Содержание

ВВЕДЕНИЕ6

1 Техническое задание на предмет разработки7

2 Обоснование используемых языков и технологий7

3 Моделирование и проектирование программного обеспечения 9

3.1 Анализ задачи9

3.2 Создание игрового движка9

3.3 Варианты использования приложения 12

3.3 Тестирование игрового движка посредством создания игры. 13

ЗАКЛЮЧЕНИЕ 16

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 16

Приложение A Java-листинг16

ВВЕДЕНИЕ

Целью курсовой работы была поставлена задача изучения графической механики компьютерных игр, в итоге был получен набор классов и методов позволяющих отрисовывать 3d модели а так же, полученную в процессе тестирования, игру.

Программа основана на графической библиотеке OpenGL, реализованная языком Java с использованием библиотеки LWJGL - Lightweight Java Game Library. Данные средства позволяют не только отрисовывать трехмерные модели но и добавлять различные графические эффекты к ним. Данная программа является реализацией простейшего движка с возможностью дальнейшего расширения пользователем.

Возможность отрисовки 3d моделей реализованна системой VAO и VBO. Vertex Buffer Object (VBO) — это такое средство OpenGL, позволяющее загружать определенные данные в память GPU. Например, если вы хотите сообщить GPU координаты вершин, цвета или нормали, нужно создать VBO и положить эти данные в него. Vertex Arrays Object (VAO) — это механизм, который говорит OpenGL, какую часть VBO следует использовать в последующих командах.

Программа реализует отрисовку шейдоров, а так же дает возможность использовать систему mipmapping.

Все возможности движка реализованы в игре “Звездные войны”. В игре продемонстрирована коллизия объектов, динамическое создание этих объектов, а так же реализована стандартными функциями java возможность мультиплеера.

# 1 Техническое задание на предмет разработки

Конечной целью в данном приложении является создание игры. Данная игра имеет возможность управления через стандартные методы ввода, отображения изменений на экране.

По сценарию пользователь имеет космический корабль, который имеет свой индикатор здоровья и умеет стрелять. Окружение корабля состоит из астероидов, при столкновении с которыми корабль теряет двадцать единиц здоровья. Задача пользователя не допустить того, чтобы показатель здоровья стал равен нулю.  
В режиме мультиплеера у пользователя есть соперник, который может уничтожить корабль пользователя, но и пользователь может уничтожить соперника. Задача пользователя та же, не допустить того, чтобы показатель здоровья упал до нуля.

**2 Обоснование используемых языков и технологий**

Разрабатывать данное приложение будем в интегрированной среде разработки NetBeans 8.2. Данная среда разработки предоставляет широкий перечень инструментов и возможностей для написания и отладки приложения. NetBeans имеет удобный и понятный пользователю интерфейс, всевозможные клавиши управления программой во время ее написания. Благодаря этому скорость написания кода увеличивается и прогресс написания программы возрастает.

После загрузки Java пользователи получают Java Runtime Environment (JRE). JRE состоит из Java Virtual Machine (JVM), базовых классов платформы Java и вспомогательных библиотек платформы Java. JRE является областью программного обеспечения Java, используемой во время выполнения, т.е. единственным компонентом, который требуется для его запуска в используемом в веб-браузере.

Java Development Kit - комплект разработчика приложений на языке Java, включающий в себя компилятор Java, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java(JRE). В состав JDK не входит интегрированная среда разработки на Java.

Lightweight Java Game Library (LWJGL) — открытая графическая библиотека, основной целью которой является предоставление простого и легковесного программного интерфейса для создателей компьютерных игр на языке Java.

LWJGL является высокопроизводительной кроссплатформенной библиотекой, широко используемой в разработке компьютерных игр и мультимедийных приложениях. Она предоставляет доступ к OpenGL, OpenAL, OpenCL и обеспечивает платформонезависимый доступ к различным манипуляторам, таким как геймпады, рули и джойстики.

Основной целью проекта является создание технологии, которая позволяла бы Java-разработчикам получить доступ к ресурсам, доступ к которым в настоящее время затруднён или вовсе отсутствует из Java платформы.

LWJGL доступна под BSD-лицензией. Будучи открытой и бесплатной, является основой многих игровых движков и библиотек.

# 3. Моделирование и проектирование программного обеспечения

## **3.1 Анализ задачи**

Процесс разработки приложения можно разделить на 2 этапа:

1) Создание игрового движка  
2) Тестирование игрового движка посредством написания игры.

## **3.2 Создание игрового движка**

Написание игрового движка сводиться к использованию возможностей графической библиотеки LWJGL что значительно упрощает его создание.

Отрисовка 3d моделей производиться посредством создания VAO массива, добавлением в него координат вершин 3d модели, созданием второго VAO массива добавлением в него карту текстур, и третьего VAO массива и добавлением в него карту нормалей. Следующим шагом является добавление этих трех массивов в массив VBO.

Этот метод эффективен тем, что для него не нужно добавлять координаты вершин текстур и нормалей так как они совпадают с первым элементом VBO(VAO координат вершин). Для каждой 3d модели нужно создавать отдельные VAO и VBO, так же для того чтобы пользователь смог увидеть модель ему нужно три указанных VAO.

Следующим этапом является отрисовка модели в окне пользователя. Создание модели начинается с добавления вершин взятых из первого элемента VBO, и последовательным их соединением с использованием Index Buffer. Index Buffer является ключом для отрисовки моделей, что уменьшает количество используемых вершин для отрисовки на две за каждый квадрат модели, то есть 30% экономии производительности от каждой модели.

Имея отрисованные вершины модели мы можем добавить карту текстур и карту нормалей.

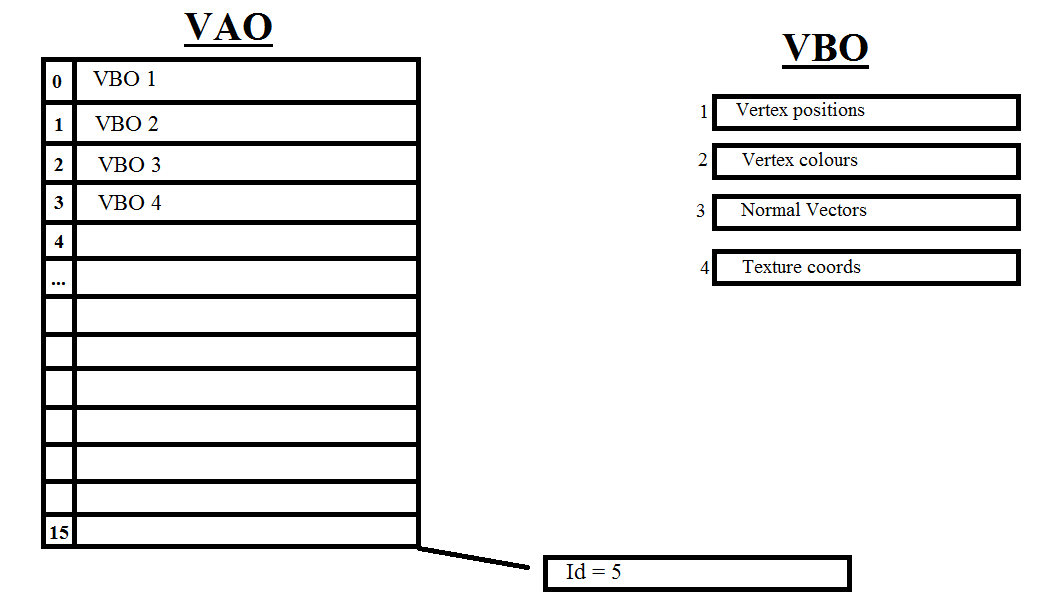


Рисунок 1 –VBO, VAO.

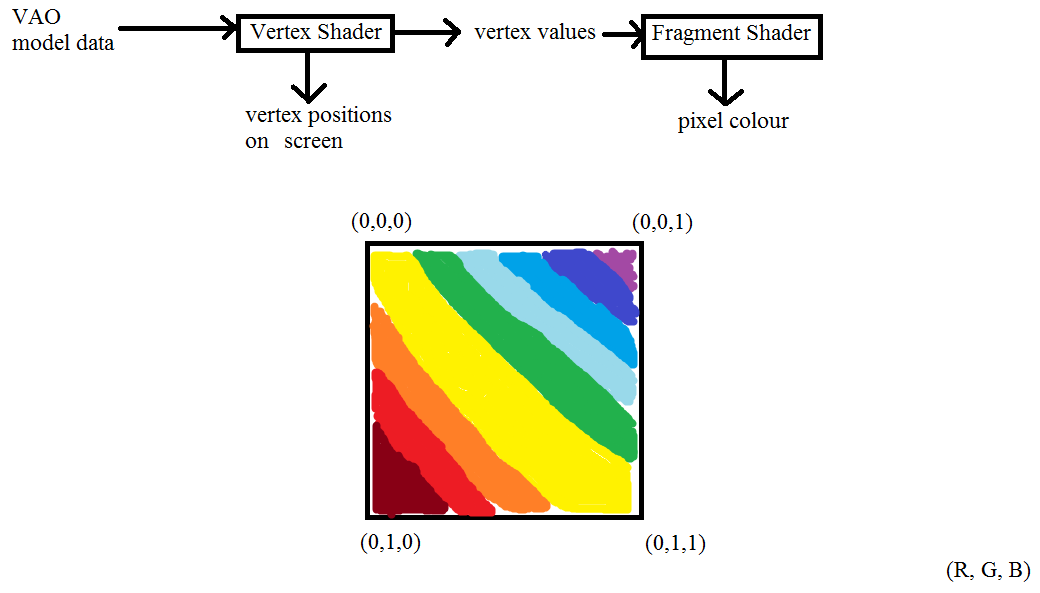
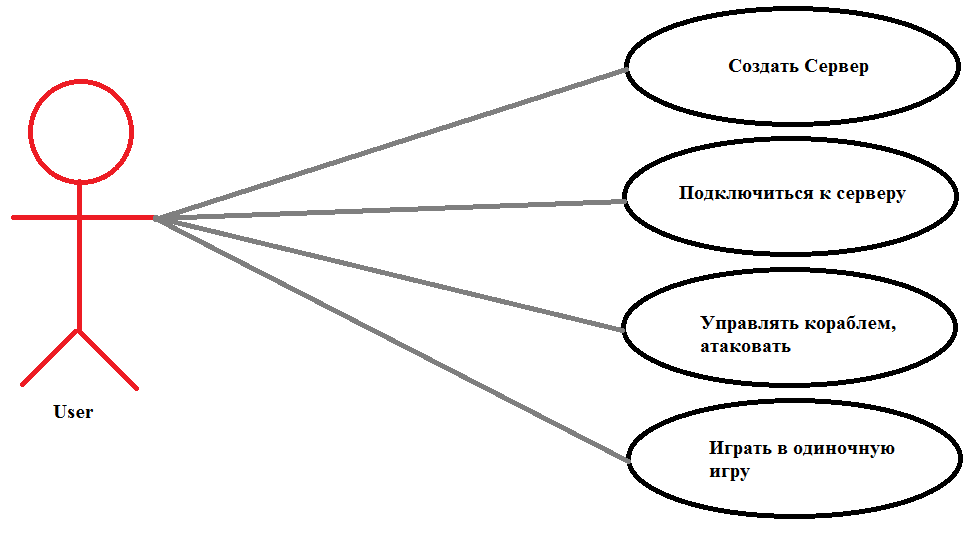


Рисунок 2 – Shaders.

**3.3 Варианты использования приложения**

На рисунке 3 приведена диаграмма вариантов использования.

**** Рисунок 3 – Диаграмма вариантов использования.

1) Создание собственного локального сервера.

2) Возможность подключения к локальному серверу.

3) Управление кораблем.

4) Создание одиночной игры.

**3.3 Тестирование игрового движка посредством создания игры**

В процессе тестирования движка была создана простейшая игра “Звёздные войны”. Игра демонстрирует графические возможности движка, имеет элементы игры по сети, демонстрирует эффекты взаимодействия и динамического создания объектов.

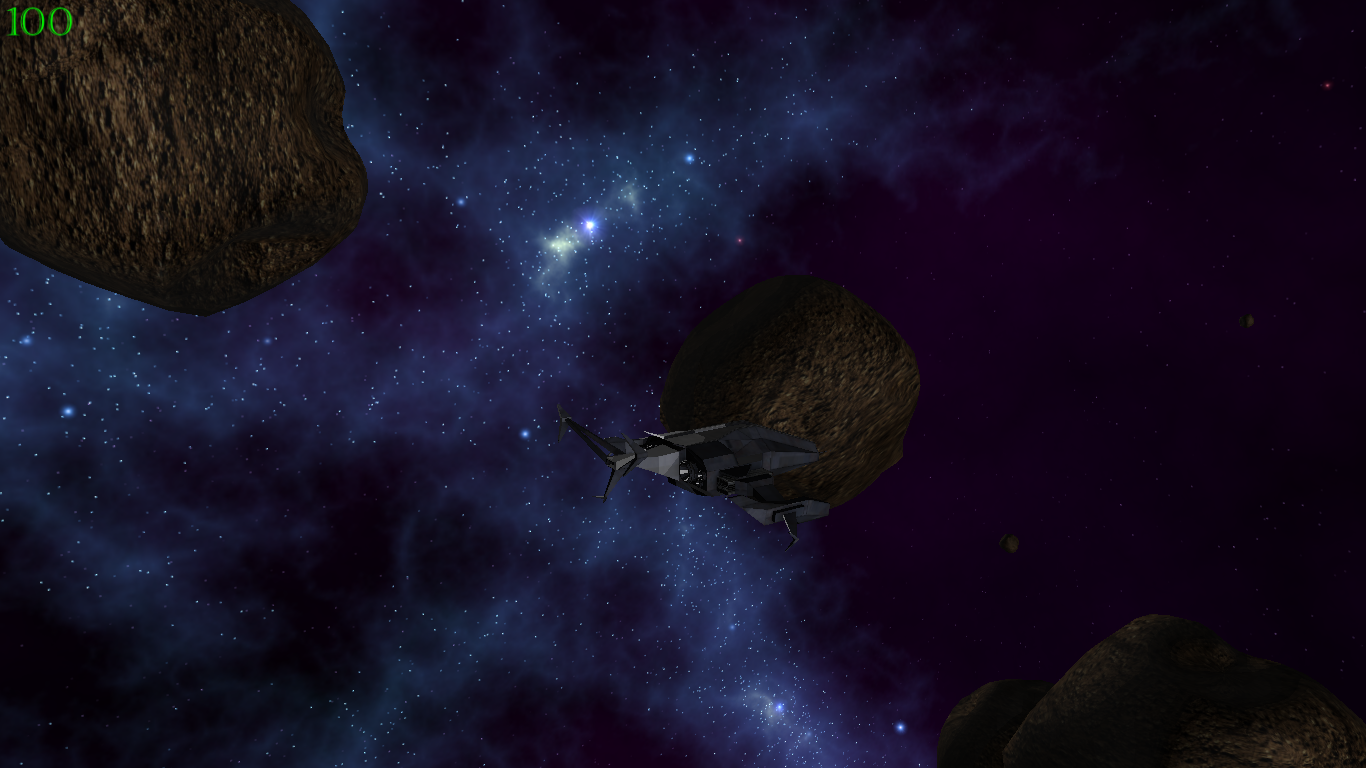


Рисунок 1 – тестирование игрового движка.

