Технически университет - София

курсова работа по дисциплина

„Програмиране на JAVA за напреднали“

изготвил

Илия Георгиев Къркъмов
991319006

София
2020

Увод

Курсовата работа по дисциплина „Програмиране на JAVA за напреднали“ представена в този документ представя приложение, което има за цел да изрисува на екрана 3D модели. Симулирана е и камера, така че обектите да могат да бъдат прегледани под различен ъгъл. Приложението използва възможностите на OpenGL, за да рисува върху екрана в реално време. Обектите, които се рисуват могат да бъдат заредени от различни източници, като върху тях се симулира отразяване на светлина чрез т.нар. Phong Lighting [1].

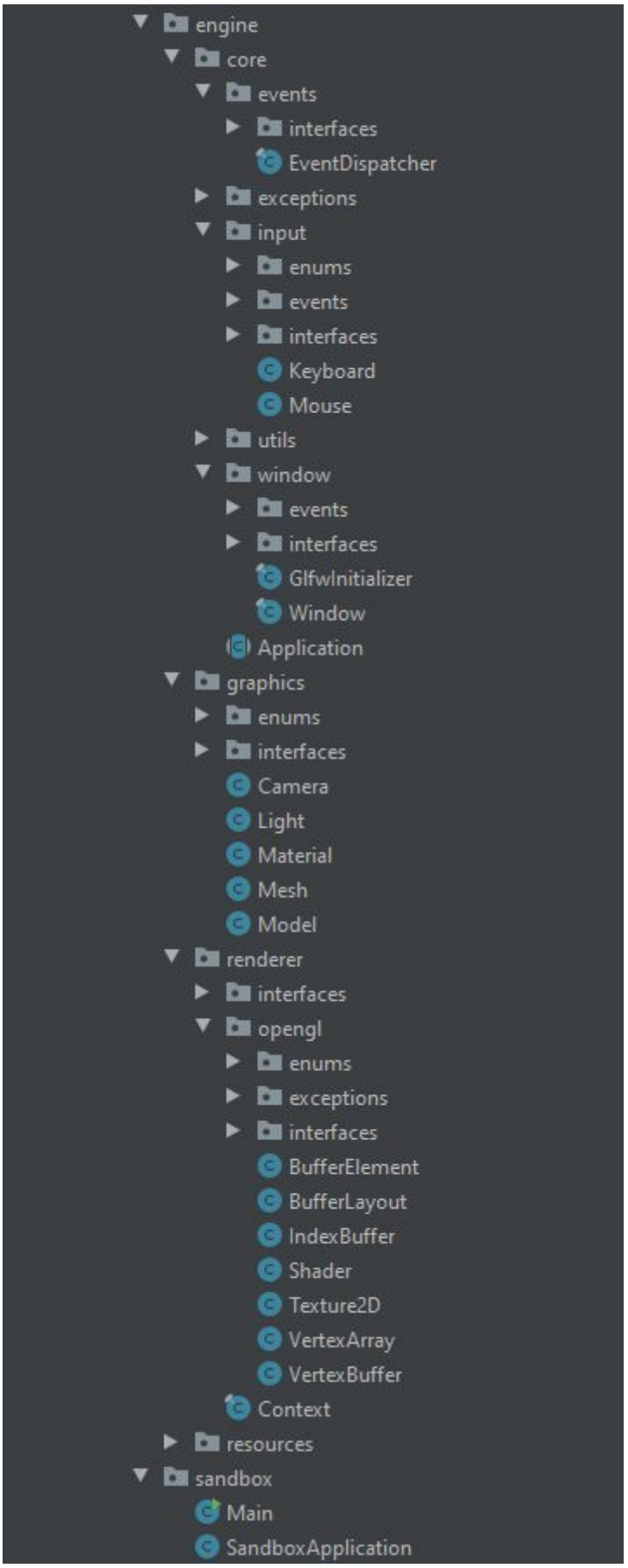
Проектиране на решение

Използвани библиотеки

За разработката на проекта са използвани няколко външни библиотеки. Основната библиотека върху, която е изграден проекта е **LWJGL (org.lwjgl)** [2]. Библиотеката предоставя основни програмни интерфейси представляващи обвивки върху библиотеки от по-ниско ниво писани на език C/C++, които предоставят възможност за създаване на Win32 прозорец поддържащ OpenGL контекст чрез библиотеката **GLFW** [3]. **LWJGL** предоставя програмен интерфейс и до библиотеката **AssImp** [4], която предоставя възможност за зареждане на различни 3D модели от различни формати за обмен на данни.

