МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 «Прикладная информатика»
(код и наименование направления подготовки, специальности)

Бизнес-информатика
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка элементов редактора 3D моделирования

Студент Д.В. Шепляков
(И.О. Фамилия) (личная подпись)

Руководитель Н.Н. Рогова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Аннотация

Тема бакалаврской работы: «Разработка элементов редактора 3D моделирования».

Ключевые слова: технологии, элементы редактора, моделирование.

Цель данной бакалаврской работы – разработка элементов редактора 3D моделирования.

Актуальность темы ВКР обусловлена тем, что графические технологии широко используются в различных крупных отраслях: медицине, промышленности, индустрии развлечений. Это делает область 3D – моделирования основой для разработки программ для работы с 3D – графикой.

На рынке программ 3D - моделирования существует большое количество аналогов представленного проекта. Некоторые позволяют спроектировать 3D – сцену, другие – отдельную модель. Главное отличие проекта – возможность проектировать 3D - сцену с внедренными объектами, созданными без использования сторонних программ моделирования и обеспечивать им взаимодействие с графическими компонентами.

Структура выпускной квалификационной работы представлена введением, тремя главами, заключением, списком используемой литературы и приложением.

В первой главе проведен анализ существующих разработок и определены основные требования программному продукту.

Во второй главе произведено логическое проектирование разрабатываемой программы и определены ее функциональные возможности.

В третье главе представлена компонентная составляющая программы и результаты ее работы в процессе тестирования.

В заключении были сделаны выводы по результатам работы над ВКР.

Работа включает: страниц 68 с приложением, рисунков 23, таблиц 3, источников 20.

Оглавление

Введение

История развития 3D – моделирования идет из 1960 года, когда в университете города Юты (США) студентами Иваном Сазерлендом и Дэвидом Эвансом были созданы первые компьютерные программы, формирующие простые трехмерные модели на основе эскизов. Такие программы никто не воспринимал всерьез при первом представлении. Но через несколько лет на основе фундаментальных исследований, проведенных студентами, стали началом развития мощнейших графических технологий, которые используются до сих пор.

Объектом исследования является изучение основ реализации 3D – технологий.

Предметом исследования является создание и работа с объектами в 3D – среде.

Цель исследования - разработка элементов 3D – редактора для возможности создания сцены с использованием внедренных объектов.

Для достижения этой цели необходимо решить следующие задачи:

- осуществить поиск и анализ учебной и учебно-методической литературе;
- исследовать и проанализировать существующие разработки;
- сформулировать требования к программе;
- составить концептуальную модель программы;
- выполнить логическое моделирование;
- определить графические технологии реализации элементов редактора 3D-моделирования;
- реализовать элементы редактора 3D-моделирования;
- описать функционал разработанных элементов редактора 3D-моделирования;
- тестирование разработанных элементов редактора 3D-моделирования.

Теоретическая значимость исследования заключается в обосновании функционала разрабатываемого приложения на базе концептуальной модели проекта.

Практическая значимость исследования определяется наглядностью демонстрации базового программного интерфейса спроектированной и разработанной программы.

В первой главе был проведен анализ существующих редакторов для изучения их технологической составляющей, языков программирования для разработки редактора и сред разработки для сборки и компиляции программы.

Во второй главе была определена концептуальная модель программы, а также построена диаграмма классов, демонстрирующая их взаимодействие и положенная в основу разработки модульной системы. На основе концептуальной модели были реализован функционал и технологические возможности редактора.

В третьей главе были реализованы графические технологии на базе программного интерфейса, внедрена модульная система графических объектов, был реализован удобный и понятный интерфейс, который определяет поведение программы.

Глава 1 Анализ программ для 3D-моделирования

1.1 Понятие редактора 3D-моделирования

Редактор 3D – моделирования – среда для создания полигональных 3D-объектов в режиме реального времени с использованием различных графических технологий. Редактор – сложный инструмент, разрабатываемый группой людей. В связи с технологическим прогрессом возникает необходимость создавать редакторы, которые не будут обременять разработчика логикой работы внутренних процессов и функций и обеспечивать разнообразный подход к поставленной задаче.

«Правильно разработанный редактор позволяет создавать различные объекты, не предоставляя пользователю знания работы внутренней логики» [18, с. 175]. Обычно он имеет набор функциональных возможностей, каждая из которых состоит из набора сложных функций, а те, в свою очередь, реализуют логику на машинном уровне при работе с видеокартой или процессором.

Также редакторы могут содержать встроенную среду разработки для поддержания проектов сложным кодом, синтаксис которого может быть привязан к различным языкам программирования.

3D-редакторы при написании не обходятся без объектно-ориентированного программирования. Оно упрощает управление ресурсами и позволяет укомплектовать их в один объект, что добавляет удобства в понимании, а соответственно и поддержке кода.

«На настоящий момент существует множество 3D-редакторов, позволяющих более упрощено подходить к созданию 3D-приложений, и все они имеют как ряд достоинств, так и недостатков. Но эти недостатки не слишком хорошо видны без знания того, как работает редактор на низком уровне.» [3, с. 35].

1.2 Анализ существующих 3D-редакторов

Все игровые редакторы направлены на создание объектов в пространстве и управление ими, без влияния каких-либо условностей со стороны – физики, освещения или типа материала. Условностей, влияющих на объекты, намного больше, но изначально упор делается на положение объекта в пространстве и его форму. Это общий признак всех существующих редакторов, но каждый из них по своему подходит к этому процессу – один позволяет лишь создать объект из заготовленных ресурсов, а другой предоставит возможность создать объект с нуля, используя необходимые данные для загрузки их в память видеокарты. Некоторые редакторы делают упор на производительность, другие – на графическую составляющую, используя при этом всю технологическую мощь. Для понимания необходимо провести анализ самых распространенных редакторов для создания 3D-сцен и сравнение в некоторых обобщенных аспектах, таких как:

- простота в освоении;
- знание языка программирования;
- уровень оптимизации;
- технологические достижения.

«Unity - имеет простой Drag&Drop интерфейс, который легко настраивать, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в редакторе. » [19, с.43]. Движок использует для написания скриптов язык C#. Очень популярен среди начинающих разработчиков, но нестабилен в плане оптимизации и потребления памяти в силу специфики своей архитектуры. Обладает большим количеством достоинств – например, межплатформенная поддержка и визуализация среды разработки, что делает его одним из самых простых в освоении. Интерфейс редактора представлен на рисунке 1.

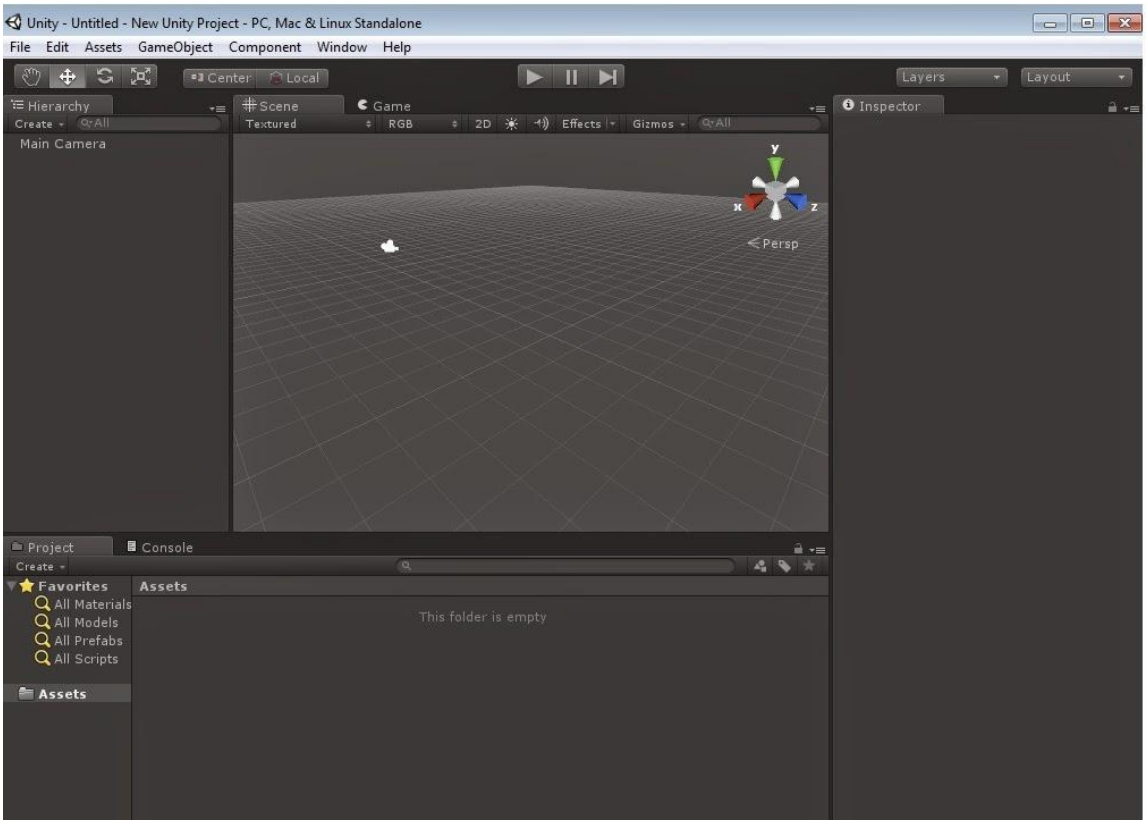

Рисунок 1 - Окно 3D-редактора Unity

Unreal Engine - один из первых универсальных игровых редакторов; он совмещал в себе графический движок, физический движок, искусственный интеллект, управление файловой и сетевой системой и готовую среду разработки для игр. Многие вещи, реализованные в данном редакторе, были революционными, а сам редактор является очень популярным. При массе достоинств он обладает лишь одним недостатком – разработка игр в данном редакторе сопровождается высокими графическими нагрузками и для каждой новой версии редактора видеокарта прошлого поколения не будет давать желаемую максимальную эффективность. Сложность разработки игры не является высокой, однако для продуктивной работы и качественной оптимизации необходимо знание C++. Пример окна интерфейса представлен на рисунке 2.

