**Разработка графического программного обеспечения для визуализации трехмерных объектов**

**[#2. 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

**[#3. Актуальность и обзор существующих решений]**

**Существующие решения из наиболее популярных:**

* Unreal Engine
* Unity
* CryEngine
* Godot

**Актуальность, из-за различных политических обстоятельств:**

* Отечественная разработка
* Импортозамещение

**[#4. Прототипирование]**

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

1. Работа и интеграция с различными операционными системами, такими как Windows и Linux. Для поддержки кроссплатформенности взята библиотека **GLFW**.
2. Отображение трехмерных объектов на любой операционной системе. Для поддержки используется библиотека **OpenGL**.
3. Удобный и практичный интерфейс для работы с трехмерными объектами на сцене. Аналогом используется библиотека **ImGUI**.
4. Загрузка текстур. Используется **SOIL**.
5. Загрузка сцены и различных трехмерных объектов. Используется библиотека **Assimp**.
6. Система рефлексии и сериализации данных. Для реализации используется библиотека **LLVM Clang**.
7. Сборщик мусора. Собственная реализация.

**[#5. Графическое окно]**

Для отображения окна была использована библиотека **Graphics Library Framework**. Она 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. На картинке представлено оконное приложение, для отображения которого использовалась данная библиотека.

**[#6. 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 состоит из 6 этапов:

* Вершинный шейдер. На вход которого приходят вершинные данные, на выход набор трехмерных координат.
* Формирование формы. Формирование набора вершин по которой будет строится геометрия.
* Геометрический шейдер. Из набора вершин формируются примитивы.
* Растеризация. Результат работы геометрического шейдера передаётся на этап растеризации, где результирующие примитивы соотносятся с пикселями на экране, формируя фрагменты.
* Фрагментный шейдер. Основная задача фрагментного шейдера заключается в вычислении конечного цвета пикселя, и именно на этом этапе чаще всего реализуются различные дополнительные эффекты.
* Тесты и смешивание. Этот этап проверяет значение глубины (и шаблона) фрагмента и использует их для определения местоположения фрагмента относительно других объектов: находится ли он спереди или сзади. Также выполняется проверка значений прозрачности и смешивание цветов

**[#7. 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:

На картинке представлены переходы: из локальных в глобальные координаты, из глобальных в видовые, из видовых в перспективную или ортографическую проекцию.

**[#8. 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. Эти модели описывают физику света на основе нашего понимания его природы и рассчитываются гораздо проще по сравнению с полным учетом всех факторов. Одной из таких моделей является модель освещения по Фонгу (Phong). Она состоит из трех основных компонентов:

1. Фоновое освещение (ambient): описывает общее освещение, которое равномерно распределяется по поверхности объекта.
2. Рассеянное/диффузное освещение (diffuse): учитывает свет, рассеивающийся равномерно по всем направлениям от источника света.
3. Бликовое освещение (specular): моделирует яркие блики, возникающие на поверхности объекта в направлении источника света.

Эти компоненты вместе позволяют создать визуально реалистичное представление освещения объектов. На картинке можете видеть, что они из себя представляют.

**[#9. 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.

На картинках можете рассмотреть два класса помеченных для рефлексии макросом CLASS(), свойства помеченные PROPERTY() и методы помеченные FUNCTION(). И на последней картинке можно посмотреть вывод системы рефлексии, где парсятся только помеченные классы, свойства и методы.

**[#10. 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

**[#11. 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. На картинке можете увидеть разработанное графическое ПО.