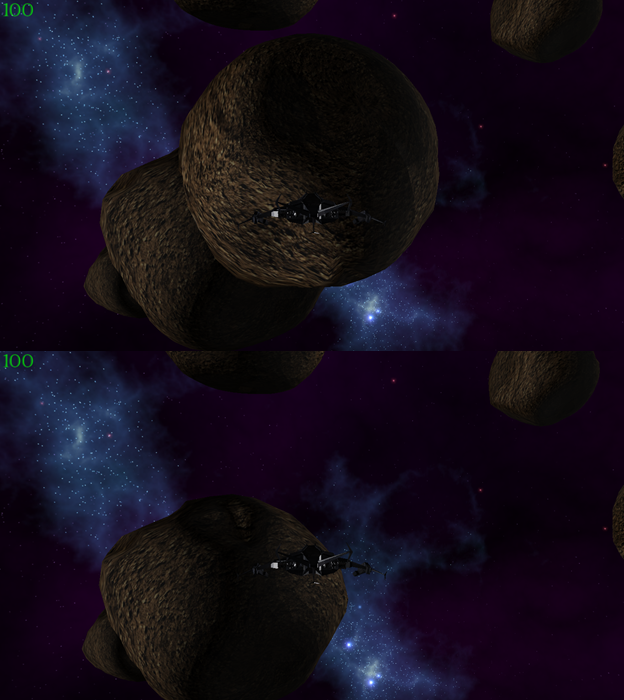


Рисунок 2 – демонстрация коллизий.

Пользователь появляется в координатах [0, 0, 0] и имеет возможность двигаться во всех направлениях. Если пользователь сталкивается с астероидом, то у него отнимается 20 единиц здоровья, так же у пользователя есть возможность уничтожить данный объект.

# ЗАКЛЮЧЕНИЕ

В итоге курсовой работы были получены общие навыки разработки игровых приложений.

Способ отрисовки моделей с использованием VBO и VAO техник значительно упрощает создание 3d моделей, а так же и увеличивает общую производительность. Index Buffer позволяет эффективно отрисовывать полигоны моделей, без повторной отрисовки линий, что увеличивает общую производительность приложения.

Использование системы mipmapping значительно увеличивает производительность проекта но и незаметно уменьшает визуальную составляющую игры. Использование же шейдеров наоборот увеличивает качество изображения но соответственно и уменьшает производительность.

Подведя итоги можно сказать, что качество игровых приложений обратно-пропорционально ее производительности. С каждым годом технологии в мире изменяются, и эти изменения хорошо видны в видеоиграх. Каждая игра, использующая современные технологии, несомненно является шедевром, что не может не привлекать людей для создания собственных чудес.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1) Базовая документация Java: http://docs.oracle.com/javase/6/docs/api/

2) Википедия Java (Software platform): https://ru.wikipedia.org/wiki/Java

3) Базовая документация NetBeans 8.2: https://netbeans.org/

Приложение A Java-листинг

Main.java

package engineTester;

import entities.SimpleEntity;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application{

@Override

public void start(Stage primaryStage) throws Exception {

Parent root = FXMLLoader.load(getClass().getResource("../StartGame/StartGameMenu.fxml"));

primaryStage.setTitle("Course Project");

primaryStage.setScene(new Scene(root, primaryStage.getWidth(), primaryStage.getHeight()));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

Файл SinglePlayer.java

package engineTester;

import entities.World;

public class SinglePlayer {

public void Start(){

World world = new World();

world.worldInit();

world.worldLoopRenderer();

world.worldCleanUp();

}

}

**Package entities**

Файл Bullet.java

package entities;

import org.lwjgl.util.vector.Vector3f;

import renderEngine.DisplayManager;

public class Bullet extends Entity {

private final float SPEED = 200;

private final int creatorID;

public int getCreatorID() {

return creatorID;

}

public Bullet(World world, Player player) {

super(world.bulletModel, new Vector3f(player.getPosition()), player.getRotX(), player.getRotY(), player.getRotZ(), 1);

this.creatorID = player.getID();

}

public void move() {

float distance = SPEED \* DisplayManager.getFarmeTimeSeconds();

float dz = (float) (distance \* Math.cos(Math.toRadians(super.getRotY())));

float dx = (float) (distance \* Math.sin(Math.toRadians(super.getRotY())));

super.increasePosition(dx, 0, dz);

}

}

Файл Camera.java

package entities;

import org.lwjgl.input.Mouse;

import org.lwjgl.util.vector.Vector3f;

import renderEngine.DisplayManager;

public class Camera {

private float distanceFromPlayer = 15;

private float angleAroundPlayer = 0;

private Vector3f position = new Vector3f(0, 0, 0);

private float pitch;

private float yaw;

private float roll;

private Player player;

public Camera(Player player) {

this.player = player;

}

public void move() {

calculateZoom();

calculatePitch();

calculateAngleAroundPlayer();

float horizantalDistance = calculateHorizontalDistance();

float verticalDistance = calculateVerticalDistance();

calculateCameraPosition(horizantalDistance, verticalDistance);

this.yaw = 180 - (player.getRotY() + angleAroundPlayer);

}

public Vector3f getPosition() {

return position;

}

public void setPosition(Vector3f position) {

this.position = position;

}

public float getPitch() {

return pitch;

}

public void setAngleAroundPlayer(float angleAroundPlayer) {

this.angleAroundPlayer = angleAroundPlayer;

}

public void setPitch(float pitch) {

this.pitch = pitch;

}

public float getYaw() {

return yaw;

}

public float getRoll() {

return roll;

}

public float getAngleAroundPlayer() {

return angleAroundPlayer;

}

private void calculateCameraPosition(float horizDistance, float verticDistance) {

float theta = player.getRotY() + angleAroundPlayer;

float offsetX = (float) (horizDistance \* Math.sin(Math.toRadians(theta)));

float offsetZ = (float) (horizDistance \* Math.cos(Math.toRadians(theta)));

position.x = player.getPosition().x - offsetX;

position.z = player.getPosition().z - offsetZ;

position.y = player.getPosition().y + verticDistance + 2;

}

private float calculateHorizontalDistance() {

return (float) (distanceFromPlayer \* Math.cos(Math.toRadians(pitch)));

}

private float calculateVerticalDistance() {

return (float) (distanceFromPlayer \* Math.sin(Math.toRadians(pitch)));

}

private void calculateZoom() {

float zoomLevel = Mouse.getDWheel() \* 0.1f \* DisplayManager.getFarmeTimeSeconds();

distanceFromPlayer -= zoomLevel;

}

private void calculatePitch() {

float pitchChange = Mouse.getDY() \* 0.1f;

pitch -= pitchChange;

}

private void calculateAngleAroundPlayer() {

float angleChange = Mouse.getDX() \* 0.3f;

angleAroundPlayer -= angleChange;

}

}

Файл Chunks.java

package entities;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.logging.Level;

import java.util.logging.Logger;

import models.TexturedModel;

import org.lwjgl.util.vector.Vector3f;

import renderEngine.Loader;

import textures.ModelTexture;

import renderEngine.OBJLoader;

public class Chunks {

private final int ZONE = 200;

private final int NUMBER = 50;

private Vector3f CURRENT\_CHUNK = new Vector3f(0, 0, 0);

private final TexturedModel asteroid;

private static final String CHUNK\_LOG = "src/logs/chunkLog.txt";

public Chunks(Loader loader) {

try {

BufferedWriter writer = new BufferedWriter(new FileWriter(CHUNK\_LOG, false));

} catch (IOException ex) {

Logger.getLogger(Chunks.class.getName()).log(Level.SEVERE, null, ex);

}

this.asteroid = new TexturedModel(OBJLoader.loadObjModel("astr\_1", loader),

new ModelTexture(loader.loadTexture("astr\_1")));

ModelTexture texture1 = this.asteroid.getTexture();

texture1.setShineDamper(1);

texture1.setReflectivity((float) 0.1);

}

public List<Entity> chunkBuilder(Vector3f playerPosition) {

List<Entity> chunk = new ArrayList<>();

if (playerPosition.z - CURRENT\_CHUNK.z \* ZONE >= ZONE / 2) {

CURRENT\_CHUNK.z += 1;

}

if (playerPosition.z - CURRENT\_CHUNK.z \* ZONE <= -ZONE / 2) {

CURRENT\_CHUNK.z -= 1;

}

if (playerPosition.x - CURRENT\_CHUNK.x \* ZONE >= ZONE / 2) {

CURRENT\_CHUNK.x += 1;

}

if (playerPosition.x - CURRENT\_CHUNK.x \* ZONE <= -ZONE / 2) {

CURRENT\_CHUNK.x -= 1;

}

if (playerPosition.y - CURRENT\_CHUNK.y \* ZONE >= ZONE / 2) {

CURRENT\_CHUNK.y += 1;

}

if (playerPosition.y - CURRENT\_CHUNK.y \* ZONE <= -ZONE / 2) {

CURRENT\_CHUNK.y -= 1;

}

CURRENT\_CHUNK = new Vector3f(CURRENT\_CHUNK.x, CURRENT\_CHUNK.y, CURRENT\_CHUNK.z);

try {

String chunkLogLine;

BufferedReader reader = new BufferedReader(new FileReader(CHUNK\_LOG));

BufferedWriter writer = new BufferedWriter(new FileWriter(CHUNK\_LOG, true));

while ((chunkLogLine = reader.readLine()) != null) {

if (chunkLogLine.equals(CURRENT\_CHUNK.x + "/" + CURRENT\_CHUNK.y + "/" + CURRENT\_CHUNK.z)) {

return chunk;

}

}

chunk = generateChunk();

writer.write(CURRENT\_CHUNK.x + "/" + CURRENT\_CHUNK.y + "/" + CURRENT\_CHUNK.z + "\n");

writer.close();

} catch (IOException ex) {

System.out.println("can't load log");

}

return chunk;

}

public List<Entity> generateChunk() {

Random random = new Random();

List<Entity> asteroids = new ArrayList<>();

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

float x = 2 \* ZONE \* (random.nextFloat() + CURRENT\_CHUNK.x - (float) 0.5);

float y = 2 \* ZONE \* (random.nextFloat() + CURRENT\_CHUNK.y - (float) 0.5);

float z = 2 \* ZONE \* (random.nextFloat() + CURRENT\_CHUNK.z - (float) 0.5);

float scale = random.nextInt(30);

asteroids.add(new Entity(this.asteroid, new Vector3f(x, y, z),

random.nextFloat() \* 180f, random.nextFloat() \* 180f, 0f, scale));

}

return asteroids;

}

}

Файл Entity.java

package entities;

import models.TexturedModel;

import org.lwjgl.util.vector.Vector3f;

public class Entity {

private TexturedModel model;

private Vector3f position;

private float rotX, rotY, rotZ;

private float scale;

private int XP = 100;

public int getXP() {

return XP;

}

public void setXP(int XP) {

this.XP = XP;

}

public Entity(TexturedModel model, Vector3f position, float rotX, float rotY, float rotZ,

float scale) {

this.model = model;

this.position = position;

this.rotX = rotX;

this.rotY = rotY;

this.rotZ = rotZ;

this.scale = scale;

}

public void increasePosition(float dx, float dy, float dz) {

this.position.x += dx;

this.position.y += dy;

this.position.z += dz;

}

public void increaseRotation(float dx, float dy, float dz) {

this.rotX += dx;

this.rotY += dy;

this.rotZ += dz;

}

public TexturedModel getModel() {

return model;

}

public void setModel(TexturedModel model) {

this.model = model;

}

public Vector3f getPosition() {

return position;

}

public void setPosition(Vector3f position) {

this.position = position;

}

public float getRotX() {

return rotX;

}

public void setRotX(float rotX) {

this.rotX = rotX;

}

public float getRotY() {

return rotY;

}

public void setRotY(float rotY) {

this.rotY = rotY;

}

public float getRotZ() {

return rotZ;

}

public void setRotZ(float rotZ) {

this.rotZ = rotZ;

}

public float getScale() {

return scale;

}

public void setScale(float scale) {

this.scale = scale;