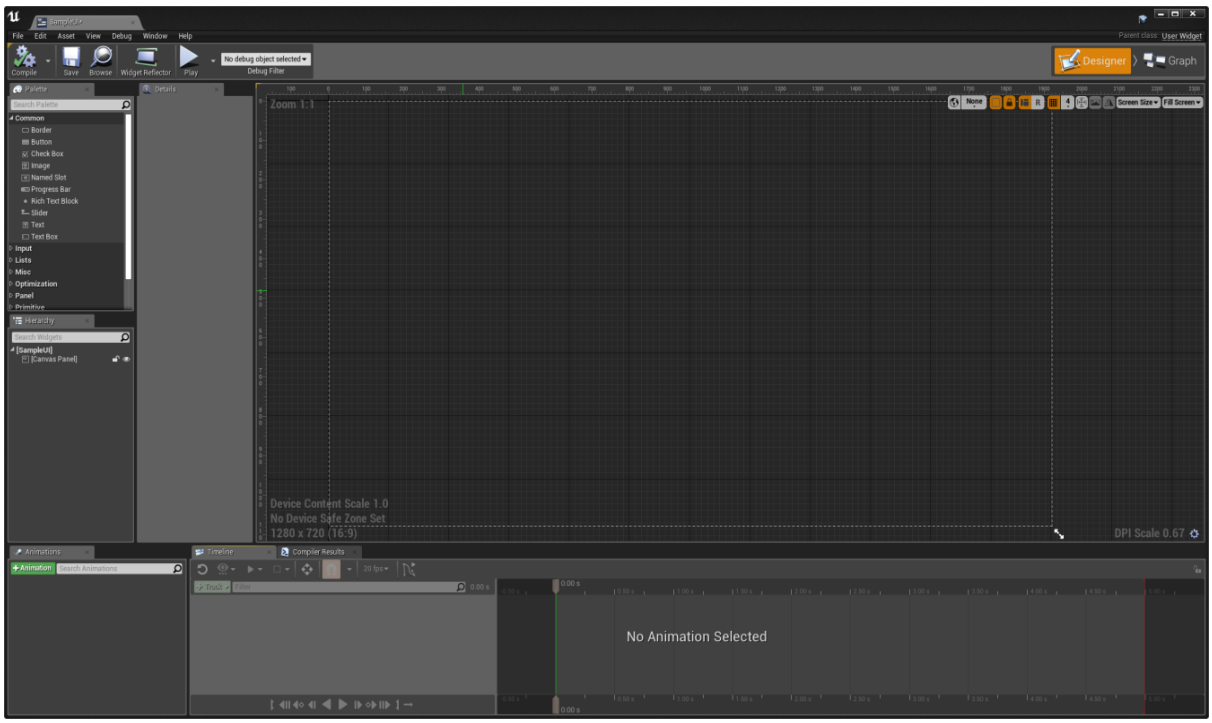

Рисунок 2 - Окно 3D-редактора Unreal Engine

Cry Engine – мощный и технологически продвинутый игровой редактор, реализовывающий различные технологические возможности. Примером могут служить многоступенчатые шейдеры для просчета освещения для сложно-освещенных объектах, аналоги которых у других редакторов имелись, но не являлись настолько проработанными. Данный редактор имеет очень продвинутую графическую основу, и оптимизация проекта является неотъемлемой частью работы и не может быть представлена в виде примитивных скриптов. Это делает его одним из сложных инструментов разработки, а знание языка C++ требуется углубленное. Окно пустого проекта показано на рисунке 3.

Рисунок 3 - Окно редактора Cry Engine

Сравнение общих показателей в 3D-редакторах представлены в таблице1.

На основе более простых функций можно изучить логику их работы. В качестве определяющих критериев были выбраны следующие:

- понятность интерфейса определяется степенью удобства и сложностью для понимания;
- технологические возможности определяются сложностью реализации элементарных технологий;
- сложность оптимизации кода основывается на работе проекта в системе без различных ошибок;
- степень оптимизации без применения языка разработки показывает, насколько возможно оптимизировать проект с помощью скриптов редактора;
- объем потребления памяти определяет возможность редактора корректно управлять и освобождать ее.

Каждый из критериев имеет степень сложности в диапазоне от низкого до высокого, который определяет результативность работы с определенным редактором.

Таблица 1 - Сравнение общих показателей в 3D-редакторах

Критерий	Unity	Cry Engine	Unreal Engine
Понятность интерфейса	высокая	низкая	низкая
Технологические возможности	средние	высокие	высокие
Сложность оптимизации кода	средняя	средняя	средняя
Степень оптимизации без применения языка разработки	низкая	средняя	средняя
Объем потребления памяти	высокая	высокая	средняя

На основе проведенного анализа вышеперечисленных редакторов была установлена общая концепция в работе с объектами, которая представляет собой элементы управления и взаимодействия на основе простых технологий, которые в свою очередь являются основополагающими для каждого редактора. Были определены положительные и отрицательные стороны каждого редактора. Данные в таблице позволяют сделать вывод, что редактор Unity наиболее подходит для изучения графических технологий и их реализации на низком уровне, чем более технологически развитые редакторы с усложненным устройством.

В работе будут реализованы основные элементы данных редакторов:

- возможность управления объектом;
- возможность текстурирования объекта;
- освещение объекта;
- возможность покраски объекта.

При работе будет сделан упор на простоту и понятность интерфейса, а также удобство сопровождения кода и контроля памятью.

1.3 Обоснование выбор средств разработки элементов редактора 3D моделирования

1.3.1 Выбор языка программирования

Выбор языка программирования является основой для разработки, определяющий результат разработки. Поэтому необходимо провести краткий анализ некоторых языков.

ООП поддерживают множество языков: Java, C++, C#. Такие языки являются основными и используют возможности ООП по максимуму. Но, все языки отличаются друг от друга в той или иной степени и предназначены для различных целей. «Само ООП представляет собой часть языка, направленная на взаимодействие объектов, созданных по шаблону класса, описывающий этот объект определяющий его параметры. » [14, с.38].

«C++ - компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции.» [6, с. 54]. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником – языком C, - наибольшее внимание уделено поддержке объектно-ориентированного и обобщенного программирования.

«Java – объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems(в последующем приобретенной компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине

вне зависимости от компьютерной архитектуры.» [15, с. 102]. Программы, написанные на Java, более медленные и занимают больше оперативной памяти, чем написанные на языке C++, что делает данный язык не очень привлекательным для разработки графических компонентов редактора.

C# - объектно-ориентированный язык программирования. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. C# полностью поддерживает концепции объектно-ориентированного программирования, включая инкапсуляцию, наследование и полиморфизм. Но данный язык спроектирован быть кроссплатформенным, однако его развитие не пошло в этом направлении. Также код на C# тяжело оптимизировать, и он требователен к ресурсам, что дает C++ большие преимущества перед данным языком.

Сравнительный анализ языков программирования представлен в таблице 2.

Таблица 2 - Сравнение общих показателей языков программирования

Критерий	C++	C#	Java
Сложность разработки	высокая	средняя	средняя
Степень оптимизации кода	высокая	средняя	средняя
Сложность переносимости кода	средняя	средняя	низкая
Степень управления ресурсами	высокая	высокая	средняя

В данной таблице были определены следующие критерии:

- сложность разработки;
- степень оптимизации кода;
- сложность переносимости кода определяется зависимостью ее от конкретной платформы;

– степень управления ресурсами определяется манипуляциями с ними с использованием различных принципов.

Каждый из критериев имеет степень сложности в диапазоне от низкого до высокого, который определяет результативность разработки программы с помощью языка разработки.

Из кратких описаний вышеперечисленных языков можно сделать вывод, что для разработки подобного рода приложения больше подходит язык C++ в связи с высоким контролем кода и его универсальностью.

1.3.2 Выбор среды разработки

«C++ поддерживается множеством компиляторов, но у многих есть проблемы на этапе подсоединения библиотек или мультиплатформенностью. »[2, с. 271] . Некоторые компиляторы подходят для сборки программ на одной платформе. Существуют так же компиляторы, специализирующиеся на сборке для различных платформ, но работа с ними сопровождается множеством настроек и включений.

Самыми популярными средами для разработки являются:

– Dev C++,
– C++ Builder,
– GCC,
– Visual Studio.

Для более полной оценки среды разработки были установлены такие критерии, как:

– простота в использовании;
– степень управляемости в разработке графических приложений;
– сложность сборки проекта;
– уровень оптимизации.

«Dev C++ является довольно мощным инструментом, разработанным на Delphi. » [17, с. 252] Эта среда состоит из редактора кода, где можно писать программу и компилятора, транслирующего написанный код в

машинный язык. Он очень прост для новичков, но создать программу для работы с графическими объектами довольно трудно. Однако компилятор позволяет с помощью базовых надстроек оптимизировать проект под нужды системы.

C++ Builder – программный продукт, ориентированный на упрощенную и быструю разработку и объединяет в себе комплекс объектных библиотек. Его недостаток – высокая сложность контролирования низкоуровневых процессов, а это является необходимым при разработке подобного рода приложений.

GCC - набор компиляторов для различных языков программирования. Это приводит к трудностям корректной компиляции и усложнению сборочных скриптов при написании кода. Но в противоположность к этому уровень оптимизации довольно высок в рамках одной системы, чего нельзя сказать о переносимости кода.

Visual Studio является хорошим выбором для разработки приложений под ОС Windows в связи с своей гибкостью и простотой. «Среда представляет собой великолепный инструмент для разработки, который располагает огромными возможностями и поддерживает внушительное количество библиотек. » [14, с.119] С обновлением SP1 для Windows 7 данная среда приобретает еще большее количество библиотек. Пакет SP1 содержит необходимую для программы библиотеку <memory.h>.

В результате проведенного анализа было установлено, что среда разработки Visual Studio подходит для разработки подобного рода программ и является компромиссом между простотой разработки и сложностью оптимизации, избегает недостатков других сред разработки и демонстрирует положительные стороны.

Для разработки графических компонентов необходимо задействовать библиотеку DirectX. Данная библиотека позволит избежать сложностей работы с графическими устройствами, предоставив набор интерфейсов.

1.3.3 Инструменты графического пакета DirectX

«DirectX – это набор программных интерфейсов приложения (далее – API), разработанных для решения задач, связанных с программированием под Microsoft Windows.» [20, с.493]. Он предоставляет большое количество функций, которые являются интерфейсами для обращения к видеокарте для исполнения низкоуровневых инструкций. Для разработки программы с реализацией элементов редактора 3D – моделирования будет использоваться основной интерфейс Direct3D – DirectX Graphics. Он подразделяется на два интерфейса – DirectDraw и DirectX3D(далее – D3D).

