Федеральное государственное автономное образовательное учреждение высшего образования
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н.Э.БАУМАНА»
(МГТУ им. Н.Э. Баумана)

# Разработка графического программного обеспечения для визуализации трехмерных объектов

Студент: Губанов Д.А.
Научный руководитель: Витюков Ф.А.

Москва, 2025

# Список обозначений и сокращений

**OpenGL** (Open Graphics Library) – Кроссплатформенный API для 2D и 3D графики.

**GLFW** (Graphics Library Framework) – библиотека для кроссплатформенного создания и открытия окон, создания OpenGL-контекста и управления вводом.

**Шейдер** (Shader) – Программа, выполняемая на GPU для обработки графических данных.

**Текстура** (Texture) — это растровое изображение (обычно в формате PNG, JPG, TGA и др.), которое накладывается на 3D-модель или 2D-объект, чтобы придать ей цвет, детализацию и реалистичность.

**Система рефлексии** (Reflection system) — это механизм, позволяющий программе анализировать и взаимодействовать с собственными свойствами и методами во время выполнения программы.

# Постановка задачи

**Цель работы:** Разработка графического программного обеспечения для визуализации трехмерных объектов.

**Задачи:**

- Создать **оконное приложение**, поддерживающее **различные ОС**;
- Реализовать **отображение** трехмерных моделей;
- Разработать **импортирование** трехмерных моделей;
- Реализовать удобный и практичный **интерфейс** для пользователя;
- Интегрировать **систему рефлексии**;

# Актуальность и обзор существующих решений

**Существующие решения:**

- Unreal Engine
- Unity
- CryEngine
- Godot
- Nau Engine

**Актуальность:**

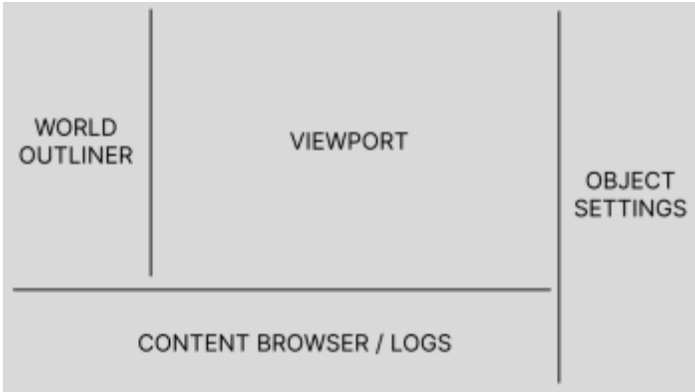
- Отечественная разработка
- Импортозамещение
- Применим в рамках образовательных программ

# Прототипирование

Для разработки графического программного обеспечения за пример взят трёхмерный движок **Unreal Engine 4**, в который входит:

1. Интеграция с операционными системами – **GLFW**
2. Отображение трехмерных объектов – **OpenGL**
3. Импортирование текстур и 3D-объектов – **SOIL** и **Assimp**
4. Интерфейс – **ImGUI**
5. Система рефлексии – на базе **LLVM Clang**