}

}

Файл Light.java

package entities;

import org.lwjgl.util.vector.Vector3f;

public class Light {

private Vector3f position;

private Vector3f colour;

private Vector3f attenuation = new Vector3f(1, 0, 0);

public Light(Vector3f position, Vector3f colour) {

this.position = position;

this.colour = colour;

}

public Light(Vector3f position, Vector3f colour, Vector3f attenuation) {

this.position = position;

this.colour = colour;

this.attenuation = attenuation;

}

public Vector3f getAttenuation() {

return attenuation;

}

public Vector3f getPosition() {

return position;

}

public void setPosition(Vector3f position) {

this.position = position;

}

public Vector3f getColour() {

return colour;

}

public void setColour(Vector3f colour) {

this.colour = colour;

}

}

Файл Player.java

package entities;

import models.TexturedModel;

import org.lwjgl.input.Keyboard;

import org.lwjgl.util.vector.Vector3f;

import renderEngine.DisplayManager;

public class Player extends Entity {

private int ID;

public float RELOAD = 500;

public float MAX\_SPEED = 100;

public float SIDE\_MAX\_SPEED = 3;

public float ROTATION\_SPEED = 50;

public float BOOST = (float) 0.5;

public float ROTATION\_BOOST = 30;

public float BRAKING = (float) 0.5;

public long currentReloadTime = 1;

private float rotationY = 0;

private float currentForwardSpeed = 0;

private float currentSideSpeed = 0;

private float currentUpSpeed = 0;

private boolean check = true;

public void setCheck(boolean check) {

this.check = check;

}

public int getID() {

return ID;

}

public Player(TexturedModel model, Vector3f position, float rotX, float rotY, float rotZ, float scale, int ID) {

super(model, position, rotX, rotY, rotZ, scale);

this.ID = ID;

}

public boolean checkReload() {

long end = System.currentTimeMillis();

if ((end - currentReloadTime) >= RELOAD) {

this.currentReloadTime = System.currentTimeMillis();

return true;

} else {

return false;

}

}

public void move() {

checkInputs();

float distanceZ = (float) (currentSideSpeed \* Math.cos(Math.toRadians(super.getRotY() - 90)));

float distanceX = (float) (currentSideSpeed \* Math.sin(Math.toRadians(super.getRotY() - 90)));

float dz = (float) (currentForwardSpeed \* Math.cos(Math.toRadians(super.getRotY())));

float dx = (float) (currentForwardSpeed \* Math.sin(Math.toRadians(super.getRotY())));

super.increaseRotation(0, rotationY, 0);

super.increasePosition(dx + distanceX, currentUpSpeed, dz + distanceZ);

}

private float boost(float currentSpeed, float maxSpeed, float boostSpeed, float direction) {

if (direction \* currentSpeed <= maxSpeed) {

currentSpeed += direction \* boostSpeed \* DisplayManager.getFarmeTimeSeconds();

}

return currentSpeed;

}

private float braking(float currentSpeed, float brakingSpeed) {

float direction = Math.abs(currentSpeed) / currentSpeed;

if (direction \* currentSpeed >= brakingSpeed \* DisplayManager.getFarmeTimeSeconds()) {

currentSpeed -= direction \* brakingSpeed \* DisplayManager.getFarmeTimeSeconds();

} else {

currentSpeed = 0;

}

return currentSpeed;

}

private void checkInputs() {

// Key Space, LControl

if (Keyboard.isKeyDown(Keyboard.KEY\_SPACE) && check) {

currentUpSpeed = boost(currentUpSpeed, SIDE\_MAX\_SPEED, BOOST, 1);

} else if (Keyboard.isKeyDown(Keyboard.KEY\_LCONTROL) && check) {

currentUpSpeed = boost(currentUpSpeed, SIDE\_MAX\_SPEED, BOOST, -1);

} else {

currentUpSpeed = braking(currentUpSpeed, BRAKING);

}

//Key W

if (Keyboard.isKeyDown(Keyboard.KEY\_W) && check) {

currentForwardSpeed = boost(currentForwardSpeed, MAX\_SPEED, BOOST, 1);

} else {

currentForwardSpeed = braking(currentForwardSpeed, BRAKING);

}

//Keys A, D

if (Keyboard.isKeyDown(Keyboard.KEY\_A) && check) {

rotationY = ROTATION\_SPEED \* DisplayManager.getFarmeTimeSeconds();

} else if (Keyboard.isKeyDown(Keyboard.KEY\_D) && check) {

rotationY = -ROTATION\_SPEED \* DisplayManager.getFarmeTimeSeconds();

} else {

rotationY = 0;

}

//Keys E, Q

if (Keyboard.isKeyDown(Keyboard.KEY\_E) && check) {

currentSideSpeed = boost(currentSideSpeed, SIDE\_MAX\_SPEED, BOOST, 1);

} else if (Keyboard.isKeyDown(Keyboard.KEY\_Q) && check) {

currentSideSpeed = boost(currentSideSpeed, SIDE\_MAX\_SPEED, BOOST, -1);

} else {

currentSideSpeed = braking(currentSideSpeed, BRAKING);

}

}

}

Файл World.java

package entities;

import fontMeshCreator.FontType;

import fontMeshCreator.GUIText;

import fontRendering.TextMaster;

import java.io.File;

import java.util.ArrayList;

import java.util.List;

import models.TexturedModel;

import org.apache.commons.collections4.ListUtils;

import org.lwjgl.input.Keyboard;

import org.lwjgl.input.Mouse;

import org.lwjgl.opengl.Display;

import org.lwjgl.util.vector.Vector2f;

import org.lwjgl.util.vector.Vector3f;

import renderEngine.DisplayManager;

import renderEngine.Loader;

import renderEngine.MasterRenderer;

import renderEngine.OBJLoader;

import textures.ModelTexture;

import toolbox.Collisions;

public class World {

public Loader loader;

private MasterRenderer renderer;

private Collisions collision;

private List<Light> lights;

public List<Bullet> bullets;

private Chunks chunks;

private FontType font;

private Camera camera;

private boolean closeRequested = false;

public List<Player> players;

private GUIText xp;

private GUIText gameOver;

private GUIText gameRule;

private List<Entity> chunk;

public TexturedModel bulletModel;

public boolean bulletCreate = false;

public void worldInit() {

// Init block

DisplayManager.createDisplay();

loader = new Loader();

renderer = new MasterRenderer(loader);

collision = new Collisions();

players = new ArrayList<>();

lights = new ArrayList<>();

bullets = new ArrayList<>();

chunks = new Chunks(loader);

font = new FontType(loader.loadTexture("Castellar"), new File("res/Castellar.fnt"));

TextMaster.init(loader);

// Player block

TexturedModel playerShip = new TexturedModel(OBJLoader.loadObjModel("space\_ship\_1", loader),

new ModelTexture(loader.loadTexture("space\_ship\_1")));

ModelTexture playerShipTexture = playerShip.getTexture();

playerShipTexture.setShineDamper(10);

playerShipTexture.setReflectivity(1);

players.add(new Player(playerShip, new Vector3f(0, 0, 0), 0, 0, 0, 1, players.size()));

camera = new Camera(players.get(0));

// Ligths

lights.add(new Light(new Vector3f(7000, 20, 0), new Vector3f(1f, 1f, 1f)));

lights.add(new Light(new Vector3f(-7000, 20, 0), new Vector3f(1f, 1f, 1f)));

// Bullets block

bulletModel = new TexturedModel(OBJLoader.loadObjModel("bullet\_1", loader),

new ModelTexture(loader.loadTexture("bullet\_1")));

// Asterods block

chunk = chunks.generateChunk();

// Texts

xp = new GUIText(Integer.toString(players.get(0).getXP()), 2, font, new Vector2f(-0.47f, 0), 1f, true);

gameOver = new GUIText("", 5, font, new Vector2f(0, 0.45f), 1f, true);

}

public void worldCleanUp() {

TextMaster.cleanUp();

loader.cleanUp();

renderer.cleanUp();

DisplayManager.closeDisplay();

}

public void worldLoopRenderer() {

while (!Display.isCloseRequested() && !closeRequested) {

renderer.render(lights, camera);

camera.move();

players.get(0).move();

if (Keyboard.isKeyDown(Keyboard.KEY\_ESCAPE)) {

closeRequested = !closeRequested;

}

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

if (players.get(i).getXP() > 0) {

renderer.processEntity(players.get(i));

}

}

xp.remove();

if (players.get(0).getXP() <= 0) {

xp = new GUIText("0", 2, font, new Vector2f(-0.47f, 0), 1f, true);

xp.setColour(0, 1, 0);

gameOver.remove();

gameOver = new GUIText("GAME OVER", 5, font, new Vector2f(0, 0.45f), 1f, true);

gameOver.setColour(1, 0, 0);

} else {

xp = new GUIText(Integer.toString(players.get(0).getXP()), 2, font, new Vector2f(-0.47f, 0), 1f, true);

xp.setColour(0, 1, 0);

}

bulletCreate = false;

if (Mouse.isButtonDown(0) && players.get(0).checkReload()) {

bullets.add(new Bullet(World.this, players.get(0)));

bulletCreate = true;

}

chunk = ListUtils.union(chunk, chunks.chunkBuilder(players.get(0).getPosition()));

for (int j = 0; j < players.size(); j++) {

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

if (collision.getCollision(bullets.get(i), players.get(j)) && players.get(j).getID() != bullets.get(i).getCreatorID()) {

players.get(j).setXP(players.get(j).getXP() - 40);

bullets.remove(i);

}

}

}

//Asteroids render and collisions

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

if (collision.getCollision(chunk.get(i), players.get(0))) {

players.get(0).setXP(players.get(0).getXP() - 20);

chunk.remove(i);

continue;

}

if (i < bullets.size()) {

bullets.get(i).move();

renderer.processEntity(bullets.get(i));

for (int j = 0; j < chunk.size(); j++) {

if (collision.getCollision(chunk.get(j), bullets.get(i))) {

chunk.remove(j);

bullets.remove(i);

break;

}

}

}

renderer.processEntity(chunk.get(i));

}

TextMaster.render();

DisplayManager.updateDisplay();

}

}

}

**Package fontMeshCreator**

Файл Character.java

package fontMeshCreator;

/\*\*

\* Simple data structure class holding information about a certain glyph in the

\* font texture atlas. All sizes are for a font-size of 1.

\*

\* @author Karl

\*

\*/

public class Character {

private int id;

private double xTextureCoord;

private double yTextureCoord;

private double xMaxTextureCoord;

private double yMaxTextureCoord;

private double xOffset;

private double yOffset;

private double sizeX;

private double sizeY;

private double xAdvance;

/\*\*

\* @param id

\* - the ASCII value of the character.

\* @param xTextureCoord

\* - the x texture coordinate for the top left corner of the

\* character in the texture atlas.

\* @param yTextureCoord

\* - the y texture coordinate for the top left corner of the

\* character in the texture atlas.

\* @param xTexSize

\* - the width of the character in the texture atlas.

\* @param yTexSize

\* - the height of the character in the texture atlas.

\* @param xOffset

\* - the x distance from the curser to the left edge of the

\* character's quad.

\* @param yOffset

\* - the y distance from the curser to the top edge of the

\* character's quad.

\* @param sizeX

\* - the width of the character's quad in screen space.

\* @param sizeY

\* - the height of the character's quad in screen space.

\* @param xAdvance

\* - how far in pixels the cursor should advance after adding

\* this character.

\*/

protected Character(int id, double xTextureCoord, double yTextureCoord, double xTexSize, double yTexSize,

double xOffset, double yOffset, double sizeX, double sizeY, double xAdvance) {

this.id = id;

this.xTextureCoord = xTextureCoord;

this.yTextureCoord = yTextureCoord;

this.xOffset = xOffset;

this.yOffset = yOffset;

this.sizeX = sizeX;

this.sizeY = sizeY;

this.xMaxTextureCoord = xTexSize + xTextureCoord;

this.yMaxTextureCoord = yTexSize + yTextureCoord;

this.xAdvance = xAdvance;

}

protected int getId() {

return id;

}

protected double getxTextureCoord() {

return xTextureCoord;

}

protected double getyTextureCoord() {

return yTextureCoord;

}

protected double getXMaxTextureCoord() {

return xMaxTextureCoord;

}

protected double getYMaxTextureCoord() {

return yMaxTextureCoord;

}

protected double getxOffset() {

return xOffset;

}

protected double getyOffset() {

return yOffset;

}

protected double getSizeX() {

return sizeX;

}

protected double getSizeY() {

return sizeY;

}

protected double getxAdvance() {

return xAdvance;

}

}

Файл FontType.java

package fontMeshCreator;

import java.io.File;

/\*\*

\* Represents a font. It holds the font's texture atlas as well as having the

\* ability to create the quad vertices for any text using this font.

\*

\* @author Karl

\*

\*/

public class FontType {

private int textureAtlas;

private TextMeshCreator loader;

/\*\*

\* Creates a new font and loads up the data about each character from the

\* font file.

\*

\* @param textureAtlas

\* - the ID of the font atlas texture.

\* @param fontFile

\* - the font file containing information about each character in

\* the texture atlas.

\*/

public FontType(int textureAtlas, File fontFile) {

this.textureAtlas = textureAtlas;

this.loader = new TextMeshCreator(fontFile);

}

/\*\*

\* @return The font texture atlas.

\*/

public int getTextureAtlas() {

return textureAtlas;

}

/\*\*

\* Takes in an unloaded text and calculate all of the vertices for the quads

\* on which this text will be rendered. The vertex positions and texture

\* coords and calculated based on the information from the font file.

\*

\* @param text

\* - the unloaded text.

\* @return Information about the vertices of all the quads.