DirectDraw является интерфейсом вывода растровой графики. Из предлагаемых данным интерфейсом инструментов будет использоваться технология отсечения поверхностей – невозможность отрисовки невидимых поверхностей для снижения потребления памяти, а также технологии работы с материалом, цветом и возможности установки экранной формы приложения.

DirectX 3D (далее – D3D) – интерфейс вывода трехмерных примитивов. Предоставляет необходимые инструменты для построения полигонов с установленными параметрами освещения, затенения и наложенными текстурами на поверхность.

После описания необходимых средств разработки следует определить задачу на разработку ПО.

1.4 Разработка модели процесса создания 3D - сцены

3D-редакторы создаются для упрощения работы с графикой, позволяя конечным пользователям избегать сложных ситуаций при взаимодействии с аппаратной системой и избавляя от знания логического устройства низкоуровневых функций. Для продуктивной работы необходимо знать логику работы внутренних программных функций. Для простоты понимания работы программы ее сопровождает техническая документация, в которой

полностью описываются методы взаимодействия программы и аппаратного составляющего. При возникновении проблем следует обратиться к помощи сотрудников технической поддержки, которые сопровождают свой продукт и исправляют его недоработки.

Это необходимо для двух конкретных целей - для создания собственного проекта или для разбора существующего с целью его изменения, которое возможно при условии открытости исходного кода, зависящее от решения разработчика проекта. Если конечным пользователем является человек, имеющий доступ к коду продукта, он может быть одновременно и пользователем, инициирующим весь бизнес-процесс заново. Схематически вся бизнес-модель представлена с помощью диаграммы нотации IDEF0 на рисунке 5 ниже. Обоснование выбора данной методологии заключается в «полноте описания бизнес-процесса, которая достигается за счет наличия средств, отображающих управляющие воздействия, обратные связи по управлению и информации. » [12, с.59]

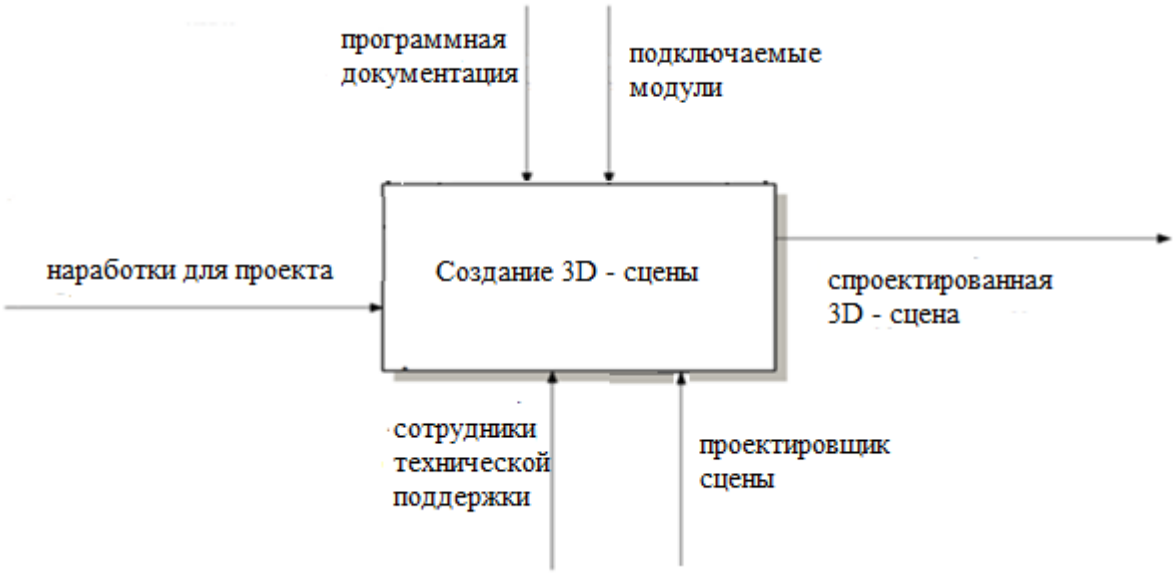

Рисунок 5 - Контекстная диаграмма бизнес-процесса «Создание 3D - сцены» в нотации IDEF0

Как видно на диаграмме, в данной модели на входе объектом преобразования являются «наработки для проекта» - графические и схематические наброски, заметки, звуковые записи и базовые примитивы объектов, которые могут использоваться в будущем проекте и представляться как сторонние модули или исходный проект, предназначенный для доработки.

Доступ к программной документации следует осуществлять из программы, упрощая доступ к ней. Сама документация представлена в виде текстового файла, который находится в корневой папке программы, что обеспечивает корректный доступ к файлу при перемещении программы.

Сторонние модули, описывающие примитивы объектов, должны быть согласованы с сотрудниками технической поддержки для включения их в составляющую программы и доработку кода программного продукта для возможности создания пользовательского объекта.

Для большей наглядности необходимо декомпозировать диаграмму бизнес-процесса «как должно быть». Она представлена на рисунке 6.

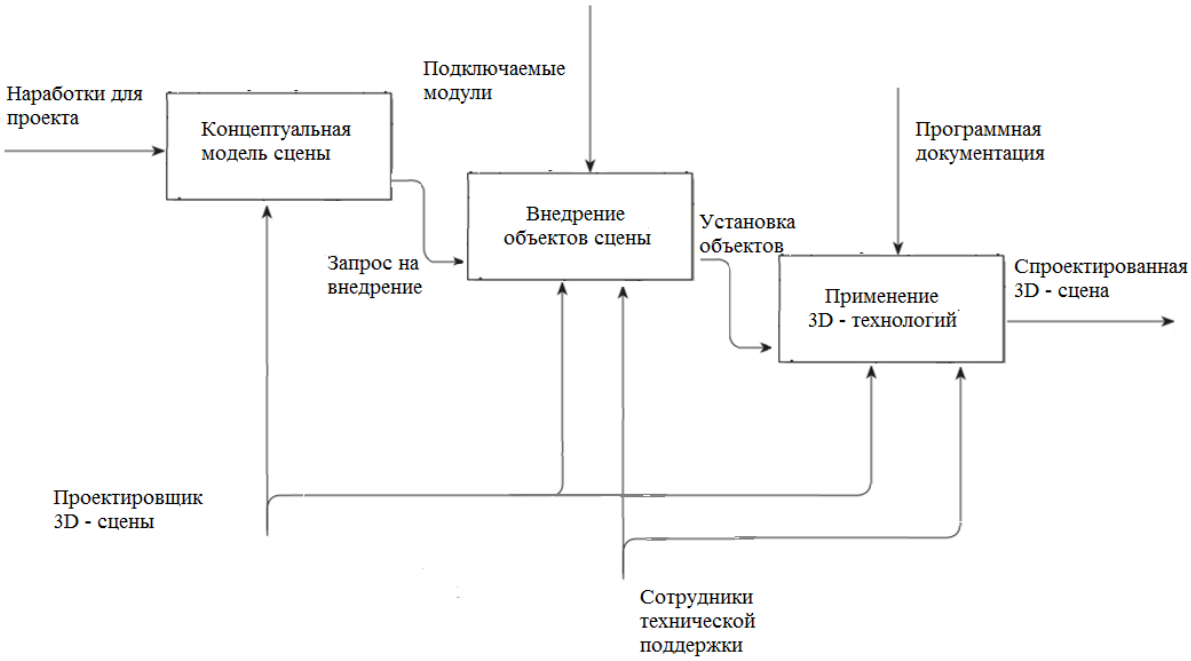

Рисунок 6 – Декомпозиция диаграммы бизнес-процесса «Создание 3D - сцены» в нотации IDEF0

Декомпозиция диаграммы бизнес – процесса представляет собой серию последовательных операций:

- создание концептуальной модели сцены – содержит наброски сцены, аудиофайлы и предполагаемые объекты для внедрения, которые будут в дальнейшем внедрены в проект;
- внедрение объектов сцены – представляет собой модули с данными объекта, которые внедряются сотрудником технической поддержки;
- применение 3D – технологий – установка настроек для объекта для демонстрации его на сцене.

Данная диаграмма демонстрирует необходимые шаги для проектирования 3D – сцены. Внедрение объектов в проект происходит при необходимости.

1.5 Постановка задачи на разработку программы

Прежде чем определить задачи на разработку программы, необходимо кратко описать модули, из которых будет состоять программа. Таких модуля будет четыре. Два из них будут описывать состояние фигур. Еще один – совмещать в себе данные, которые являются общими для всех типов фигур. Последний модуль будет содержать в себе основной код программы, который определит ее поведение и взаимодействие с другими модулями.

Такая программная организация позволяет определить еще одно эксплуатационное назначение. Если основное значение программы - создание 3D-сцены с объектами и с возможностью их контроля, то дополнительное определяется возможностью присоединения сторонних модулей к основной программе с незначительными изменениями в ней.

Всю работу программы можно представить графически с помощью блок-схем (рисунок 7).

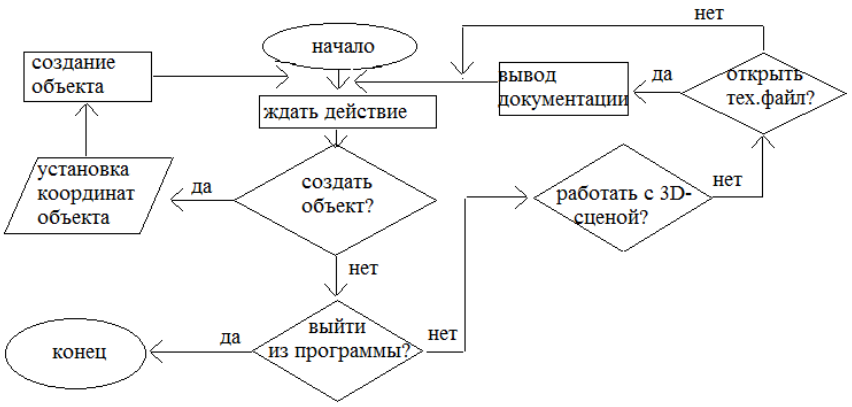

Рисунок 7 – Блок-схема алгоритма работы программы

Особенностью такой программы является ее способность к автономной работе без участия пользователя. Это означает, что редактор должен работать в цикле, обновляя сцену каждый шаг цикла.