Структура на проекта

Фигура 1 показва структурата на проекта разглеждан в настоящият документ. Основната функционалност на проекта е разработена в пакет **engine**. В него са разработени функционалности за предаване на събития, функционалност за достъп до периферни устройства - клавиатура и мишка, както и функционалност за създаване на прозорец където да бъдат изрисувани 3D обектите. Пакетът **graphics** предоставя абстракции за създаване на камера, светлина и други графични обекти. Пакетът **renderer** предоставя абстракции над различните OpenGL обекти, така че да е необходим по-малко програмен код за описването на 3D обекти. Пакетът **resources** реализира абстракции за зареждане и управление на различни ресурси като модели, shader програми и текстури. Целта е тези ресурси да бъдат преизползвани ефективно. Например един 3D модел може да бъде изрисуван множество пъти в един кадър. Би било неефективно моделът да бъде зареждан от файловата система всеки път когато е необходим. Накрая пакетът **sandbox** има за цел да сглоби функционалностите предоставени от пакета **engine**, така че да бъдат демонстрирани възможностите пакета.



Фигура 1 - Структура на проекта

Основни функционалности

Пакет за предаване на събития

Събитията са основна част от сглобяването на различните компоненти в една работеща система. Например чрез събития се уведомява когато даден клавиш бъде натиснат, а в контекста на компютърна игра чрез събития се уведомява когато два обекта се сблъскат или когато обект бъде убит/премахнат. Модулът за предаване на събития реализиран в рамките текущата курсова работа има за цел да избегне нуждата от директно свързване (coupling) на различните системи, т.е. вместо две системи да имат директна връзка помежду им, те да имат достъп до обект за предаване на събития, така че системите да комуникират чрез събития вместо да зависят пряко една от друга.

Основните интерфейси в този пакет са следните:

- **IEvent** - интерфейс за събитие;
- **IEventListener** - интерфейс за описване на слушател за конкретно събитие, т.е. имплементиращите този интерфейс реализират конкретна функционалност, която да бъде изпълнена при настъпване на събитие;
- **IEventDispatcher** - интерфейс позволяващ публикуване на събития и абониране за конкретни събития.

Фигура 2 показва интерфейсът за публикуване и абониране за събития - IEventDispatcher.

```
public interface IEventDispatcher {
    <T extends IEvent> void publish(T event);

    void addListener(Class<? extends IEvent> eventClass, IEventListener listener);

    void removeListener(Class<? extends IEvent> eventClass, IEventListener listener);
}
```

Фигура 2 - Интерфейс IEventDispatcher

Методът **publish** служи за публикуване на събитие, а методите **addListener** и **removeListener** служат за абониране и деабониране за конкретно събитие.

Пакет за управление на ресурси

Зареждането и управлението на ресурси е основна част от ефективната употреба на ресурсите. Възможността за кеширане в паметта на различни типове ресурси позволява бързия достъп до тях когато те са необходими. Например ако на един 3D терен има 100 дървета то 3D моделът на дървото, както и на текстурите на дървото могат да бъдат достъпвани директно от паметта вместо да бъдат зареждани 100 пъти от файловата система.

Основните интерфейси в този пакет са следните:

- **IResourceFactory** - интерфейс за създаване/зареждане на ресурси.
- **IResourceManager** - интерфейс за достъп до ресурси.

Фигури 3 и 4 показват двата основни интерфейса:

```
public interface IResourceFactory<T> {
    Class<T> getType();

    T create(String resource) throws ResourceLoadException;

    IResourceManager getResourceManager();
}
```

Фигура 3 - Интерфейс IResourceFactory

Основният метод в интерфейс IResourceManager е методът **create**. Реализиращите този интерфейс трябва да предоставят конкретна логика по създаване/зареждане на ресурс идентифициран от параметъра resource.

```
public interface IResourceManager {
    <T> T get(Class<T> resourceClass, String resourceKey) throws ResourceLoadException;

    void registerFactory(IResourceFactory<?> factory);
}
```

Фигура 4 - Интерфейс IResourceManager

Методът **get** на интерфейс IResourceManager служи за осъществяване на достъп до конкретен ресурс. Реализиращите този интерфейс трябва да предоставят логика за достъп до ресурси. Например ресурсите могат да бъдат кеширани в паметта и в случай, че ресурс идентифициран чрез параметъра resourceKey не е кеширан, то той да бъде зареден чрез съответната имплементация на интерфейса IResourceFactory. Методът **registerFactory** служи за регистриране на конкретни имплементации на IResourceFactory, така че те да бъдат използвани за зареждане на ресурси.