\*/

public TextMeshData loadText(GUIText text) {

return loader.createTextMesh(text);

}

}

Файл GUIText.java

package fontMeshCreator;

import org.lwjgl.util.vector.Vector2f;

import org.lwjgl.util.vector.Vector3f;

import fontRendering.TextMaster;

/\*\*

\* Represents a piece of text in the game.

\*

\* @author Karl

\*

\*/

public class GUIText {

private String textString;

private float fontSize;

private int textMeshVao;

private int vertexCount;

private Vector3f colour = new Vector3f(0f, 0f, 0f);

private Vector2f position;

private float lineMaxSize;

private int numberOfLines;

private FontType font;

private boolean centerText = false;

/\*\*

\* Creates a new text, loads the text's quads into a VAO, and adds the text

\* to the screen.

\*

\* @param text - the text.

\* @param fontSize - the font size of the text, where a font size of 1 is

\* the default size.

\* @param font - the font that this text should use.

\* @param position - the position on the screen where the top left corner of

\* the text should be rendered. The top left corner of the screen is (0, 0)

\* and the bottom right is (1, 1).

\* @param maxLineLength - basically the width of the virtual page in terms

\* of screen width (1 is full screen width, 0.5 is half the width of the

\* screen, etc.) Text cannot go off the edge of the page, so if the text is

\* longer than this length it will go onto the next line. When text is

\* centered it is centered into the middle of the line, based on this line

\* length value.

\* @param centered - whether the text should be centered or not.

\*/

public GUIText(String text, float fontSize, FontType font, Vector2f position, float maxLineLength,

boolean centered) {

this.textString = text;

this.fontSize = fontSize;

this.font = font;

this.position = position;

this.lineMaxSize = maxLineLength;

this.centerText = centered;

TextMaster.loadText(this);;

}

/\*\*

\* Remove the text from the screen.

\*/

public void remove() {

TextMaster.removeText(this);

}

/\*\*

\* @return The font used by this text.

\*/

public FontType getFont() {

return font;

}

/\*\*

\* Set the colour of the text.

\*

\* @param r - red value, between 0 and 1.

\* @param g - green value, between 0 and 1.

\* @param b - blue value, between 0 and 1.

\*/

public void setColour(float r, float g, float b) {

colour.set(r, g, b);

}

/\*\*

\* @return the colour of the text.

\*/

public Vector3f getColour() {

return colour;

}

/\*\*

\* @return The number of lines of text. This is determined when the text is

\* loaded, based on the length of the text and the max line length that is

\* set.

\*/

public int getNumberOfLines() {

return numberOfLines;

}

/\*\*

\* @return The position of the top-left corner of the text in screen-space.

\* (0, 0) is the top left corner of the screen, (1, 1) is the bottom right.

\*/

public Vector2f getPosition() {

return position;

}

/\*\*

\* @return the ID of the text's VAO, which contains all the vertex data for

\* the quads on which the text will be rendered.

\*/

public int getMesh() {

return textMeshVao;

}

/\*\*

\* Set the VAO and vertex count for this text.

\*

\* @param vao - the VAO containing all the vertex data for the quads on

\* which the text will be rendered.

\* @param verticesCount - the total number of vertices in all of the quads.

\*/

public void setMeshInfo(int vao, int verticesCount) {

this.textMeshVao = vao;

this.vertexCount = verticesCount;

}

/\*\*

\* @return The total number of vertices of all the text's quads.

\*/

public int getVertexCount() {

return this.vertexCount;

}

/\*\*

\* @return the font size of the text (a font size of 1 is normal).

\*/

protected float getFontSize() {

return fontSize;

}

/\*\*

\* Sets the number of lines that this text covers (method used only in

\* loading).

\*

\* @param number

\*/

protected void setNumberOfLines(int number) {

this.numberOfLines = number;

}

/\*\*

\* @return {@code true} if the text should be centered.

\*/

protected boolean isCentered() {

return centerText;

}

/\*\*

\* @return The maximum length of a line of this text.

\*/

protected float getMaxLineSize() {

return lineMaxSize;

}

/\*\*

\* @return The string of text.

\*/

protected String getTextString() {

return textString;

}

public void setTextString(String textString) {

this.textString = textString;

}

}

Файл Line.java

package fontMeshCreator;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Represents a line of text during the loading of a text.

\*

\* @author Karl

\*

\*/

public class Line {

private double maxLength;

private double spaceSize;

private List<Word> words = new ArrayList<Word>();

private double currentLineLength = 0;

/\*\*

\* Creates an empty line.

\*

\* @param spaceWidth

\* - the screen-space width of a space character.

\* @param fontSize

\* - the size of font being used.

\* @param maxLength

\* - the screen-space maximum length of a line.

\*/

protected Line(double spaceWidth, double fontSize, double maxLength) {

this.spaceSize = spaceWidth \* fontSize;

this.maxLength = maxLength;

}

/\*\*

\* Attempt to add a word to the line. If the line can fit the word in

\* without reaching the maximum line length then the word is added and the

\* line length increased.

\*

\* @param word

\* - the word to try to add.

\* @return {@code true} if the word has successfully been added to the line.

\*/

protected boolean attemptToAddWord(Word word) {

double additionalLength = word.getWordWidth();

additionalLength += !words.isEmpty() ? spaceSize : 0;

if (currentLineLength + additionalLength <= maxLength) {

words.add(word);

currentLineLength += additionalLength;

return true;

} else {

return false;

}

}

/\*\*

\* @return The max length of the line.

\*/

protected double getMaxLength() {

return maxLength;

}

/\*\*

\* @return The current screen-space length of the line.

\*/

protected double getLineLength() {

return currentLineLength;

}

/\*\*

\* @return The list of words in the line.

\*/

protected List<Word> getWords() {

return words;

}

}

Файл MetaFile.java

package fontMeshCreator;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import org.lwjgl.opengl.Display;

/\*\*

\* Provides functionality for getting the values from a font file.

\*

\* @author Karl

\*

\*/

public class MetaFile {

private static final int PAD\_TOP = 0;

private static final int PAD\_LEFT = 1;

private static final int PAD\_BOTTOM = 2;

private static final int PAD\_RIGHT = 3;

private static final int DESIRED\_PADDING = 3;

private static final String SPLITTER = " ";

private static final String NUMBER\_SEPARATOR = ",";

private double aspectRatio;

private double verticalPerPixelSize;

private double horizontalPerPixelSize;

private double spaceWidth;

private int[] padding;

private int paddingWidth;

private int paddingHeight;

private Map<Integer, Character> metaData = new HashMap<Integer, Character>();

private BufferedReader reader;

private Map<String, String> values = new HashMap<String, String>();

/\*\*

\* Opens a font file in preparation for reading.

\*

\* @param file

\* - the font file.

\*/

protected MetaFile(File file) {

this.aspectRatio = (double) Display.getWidth() / (double) Display.getHeight();

openFile(file);

loadPaddingData();

loadLineSizes();

int imageWidth = getValueOfVariable("scaleW");

loadCharacterData(imageWidth);

close();

}

protected double getSpaceWidth() {

return spaceWidth;

}

protected Character getCharacter(int ascii) {

return metaData.get(ascii);

}

/\*\*

\* Read in the next line and store the variable values.

\*

\* @return {@code true} if the end of the file hasn't been reached.

\*/

private boolean processNextLine() {

values.clear();

String line = null;

try {

line = reader.readLine();

} catch (IOException e1) {

}

if (line == null) {

return false;

}

for (String part : line.split(SPLITTER)) {

String[] valuePairs = part.split("=");

if (valuePairs.length == 2) {

values.put(valuePairs[0], valuePairs[1]);

}

}

return true;

}

/\*\*

\* Gets the {@code int} value of the variable with a certain name on the

\* current line.

\*

\* @param variable

\* - the name of the variable.

\* @return The value of the variable.

\*/

private int getValueOfVariable(String variable) {

return Integer.parseInt(values.get(variable));

}

/\*\*

\* Gets the array of ints associated with a variable on the current line.

\*

\* @param variable

\* - the name of the variable.

\* @return The int array of values associated with the variable.

\*/

private int[] getValuesOfVariable(String variable) {

String[] numbers = values.get(variable).split(NUMBER\_SEPARATOR);

int[] actualValues = new int[numbers.length];

for (int i = 0; i < actualValues.length; i++) {

actualValues[i] = Integer.parseInt(numbers[i]);

}

return actualValues;

}

/\*\*

\* Closes the font file after finishing reading.

\*/

private void close() {

try {

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Opens the font file, ready for reading.

\*

\* @param file

\* - the font file.

\*/

private void openFile(File file) {

try {

reader = new BufferedReader(new FileReader(file));

} catch (Exception e) {

e.printStackTrace();

System.err.println("Couldn't read font meta file!");

}

}

/\*\*

\* Loads the data about how much padding is used around each character in

\* the texture atlas.

\*/

private void loadPaddingData() {

processNextLine();

this.padding = getValuesOfVariable("padding");

this.paddingWidth = padding[PAD\_LEFT] + padding[PAD\_RIGHT];

this.paddingHeight = padding[PAD\_TOP] + padding[PAD\_BOTTOM];

}

/\*\*

\* Loads information about the line height for this font in pixels, and uses

\* this as a way to find the conversion rate between pixels in the texture

\* atlas and screen-space.

\*/

private void loadLineSizes() {

processNextLine();

int lineHeightPixels = getValueOfVariable("lineHeight") - paddingHeight;

verticalPerPixelSize = TextMeshCreator.LINE\_HEIGHT / (double) lineHeightPixels;

horizontalPerPixelSize = verticalPerPixelSize / aspectRatio;

}

/\*\*

\* Loads in data about each character and stores the data in the

\* {@link Character} class.

\*

\* @param imageWidth

\* - the width of the texture atlas in pixels.

\*/

private void loadCharacterData(int imageWidth) {

processNextLine();

processNextLine();

while (processNextLine()) {

Character c = loadCharacter(imageWidth);

if (c != null) {

metaData.put(c.getId(), c);

}

}

}

/\*\*

\* Loads all the data about one character in the texture atlas and converts

\* it all from 'pixels' to 'screen-space' before storing. The effects of

\* padding are also removed from the data.

\*

\* @param imageSize

\* - the size of the texture atlas in pixels.

\* @return The data about the character.

\*/

private Character loadCharacter(int imageSize) {

int id = getValueOfVariable("id");

if (id == TextMeshCreator.SPACE\_ASCII) {

this.spaceWidth = (getValueOfVariable("xadvance") - paddingWidth) \* horizontalPerPixelSize;

return null;

}

double xTex = ((double) getValueOfVariable("x") + (padding[PAD\_LEFT] - DESIRED\_PADDING)) / imageSize;

double yTex = ((double) getValueOfVariable("y") + (padding[PAD\_TOP] - DESIRED\_PADDING)) / imageSize;

int width = getValueOfVariable("width") - (paddingWidth - (2 \* DESIRED\_PADDING));

int height = getValueOfVariable("height") - ((paddingHeight) - (2 \* DESIRED\_PADDING));

double quadWidth = width \* horizontalPerPixelSize;

double quadHeight = height \* verticalPerPixelSize;

double xTexSize = (double) width / imageSize;

double yTexSize = (double) height / imageSize;

double xOff = (getValueOfVariable("xoffset") + padding[PAD\_LEFT] - DESIRED\_PADDING) \* horizontalPerPixelSize;

double yOff = (getValueOfVariable("yoffset") + (padding[PAD\_TOP] - DESIRED\_PADDING)) \* verticalPerPixelSize;

double xAdvance = (getValueOfVariable("xadvance") - paddingWidth) \* horizontalPerPixelSize;

return new Character(id, xTex, yTex, xTexSize, yTexSize, xOff, yOff, quadWidth, quadHeight, xAdvance);

}

}

Файл TextMeshCreator.java

package fontMeshCreator;

import java.io.File;

import java.util.ArrayList;

import java.util.List;

public class TextMeshCreator {

protected static final double LINE\_HEIGHT = 0.03f;

protected static final int SPACE\_ASCII = 32;

private MetaFile metaData;

protected TextMeshCreator(File metaFile) {

metaData = new MetaFile(metaFile);

}

protected TextMeshData createTextMesh(GUIText text) {

List<Line> lines = createStructure(text);

TextMeshData data = createQuadVertices(text, lines);

return data;

}

private List<Line> createStructure(GUIText text) {

char[] chars = text.getTextString().toCharArray();

List<Line> lines = new ArrayList<Line>();

Line currentLine = new Line(metaData.getSpaceWidth(), text.getFontSize(), text.getMaxLineSize());

Word currentWord = new Word(text.getFontSize());

for (char c : chars) {

int ascii = (int) c;

if (ascii == SPACE\_ASCII) {

boolean added = currentLine.attemptToAddWord(currentWord);

if (!added) {

lines.add(currentLine);

currentLine = new Line(metaData.getSpaceWidth(), text.getFontSize(), text.getMaxLineSize());

currentLine.attemptToAddWord(currentWord);

}

currentWord = new Word(text.getFontSize());

continue;

}

Character character = metaData.getCharacter(ascii);

currentWord.addCharacter(character);

}

completeStructure(lines, currentLine, currentWord, text);

return lines;

}

private void completeStructure(List<Line> lines, Line currentLine, Word currentWord, GUIText text) {

boolean added = currentLine.attemptToAddWord(currentWord);

if (!added) {

lines.add(currentLine);

currentLine = new Line(metaData.getSpaceWidth(), text.getFontSize(), text.getMaxLineSize());

currentLine.attemptToAddWord(currentWord);

}

lines.add(currentLine);

}

private TextMeshData createQuadVertices(GUIText text, List<Line> lines) {

text.setNumberOfLines(lines.size());

double curserX = 0f;

double curserY = 0f;

List<Float> vertices = new ArrayList<Float>();