Фоновый буфер используется в качестве памяти, в которую временно записывается изображение, подлежащее отображению на экране, без его выдачи на видеокарту. Если изображение выдавать непосредственно в видеопамять экрана, то будет наблюдаться мерцание; поэтому информация записывается и быстро обрабатывается в фоновом буфере. Указанный «фоновый буфер обычно расположен внутри физической памяти оперативного запоминающего устройства ОЗУ (random-access memory - RAM) видеокарты и карты графического ускорителя» [20, с. 9] Пользователь на данный процесс не имеет влияния, и единственная остановка программы происходит по желанию пользователя. Исключением может быть аварийная остановка программы по техническим причинам.

После определения алгоритма работы программы нужно определить задачи, необходимые к выполнению в процессе разработки:

– разработать главное окно программы;

- разработать меню программы;
- программно определить возможность создания объекта с параметрами;
- реализовать возможность открытия технического файла;
- разработать модель рендеринга 3D – сцены;
- реализовать графические технологии с их последующим внедрением;
- разработать модульные части проекта с содержанием классов объекта.

По завершению разработки необходимо протестировать программу на наличие ошибок исполнения и работоспособность функциональной части программы.

Вывод по главе

В первой главе проанализированы достоинства и недостатки популярных редакторов. В процессе анализа определены основные технологии для реализации. Обоснован выбор средств разработки. Установлен алгоритм работы программы и на его основе определены задачи разработки.

Глава 2 Логическое проектирование элементов редактора 3D-моделирования

2.1 Выбор используемых подходов к проектированию

Следующим этапом разработки должно стать рабочее проектирование. Поскольку большая часть современных программных решений реализуется с использованием объектно-ориентированных языков программирования, то использование объектно-ориентированных подходов к проектированию является наиболее оправданным.

Для унификации описания различных этапов и результатов анализа и проектирования программной системы в рамках объектно-ориентированного подхода используют UML-моделирование.

«Унифицированный язык моделирования (UML) - это семейство графических нотаций, в основе которого лежит единая метамодель. Он помогает в описании и проектировании программных систем, в особенности систем, построенных с использованием объектно-ориентированных технологий. » [4,с.36]. Существует несколько графических нотаций стандарта UML:

- диаграмма классов - статическая структурная диаграмма, описывающая структуру системы, демонстрирующая классы системы, их атрибуты, методы и зависимости между классами;
- диаграмма компонентов - статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты;
- диаграмма композитной/составной структуры - статическая структурная диаграмма, демонстрирует внутреннюю структуру

классов и, по возможности, взаимодействие элементов (частей) внутренней структуры класса;

- диаграмма объектов - демонстрирует полный или частичный снимок моделируемой системы в заданный момент времени. На диаграмме объектов отображаются экземпляры классов (объекты) системы с указанием текущих значений их атрибутов и связей между объектами;
- «диаграмма деятельности - диаграмма, на которой показано разложение некоторой деятельности на её составные части. » [5, с.203]. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий, соединённых между собой потоками, которые идут от выходов одного узла к входам другого;
- диаграмма вариантов использования - диаграмма, на которой отражены отношения, существующие между актерами и вариантами использования.

В данной работе для общего описания функционала продукта и существующих ролей используется диаграмма взаимодействия; для определения архитектуры и составляющих ее компонентов используется диаграмма компонентов; для уточнения разработанной концептуальной модели и создания спецификации разрабатываемого программного продукта используется диаграмма классов; для отображения взаимодействия во времени между модулями системы используется диаграмма последовательности; для отображения возможных вариантов работы программы используется диаграмма деятельности.

2.2 Разработка логической модели программы

2.2.1 Диаграмма вариантов использования

Первым шагом моделирования является создание концептуальной модели в виде диаграммы вариантов использования. Определяем актеров и варианты использования. Актерами данной системы являются:

– проектировщик 3D-сцены;

– сотрудник технической поддержки – при обращении проектировщика присоединяет необходимый модуль с данными;

– пользователь – является конечным пользователем и оценивает конечный продукт.

У проектировщика может быть несколько вариантов использования редактора. Ими являются:

– возможность обращения в техническую поддержку;

– просмотр технической документации;

– создание 3D-сцены.

На рисунке 8 представлена диаграмма вариантов использования программы и взаимодействие модулей в ней. Для ее представления использовался язык UML.

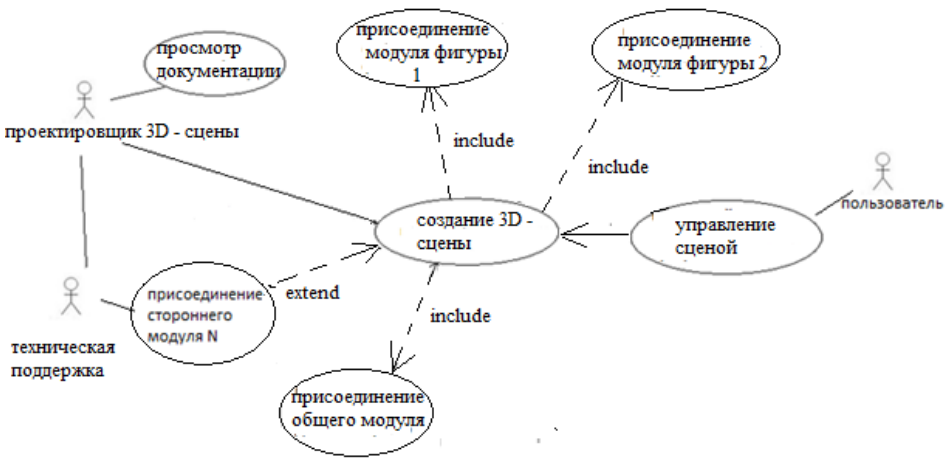

Рисунок 8 – UML – диаграмма взаимодействия актеров с редактором

При работе с редактором пользователь может совершать некоторые действия, определяющие поведение программы. Он имеет возможность присоединить модуль к редактору и изучить техническую документацию. Основной работой с редактором является создание 3D-сцены, в которой пользователь использует свои наработки и реализованные в программе технологии. В результате объединения данных модулей и выбора способа изменения состояния 3D-сцены происходит обновление изображения и вывода содержимого фонового буфера на экран.

Полученная диаграмма взаимодействия отображает варианты использования проектировщиком программного продукта и позволяет определить результаты взаимодействия проектировщика и других актеров, что дает более точное представление того, какие основные задачи должна выполнять программа.

2.2.2 Диаграмма классов

Следующий этап проектирования - разработка внутренней статической структуры программного продукта как совокупности классов-сущностей, обладающих внутренним состоянием и поведением.

На диаграмме классов (рисунок 9) основным классом программного продукта является родительский класс Object, обеспечивающий последовательное заполнение полей объекта данными фигуры. Родительский класс является общим для наследуемых и содержит общие переменные для всех типов фигур, которые могут изменяться и содержать отличную от других объектов информацию, которая не может объявляться как static.

Класс является абстрактным. «Абстрактность позволяет описать класс так, что экземпляр данного класса невозможно сконструировать, а класс определяется лишь как часть контейнера для основного объекта. » [5, с.205]

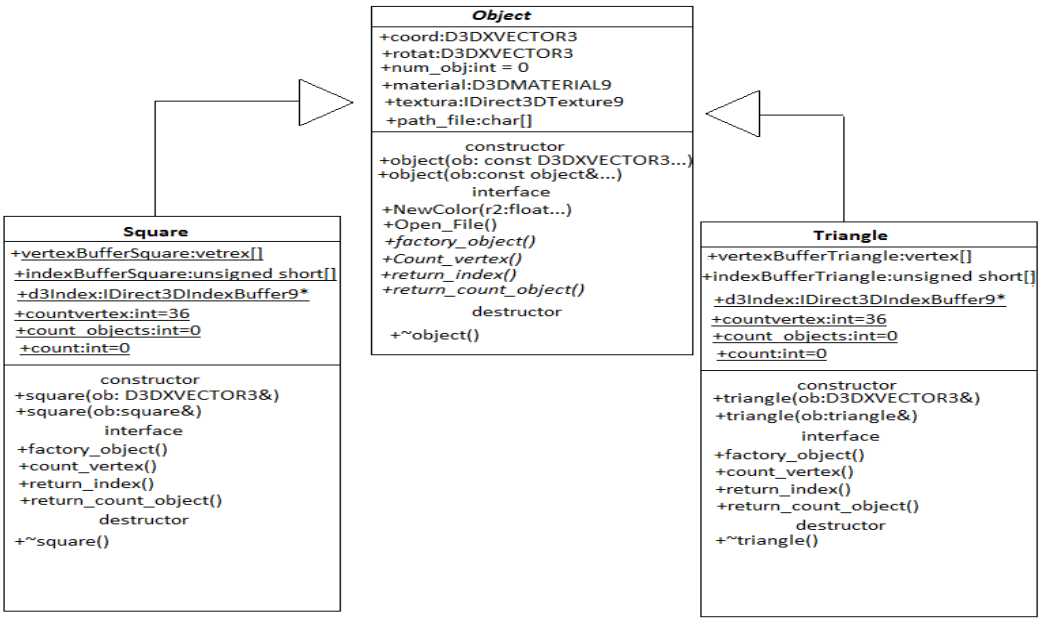

Рисунок 9 – Диаграмма классов

Компоненты класса представляют собой структурированные данные и методы для работы с ними. Метод NewColor() определяет цвет материала material, OpenFile() позволяет получить путь к файлу текстуры и представить его как массив символов path_file[]. Создание и копирование объектов происходит с помощью метода factory_object(), что вызывает конструкторы класса и инициализирует поля родительского и наследуемого класса. «Задача конструктора - выделение памяти под объект и незамедлительная инициализация всех переменных. » [3, с. 73]. Переменные coord и rotat отвечают за сохранение координат и состояния поворота объекта. Деструктор ~object, выполняет стандартные процедуры освобождения памяти. В проекте определены несколько указателей на участки памяти, которые без необходимых инструкций не могут освободить зарезервированную память самостоятельно. Вместе с очисткой памяти деструктор уменьшает счетчик объектов своего класса. На диаграмме классов присутствуют вспомогательные наследуемые классы. Они содержат информацию, присущую только определенному типу фигур. Указатели содержат ссылки на

блоки памяти с данными объекта для видеокарты, переменная countvertex определяет количество вершин у каждого объекта, count_objects и count отвечают за подсчет количества объектов в программе. Все методы в наследуемых классах являются виртуальными и их реализация происходит в родительском классе, получая от наследуемых объектов данные, определяющие тип объекта.

