num = 1;

//for (Physics x : Boids)

//{

// //cout << "in .."<<num<<" "<<x.velocity.x<<" " << x.velocity.y << " " << x.velocity.z << " " <<endl;

// //x.updateBoids(delta, Boids);

// x.updateflock(delta);

// num++;

//}

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

{

Boids.at(i).updateBoids(delta, Boids);

}

}

void Flock::addBoid()

{

Physics s = Physics(3);

s.position = glm::vec3(2.5f, 2.0f, -6.0f);

float angle = 0 + (rand() / (RAND\_MAX / (M\_PI)));

int min = -1;

int max = 1;

float rt = (float)rand() / (float)RAND\_MAX;

float z = min + rt \* (max - min);

float x = sqrt(1 - pow(z,2)) \* cos(angle);

float y = sqrt(1 - pow(z,2)) \* sin(angle);

glm::vec3 speed3 = glm::vec3(x,y,z);

glm::vec3 speed2 = glm::vec3(0.0f, 0.0f, 0.0f);

glm::vec3 speed1(glm::normalize(glm::vec3((rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f)));;

s.velocity = speed3;

//cout << " " << s.velocity.x << " " << s.velocity.y << " " << s.velocity.z << " " << s.velocity.x << "\n";

s.acceleration = glm::vec3(0.0f, 0.0f, 0.0f);

s.maxspeed = 0.1;

s.r = 2.0;

s.maxforce = 0.03;

Boids.push\_back(s);

count++;

}

glm::vec3 Flock::GetPos(int index)

{

return Boids.at(index).GetPos1();

}

#include "Physics.h"

#include "Shapes.h"

#include "Explode.h"

#include <ctime>

#include <cstdlib>

#include <iostream>

Explode::Explode(int num)

{

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

{

Physics s = Physics(2);

s.position = glm::vec3(-2.0f, 2.0f, -6.0f);

glm::vec3 speed1(glm::normalize(glm::vec3((rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f, (rand() % 1500 - 750) / 10000.0f)));;

s.velocity = speed1;

Spheres.push\_back(s);

}

}

void Explode::update\_balls(double delta, int num, std::vector <Sphere> & all)

{

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

{

Spheres.at(i).updateEx(delta, all.at(i).fillColor.a);

Spheres.at(i).LessColor(all.at(i).fillColor.a);

Spheres.at(i).Regen(all.at(i).fillColor.a, i);

}

}

glm::vec3 Explode::GetPos(int index)

{

return Spheres.at(index).GetPos();

}

### A\* Path planning search

**from** im **import** \*

start, goal

came\_from, cost\_so\_far = a\_star\_search(grid, start, goal)

# draw\_grid(grid, width=3, point\_to=came\_from, start=start, goal=goal)

**print**()

draw\_grid(grid, width=3, number=cost\_so\_far, start=start, goal=goal)

**print**()

draw\_grid(grid, width=3, path=reconstruct\_path(came\_from, start=start, goal=goal))

# Sample code from https://www.redblobgames.com/pathfinding/a-star/

# Copyright 2014 Red Blob Games <redblobgames@gmail.com>

#

# Feel free to use this code in your own projects, including commercial projects

# License: Apache v2.0 <http://www.apache.org/licenses/LICENSE-2.0.html>

**import** collections

**import** heapq

**import** random

**class** SimpleGraph:

**def** \_\_init\_\_(self):

self.edges = {}

**def** neighbors(self, id):

**return** self.edges[id]

**class** Queue:

**def** \_\_init\_\_(self):

self.elements = collections.deque()

**def** empty(self):

**return** len(self.elements) == 0

**def** put(self, x):

self.elements.append(x)

**def** get(self):

**return** self.elements.popleft()

# utility functions for dealing with square grids

**def** from\_id\_width(id, width):

**return** (id % width, id // width)

**def** draw\_tile(graph, id, style, width):

r = "."