List<Float> textureCoords = new ArrayList<Float>();

for (Line line : lines) {

if (text.isCentered()) {

curserX = (line.getMaxLength() - line.getLineLength()) / 2;

}

for (Word word : line.getWords()) {

for (Character letter : word.getCharacters()) {

addVerticesForCharacter(curserX, curserY, letter, text.getFontSize(), vertices);

addTexCoords(textureCoords, letter.getxTextureCoord(), letter.getyTextureCoord(),

letter.getXMaxTextureCoord(), letter.getYMaxTextureCoord());

curserX += letter.getxAdvance() \* text.getFontSize();

}

curserX += metaData.getSpaceWidth() \* text.getFontSize();

}

curserX = 0;

curserY += LINE\_HEIGHT \* text.getFontSize();

}

return new TextMeshData(listToArray(vertices), listToArray(textureCoords));

}

private void addVerticesForCharacter(double curserX, double curserY, Character character, double fontSize,

List<Float> vertices) {

double x = curserX + (character.getxOffset() \* fontSize);

double y = curserY + (character.getyOffset() \* fontSize);

double maxX = x + (character.getSizeX() \* fontSize);

double maxY = y + (character.getSizeY() \* fontSize);

double properX = (2 \* x) - 1;

double properY = (-2 \* y) + 1;

double properMaxX = (2 \* maxX) - 1;

double properMaxY = (-2 \* maxY) + 1;

addVertices(vertices, properX, properY, properMaxX, properMaxY);

}

private static void addVertices(List<Float> vertices, double x, double y, double maxX, double maxY) {

vertices.add((float) x);

vertices.add((float) y);

vertices.add((float) x);

vertices.add((float) maxY);

vertices.add((float) maxX);

vertices.add((float) maxY);

vertices.add((float) maxX);

vertices.add((float) maxY);

vertices.add((float) maxX);

vertices.add((float) y);

vertices.add((float) x);

vertices.add((float) y);

}

private static void addTexCoords(List<Float> texCoords, double x, double y, double maxX, double maxY) {

texCoords.add((float) x);

texCoords.add((float) y);

texCoords.add((float) x);

texCoords.add((float) maxY);

texCoords.add((float) maxX);

texCoords.add((float) maxY);

texCoords.add((float) maxX);

texCoords.add((float) maxY);

texCoords.add((float) maxX);

texCoords.add((float) y);

texCoords.add((float) x);

texCoords.add((float) y);

}

private static float[] listToArray(List<Float> listOfFloats) {

float[] array = new float[listOfFloats.size()];

for (int i = 0; i < array.length; i++) {

array[i] = listOfFloats.get(i);

}

return array;

}

}

Файл TextMeshData.java

package fontMeshCreator;

/\*\*

\* Stores the vertex data for all the quads on which a text will be rendered.

\* @author Karl

\*

\*/

public class TextMeshData {

private float[] vertexPositions;

private float[] textureCoords;

protected TextMeshData(float[] vertexPositions, float[] textureCoords){

this.vertexPositions = vertexPositions;

this.textureCoords = textureCoords;

}

public float[] getVertexPositions() {

return vertexPositions;

}

public float[] getTextureCoords() {

return textureCoords;

}

public int getVertexCount() {

return vertexPositions.length/2;

}

}

Файл Word.java

package fontMeshCreator;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* During the loading of a text this represents one word in the text.

\* @author Karl

\*

\*/

public class Word {

private List<Character> characters = new ArrayList<Character>();

private double width = 0;

private double fontSize;

/\*\*

\* Create a new empty word.

\* @param fontSize - the font size of the text which this word is in.

\*/

protected Word(double fontSize){

this.fontSize = fontSize;

}

/\*\*

\* Adds a character to the end of the current word and increases the screen-space width of the word.

\* @param character - the character to be added.

\*/

protected void addCharacter(Character character){

characters.add(character);

width += character.getxAdvance() \* fontSize;

}

/\*\*

\* @return The list of characters in the word.

\*/

protected List<Character> getCharacters(){

return characters;

}

/\*\*

\* @return The width of the word in terms of screen size.

\*/

protected double getWordWidth(){

return width;

}

}

**Package fontRendering**

Файл fontFragment.txt

#version 330

in vec2 pass\_textureCoords;

out vec4 out\_colour;

uniform vec3 colour;

uniform sampler2D fontAtlas;

void main(void){

out\_colour = vec4(colour, texture(fontAtlas, pass\_textureCoords).a);

}

Файл fontVertex.txt

#version 330

in vec2 position;

in vec2 textureCoords;

out vec2 pass\_textureCoords;

uniform vec2 translation;

void main(void){

gl\_Position = vec4(position + translation \* vec2(2.0, -2.0), 0.0, 1.0);

pass\_textureCoords = textureCoords;

}

Файл FontRenderer.java

package fontRendering;

import java.util.List;

import java.util.Map;

import org.lwjgl.opengl.GL11;

import org.lwjgl.opengl.GL13;

import org.lwjgl.opengl.GL20;

import org.lwjgl.opengl.GL30;

import fontMeshCreator.FontType;

import fontMeshCreator.GUIText;

public class FontRenderer {

private FontShader shader;

public FontRenderer() {

shader = new FontShader();

}

public void render(Map<FontType, List<GUIText>> texts){

prepare();

for(FontType font : texts.keySet()){

GL13.glActiveTexture(GL13.GL\_TEXTURE0);

GL11.glBindTexture(GL11.GL\_TEXTURE\_2D, font.getTextureAtlas());

for(GUIText text : texts.get(font)){

renderText(text);

}

}

endRendering();

}

public void cleanUp(){

shader.cleanUp();

}

private void prepare(){

GL11.glEnable(GL11.GL\_BLEND);

GL11.glBlendFunc(GL11.GL\_SRC\_ALPHA, GL11.GL\_ONE\_MINUS\_SRC\_ALPHA);

GL11.glDisable(GL11.GL\_DEPTH\_TEST);

shader.start();

}

private void renderText(GUIText text){

GL30.glBindVertexArray(text.getMesh());

GL20.glEnableVertexAttribArray(0);

GL20.glEnableVertexAttribArray(1);

shader.loadColour(text.getColour());

shader.loadTranslation(text.getPosition());

GL11.glDrawArrays(GL11.GL\_TRIANGLES, 0, text.getVertexCount());

GL20.glDisableVertexAttribArray(0);

GL20.glDisableVertexAttribArray(1);

GL30.glBindVertexArray(0);

}

private void endRendering(){

shader.stop();

GL11.glDisable(GL11.GL\_BLEND);

GL11.glEnable(GL11.GL\_DEPTH\_TEST);

}

}

Файл FontShader.java

package fontRendering;

import org.lwjgl.util.vector.Vector2f;

import org.lwjgl.util.vector.Vector3f;

import shaders.ShaderProgram;

public class FontShader extends ShaderProgram{

private static final String VERTEX\_FILE = "src/fontRendering/fontVertex.txt";

private static final String FRAGMENT\_FILE = "src/fontRendering/fontFragment.txt";

private int location\_colour;

private int location\_translation;

public FontShader() {

super(VERTEX\_FILE, FRAGMENT\_FILE);

}

@Override

protected void getAllUniformLocations() {

location\_colour = super.getUniformLocation("colour");

location\_translation = super.getUniformLocation("translation");

}

@Override

protected void bindAttributes() {

super.bindAttribute(0, "position");

super.bindAttribute(1, "textureCoords");

}

protected void loadColour(Vector3f colour){

super.loadVector(location\_colour, colour);

}

protected void loadTranslation(Vector2f translation){

super.load2DVector(location\_translation, translation);

}

}

Файл TextMaster.java

package fontRendering;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import fontMeshCreator.FontType;

import fontMeshCreator.GUIText;

import fontMeshCreator.TextMeshData;

import renderEngine.Loader;

public class TextMaster {

private static Loader loader;

private static Map<FontType, List<GUIText>> texts = new HashMap<FontType, List<GUIText>>();

private static FontRenderer renderer;

public static void init(Loader theLoader){

renderer = new FontRenderer();

loader = theLoader;

}

public static void render(){

renderer.render(texts);

}

public static void loadText(GUIText text){

FontType font = text.getFont();

TextMeshData data = font.loadText(text);

int vao = loader.loadToVAO(data.getVertexPositions(), data.getTextureCoords());

text.setMeshInfo(vao, data.getVertexCount());

List<GUIText> textBatch = texts.get(font);

if(textBatch == null){

textBatch = new ArrayList<GUIText>();

texts.put(font, textBatch);

}

textBatch.add(text);

}

public static void removeText(GUIText text){

List<GUIText> textBatch = texts.get(text.getFont());

textBatch.remove(text);

if(textBatch.isEmpty()){

texts.remove(texts.get(text.getFont()));

}

}

public static void cleanUp(){

renderer.cleanUp();

}

}

**Package guis**

Файл guiFragmentShader.txt

#version 140

in vec2 textureCoords;

out vec4 out\_Color;

uniform sampler2D guiTexture;

void main(void){

out\_Color = texture(guiTexture,textureCoords);

}

Файл guiVertexShader.txt

#version 140

in vec2 position;

out vec2 textureCoords;

uniform mat4 transformationMatrix;

void main(void){

gl\_Position = transformationMatrix \* vec4(position, 0.0, 1.0);

textureCoords = vec2((position.x+1.0)/2.0, 1 - (position.y+1.0)/2.0);

}

Файл GuiRenderer.java

package guis;

import java.util.List;

import models.RawModel;

import org.lwjgl.opengl.GL11;

import org.lwjgl.opengl.GL13;

import org.lwjgl.opengl.GL20;

import org.lwjgl.opengl.GL30;

import org.lwjgl.util.vector.Matrix4f;

import renderEngine.Loader;

import toolbox.Maths;

public class GuiRenderer {

private final RawModel quad;

private GuiShader shader;

public GuiRenderer(Loader loader) {

float[] positions = {-1, 1, -1, -1, 1, 1, 1, -1};

quad = loader.loadToVAO(positions, 2);

shader = new GuiShader();

}

public void render(List<GuiTexture> guis){

shader.start();

GL30.glBindVertexArray(quad.getVaoID());

GL20.glEnableVertexAttribArray(0);

GL11.glEnable(GL11.GL\_BLEND);

GL11.glBlendFunc(GL11.GL\_SRC\_ALPHA, GL11.GL\_ONE\_MINUS\_SRC\_ALPHA);

GL11.glDisable(GL11.GL\_DEPTH\_TEST);

for(GuiTexture gui: guis){

GL13.glActiveTexture(GL13.GL\_TEXTURE0);

GL11.glBindTexture(GL11.GL\_TEXTURE\_2D, gui.getTexture());

Matrix4f matrix = Maths.createTransformationMatrix(gui.getPosition(), gui.getScale());

shader.loadTransformation(matrix);

GL11.glDrawArrays(GL11.GL\_TRIANGLE\_STRIP, 0, quad.getVertexCount());

}

GL11.glEnable(GL11.GL\_DEPTH\_TEST);

GL11.glDisable(GL11.GL\_BLEND);

GL20.glDisableVertexAttribArray(0);

GL30.glBindVertexArray(0);

}

public void cleanUp(){

shader.cleanUp();

}

}

Файл GuiShader.java

package guis;

import org.lwjgl.util.vector.Matrix4f;

import shaders.ShaderProgram;

public class GuiShader extends ShaderProgram{

private static final String VERTEX\_FILE = "src/guis/guiVertexShader.txt";

private static final String FRAGMENT\_FILE = "src/guis/guiFragmentShader.txt";

private int location\_transformationMatrix;

public GuiShader() {

super(VERTEX\_FILE, FRAGMENT\_FILE);

}

public void loadTransformation(Matrix4f matrix){

super.loadMatrix(location\_transformationMatrix, matrix);

}

@Override

protected void getAllUniformLocations() {

location\_transformationMatrix = super.getUniformLocation("transformationMatrix");

}

@Override

protected void bindAttributes() {

super.bindAttribute(0, "position");

}

}

Файл GuiTexture.java

package guis;

import org.lwjgl.util.vector.Vector2f;

public class GuiTexture {

private int texture;

private Vector2f position;

private Vector2f scale;

public GuiTexture(int texture, Vector2f position, Vector2f scale) {

this.texture = texture;

this.position = position;

this.scale = scale;

}

public void setTexture(int texture) {

this.texture = texture;

}

public void setScale(Vector2f scale) {

this.scale = scale;

}

public void increasePosition(float dx, float dy) {

this.position.x += dx;

this.position.y += dy;

}

public void setPosition(Vector2f position) {

this.position = position;

}

public int getTexture() {

return texture;

}

public Vector2f getPosition() {

return position;

}

public Vector2f getScale() {

return scale;

}

}

**Package models**

Файл RawModel.java

package models;

public class RawModel {

private int vaoID;

private int vertexCount;

public RawModel(int vaoID, int vertexCount){

this.vaoID = vaoID;

this.vertexCount = vertexCount;

}

public int getVaoID() {

return vaoID;

}

public int getVertexCount() {

return vertexCount;

}

}

Файл TexturedModel.java

package models;

import textures.ModelTexture;

public class TexturedModel {

private RawModel rawModel;

private ModelTexture texture;

public TexturedModel(RawModel model, ModelTexture texture){

this.rawModel = model;

this.texture = texture;

}

public RawModel getRawModel() {

return rawModel;

}

public ModelTexture getTexture() {

return texture;

}

}

Package multiplayer

Файл DataThread.java

package Multiplayer;

import entities.Bullet;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import org.lwjgl.util.vector.Vector3f;

public class DataThread extends Thread {

ObjectInputStream in;

ObjectOutputStream out;