Все классы расположены в подключаемых модулях и выполняют функции определения объекта как набор данных для отображения на сцене. Данные в классе для упрощения доступа к ним объявлены как public – это обеспечивает прямой доступ без промежуточных методов.

«Некоторые члены в классах обладают модификатором static, что позволяет определить единственную копию переменной для всех объектов. » [9, с. 64] Такие переменные инициализируются вне класса в глобальной области программы. Это «упрощает читаемость кода и обеспечивает эффективное управление памятью, избавляя от надобности выделения памяти копиям переменной, которая не изменяется в течение всей программы. » [14, с.403]. Из того, что все переменные наследуемых классов обеспечивают неизменное состояние создаваемых объектов касаемо их формы, следует, что все переменные в наследуемых классах будут объявлены как static.

Инициализирующее значение некоторых переменных в диаграмме опущено по причине сложности самой инициализации. Указатели на буфер переменных, определяющих вершины объектов, имеют сложную структуру и инициализируются при запуске программы. Инициализирующее значение представляет собой набор структур, каждая из которых имеет четыре значения типа unsigned int, определяющая точку вершины в пространстве с локальной системой координат. Это обусловлено тем, что каждый объект имеет свою локальную систему координат, которая располагается в глобальной системе координат. Объект имеет свои собственные глобальные

координаты, но его вершины задаются исходя из локальной системы координат.

Каждый из определяемых классов находится в отдельном заголовочном файле и пользователь для создания собственного объекта может подключить заголовочный файл к программе при условии, что его организация совпадает с организацией, представленной на диаграмме.

2.2.3 Диаграмма последовательности

Динамика выполнения любого сценария или метода класса может быть описана в нотации диаграммы последовательности. На рисунке 10 представлена диаграмма последовательности для варианта использования «Создание новой сцены».

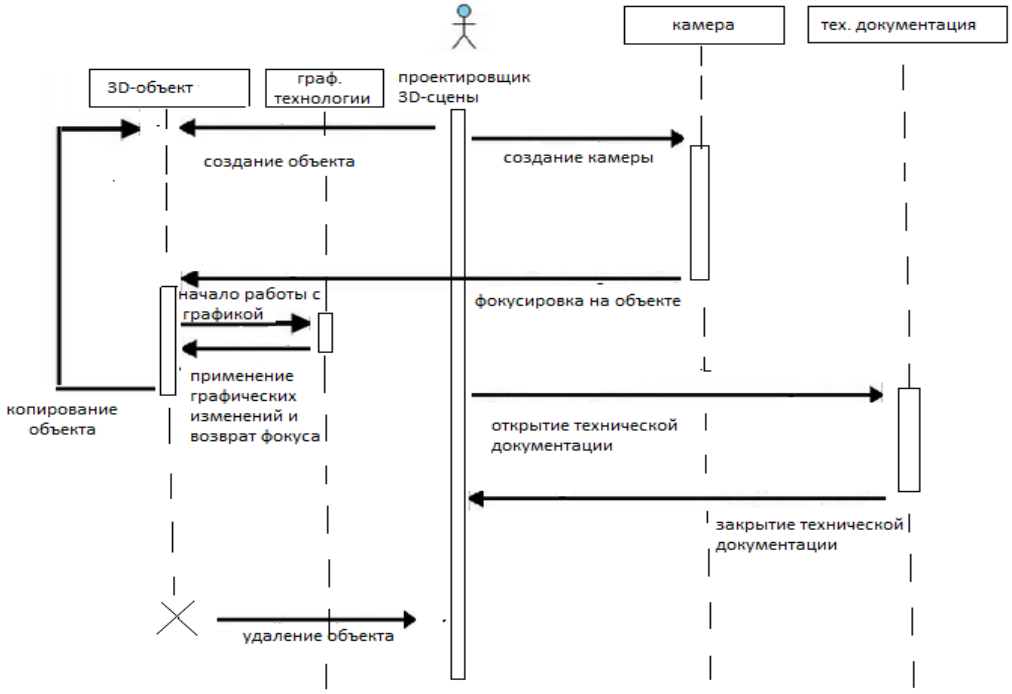

Рисунок 10 – Диаграмма последовательности для варианта использования «Создание новой сцены»

На диаграмме последовательности для вариантов использования «Создание новой сцены» показан жизненный цикл сцены, управляемой

проектировщиком. Каждый шаг цикла сохраняет изменения сцены с предыдущего шага, а действия, доступные для выполнения, определяются наличием объектов, для которых данное действие может быть применено. Взаимодействие с программой возвращает управление ею проектировщиком.

2.3 Определение графических технологий для реализации

В процессе анализа редактора Unity были определены простые графические технологии, которые определяют принцип работы редактора. Для реализации в программе простые технологии следует разделить на необходимые и второстепенные. Второстепенные технологии работают на основе необходимых, используя результаты их расчетов. Из этого следует то, что второстепенные технологии будут реализованы после успешной разработки и внедрения необходимых технологий.

Необходимыми технологиями являются:

- управление объектом – определение угла вращения объекта вокруг своей оси;
- копирование объекта – создание нового объекта на основе имеющегося со всеми его атрибутами;
- текстурирование – возможность наложения картинки на поверхность объекта;

Второстепенными технологиями являются:

- освещение объекта;
- затенение объекта – под действием освещения предоставляет возможность создания тени;
- покраска объекта.

Такой подход к реализации обеспечит поэтапную разработку технологий, дополняя существующие методы и функции программы.

Вывод по главе

В данной главе была произведено логическое проектирование программы, определено содержание модулей программы и их взаимодействие между собой. Определен порядок реализации технологий в проекте по уровню их сложности и взаимосвязанности.

Глава 3 Физическое проектирование элементов редактора 3D – моделирования

3.1 Разработка диаграммы компонентов программы

Для определения физической модели разрабатываемой программы и выделения основных этапов проектирования необходимо создать диаграмму компонентов. Диаграмма компонентов показывает связи между модулями, из которых состоит моделируемая программа (рисунок 11).

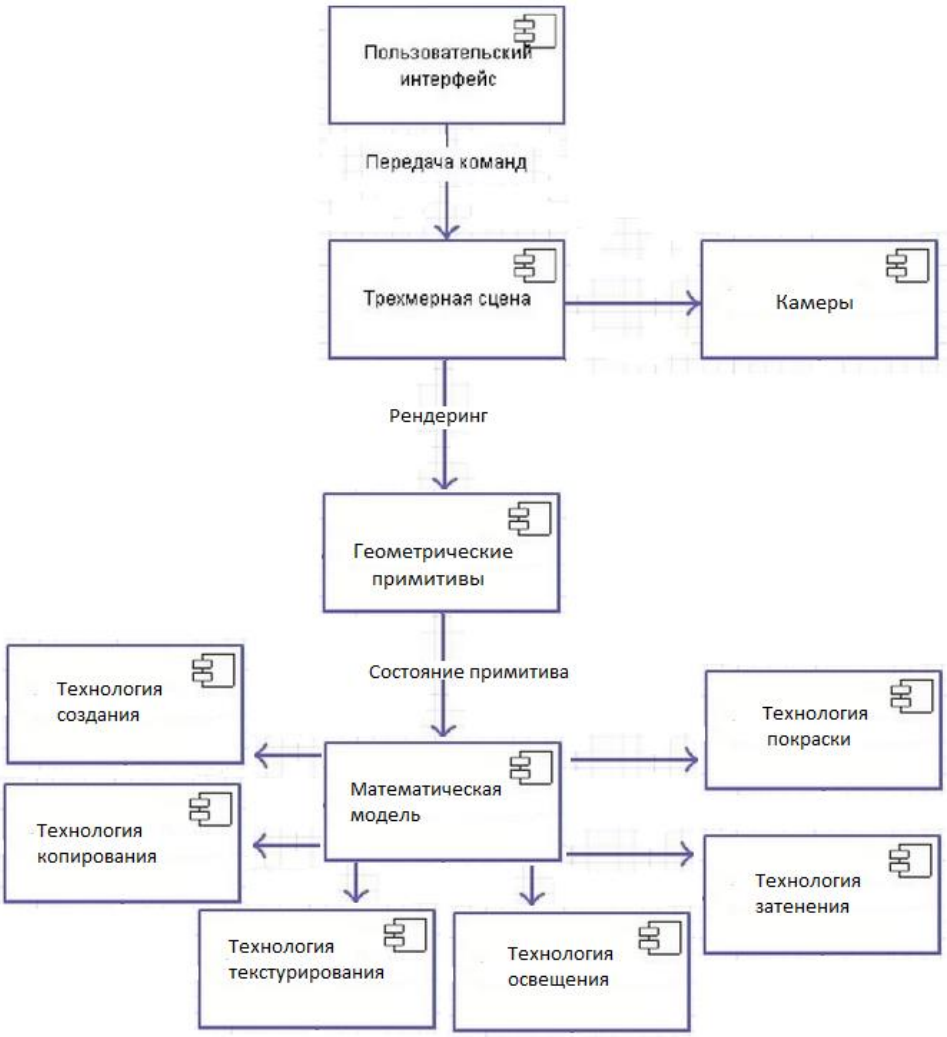

Рисунок 11 – Диаграмма компонентов моделируемой программы

Из диаграммы компонентов следует, что технологии работают по определенной математической модели, расчеты которой позволяют получить результат при любом изменении состояния сцены. Геометрические примитивы определяют работу математической модели в зависимости от шаблона данных примитива. Трехмерная сцена путем постоянного непрерывающегося рендеринга обеспечивает состояние примитивов и камер, а интерфейс программы при помощи команд от пользователя или системы влияет на изменения сцены.

В рамках данной работы было определена физическая модель, представленная на диаграмме компонентов и способ разработки программы «сверху вниз». Это означает, что сначала будет разработан интерфейс приложения, затем 3D – сцена с набором данных примитива, а после с помощью функций API Directx определена математическая модель расчетов построения объектов и базирующиеся на ней графические технологии.