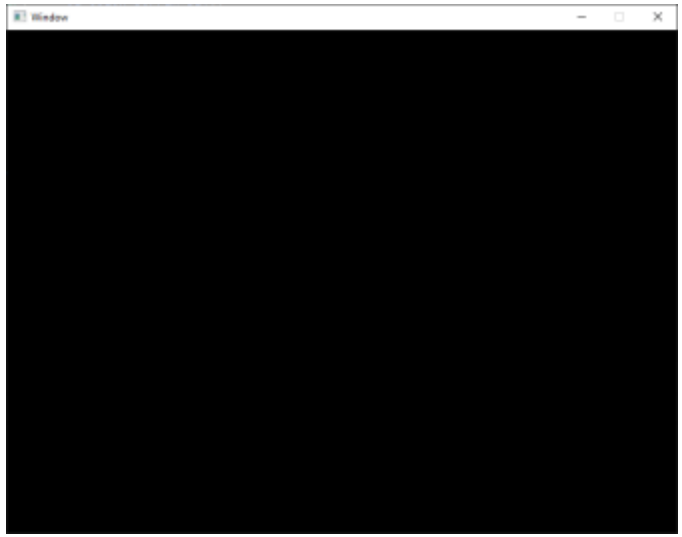
Интерфейс Unreal Engine 4




Прототип интерфейса

# Графическое окно

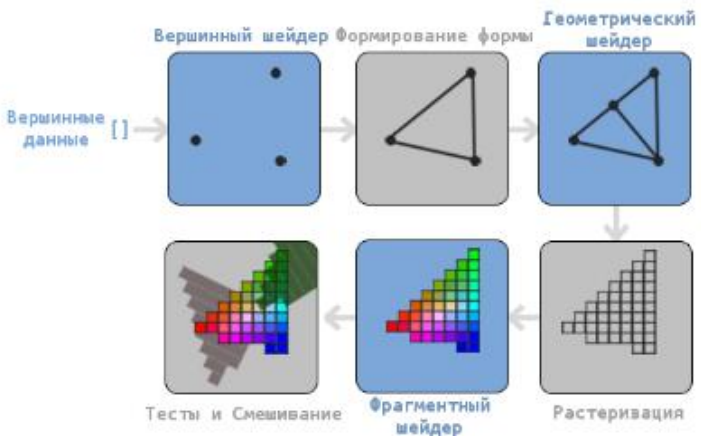
**GLFW** представляет собой библиотеку, написанную на языке программирования C, предназначенную для обеспечения **OpenGL** необходимыми функциональностями для работы с операционной системой. Данная библиотека позволяет создавать контекст, задавать параметры оконного приложения и обрабатывать пользовательский ввод.



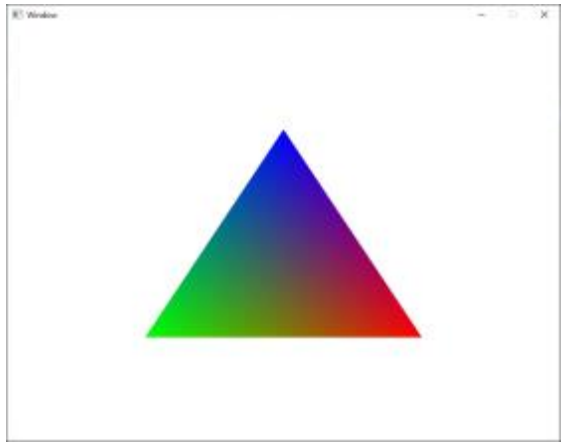
Оконное приложение

# Графический конвейер

В **OpenGL** все объекты находятся в трёхмерном пространстве, однако экран и окно представляют собой двумерную матрицу пикселей. Следовательно, значительная часть задач OpenGL связана с преобразованием трёхмерных координат в двумерные для отображения на экране. Этот процесс преобразования управляется **графическим конвейером** OpenGL.



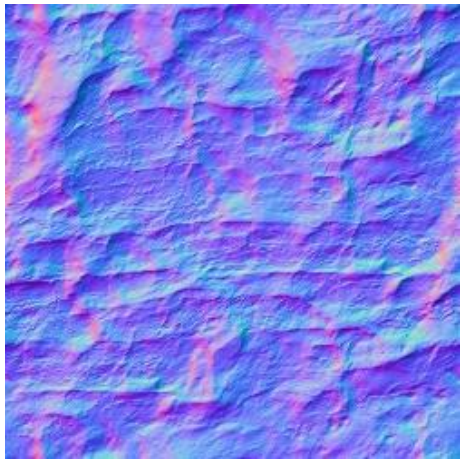

Графический конвейер



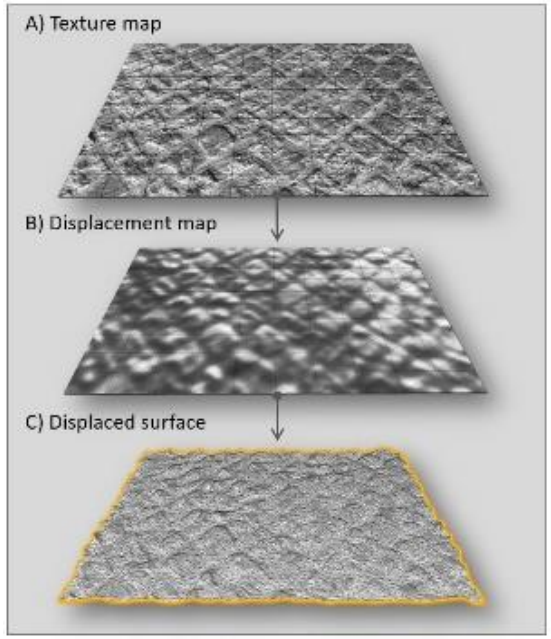
Пример работы

# Текстурирование

**Текстурирование** — процесс, заключающийся в наложении двумерных изображений (**текстур**) на поверхность модели для придания ей визуальной сложности и реалистичности.