Multiplayer world;

public DataThread(Multiplayer world, ObjectOutputStream out, ObjectInputStream in) {

this.in = in;

this.out = out;

this.world = world;

}

@Override

public void run() {

while (true) {

try {

if (!isInterrupted()) {

out.writeObject("" + world.players.get(0).getPosition().x);

out.writeObject("" + world.players.get(0).getPosition().y);

out.writeObject("" + world.players.get(0).getPosition().z);

out.writeObject("" + world.players.get(0).getRotX());

out.writeObject("" + world.players.get(0).getRotY());

out.writeObject("" + world.players.get(0).getRotZ());

out.writeObject("" + world.players.get(0).getXP());

out.writeObject("" + world.bulletCreate);

float x = Float.parseFloat((String) in.readObject());

float y = Float.parseFloat((String) in.readObject());

float z = Float.parseFloat((String) in.readObject());

world.players.get(1).setRotX(Float.parseFloat((String) in.readObject()));

world.players.get(1).setRotY(Float.parseFloat((String) in.readObject()));

world.players.get(1).setRotZ(Float.parseFloat((String) in.readObject()));

world.players.get(1).setXP(Integer.parseInt((String) in.readObject()));

world.players.get(1).setPosition(new Vector3f(x, y, z));

String s = (String) in.readObject();

if (Boolean.parseBoolean(s)) {

world.bullets.add(new Bullet(world, world.players.get(1)));

}

} else {

throw new InterruptedException();

}

} catch (IOException | NumberFormatException | ClassNotFoundException | InterruptedException e) {

}

}

}

}

Файл Multiplayer.java

**package Multiplayer;**

import entities.Player;

import entities.World;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.InetAddress;

import java.net.ServerSocket;

import java.net.Socket;

import models.TexturedModel;

import org.lwjgl.util.vector.Vector3f;

import renderEngine.OBJLoader;

import textures.ModelTexture;

public class Multiplayer extends World {

static ObjectOutputStream out;

static ObjectInputStream in;

static ServerSocket server;

static Socket client;

static Socket socket;

static String IP;

static int port = 25566;

static DataThread dataThread;

static DataThreadSend dataThreadSend;

public Multiplayer(boolean host, String IP) {

this.IP = IP;

worldInit();

TexturedModel playerShip = new TexturedModel(OBJLoader.loadObjModel("space\_ship\_1", loader),

new ModelTexture(loader.loadTexture("space\_ship\_1")));

ModelTexture playerShipTexture = playerShip.getTexture();

playerShipTexture.setShineDamper(10);

playerShipTexture.setReflectivity(1);

players.add(new Player(playerShip, new Vector3f(0, 0, 0), 0, 0, 0, 1, players.size()));

if (host) {

try {

server = new ServerSocket(port, 4, InetAddress.getByName(IP));

client = server.accept();

out = new ObjectOutputStream(client.getOutputStream());

in = new ObjectInputStream(client.getInputStream());

dataThread = new DataThread(Multiplayer.this, out, in);

dataThread.start();

} catch (IOException e) {

}

} else {

try {

socket = new Socket(IP, port);

in = new ObjectInputStream(socket.getInputStream());

out = new ObjectOutputStream(socket.getOutputStream());

dataThread = new DataThread(Multiplayer.this, out, in);

dataThread.start();

} catch (IOException e) {

}

}

worldLoopRenderer();

worldCleanUp();

dataThread.interrupt();

dataThread.stop();

}

}

**Package renderEngine**

Файл DisplayManager.java

package renderEngine;

import org.lwjgl.LWJGLException;

import org.lwjgl.Sys;

import org.lwjgl.input.Mouse;

import org.lwjgl.opengl.ContextAttribs;

import org.lwjgl.opengl.Display;

import org.lwjgl.opengl.DisplayMode;

import org.lwjgl.opengl.PixelFormat;

public class DisplayManager {

private static final int FPS\_CAP = 120;

private static long lastFrameTime;

private static float delta;

public static void createDisplay() {

ContextAttribs attribs = new ContextAttribs(3, 2)

.withForwardCompatible(true)

.withProfileCore(true);

try {

Mouse.setGrabbed(true);

Display.setFullscreen(true);

DisplayMode mode = Display.getDesktopDisplayMode();

Display.setDisplayMode(mode);

Display.create(new PixelFormat(), attribs);

Display.setTitle("Course Project");

} catch (LWJGLException e) {

e.printStackTrace();

}

lastFrameTime = getCurrentTime();

}

public static void updateDisplay() {

Display.sync(FPS\_CAP);

Display.update();

long currentFrameTime = getCurrentTime();

delta = (currentFrameTime - lastFrameTime)/500f;

lastFrameTime = currentFrameTime;

}

public static float getFarmeTimeSeconds(){

return delta;

}

public static void closeDisplay() {

Display.destroy();

}

private static long getCurrentTime(){

return Sys.getTime()\*1000/Sys.getTimerResolution();

}

}

Файл EntityRenderer.java

**package renderEngine;**

import models.RawModel;

import models.TexturedModel;

import org.lwjgl.opengl.GL11;

import org.lwjgl.opengl.GL13;

import org.lwjgl.opengl.GL20;

import org.lwjgl.opengl.GL30;

import org.lwjgl.util.vector.Matrix4f;

import shaders.StaticShader;

import toolbox.Maths;

import entities.Entity;

import java.util.List;

import java.util.Map;

import textures.ModelTexture;

public class EntityRenderer {

private StaticShader shader;

public EntityRenderer(StaticShader shader, Matrix4f projectionMatrix) {

this.shader = shader;

shader.start();

shader.loadProjectionMatrix(projectionMatrix);

shader.stop();

}

public void render(Map<TexturedModel, List<Entity>> entities) {

for (TexturedModel model : entities.keySet()) {

prepareTexturedModel(model);

List<Entity> batch = entities.get(model);

for (Entity entity : batch) {

prepareInstance(entity);

GL11.glDrawElements(GL11.GL\_TRIANGLES,

model.getRawModel().getVertexCount(), GL11.GL\_UNSIGNED\_INT, 0);

}

unbindTexturedModel();

}

}

private void prepareTexturedModel(TexturedModel model) {

RawModel rawModel = model.getRawModel();

GL30.glBindVertexArray(rawModel.getVaoID());

GL20.glEnableVertexAttribArray(0);

GL20.glEnableVertexAttribArray(1);

GL20.glEnableVertexAttribArray(2);

ModelTexture texture = model.getTexture();

if(texture.isHasTransparency()){

MasterRenderer.disableCulling();

}

shader.loadFakeLightingVariable(texture.isUseFakeLighting());

shader.loadShineVariables(texture.getShineDamper(), texture.getReflectivity());

GL13.glActiveTexture(GL13.GL\_TEXTURE0);

GL11.glBindTexture(GL11.GL\_TEXTURE\_2D, model.getTexture().getID());

}

private void unbindTexturedModel() {

MasterRenderer.enableCulling();

GL20.glDisableVertexAttribArray(0);

GL20.glDisableVertexAttribArray(1);

GL20.glDisableVertexAttribArray(2);

GL30.glBindVertexArray(0);

}

private void prepareInstance(Entity entity) {

Matrix4f transformationMatrix = Maths.createTransformationMatrix(entity.getPosition(),

entity.getRotX(), entity.getRotY(), entity.getRotZ(), entity.getScale());

shader.loadTransformationMatrix(transformationMatrix);

}

}

Файл Loader.java

package renderEngine;

import java.io.FileInputStream;

import java.nio.ByteBuffer;

import java.nio.FloatBuffer;

import java.nio.IntBuffer;

import java.util.ArrayList;

import java.util.List;

import models.RawModel;

import org.lwjgl.BufferUtils;

import org.lwjgl.opengl.GL11;

import org.lwjgl.opengl.GL12;

import org.lwjgl.opengl.GL13;

import org.lwjgl.opengl.GL14;

import org.lwjgl.opengl.GL15;

import org.lwjgl.opengl.GL20;

import org.lwjgl.opengl.GL30;

import org.newdawn.slick.opengl.Texture;

import org.newdawn.slick.opengl.TextureLoader;

import textures.TextureData;

import de.matthiasmann.twl.utils.PNGDecoder;

import de.matthiasmann.twl.utils.PNGDecoder.Format;

public class Loader {

private List<Integer> vaos = new ArrayList<Integer>();

private List<Integer> vbos = new ArrayList<Integer>();

private List<Integer> textures = new ArrayList<Integer>();

public int loadToVAO(float[] positions, float[] textureCoords) {

int vaoID = createVAO();

storeDataInAttributeList(0, 2, positions);

storeDataInAttributeList(1, 2, textureCoords);

unbindVAO();

return vaoID;

}

public RawModel loadToVAO(float[] positions, float[] textureCoords, float[] normals,

int[] indices) {

int vaoID = createVAO();

bindIndicesBuffer(indices);

storeDataInAttributeList(0, 3, positions);

storeDataInAttributeList(1, 2, textureCoords);

storeDataInAttributeList(2, 3, normals);

unbindVAO();

return new RawModel(vaoID, indices.length);

}

public RawModel loadToVAO(float[] positions, float[] textureCoords, float[] normals, float[] tangents,

int[] indices) {

int vaoID = createVAO();

bindIndicesBuffer(indices);

storeDataInAttributeList(0, 3, positions);

storeDataInAttributeList(1, 2, textureCoords);

storeDataInAttributeList(2, 3, normals);

storeDataInAttributeList(3, 3, tangents);

unbindVAO();

return new RawModel(vaoID, indices.length);

}

public RawModel loadToVAO(float[] positions, int dimensions) {

int vaoID = createVAO();

this.storeDataInAttributeList(0, dimensions, positions);

unbindVAO();

return new RawModel(vaoID, positions.length / dimensions);

}

public int loadTexture(String fileName) {

Texture texture = null;

try {

texture = TextureLoader.getTexture("PNG", new FileInputStream("res/" + fileName

+ ".png"));

GL30.glGenerateMipmap(GL11.GL\_TEXTURE\_2D);

GL11.glTexParameteri(GL11.GL\_TEXTURE\_2D, GL11.GL\_TEXTURE\_MIN\_FILTER,

GL11.GL\_LINEAR\_MIPMAP\_LINEAR);

GL11.glTexParameterf(GL11.GL\_TEXTURE\_2D, GL14.GL\_TEXTURE\_LOD\_BIAS, 0f);

} catch (Exception e) {

e.printStackTrace();

System.err.println("Tried to load texture " + fileName + ".png , didn't work");

System.exit(-1);

}

textures.add(texture.getTextureID());

return texture.getTextureID();

}

public void cleanUp() {

for (int vao : vaos) {

GL30.glDeleteVertexArrays(vao);

}

for (int vbo : vbos) {

GL15.glDeleteBuffers(vbo);

}

for (int texture : textures) {

GL11.glDeleteTextures(texture);

}

}

public int loadCubeMap(String[] textureFiles) {

int texID = GL11.glGenTextures();

GL13.glActiveTexture(GL13.GL\_TEXTURE0);

GL11.glBindTexture(GL13.GL\_TEXTURE\_CUBE\_MAP, texID);

for (int i = 0; i < textureFiles.length; i++) {

TextureData data = decodeTextureFile("res/" + textureFiles[i] + ".png");

GL11.glTexImage2D(GL13.GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, 0, GL11.GL\_RGBA, data.getWidth(), data.getHeight(), 0,

GL11.GL\_RGBA, GL11.GL\_UNSIGNED\_BYTE, data.getBuffer());

}

GL11.glTexParameteri(GL13.GL\_TEXTURE\_CUBE\_MAP, GL11.GL\_TEXTURE\_MAG\_FILTER, GL11.GL\_LINEAR);

GL11.glTexParameteri(GL13.GL\_TEXTURE\_CUBE\_MAP, GL11.GL\_TEXTURE\_MIN\_FILTER, GL11.GL\_LINEAR);

GL11.glTexParameteri(GL13.GL\_TEXTURE\_CUBE\_MAP, GL11.GL\_TEXTURE\_WRAP\_S, GL12.GL\_CLAMP\_TO\_EDGE);

GL11.glTexParameteri(GL13.GL\_TEXTURE\_CUBE\_MAP, GL11.GL\_TEXTURE\_WRAP\_T, GL12.GL\_CLAMP\_TO\_EDGE);

textures.add(texID);

return texID;

}

private TextureData decodeTextureFile(String fileName) {

int width = 0;

int height = 0;

ByteBuffer buffer = null;

try {

FileInputStream in = new FileInputStream(fileName);

PNGDecoder decoder = new PNGDecoder(in);

width = decoder.getWidth();

height = decoder.getHeight();

buffer = ByteBuffer.allocateDirect(4 \* width \* height);

decoder.decode(buffer, width \* 4, Format.RGBA);

buffer.flip();

in.close();

} catch (Exception e) {

e.printStackTrace();

System.err.println("Tried to load texture " + fileName + ", didn't work");

System.exit(-1);

}

return new TextureData(buffer, width, height);

}

private int createVAO() {

int vaoID = GL30.glGenVertexArrays();

vaos.add(vaoID);

GL30.glBindVertexArray(vaoID);

return vaoID;

}

private void storeDataInAttributeList(int attributeNumber, int coordinateSize, float[] data) {

int vboID = GL15.glGenBuffers();

vbos.add(vboID);

GL15.glBindBuffer(GL15.GL\_ARRAY\_BUFFER, vboID);

FloatBuffer buffer = storeDataInFloatBuffer(data);

GL15.glBufferData(GL15.GL\_ARRAY\_BUFFER, buffer, GL15.GL\_STATIC\_DRAW);

GL20.glVertexAttribPointer(attributeNumber, coordinateSize, GL11.GL\_FLOAT, false, 0, 0);

GL15.glBindBuffer(GL15.GL\_ARRAY\_BUFFER, 0);