3.2 Разработка диаграммы деятельности программы

3D-редактор должен осуществлять как работоспособность функций по работе в 3D-пространстве, так и предоставлять необходимую степень удобства при использовании. Для достижения этой цели необходимо рассмотреть оформление программы как часть функционала. Если управление объектом с клавиатуры и копирование – это функции, на которых основывается логика программы, то меню, списки объектов, флаги выбора – функции, определяющие степень доступности и удобства для пользователя. Из этого следует, что необходимо разделить задачи на подзадачи.

Для обеспечения удобства посредством создания понятного интерфейса необходимо выполнить следующие подзадачи:

– создать окно с рамкой и возможностью закрытия программы;
– создание оконного меню с возможностью выбора подменю;

– для определения формы объекта создать отдельное дочернее окно с флагами выбора;
– определить в меню позицию для открытия файла с техническим описанием;
– создать списки для отображения создаваемых объектов.

Определение логики программы так же можно разделить на две категории – выполнение базовых функций и функции оптимизации. Если в первом случае можно увидеть результат по написанию программы и ее сборке, не зная о ее внутренней реализации, то во втором случае пользователь не сможет обнаружить без сторонних утилит недоработки в оптимизации продукта.

Выполнение базовых функций было определено в ТЗ в разделе функциональные требования. Функции оптимизации могут быть достигнуты выполнением следующих подзадач:

– обеспечить модульность данных (это способствует не только читаемости кода и возможности избавить загрузку ненужных участков кода, но так же позволяет на ходу создавать свой модуль и включать его в проект с небольшими изменениями в самой программе);
– избежать фрагментации памяти или предотвратить повторное резервирование уже используемой памяти;
– обеспечить корректное освобождение памяти при обычном или аварийном закрытии программы;
– изолировать память видеокарты от общей памяти;
– реализация очистки и освобождения памяти при удалении объекта.

Все вышеперечисленные диаграммы демонстрируют принцип работы и возможности действий в рамках проектируемой системы. Но таким способом трудно описать последовательность шагов, которые необходимы для

выполнения того или иного действия. Примером может служить установка объекта по соответствующим координатам пространства, которая имеет более сложную логику выполнения и для представления порядка выполнения действий для достижения нужного результата нужно создать обобщенную диаграмму деятельности.

Диаграмма на рисунке 12 определяет все возможные вариации работы с программой и демонстрирует ее цикличность.

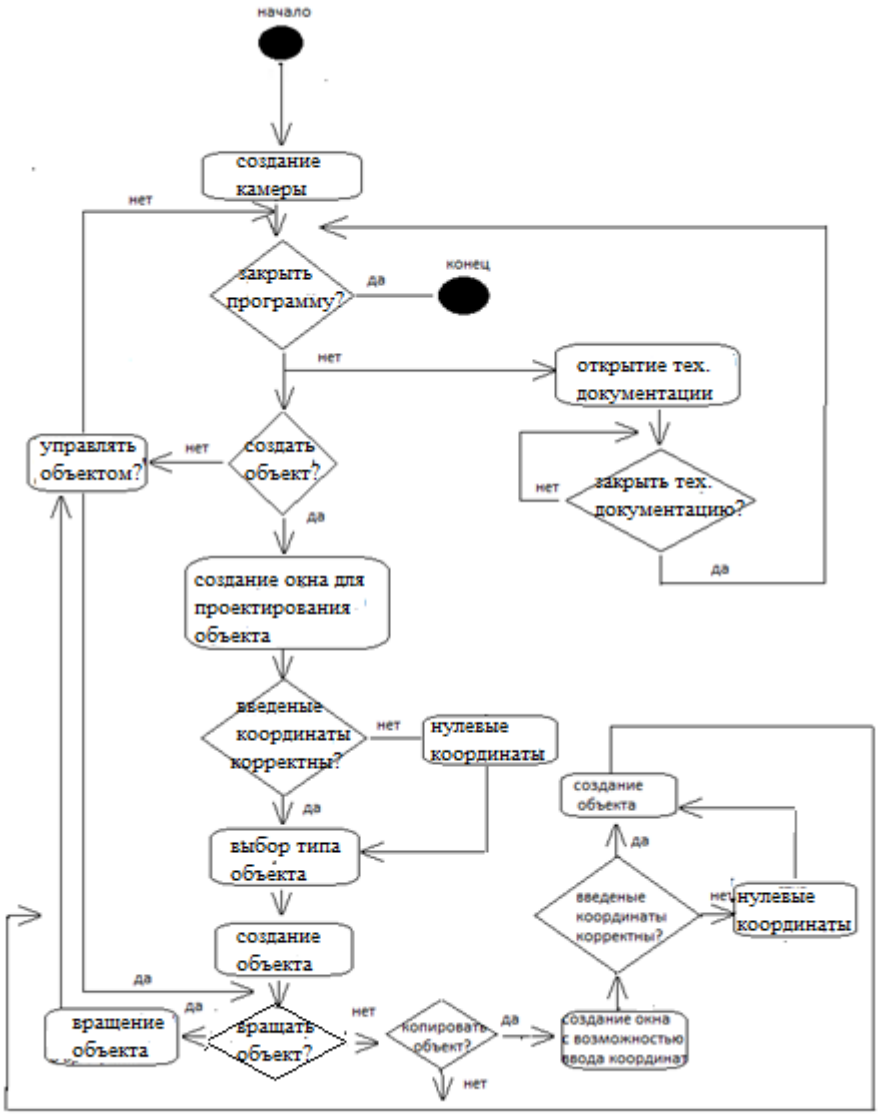

Рисунок 12 – Диаграмма деятельности при работе с 3D-редактором

Такая диаграмма имеет сложную структуру, полученную в результате изучения редакторов и процедуры, доступные для исполнения, определены в ней с каждым доступным шагом выполнения и последующей логики работы от выбранного действия.

3.3 Определение модульной системы программы

Реализация модульной системы основана на подключении к основному файлу программы формата .cpp заголовочного файла формата .h. Заголовочные файлы подключаются командой #include, не учитывая случаи использования различных макросов.

Каждый производный класс зависит от родительского класса, а родительский конструктор не может конструировать объект своего класса. При подключении файлов к программе выстраивается классическая иерархическая система наследования.

Сложность такого иерархического подключения - файлы, содержащие логику для определенного типа фигур, подключаются не один, а несколько раз, что негативно сказывается на оптимизации. Эта ситуация избегается путем определения макросов, позволяющих запретить множественное подключение заголовочного файла с родительским классом.

```
#ifndef ___Имя_файла_
#define __Имя_файла_
Тело файла
#endif
```

Файл с родительским классом объявлен в заголовочных файлах с производными классами, а файлы с производными классами в исполняемом файле. После проделанных манипуляций основной файл знает о существовании родительского класса и объект можно создать без применения лишних ссылок.

3.4 Организация кода программы

Файл формата .cpp является центром всей программы и содержит весь основной код, на котором основана логика интерфейса и обеспечивается инкапсуляция данных, суть которого заключается в объявлении сложных переменных, не имея представление об их внутренней реализации. При написании программы реализовано множество функций для использования, а также обеспечено необходимая степень оптимизации и устойчивости системы, что предполагает большое количество кода.

Способов организации кода существует несколько:

- внесение комментариев для уточнения;
- разбиение кода на модули для обособления зависимых друг от друга участков кода;
- разделение на блоки.

«Структурированность кода достигается приданием ему уровня сложности его восприятия.» [3, с. 58]. Самыми простыми для понимания являются различные объявления библиотек и макросов. Затем можно объявить глобальные переменные, действие которых распространяется на всю программу. Следующими по сложности являются простые функции, далее структуры и классы.

После определения классов следует описать функцию обратного вызова, которая выполняет работу по контролю низкоуровневых сигналов от ОС, посылаемых программе. В функции обратного вызова большая степень разветвления условий и каждой такой ветке необходимо уделить особое внимание. Необходимо очень внимательно определять начало и конец каждой ветки, иначе условия выполнения могут проникать в другие участки кода.

Заключающим элементом иерархии следует описать главную функцию программы – порядок выполнения функций и применения объектов вышеописанных классов определяется в ней.

В каждой функции или классе следует определить локальные переменные, функции и методы, а также классы, в которых могут быть свои локальные переменные. Для наглядности организационную модель кода можно представить с помощью иерархической модели данных, которая позволит в упрощенном варианте представить структуру программы (рисунок 13).

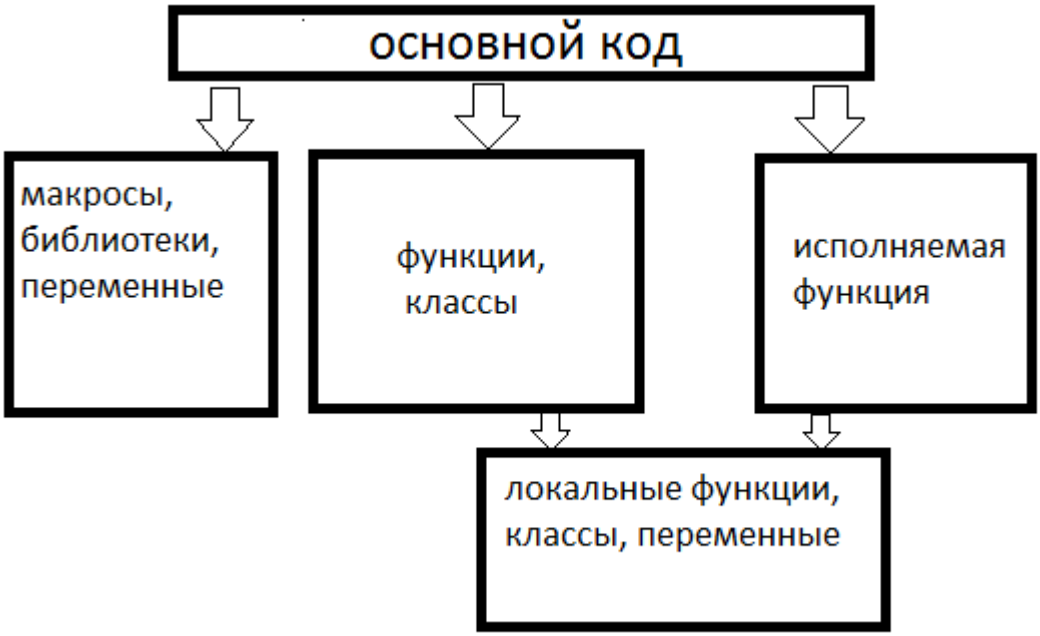

Рисунок 13 – Организация исполняемого файла

3.5 Алгоритм работы программы

Разработка программы представляет собой последовательное выполнение действий. Сначала происходит разработка окна приложения и регистрация окна в системе. Далее в функции обратного вызова для главного окна создается меню с параметрами выполнения определенных функций.