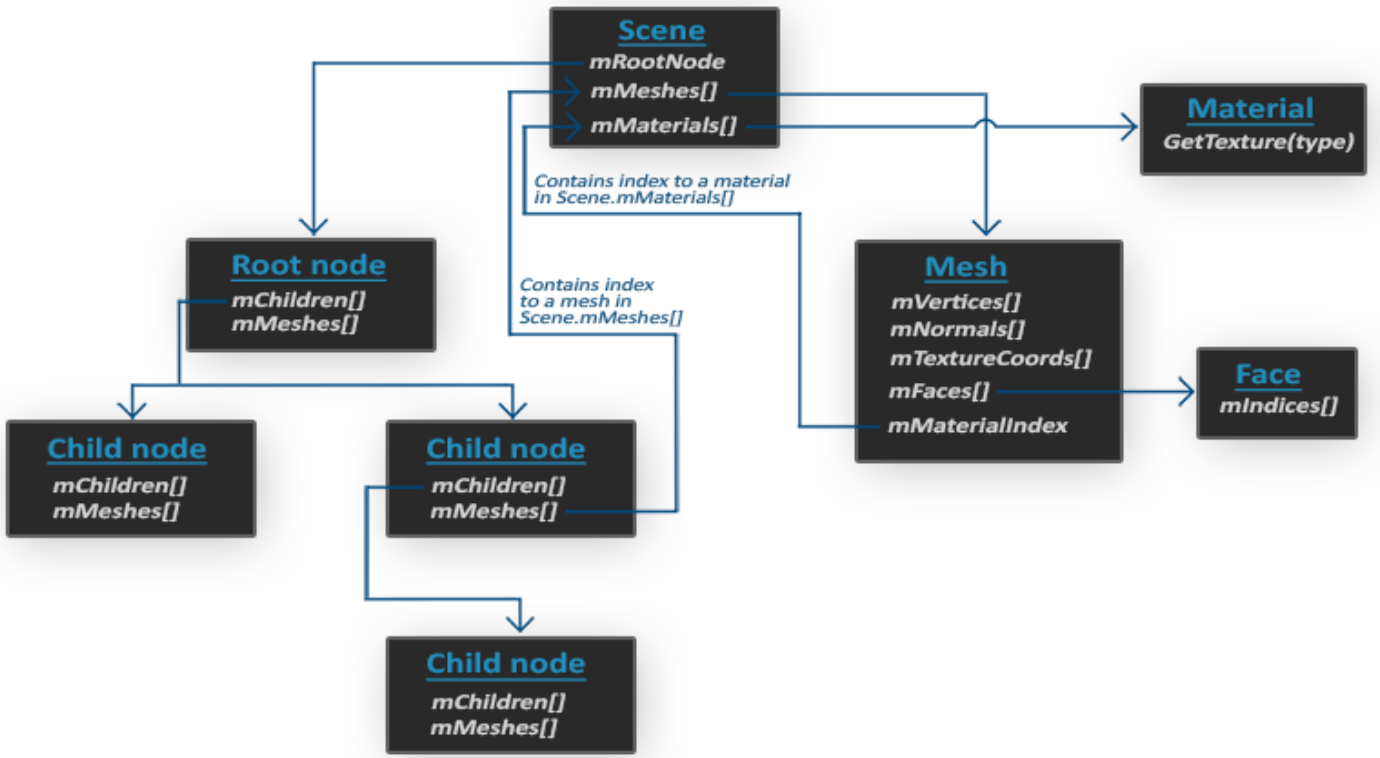
Карта нормалей




Карта смещений

Пример работы

# Импортирование трехмерных моделей




Модель структуры Assimp

# Аффинные преобразования

Преобразование точки в пространстве описывается формулой:

$$\boldsymbol{P' = M \times P = T \times R \times S \times P}$$

Глобальная трансформация определяется как произведение матриц преобразований всех родительских объектов:

$$\boldsymbol{M_{global} = M_{parentN} \times \ldots \times M_{parent1} \times M_{local}}$$

$$\mathbf{T} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \boldsymbol{t_x} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \boldsymbol{t_y} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \boldsymbol{t_z} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