}

private void unbindVAO() {

GL30.glBindVertexArray(0);

}

private void bindIndicesBuffer(int[] indices) {

int vboID = GL15.glGenBuffers();

vbos.add(vboID);

GL15.glBindBuffer(GL15.GL\_ELEMENT\_ARRAY\_BUFFER, vboID);

IntBuffer buffer = storeDataInIntBuffer(indices);

GL15.glBufferData(GL15.GL\_ELEMENT\_ARRAY\_BUFFER, buffer, GL15.GL\_STATIC\_DRAW);

}

private IntBuffer storeDataInIntBuffer(int[] data) {

IntBuffer buffer = BufferUtils.createIntBuffer(data.length);

buffer.put(data);

buffer.flip();

return buffer;

}

private FloatBuffer storeDataInFloatBuffer(float[] data) {

FloatBuffer buffer = BufferUtils.createFloatBuffer(data.length);

buffer.put(data);

buffer.flip();

return buffer;

}

}

Файл MasterRenderer.java

package renderEngine;

import entities.Camera;

import entities.Entity;

import entities.Light;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import models.TexturedModel;

import org.lwjgl.opengl.Display;

import org.lwjgl.opengl.GL11;

import org.lwjgl.util.vector.Matrix4f;

import shaders.StaticShader;

import skybox.SkyboxRenderer;

public class MasterRenderer {

private static final float FOV = 70;

private static final float NEAR\_PLANE = 0.1f;

private static final float FAR\_PLANE = 1000;

private static final float RED = 0f;

private static final float GREEN = 0f;

private static final float BLUE = 0f;

private Matrix4f projectionMatrix;

private StaticShader shader = new StaticShader();

private EntityRenderer renderer;

private Map<TexturedModel, List<Entity>> entities = new HashMap<TexturedModel, List<Entity>>();

private SkyboxRenderer skyboxRenderer;

public MasterRenderer(Loader loader) {

enableCulling();

createProjectionMatrix();

renderer = new EntityRenderer(shader, projectionMatrix);

skyboxRenderer = new SkyboxRenderer(loader, projectionMatrix);

}

public static void enableCulling(){

GL11.glEnable(GL11.GL\_CULL\_FACE);

GL11.glCullFace(GL11.GL\_BACK);

}

public static void disableCulling(){

GL11.glDisable(GL11.GL\_CULL\_FACE);

}

public void render(List<Light> lights, Camera camera) {

prepare();

shader.start();

shader.loadSkyColour(RED, GREEN, BLUE);

shader.loadLights(lights);

shader.loadViewMatrix(camera);

renderer.render(entities);

shader.stop();

skyboxRenderer.render(camera);

entities.clear();

}

public void processEntity(Entity entity) {

TexturedModel entityModel = entity.getModel();

List<Entity> batch = entities.get(entityModel);

if (batch != null) {

batch.add(entity);

} else {

List<Entity> newBatch = new ArrayList<Entity>();

newBatch.add(entity);

entities.put(entityModel, newBatch);

}

}

public void cleanUp() {

shader.cleanUp();

}

public void prepare() {

GL11.glEnable(GL11.GL\_DEPTH\_TEST);

GL11.glClear(GL11.GL\_COLOR\_BUFFER\_BIT | GL11.GL\_DEPTH\_BUFFER\_BIT);

GL11.glClearColor(RED, GREEN, BLUE, 0);

}

private void createProjectionMatrix() {

float aspectRatio = (float) Display.getWidth() / (float) Display.getHeight();

float y\_scale = (float) ((1f / Math.tan(Math.toRadians(FOV / 2f))) \* aspectRatio);

float x\_scale = y\_scale / aspectRatio;

float frustum\_length = FAR\_PLANE - NEAR\_PLANE;

projectionMatrix = new Matrix4f();

projectionMatrix.m00 = x\_scale;

projectionMatrix.m11 = y\_scale;

projectionMatrix.m22 = -((FAR\_PLANE + NEAR\_PLANE) / frustum\_length);

projectionMatrix.m23 = -1;

projectionMatrix.m32 = -((2 \* NEAR\_PLANE \* FAR\_PLANE) / frustum\_length);

projectionMatrix.m33 = 0;

}

}

Файл OBJLoader.java

package renderEngine;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.util.ArrayList;

import java.util.List;

import models.RawModel;

import org.lwjgl.util.vector.Vector2f;

import org.lwjgl.util.vector.Vector3f;

public class OBJLoader {

public static RawModel loadObjModel(String fileName, Loader loader) {

FileReader fr = null;

try {

fr = new FileReader(new File("res/" + fileName + ".obj"));

} catch (FileNotFoundException e) {

System.err.println("Couldn't load file!");

e.printStackTrace();

}

BufferedReader reader = new BufferedReader(fr);

String line;

List<Vector3f> vertices = new ArrayList<Vector3f>();

List<Vector2f> textures = new ArrayList<Vector2f>();

List<Vector3f> normals = new ArrayList<Vector3f>();

List<Integer> indices = new ArrayList<Integer>();

float[] verticesArray = null;

float[] normalsArray = null;

float[] textureArray = null;

int[] indicesArray = null;

try {

while (true) {

line = reader.readLine();

String[] currentLine = line.split(" ");

if (line.startsWith("v ")) {

Vector3f vertex = new Vector3f(Float.parseFloat(currentLine[1]),

Float.parseFloat(currentLine[2]), Float.parseFloat(currentLine[3]));

vertices.add(vertex);

} else if (line.startsWith("vt ")) {

Vector2f texture = new Vector2f(Float.parseFloat(currentLine[1]),

Float.parseFloat(currentLine[2]));

textures.add(texture);

} else if (line.startsWith("vn ")) {

Vector3f normal = new Vector3f(Float.parseFloat(currentLine[1]),

Float.parseFloat(currentLine[2]), Float.parseFloat(currentLine[3]));

normals.add(normal);

} else if (line.startsWith("f ")) {

textureArray = new float[vertices.size() \* 2];

normalsArray = new float[vertices.size() \* 3];

break;

}

}

while (line != null) {

if (!line.startsWith("f ")) {

line = reader.readLine();

continue;

}

String[] currentLine = line.split(" ");

String[] vertex1 = currentLine[1].split("/");

String[] vertex2 = currentLine[2].split("/");

String[] vertex3 = currentLine[3].split("/");

processVertex(vertex1,indices,textures,normals,textureArray,normalsArray);

processVertex(vertex2,indices,textures,normals,textureArray,normalsArray);

processVertex(vertex3,indices,textures,normals,textureArray,normalsArray);

line = reader.readLine();

}

reader.close();

} catch (Exception e) {

e.printStackTrace();

}

verticesArray = new float[vertices.size()\*3];

indicesArray = new int[indices.size()];

int vertexPointer = 0;

for(Vector3f vertex:vertices){

verticesArray[vertexPointer++] = vertex.x;

verticesArray[vertexPointer++] = vertex.y;

verticesArray[vertexPointer++] = vertex.z;

}

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

indicesArray[i] = indices.get(i);

}

return loader.loadToVAO(verticesArray, textureArray, normalsArray, indicesArray);

}

private static void processVertex(String[] vertexData, List<Integer> indices,

List<Vector2f> textures, List<Vector3f> normals, float[] textureArray,

float[] normalsArray) {

int currentVertexPointer = Integer.parseInt(vertexData[0]) - 1;

indices.add(currentVertexPointer);

Vector2f currentTex = textures.get(Integer.parseInt(vertexData[1])-1);

textureArray[currentVertexPointer\*2] = currentTex.x;

textureArray[currentVertexPointer\*2+1] = 1 - currentTex.y;

Vector3f currentNorm = normals.get(Integer.parseInt(vertexData[2])-1);

normalsArray[currentVertexPointer\*3] = currentNorm.x;

normalsArray[currentVertexPointer\*3+1] = currentNorm.y;

normalsArray[currentVertexPointer\*3+2] = currentNorm.z;

}

}

**Package shaders**

Файл fragmentShader.txt

#version 400 core

in vec2 pass\_textureCoordinates;

in vec3 surfaceNormal;

in vec3 toLightVector[4];

in vec3 toCameraVector;

in float visibility;

out vec4 out\_Color;

uniform sampler2D modelTexture;

uniform vec3 lightColour[4];

uniform vec3 attenuation[4];

uniform float shineDamper;

uniform float reflectivity;

uniform vec3 skyColour;

const float levels = 10.0;

void main(void){

vec3 unitNormal = normalize(surfaceNormal);

vec3 unitVectorToCamera = normalize(toCameraVector);

vec3 totalDiffuse = vec3(0.0);

vec3 totalSpecular = vec3(0.0);

for(int i = 0; i<4; i++){

float distance = length(toLightVector[i]);

float attFactor = attenuation[i].x + (attenuation[i].y \* distance) + (attenuation[i].z \* distance \* distance);

vec3 unitLightVector = normalize(toLightVector[i]);

float nDotl = dot(unitNormal, unitLightVector);

float brightness = max(nDotl, 0.0);

float level = floor(brightness \* levels);

brightness = level / levels;

vec3 lightDirection = -unitLightVector;

vec3 reflectedLightDirection = reflect(lightDirection, unitNormal);

float specularFactor = dot(reflectedLightDirection, unitVectorToCamera);

specularFactor = max(specularFactor, 0.0);

float dampedFactor = pow(specularFactor, shineDamper);

level = floor(dampedFactor \* levels);

dampedFactor = level / levels;

totalDiffuse = totalDiffuse + (brightness \* lightColour[i]) / attFactor;

totalSpecular = totalSpecular + (dampedFactor \* reflectivity \* lightColour[i]) / attFactor;

}

totalDiffuse = max(totalDiffuse, 0.1);

vec4 textureColour = texture(modelTexture,pass\_textureCoordinates);

if(textureColour.a < 0.5){

discard;

}

out\_Color = vec4(totalDiffuse, 1.0) \* textureColour + vec4(totalSpecular, 1.0);

out\_Color = mix(vec4(skyColour, 1.0), out\_Color, visibility);

}

Файл vertexShader.txt

#version 400 core

in vec3 position;

in vec2 textureCoordinates;

in vec3 normal;

out vec2 pass\_textureCoordinates;

out vec3 surfaceNormal;

out vec3 toLightVector[4];

out vec3 toCameraVector;

out float visibility;

uniform mat4 transformationMatrix;

uniform mat4 projectionMatrix;

uniform mat4 viewMatrix;

uniform vec3 lightPosition[4];

uniform float useFakeLighting;

const float density = 0.0035;

const float gradient = 0.5;

void main(void){

vec4 worldPosition = transformationMatrix \* vec4(position,1.0);

vec4 positionRelativeToCam = viewMatrix \* worldPosition;

gl\_Position = projectionMatrix \* positionRelativeToCam;

pass\_textureCoordinates = textureCoordinates;

vec3 actualNormal = normal;

if(useFakeLighting > 0.5){

actualNormal = vec3(0.0, 1.0, 0.0);

}

surfaceNormal = (transformationMatrix \* vec4(normal, 0.0)).xyz;

for(int i = 0; i < 4; i++){

toLightVector[i] = lightPosition[i] - worldPosition.xyz;

}

toCameraVector = (inverse(viewMatrix)\*vec4(0.0,0.0,0.0,1.0)).xyz - worldPosition.xyz;

float distance = length(positionRelativeToCam.xyz);

visibility = exp(-pow((distance\*density), gradient));

visibility = clamp(visibility, 0.0, 1.0);

}

Файл ShaderProgram.java

package shaders;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.nio.FloatBuffer;

import org.lwjgl.BufferUtils;

import org.lwjgl.opengl.GL11;

import org.lwjgl.opengl.GL20;

import org.lwjgl.util.vector.Matrix4f;

import org.lwjgl.util.vector.Vector2f;

import org.lwjgl.util.vector.Vector3f;

import org.lwjgl.util.vector.Vector4f;

public abstract class ShaderProgram {

private int programID;

private int vertexShaderID;

private int fragmentShaderID;

private static FloatBuffer matrixBuffer = BufferUtils.createFloatBuffer(16);

public ShaderProgram(String vertexFile,String fragmentFile){

vertexShaderID = loadShader(vertexFile,GL20.GL\_VERTEX\_SHADER);

fragmentShaderID = loadShader(fragmentFile,GL20.GL\_FRAGMENT\_SHADER);

programID = GL20.glCreateProgram();

GL20.glAttachShader(programID, vertexShaderID);

GL20.glAttachShader(programID, fragmentShaderID);

bindAttributes();

GL20.glLinkProgram(programID);

GL20.glValidateProgram(programID);

getAllUniformLocations();

}

protected abstract void getAllUniformLocations();

protected int getUniformLocation(String uniformName){

return GL20.glGetUniformLocation(programID,uniformName);

}

public void start(){

GL20.glUseProgram(programID);

}

public void stop(){

GL20.glUseProgram(0);

}

public void cleanUp(){

stop();

GL20.glDetachShader(programID, vertexShaderID);

GL20.glDetachShader(programID, fragmentShaderID);

GL20.glDeleteShader(vertexShaderID);

GL20.glDeleteShader(fragmentShaderID);

GL20.glDeleteProgram(programID);

}

protected abstract void bindAttributes();

protected void bindAttribute(int attribute, String variableName){

GL20.glBindAttribLocation(programID, attribute, variableName);

}

protected void loadFloat(int location, float value){

GL20.glUniform1f(location, value);

}

protected void loadInt(int location, int value){

GL20.glUniform1i(location, value);

}

protected void loadVector(int location, Vector3f vector){

GL20.glUniform3f(location,vector.x,vector.y,vector.z);

}

protected void loadVector(int location, Vector4f vector){

GL20.glUniform4f(location,vector.x,vector.y,vector.z, vector.w);