Разработанное и зарегистрированное окно ввода параметров для инициализации объекта имеет свою функцию обратного вызова и является дочерним от родительского главного окна программы.

Основной цикл программы реализован после завершения разработки окон. Логика цикла заключается в проверке потока сообщений на их наличие. При отсутствии сообщений в потоке происходит рендеринг сцены. Такой подход к построению сцены позволяет не занимать процессорное время ожиданием входящего сообщения.

В цикле расположены основные методы для рендеринга сцены, а именно:

- очистка окна – уничтожение всей отрисовки и очистка буфера;
- начало отрисовки сцены – этот шаг задействует все процедуры, участвующие в работе с графикой. Эта функция так же блокирует буфер для корректной отрисовки;
- конец отрисовки сцены – буфер разблокируется и отрисовка в нем заканчивается;
- представление сцены – содержимое буфера выводится на экран.

При возникновении сообщения о закрытии программы происходит очистка сцены и освобождение всех удерживаемых ресурсов: освобождается память видеокарты и оперативная память, удаляя объекты контейнера и все связанные с ними данные. Весь процесс рендеринга продемонстрирован на рисунке 17. Код окон приложения и процесса рендеринга сцены представлен в приложении Б.

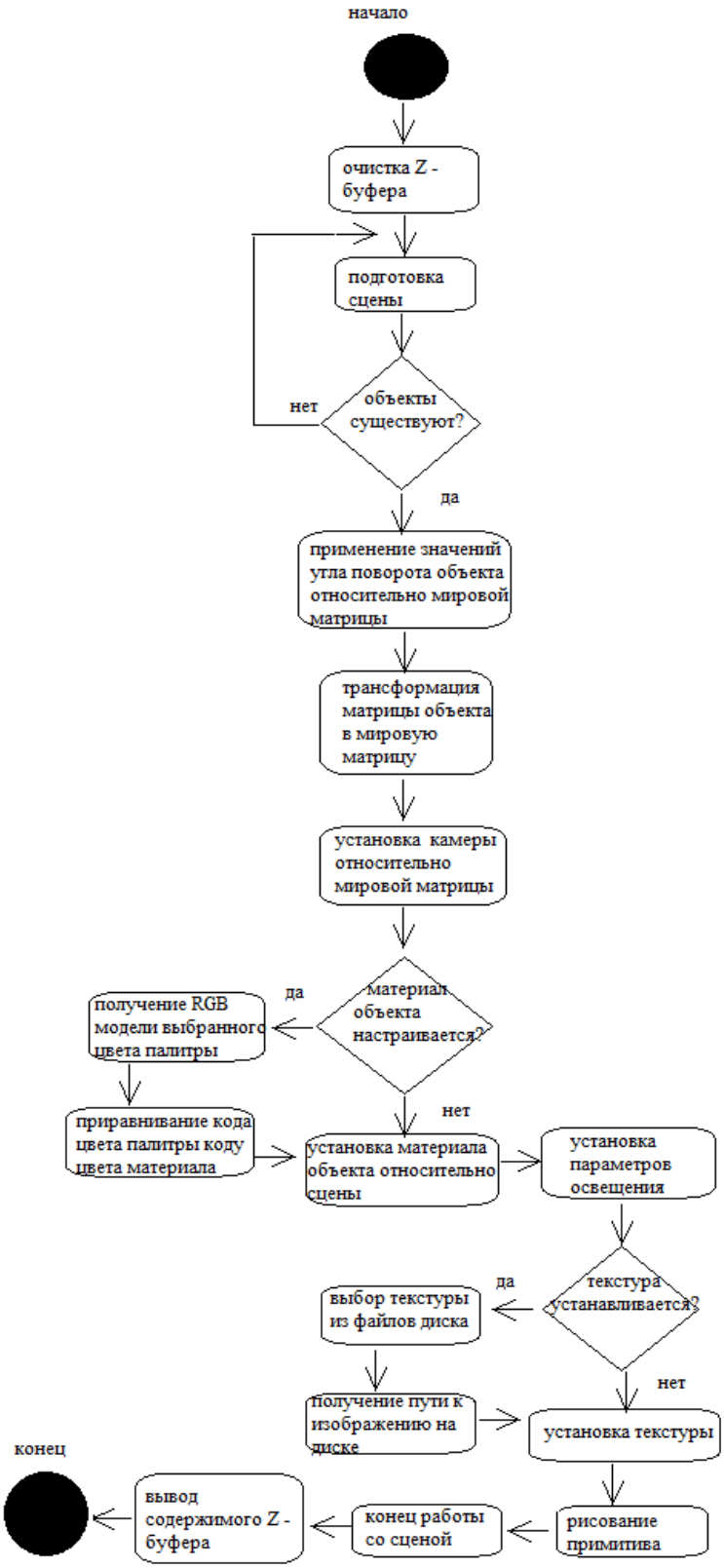

Рисунок 17 – Алгоритм рендеринга сцены

Разработка модульной системы заключается в размещении классов объекта в заголовочных файлах формата .h. Родительский класс помещен отдельно и подключается в файлах наследуемых классов.

В родительском классе также определен метод, работающий с открытием графического файла с использованием стандартных служб Windows, представляющие собой классические способы выбора файла на компьютере. Из этого следует, что данный метод выполняет общую работу для каждого объекта. Логика работы представлена в приложении В.

Разработка наследуемых классов также представляет собой описание класса с инициализацией имеющихся статических переменных и расположением макросов. Заголовочные файлы наследуемых классов включаются в основной файл формата .cpp. Включение родительского класса в исполняемый файл не требуется из-за наличия его в наследуемых файлах.

Код наследуемых классов и пояснения к нему представлены в приложении Г и Д.

3.6 Функционал разработанных элементов редактора 3D моделирования

По завершению разработки была реализована полностью работоспособная программа, предоставляющая возможность применить разработанные графические компоненты на создаваемые объекты. При создании объекта происходит построение примитивов по заданному порядку, определяемому типом объекта. Установки сцены позволяют строить объект с применением технологии отсечения, что существенно сказывается на оптимизации самой программы и количеством потребляемой памяти на один объект. Процесс отсечения определяется порядком установки вершин согласно настройкам сцены.

Построенный объект имеет свой материал и координатные вершины для установки текстуры, а также нормали вершин, заданные значения которых реагируют на источники света на каждой оси пространства.

С помощью меню можно установить координаты нового объекта, установить текстуру на имеющийся или установить цвет материала объекта. Данные элементы являются частью выпадающего списка пункта «Work with 3D object». «Working with light» позволяет определить тип освещения, а «Instruction» вызывает техническую документацию. Реализованное меню продемонстрировано на рисунке 18.

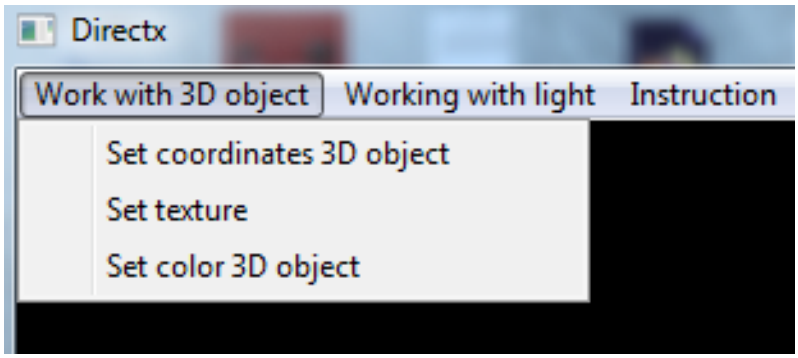

Рисунок 18 – Реализованное меню программы

При создании объекта вызывается окно ввода координат и выбора типа объекта. Тип объекта определяется выбранным флажком, а координаты устанавливаются согласно данным, полученным из полей ввода. При неверно введенных данных объект создается с нулевыми координатами. При нажатии кнопки “ОК” создается объект. Если объект данного типа один, его данные вершин загружаются в видеокарту и остаются там до его удаления. Подобный подход позволит не перезаписывать память много раз при создании объектов одного типа.

Рисунок 19 – Окно создания объекта

Для созданного объекта доступны все реализованные графические элементы. Уничтожение объекта не приводит к отключению освещения и затенения.

Демонстрация реализованных графических элементов редактора приводится в подпункте «3.6.1 Функциональное тестирование программы».

3.7 Тестирование разработанных элементов редактора 3D моделирования

Первоначально программа компилировалась с настройками Debug – версии, что позволяло с минимальными условиями оптимизации собрать проект для текущей платформы. Debug – версию программы невозможно запустить на другой платформе без предварительной настройки ее через среду разработки.

«Для устранения сложности переноса программ в среды разработки были добавлены возможность скомпилировать проект в Release – версии.» [12, с. 114] Такой режим компиляции позволяет скомпилировать программу с настройками для запуска на всех версиях текущей платформы.

Разработанная программа, включающая реализацию компонентов редактора, представляет собой объект для тестирования с возможностью испытания как технологической составляющей, так и производительности относительно системы.

Существует множество видов тестирования программных объектов. Обычно выделяют следующие:

- функциональное тестирование – тестирование ПО в целях проверки реализуемости функциональных требований;
- тестирование производительности – определяет, как быстро работает вычислительная система или её часть под определённой нагрузкой;
- тестирование безопасности – оценка системы на уязвимости к ошибкам.

Далее приводится описание процессов тестирования и их результаты.

3.7.1 Функциональное тестирование программы

Разработанная программа предоставляет не только базовые возможности для создания и управления стандартными примитивами, но и позволяет применять базовые графические технологии, расширяя возможности взаимодействия с объектами. Возможность создавать собственные объекты определяется разработанной модульной системой, позволяющей подключить к программе неограниченное количество модулей с данными объектов.