Матрица трансляции

$$\mathbf{S} = \begin{pmatrix} \boldsymbol{s_x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{s_y} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{s_z} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

Матрица масштабирования

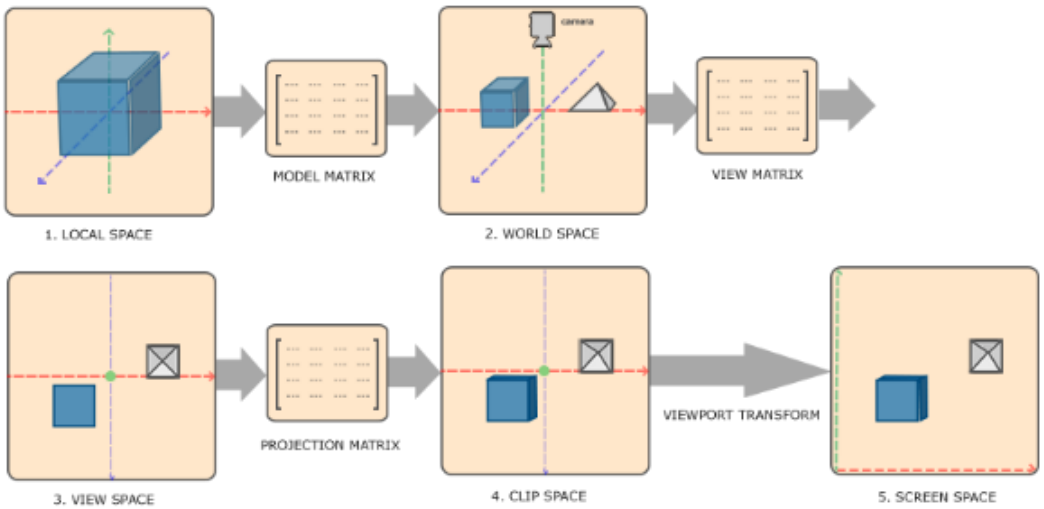
$$\boldsymbol{R_x(\theta)} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\cos\theta} & \mathbf{-\sin\theta} & \mathbf{0} \\ \mathbf{0} & \mathbf{\sin\theta} & \mathbf{\cos\theta} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

Матрица вращения

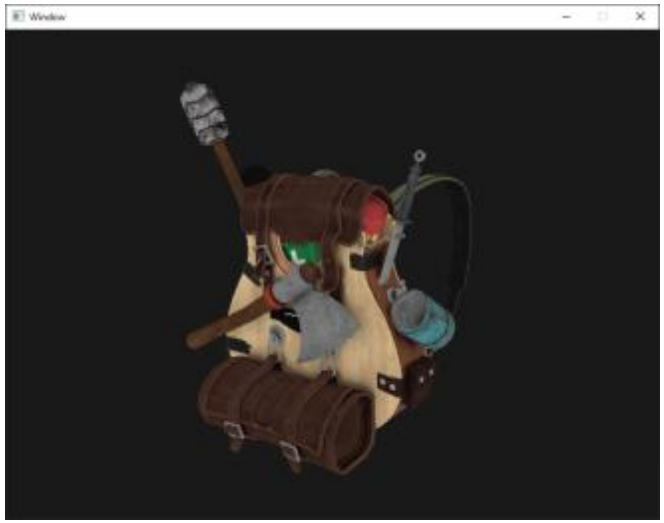
# Системы координат

Координата вершины преобразуется в координаты пространства отсечения следующим образом:

$$V_{clip} = M_{projection} \times M_{view} \times M_{model} \times V_{local}$$



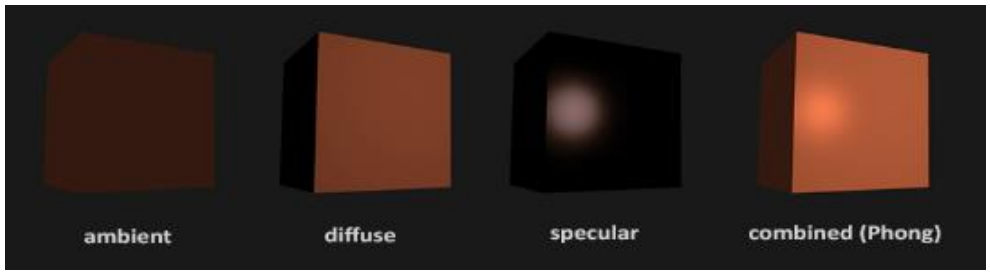

Преобразование координат



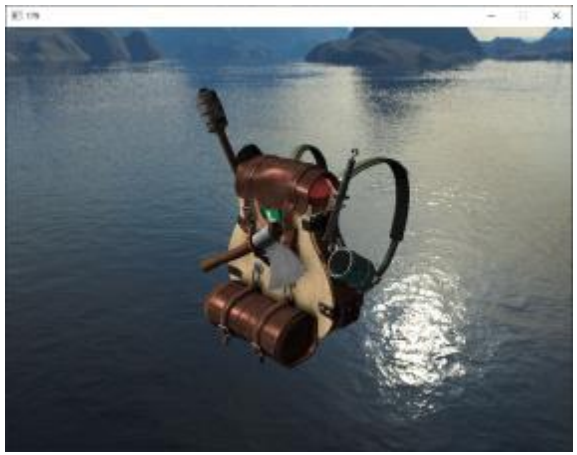
Пример работы

# Освещение

Распространение света в реальном мире представляет собой чрезвычайно сложное явление, зависящее от множества факторов. В условиях ограниченных вычислительных ресурсов мы не можем позволить себе учитывать все нюансы в расчетах. Поэтому освещение в OpenGL базируется на использовании упрощенных математических моделей, приближенных к реальности. Одной из таких моделей является модель освещения по **Фонгу**. Она состоит из трех основных компонентов: **ambient**, **diffuse**, **specular**.

