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

Содержание

Постановка задачи

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

Задачи:

- Отображение **трехмерных объектов** с возможностью кастомизации
- **Система рефлексии** пользовательских типов данных
- **Сборщик мусора** для контроля над пользовательскими данными

Актуальность и обзор существующих решений

Существующие решения:

- Unreal Engine
- Unity
- CryEngine
- Godot

Актуальность:

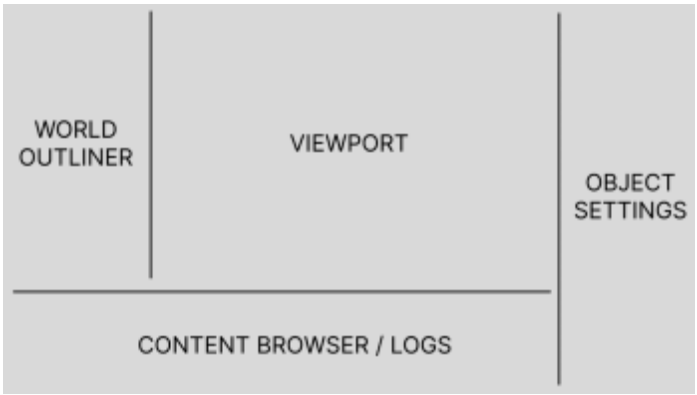
- Отечественная разработка
- Импортозамещение

Прототипирование

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

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

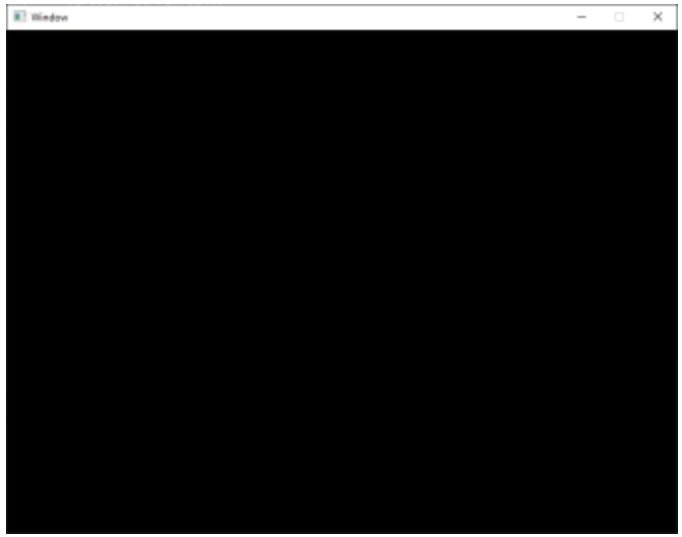
Интерфейс Unreal Engine 4



Прототип интерфейса

Графическое окно

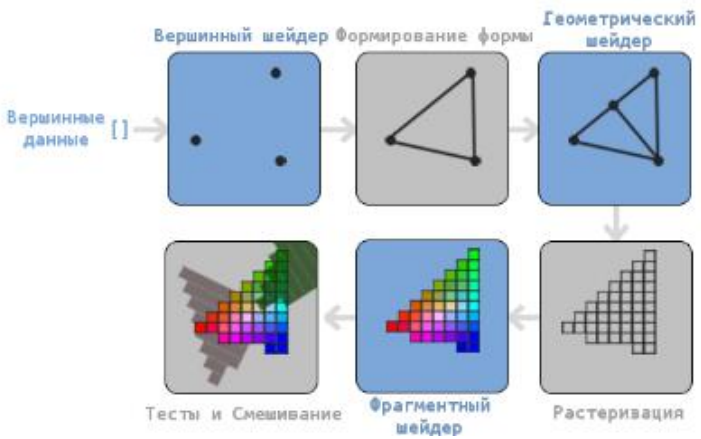
Для отображения окна была использована библиотека **GLFW**. GLFW представляет собой библиотеку, написанную на языке программирования C, предназначенную для обеспечения **OpenGL** необходимыми функциональностями для отрисовки контента на экране. Данная библиотека позволяет создавать контекст, задавать параметры окна и обрабатывать пользовательский ввод.