Свързване на пакетите в демонстративно приложение

Свързването на всички разработени пакети е реализирано в пакет **sandbox**. Клас **SandboxApplication** демонстрира реализация на приложение зареждащо 3D модел на робот, което изрисува грид от роботи на екрана. Следващите фигури показват реализираната функционалност.

```java
public class SandboxApplication extends Application {
    private static final float cameraSensitivity = 0.05f;

    private IShader shader;
    private ICamera camera;

    private IModel model;
    private Light light;

    SandboxApplication() {
        getEventDispatcher().addListener(WindowCloseEvent.class, event -> {
            setRunning(false);
            return false;
        });
    }

    @Override
    protected void init() throws ApplicationInitException {
        try {
            getWindow().setVerticalSync(false);

            shader = getResourceManager().get(IShader.class, resourceKey: "shaders/simple/simple");
            camera = new Camera(new Vector3f( x: 100.f, y: 100.f, z: 50f));

            model = getResourceManager().get(IModel.class, resourceKey: "assets/models/ba2/Quandtum_BA-2_v1_1.dae");

            light = new Light();
            light.setPosition(new Vector3f( x: 150.f, y: 200.f, z: -50.f));
            light.setAmbient(new Vector3f( x: 0.2f, y: 0.2f, z: 0.2f));
            light.setDiffuse(new Vector3f( x: 0.5f, y: 0.5f, z: 0.5f));
            light.setSpecular(new Vector3f( x: 1.0f, y: 1.0f, z: 1.0f));

            getContext().enable(Capability.CullFace);
            getContext().enable(Capability.DepthTest);
        } catch (ResourceLoadException e) {
            throw new ApplicationInitException("Failed to initialize the application.", e);
        }
    }

    @Override
    protected void close() {
    }

    @Override
    protected void update(float delta) {
```

```
final float speed = 15.f * delta;

if (getKeyboard().isKeyPressed(KeyboardButton.A)) {
    camera.move(MoveDirection.Left, speed);
}

if (getKeyboard().isKeyPressed(KeyboardButton.D)) {
    camera.move(MoveDirection.Right, speed);
}

if (getKeyboard().isKeyPressed(KeyboardButton.W)) {
    camera.move(MoveDirection.Forward, speed);
}

if (getKeyboard().isKeyPressed(KeyboardButton.S)) {
    camera.move(MoveDirection.Backward, speed);
}

if (getMouse().isButtonPressed(MouseButton.MouseButtonLeft)) {
    Vector2f offset = getMouse().getPositionOffset();
    camera.updateYaw(offset.x * cameraSensitivity);
    camera.updatePitch(offset.y * cameraSensitivity);
}

getWindow().setTitle("FPS: " + 1 / delta);
getContext().clear();

shader.bind();
shader.setUniform( name: "light.position", light.getPosition());
shader.setUniform( name: "light.ambient", light.getAmbient());
shader.setUniform( name: "light.diffuse", light.getDiffuse());
shader.setUniform( name: "light.specular", light.getSpecular());
shader.setUniform( name: "viewPos", camera.getPosition());

Matrix4f projection = new Matrix4f();
projection.perspective((float) Math.toRadians(camera.getZoom()),  aspect: 16.f / 9.f,  zNear: 0.01f,  zFar: 100.0f);

shader.setUniform( name: "projection", projection);
shader.setUniform( name: "view", camera.getViewMatrix());

float x = 100.f;
float y = 100.f;
float z = -20.f;
final float offset = 10.f;
```

```
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    Matrix4f model = new Matrix4f();
                    model = model.translate(x, y, z).rotate((float) Math.toRadians(-90.f), new Vector3f( x: 1,  y: 0,  z: 0));
                    shader.setUniform( name: "model", model);

                    this.model.draw(getContext(), shader);

                    z -= offset;
                }

                y += offset;
                z = -20.f;
            }

            x += offset;
            y = 100.f;
        }
    }
}
```

Резултати

Към текущия документ е прикачен видеоклип демонстриращ работата на приложението.

Използвана литература

[1] Phong lighting - https://learnopengl.com/Lighting/Basic-Lighting
[2] LWJGL - https://www.lwjgl.org/
[3] GLFW - https://www.glfw.org/
[4] AssImp - http://www.assimp.org/