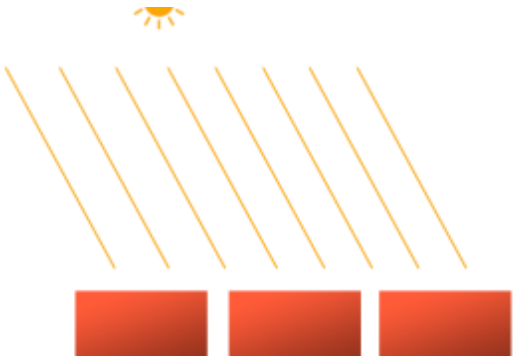



Компоненты освещения

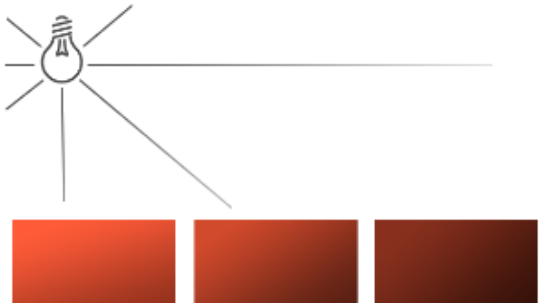


Пример работы

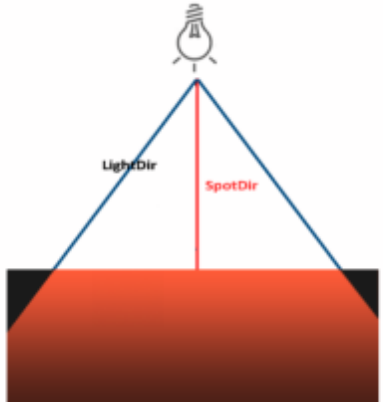
# Виды источников освещения



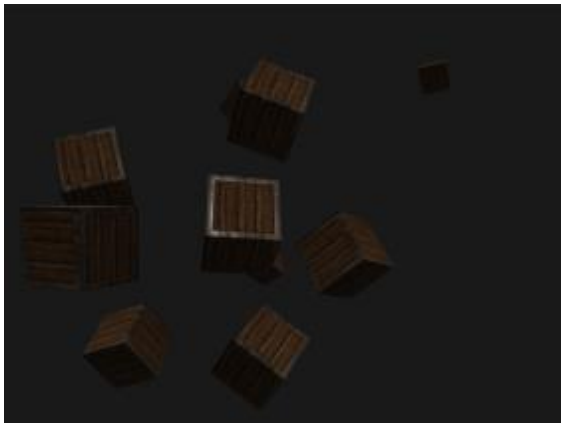
Направленный источник освещения

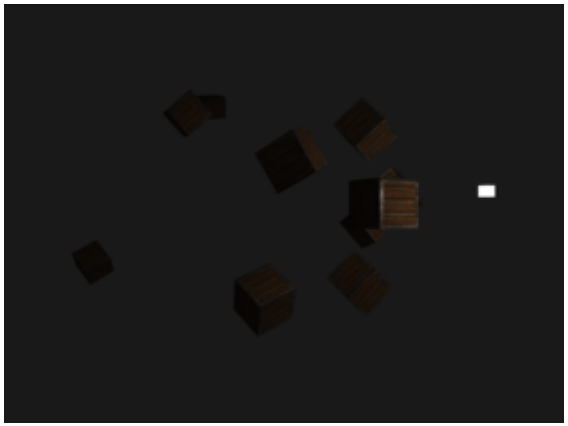


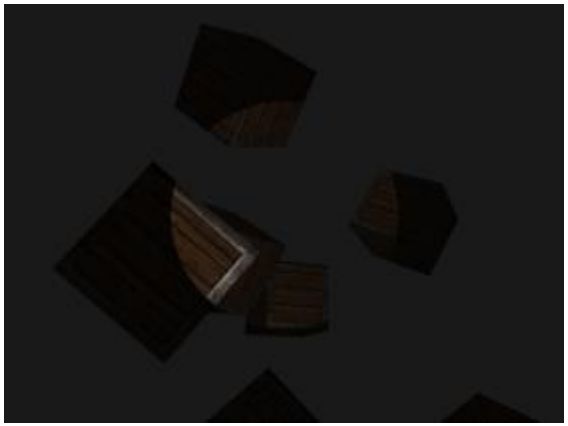
Точечный источник освещения



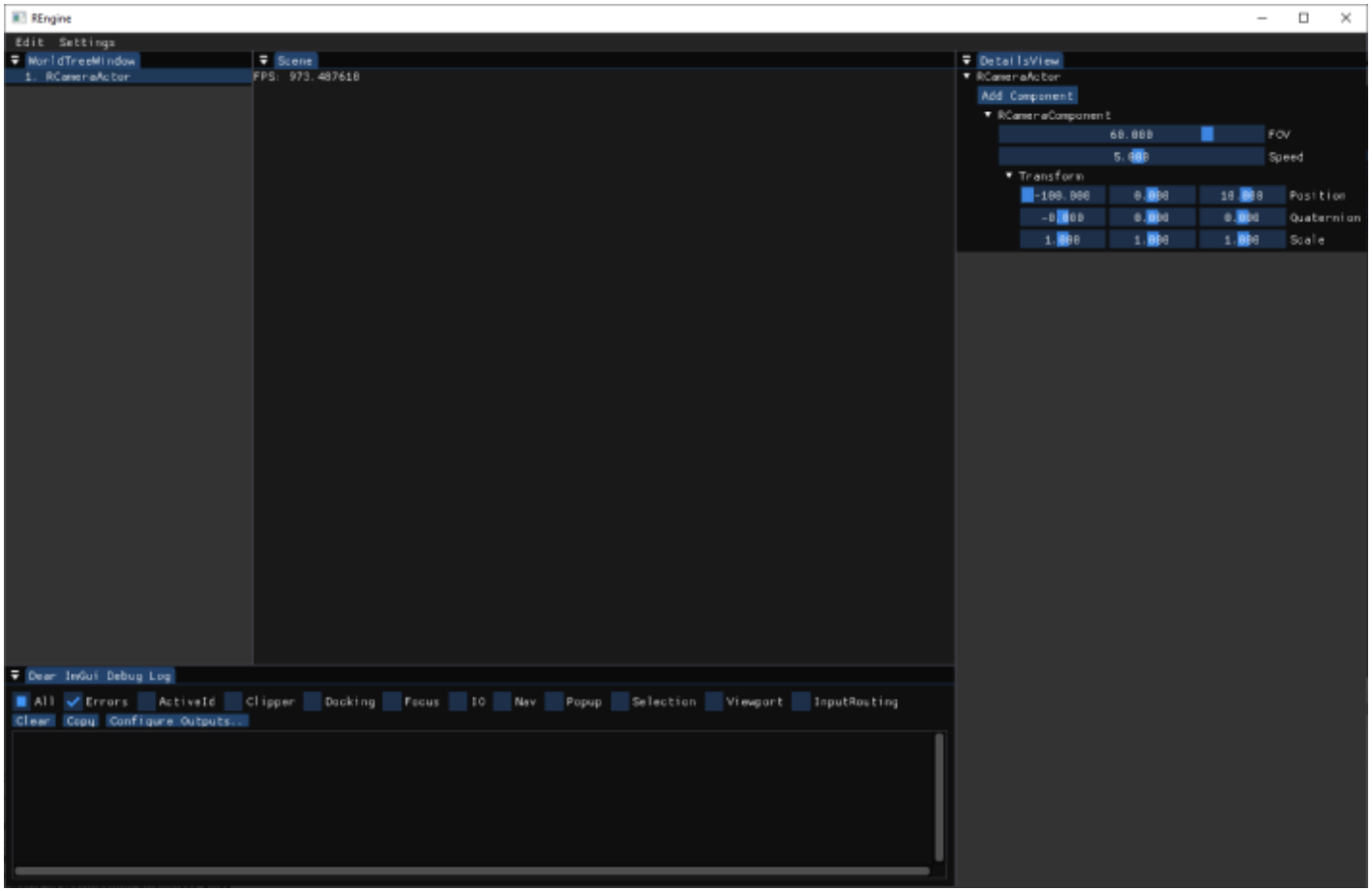

Прожекторный источник освещения







# Интерфейс




Интерфейс разработанного графического ПО

# Система рефлексии

```cpp
#include "LYDAIController.generated.h"

class ULYDAIPerceptionComponent;

UCLASS()
1 derived blueprint class
class LYD_API ALYDAIController : public AAIController
{
    GENERATED_BODY()

public:
    ALYDAIController();

    UPROPERTY()
    AActor* Target;

protected:
    UPROPERTY(EditDefaultsOnly,Category="Components")
    ULYDAIPerceptionComponent* AIPerceptionComponent;

    UFUNCTION()
    virtual void OnPossess(APawn* InPawn) override;
};
```