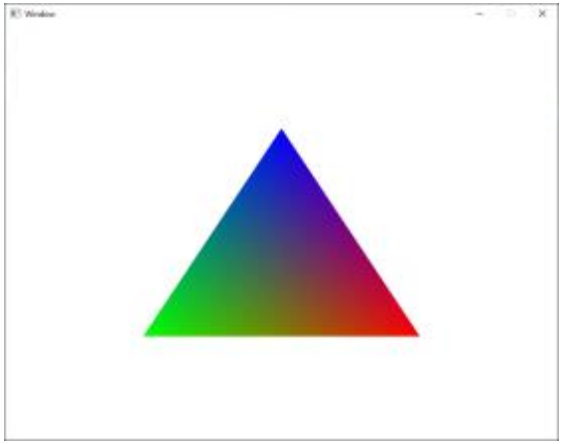
Оконное приложение

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

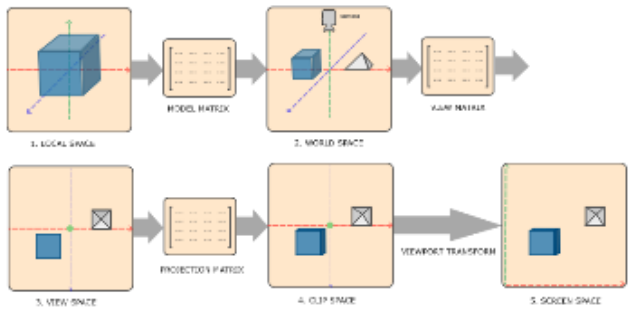


Пример работы

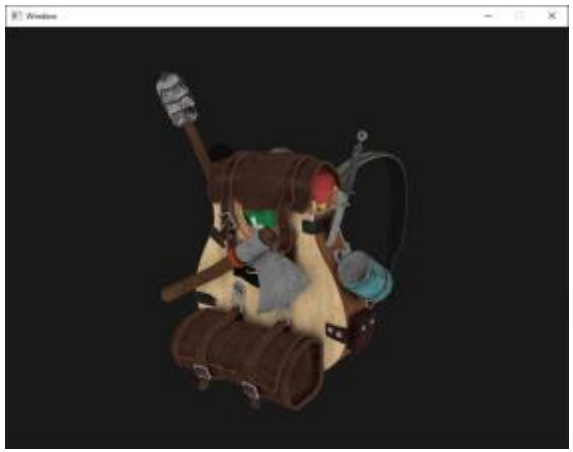
Системы координат

Преобразование координат происходит в несколько этапов: из нормализованных координат в экранные координаты через промежуточные координатные системы. Для преобразования координат из одного пространства в другое используются несколько матриц трансформации, среди которых являются матрицы **Модели**, **Вида** и **Проекции**. Координата вершины преобразуется в координаты пространства отсечения следующим образом:

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



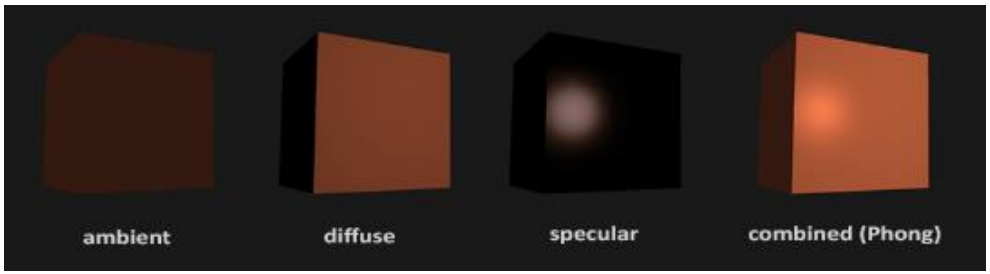
Преобразование координат



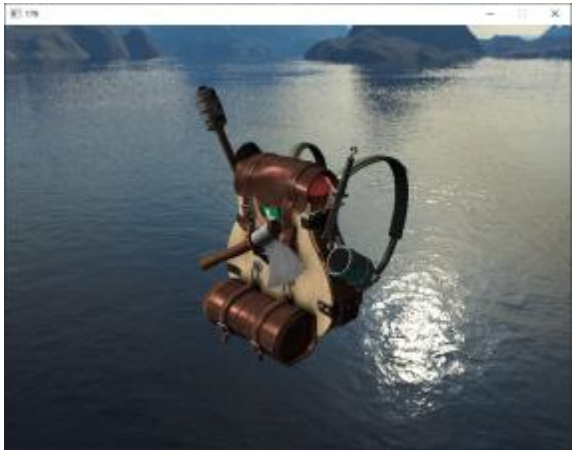
Пример работы

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

Система рефлексии

Система рефлексии — это механизм, позволяющий программе анализировать и взаимодействовать с собственными типами данных, функциями и структурами во время выполнения. C++ не имеет встроенной поддержки рефлексии, поэтому её реализация требует ручного подхода. Одним, из которых является обработка абстрактного синтаксического дерева (**AST**), которая предоставляет библиотека **LLVM Clang**. AST содержит полную информацию о структуре программы, включая типы данных, функции, классы, поля и методы.

```
CLASS() Point
{
public:
    PROPERTY()
    int x;

    PROPERTY()
    int y;

    PROPERTY()
    int z;

    FUNCTION()
    size_t Hash() const
    {
        return x ^ y ^ z;
    }
};
```

Первый класс

```
CLASS() NPCObject
{
public:
    PROPERTY()
    Point position;

    FUNCTION()
    char const *GetName() const noexcept
    {
        return m_name;
    }

    FUNCTION()
    void SetName(char const *name) noexcept
    {
        free(m_name);
        m_name = _strdup(name);
    }

private:
    char *m_name = nullptr;
};
```

Второй класс

```
------- START PARSE CLASS -------
Class name: Point
Property name: x
Property name: y
Property name: z
Function name: Hash()
------- END PARSE CLASS -------
------- START PARSE CLASS -------
Class name: NPCObject
Property name: position
Function name: GetName()
Function name: SetName(const char *)
------- END PARSE CLASS -------
```

Вывод информации классов

Сборщик мусора

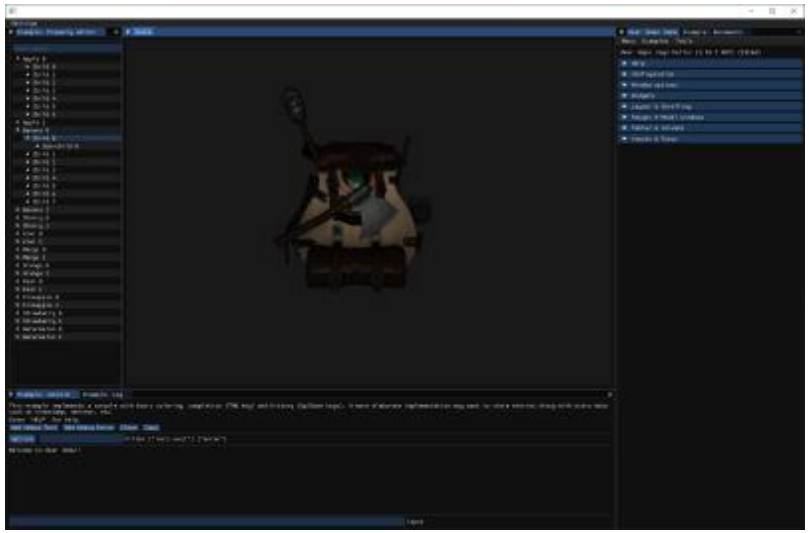
Сборщик мусора — это механизм автоматического управления памятью, который освобождает память, занятую объектами, больше не используемыми в программе. Он избавляет разработчика от необходимости вручную освобождать память, что помогает избежать утечек памяти и ошибок, связанных с неправильным управлением памятью.

Основные этапы работы:

1. Сборщик мусора начинает с корневого объекта наследованного от **RObject**
2. Он рекурсивно проходит по всем дочерним объектам, которые тоже наследуются от **RObject**, чтобы найти все достижимые объекты
3. Объекты, которые не могут быть достигнуты через цепочку ссылок из корней, считаются **"мусором"**
4. **Освобождение памяти** всех "мусорных" указателей

Заключение

В ходе выполнения данной работы были достигнуты все поставленные цели и задачи. В результате было создано эффективное и гибкое графическое программное обеспечение для визуализации трехмерных объектов на базе библиотеки OpenGL 4.6 и языка программирования C++. результаты данной работы имеют высокую практическую значимость и могут быть успешно применены в различных сферах, требующих визуализации трехмерных объектов.



Графическое ПО