**if** 'number' **in** style **and** id **in** style['number']: r = "%d" % style['number'][id]

**if** 'point\_to' **in** style **and** style['point\_to'].get(id, None) **is** **not** None:

(x1, y1) = id

(x2, y2) = style['point\_to'][id]

**if** x2 == x1 + 1: r = ">"

**if** x2 == x1 - 1: r = "<"

**if** y2 == y1 + 1: r = "v"

**if** y2 == y1 - 1: r = "^"

**if** 'start' **in** style **and** id == style['start']: r = "S"

**if** 'goal' **in** style **and** id == style['goal']: r = "G"

**if** 'path' **in** style **and** id **in** style['path']: r = "X"

**if** id **in** graph.walls: r = "||"

**return** r

**def** draw\_grid(graph, width=2, \*\*style):

**for** y **in** range(graph.height):

**for** x **in** range(graph.width):

**print**("%%-%ds" % width % draw\_tile(graph, (x, y), style, width), end="")

**print**()

**class** SquareGrid:

**def** \_\_init\_\_(self, width, height):

self.width = width

self.height = height

self.walls = []

**def** in\_bounds(self, id):

(x, y) = id

**return** 0 <= x < self.width **and** 0 <= y < self.height

**def** passable(self, id):

**return** id **not** **in** self.walls

**def** neighbors(self, id):

(x, y) = id

results = [(x + 1, y), (x, y - 1), (x - 1, y), (x, y + 1)]

**if** (x + y) % 2 == 0: results.reverse() # aesthetics

results = filter(self.in\_bounds, results)

results = filter(self.passable, results)

**return** results

**class** GridWithWeights(SquareGrid):

**def** \_\_init\_\_(self, width, height):

super().\_\_init\_\_(width, height)

self.weights = {}

**def** cost(self, from\_node, to\_node):

**return** self.weights.get(to\_node, 1)

grid = GridWithWeights(20, 20)

grid.walls = []

sx = random.randint(0, 19)

sy = random.randint(0, 19)

gx = random.randint(0, 19)

gy = random.randint(0, 19)

start, goal = (0, 0), (19, 19)

w = random.randint(80, 100)

**for** n **in** range(w):

x = 100\*random.uniform(0, 0.19)

y = 100\*random.uniform(0, 0.19)

r = (int(x), int(y))

**while** r == (0, 0) **and** r == (19, 19) **and** r == grid.walls[n-1]:

x = 100 \* random.uniform(0, 0.19)

y = 100 \* random.uniform(0, 0.19)

r = (int(x), int(y))

# for n in range(len(a)):

# if r == grid.walls[n]:

# x = 100 \* random.uniform(0, 0.19)

# y = 100 \* random.uniform(0, 0.19)

# r = (int(x), int(y))

grid.walls.append(r)

a = grid.walls

**class** PriorityQueue:

**def** \_\_init\_\_(self):

self.elements = []

**def** empty(self):

**return** len(self.elements) == 0

**def** put(self, item, priority):

heapq.heappush(self.elements, (priority, item))

**def** get(self):

**return** heapq.heappop(self.elements)[1]

**def** reconstruct\_path(came\_from, start, goal):

current = goal

path = []

**while** current != start:

path.append(current)

current = came\_from[current]

path.append(start) # optional

path.reverse() # optional

**return** path

# Euclidean distance

**def** heuristic(a, b):

(x1, y1) = a

(x2, y2) = b

**return** abs(x1 - x2) + abs(y1 - y2)

**def** a\_star\_search(graph, start, goal):

frontier = PriorityQueue()

frontier.put(start, 0)

came\_from = {}

cost\_so\_far = {}

came\_from[start] = None

cost\_so\_far[start] = 0

**while** **not** frontier.empty():

current = frontier.get()

**if** current == goal:

**break**

**for** next **in** graph.neighbors(current):

new\_cost = cost\_so\_far[current] + graph.cost(current, next)

**if** next **not** **in** cost\_so\_far **or** new\_cost < cost\_so\_far[next]:

cost\_so\_far[next] = new\_cost

priority = new\_cost + heuristic(goal, next)

frontier.put(next, priority)

came\_from[next] = current

**return** came\_from, cost\_so\_far