Пример реализации класса в Unreal Engine

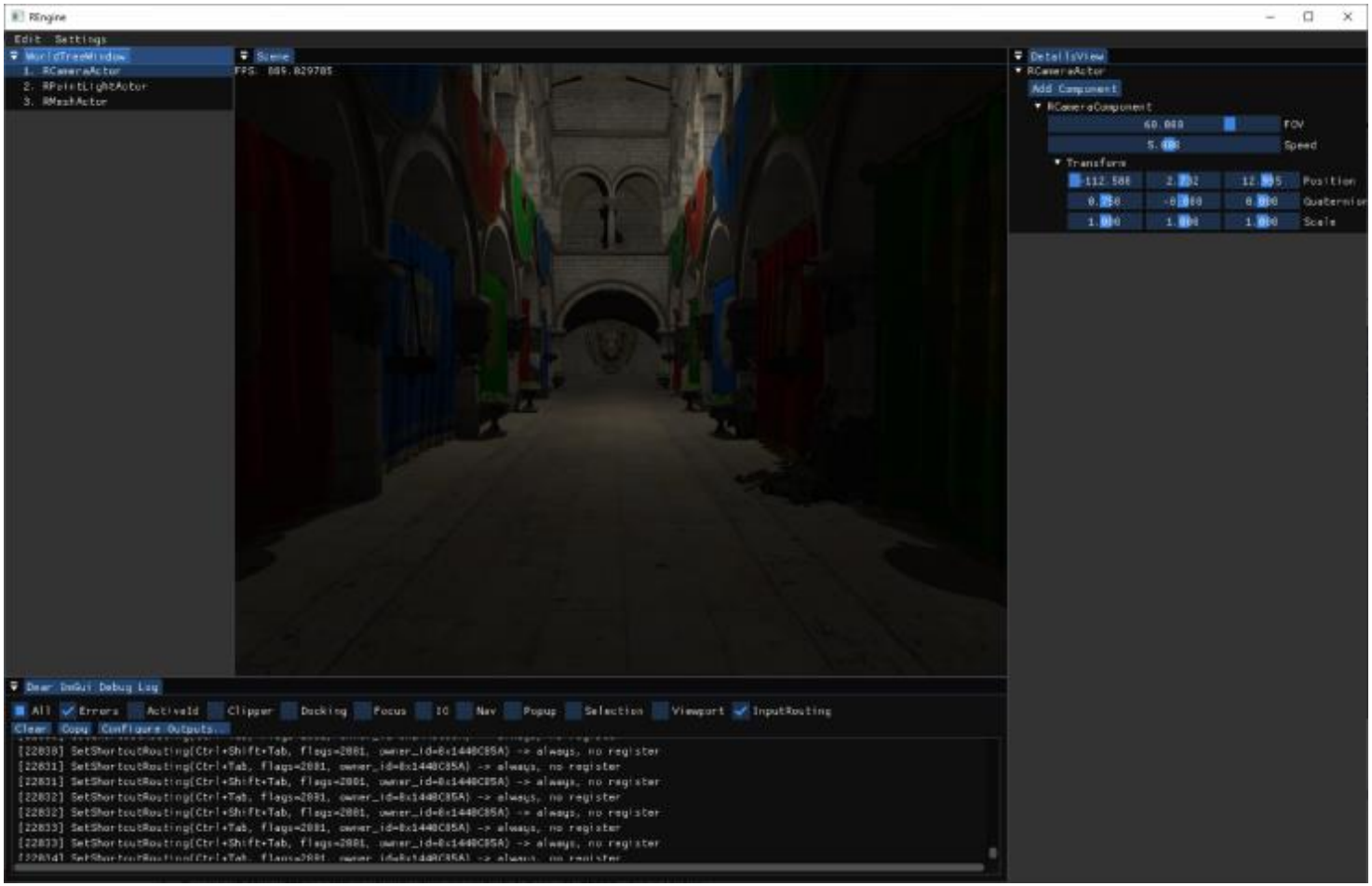
```cpp
#include "Generated/RCameraComponent.generated.h"

class CLASS() RCameraComponent : public RSceneComponent
{
    GENERATED_BODY()

public:
    void Construct() override;

    FMatrix GetViewMatrix() const;
    FMatrix GetProjectionMatrix() const;

    void SetFOV(float InFOV);
    float GetFOV() const;

    void SetSpeed(float InSpeed);
    float GetSpeed() const;

protected:
    PROPERTY()
    float FOV = 60.0f;

    PROPERTY()
    float Speed = 5.0f;

    FUNCTION()
    void Review();

};
```

Пример реализации класс в разработанном графическом ПО

```cpp
#pragma once
#include "MetaReflection/MetaReflection.h"

#ifdef GENERATED_BODY
#undef GENERATED_BODY
#endif

#define GENERATED_BODY(...)\
public:\
    using RSceneComponent::RSceneComponent;\
    std::string GetClassName() const override { return "RCameraComponent" ; }\
    static std::string GetStaticClassName() { return "RCameraComponent" ; }\
private:\
    using ThisClass = RCameraComponent;\
    using SuperClass = RSceneComponent;\
    friend struct refl_impl::metadata::type_info__<RCameraComponent>;\
private:

#ifdef META_REFLECT
#undef META_REFLECT
#endif

#define META_REFLECT(...)\
REFL_AUTO(\
    type(RCameraComponent, bases<RSceneComponent>), \
    field(FOV, Serializable()), \
    field(Speed, Serializable()))\
)
```

Сгенерированный h-файл

# Заключение




Графическое ПО

# Спасибо за внимание

Разработка графического программного обеспечения для визуализации трехмерных объектов

Студент: Губанов Д.А.
Научный руководитель: Витюков Ф.А.

Москва, 2025