}

protected void load2DVector(int location, Vector2f vector){

GL20.glUniform2f(location,vector.x,vector.y);

}

protected void loadBoolean(int location, boolean value){

float toLoad = 0;

if(value){

toLoad = 1;

}

GL20.glUniform1f(location, toLoad);

}

protected void loadMatrix(int location, Matrix4f matrix){

matrix.store(matrixBuffer);

matrixBuffer.flip();

GL20.glUniformMatrix4(location, false, matrixBuffer);

}

private static int loadShader(String file, int type){

StringBuilder shaderSource = new StringBuilder();

try{

BufferedReader reader = new BufferedReader(new FileReader(file));

String line;

while((line = reader.readLine())!=null){

shaderSource.append(line).append("//\n");

}

reader.close();

}catch(IOException e){

e.printStackTrace();

System.exit(-1);

}

int shaderID = GL20.glCreateShader(type);

GL20.glShaderSource(shaderID, shaderSource);

GL20.glCompileShader(shaderID);

if(GL20.glGetShaderi(shaderID, GL20.GL\_COMPILE\_STATUS )== GL11.GL\_FALSE){

System.out.println(GL20.glGetShaderInfoLog(shaderID, 500));

System.err.println("Could not compile shader!");

System.exit(-1);

}

return shaderID;

}

}

Файл StaticShader.java

package shaders;

import org.lwjgl.util.vector.Matrix4f;

import toolbox.Maths;

import entities.Camera;

import entities.Light;

import java.util.List;

import org.lwjgl.util.vector.Vector3f;

public class StaticShader extends ShaderProgram {

private static final int MAX\_LIGHTS = 4;

private static final String VERTEX\_FILE = "src/shaders/vertexShader.txt";

private static final String FRAGMENT\_FILE = "src/shaders/fragmentShader.txt";

private int location\_transformationMatrix;

private int location\_projectionMatrix;

private int location\_viewMatrix;

private int location\_lightPosition[];

private int location\_lightColour[];

private int location\_attenuation[];

private int location\_shineDamper;

private int location\_reflectivity;

private int location\_skyColour;

private int location\_useFakeLighting;

public StaticShader() {

super(VERTEX\_FILE, FRAGMENT\_FILE);

}

@Override

protected void bindAttributes() {

super.bindAttribute(0, "position");

super.bindAttribute(1, "textureCoordinates");

super.bindAttribute(2, "normal");

}

@Override

protected void getAllUniformLocations() {

location\_transformationMatrix = super.getUniformLocation("transformationMatrix");

location\_projectionMatrix = super.getUniformLocation("projectionMatrix");

location\_viewMatrix = super.getUniformLocation("viewMatrix");

location\_shineDamper = super.getUniformLocation("shineDamper");

location\_reflectivity = super.getUniformLocation("reflectivity");

location\_skyColour = super.getUniformLocation("skyColour");

location\_useFakeLighting = super.getUniformLocation("useFakeLighting");

location\_lightPosition = new int[MAX\_LIGHTS];

location\_lightColour = new int[MAX\_LIGHTS];

location\_attenuation = new int[MAX\_LIGHTS];

for(int i = 0; i<MAX\_LIGHTS; i++){

location\_lightPosition[i] = super.getUniformLocation("lightPosition[" + i + "]");

location\_lightColour[i] = super.getUniformLocation("lightColour[" + i + "]");

location\_attenuation[i] = super.getUniformLocation("attenuation[" + i + "]");

}

}

public void loadFakeLightingVariable(boolean useFake){

super.loadBoolean(location\_useFakeLighting, useFake);

}

public void loadSkyColour(float r, float g, float b) {

super.loadVector(location\_skyColour, new Vector3f(r, g, b));

}

public void loadShineVariables(float damper, float reflectivity) {

super.loadFloat(location\_shineDamper, damper);

super.loadFloat(location\_reflectivity, reflectivity);

}

public void loadTransformationMatrix(Matrix4f matrix) {

super.loadMatrix(location\_transformationMatrix, matrix);

}

public void loadLights(List<Light> lights) {

for(int i = 0; i < MAX\_LIGHTS; i++){

if(i<lights.size()){

super.loadVector(location\_lightPosition[i], lights.get(i).getPosition());

super.loadVector(location\_lightColour[i], lights.get(i).getColour());

super.loadVector(location\_attenuation[i], lights.get(i).getAttenuation());

}else {

super.loadVector(location\_lightPosition[i], new Vector3f(0, 0, 0));

super.loadVector(location\_lightColour[i], new Vector3f(0, 0, 0));

super.loadVector(location\_attenuation[i], new Vector3f(1, 0, 0));

}

}

}

public void loadViewMatrix(Camera camera) {

Matrix4f viewMatrix = Maths.createViewMatrix(camera);

super.loadMatrix(location\_viewMatrix, viewMatrix);

}

public void loadProjectionMatrix(Matrix4f projection) {

super.loadMatrix(location\_projectionMatrix, projection);

}

}

**Package skybox**

Файл skyboxFragmentShader.txt

#version 400

in vec3 textureCoords;

out vec4 out\_Color;

uniform samplerCube cubeMap;

void main(void){

out\_Color = texture(cubeMap, textureCoords);

}

Файл skyboxVertexShader.txt

#version 400

in vec3 position;

out vec3 textureCoords;

uniform mat4 projectionMatrix;

uniform mat4 viewMatrix;

void main(void){

gl\_Position = projectionMatrix \* viewMatrix \* vec4(position, 1.0);

textureCoords = position;

}

Файл SkyboxRenderer.java

package skybox;

import entities.Camera;

import models.RawModel;

import org.lwjgl.opengl.GL11;

import org.lwjgl.opengl.GL13;

import org.lwjgl.opengl.GL20;

import org.lwjgl.opengl.GL30;

import org.lwjgl.util.vector.Matrix4f;

import renderEngine.Loader;

public class SkyboxRenderer {

private static final float SIZE = 500f;

private static final float[] VERTICES = {

-SIZE, SIZE, -SIZE,

-SIZE, -SIZE, -SIZE,

SIZE, -SIZE, -SIZE,

SIZE, -SIZE, -SIZE,

SIZE, SIZE, -SIZE,

-SIZE, SIZE, -SIZE,

-SIZE, -SIZE, SIZE,

-SIZE, -SIZE, -SIZE,

-SIZE, SIZE, -SIZE,

-SIZE, SIZE, -SIZE,

-SIZE, SIZE, SIZE,

-SIZE, -SIZE, SIZE,

SIZE, -SIZE, -SIZE,

SIZE, -SIZE, SIZE,

SIZE, SIZE, SIZE,

SIZE, SIZE, SIZE,

SIZE, SIZE, -SIZE,

SIZE, -SIZE, -SIZE,

-SIZE, -SIZE, SIZE,

-SIZE, SIZE, SIZE,

SIZE, SIZE, SIZE,

SIZE, SIZE, SIZE,

SIZE, -SIZE, SIZE,

-SIZE, -SIZE, SIZE,

-SIZE, SIZE, -SIZE,

SIZE, SIZE, -SIZE,

SIZE, SIZE, SIZE,

SIZE, SIZE, SIZE,

-SIZE, SIZE, SIZE,

-SIZE, SIZE, -SIZE,

-SIZE, -SIZE, -SIZE,

-SIZE, -SIZE, SIZE,

SIZE, -SIZE, -SIZE,

SIZE, -SIZE, -SIZE,

-SIZE, -SIZE, SIZE,

SIZE, -SIZE, SIZE

};

private static String[] TEXTURE\_FILES= {"skybox/right", "skybox/left", "skybox/top", "skybox/bottom", "skybox/front", "skybox/back"};

private RawModel cube;

private int texture;

private SkyboxShader shader;

public SkyboxRenderer(Loader loader, Matrix4f projectionMatrix){

cube = loader.loadToVAO(VERTICES, 3);

texture = loader.loadCubeMap(TEXTURE\_FILES);

shader = new SkyboxShader();

shader.start();

shader.loadProjectionMatrix(projectionMatrix);

shader.stop();

}

public void render(Camera camera){

shader.start();

shader.loadViewMatrix(camera);

GL30.glBindVertexArray(cube.getVaoID());

GL20.glEnableVertexAttribArray(0);

GL13.glActiveTexture(GL13.GL\_TEXTURE0);

GL11.glBindTexture(GL13.GL\_TEXTURE\_CUBE\_MAP, texture);

GL11.glDrawArrays(GL11.GL\_TRIANGLES, 0, cube.getVertexCount());

GL20.glDisableVertexAttribArray(0);

GL30.glBindVertexArray(0);

shader.stop();

}

}

Файл SkyboxShader.java

package skybox;

import org.lwjgl.util.vector.Matrix4f;

import entities.Camera;

import shaders.ShaderProgram;

import toolbox.Maths;

public class SkyboxShader extends ShaderProgram{

private static final String VERTEX\_FILE = "src/skybox/skyboxVertexShader.txt";

private static final String FRAGMENT\_FILE = "src/skybox/skyboxFragmentShader.txt";

private int location\_projectionMatrix;

private int location\_viewMatrix;

public SkyboxShader() {

super(VERTEX\_FILE, FRAGMENT\_FILE);

}

public void loadProjectionMatrix(Matrix4f matrix){

super.loadMatrix(location\_projectionMatrix, matrix);

}

public void loadViewMatrix(Camera camera){

Matrix4f matrix = Maths.createViewMatrix(camera);

matrix.m30 = 0;

matrix.m31 = 0;

matrix.m32 = 0;

super.loadMatrix(location\_viewMatrix, matrix);

}

@Override

protected void getAllUniformLocations() {

location\_projectionMatrix = super.getUniformLocation("projectionMatrix");

location\_viewMatrix = super.getUniformLocation("viewMatrix");

}

@Override

protected void bindAttributes() {

super.bindAttribute(0, "position");

}

}

**Package textures**

Файл ModelTexture.java

package textures;

public class ModelTexture {

private int textureID;

private float shineDamper = 1;

private float reflectivity = 0;

private boolean hasTransparency = false;

private boolean useFakeLighting = false;

public boolean isUseFakeLighting() {

return useFakeLighting;

}

public void setUseFakeLighting(boolean useFakeLighting) {

this.useFakeLighting = useFakeLighting;

}

public boolean isHasTransparency() {

return hasTransparency;

}

public void setHasTransparency(boolean hasTransparency) {

this.hasTransparency = hasTransparency;

}

public float getShineDamper() {

return shineDamper;

}

public void setShineDamper(float shineDamper) {

this.shineDamper = shineDamper;

}

public float getReflectivity() {

return reflectivity;

}

public void setReflectivity(float reflectivity) {

this.reflectivity = reflectivity;

}

public ModelTexture(int texture) {

this.textureID = texture;

}

public int getID() {

return textureID;

}

}

Файл TextureData.java

package textures;

import java.nio.ByteBuffer;

public class TextureData {

private int width;

private int height;

private ByteBuffer buffer;

public TextureData(ByteBuffer buffer, int width, int height){

this.buffer = buffer;

this.width = width;

this.height = height;

}

public int getWidth(){

return width;

}

public int getHeight(){

return height;

}

public ByteBuffer getBuffer(){

return buffer;

}

}

**Package toolbox**

Файл Collisions.java

package toolbox;

import entities.Entity;

public class Collisions {

public boolean getCollision(Entity objectA, Entity objectB) {

float distance = (float) Math.sqrt(Math.pow(objectA.getPosition().x - objectB.getPosition().x, 2)

+ Math.pow(objectA.getPosition().y - objectB.getPosition().y, 2)

+ Math.pow(objectA.getPosition().z - objectB.getPosition().z, 2));

return distance <= (objectA.getScale() + objectB.getScale()) \* 1.5;

}

}

Файл Maths.java

package toolbox;

import org.lwjgl.util.vector.Matrix4f;

import org.lwjgl.util.vector.Vector3f;

import entities.Camera;

import org.lwjgl.util.vector.Vector2f;

public class Maths {

public static Matrix4f createTransformationMatrix(Vector2f translation, Vector2f scale) {

Matrix4f matrix = new Matrix4f();

matrix.setIdentity();

Matrix4f.translate(translation, matrix, matrix);

Matrix4f.scale(new Vector3f(scale.x, scale.y, 1f), matrix, matrix);

return matrix;

}

public static Matrix4f createTransformationMatrix(Vector3f translation, float rx, float ry,

float rz, float scale) {

Matrix4f matrix = new Matrix4f();

matrix.setIdentity();

Matrix4f.translate(translation, matrix, matrix);

Matrix4f.rotate((float) Math.toRadians(rx), new Vector3f(1,0,0), matrix, matrix);

Matrix4f.rotate((float) Math.toRadians(ry), new Vector3f(0,1,0), matrix, matrix);

Matrix4f.rotate((float) Math.toRadians(rz), new Vector3f(0,0,1), matrix, matrix);

Matrix4f.scale(new Vector3f(scale,scale,scale), matrix, matrix);

return matrix;

}

public static Matrix4f createViewMatrix(Camera camera) {

Matrix4f viewMatrix = new Matrix4f();

viewMatrix.setIdentity();

Matrix4f.rotate((float) Math.toRadians(camera.getPitch()), new Vector3f(1, 0, 0), viewMatrix,

viewMatrix);

Matrix4f.rotate((float) Math.toRadians(camera.getYaw()), new Vector3f(0, 1, 0), viewMatrix, viewMatrix);

Vector3f cameraPos = camera.getPosition();

Vector3f negativeCameraPos = new Vector3f(-cameraPos.x,-cameraPos.y,-cameraPos.z);

Matrix4f.translate(negativeCameraPos, viewMatrix, viewMatrix);

return viewMatrix;

}

}