Функциональное тестирование было проведено с помощью метода предположения об ошибке. Данный метод заключается в предположении мест, подверженных ошибкам и проверке их на работоспособность. Основными предполагаемыми местами возникновения ошибок могут быть:

- корректная работа списка;
- возможность создания объекта;
- возможность наложения текстуры на объект;

– корректная работа палитры.

Все действия, доступные пользователю, продемонстрированы спроектированным меню, которое предоставляет необходимые возможности для работы тогда, когда это возможно. Пункты меню, ответственные за действия, связанные с наличием объекта на сцене (текстурирование, покраска) не будут доступны, если объекта в 3D – пространстве не существует (рисунок 20).

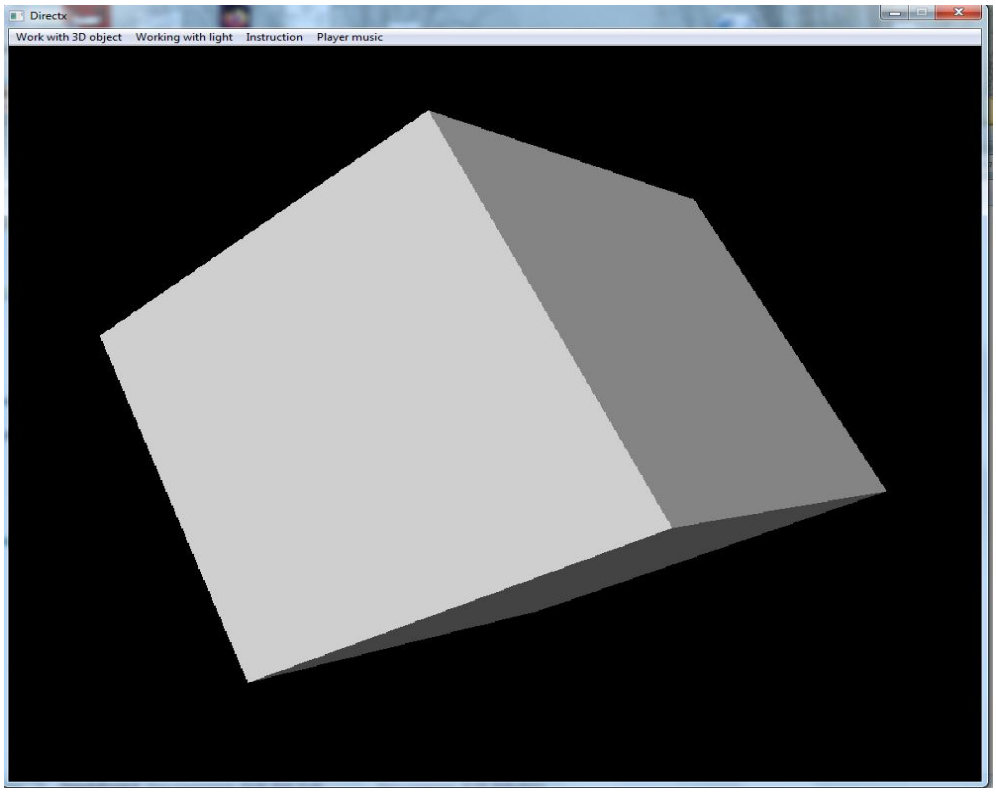

Рисунок 20 – Созданный объект типа «квадрат»

Технология текстурирования позволяет наложить на объект любое изображение, которое было выбрано с помощью стандартных окон Windows на жестком диске ПК. Для наглядности было реализовано текстурирование двух сторон квадрата. Рисунок 21 демонстрирует, что технология текстурирования реализована правильно и не имеет ошибок.

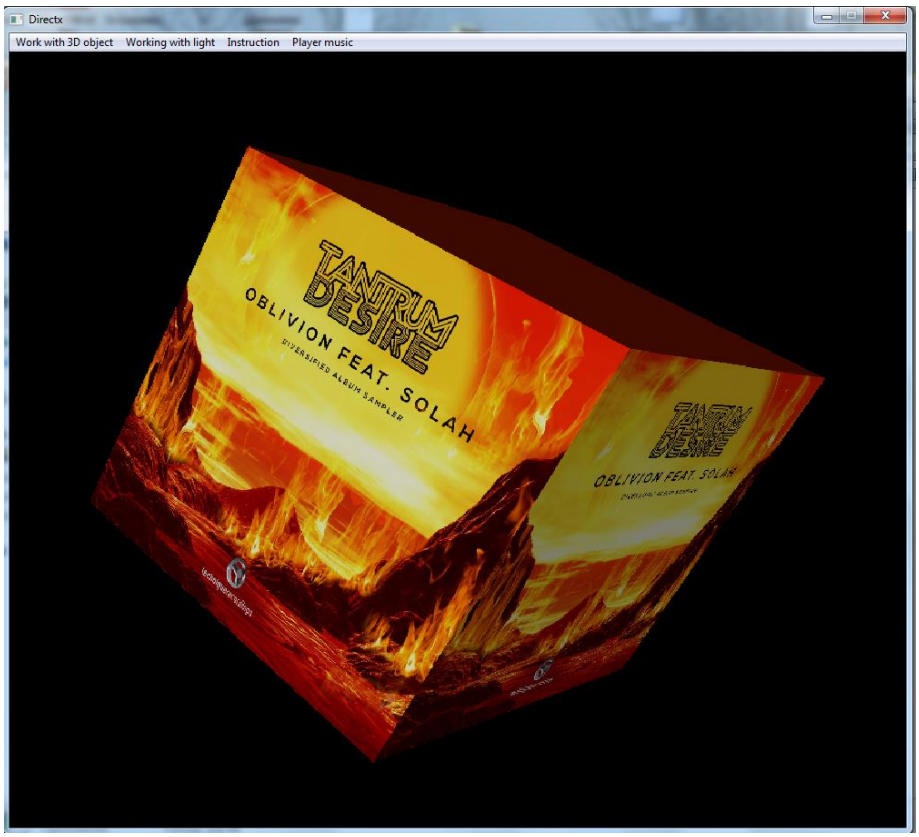

Рисунок 21 – Текстурирование квадрата

Покраска объекта осуществлена с помощью палитры, которая предоставляет несколько цветов на выбор. Выбор цвета создает RGB – структуру с кодом цвета и приравнивается к коду цвета материала объекта.

Управление памятью, реализованное в программе, позволяет избегать ее потерь и каждый объект обладает функцией полного стирания – из памяти удаляется копия объекта с ненужными данными. Общие данные остаются в системе при наличии одного объекта. При удалении последнего объекта память очищается полностью, а контейнер объектов становится пустым.

В процессе разработки был реализован понятный и удобный интерфейс. Он представляет собой простое меню с выпадающими списками возможных действий. Была реализована стандартная палитра Windows, позволяющая выбрать цвет для покраски материала из доступных. Такая палитра продемонстрирована на рисунке 22.

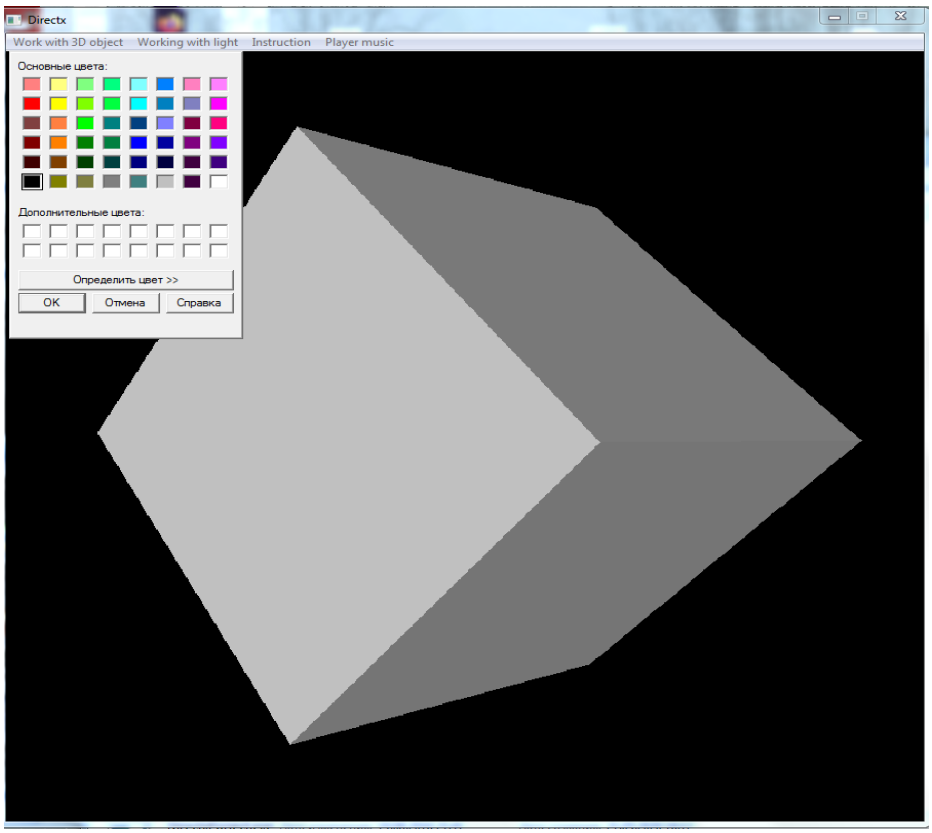

Рисунок 22 – Наличие палитры в программе

Функционал программы не отображает всех возможностей API DirectX, но реализованные в ней технологии демонстрируют широкие возможности данного набора программных интерфейсов.

3.7.2 Тестирование производительности программы

Тестирование производительности было проведено с помощью метода тестирования стабильности. Данный метод заключается в наблюдении за потреблением программой ресурсов и выявлением утечек памяти. Для данного метода не имеет значения длительность тестирования. Вся представленная информация о производительности получена с помощью диспетчера задач Windows.

Тестирование производительности основывалось на нагрузке системы путем создания объектов. После каждого нового объекта производились замеры, а результаты вносились в таблицу (таблица 3).

Таблица 3 – Результаты тестирования производительности

Количество объектов	Загрузка ЦП	Потребление физической памяти (Кб)
0	2 %	4 450
1	2 %	4 520
2	2%	4 600
3	2 – 3%	4 658
2	2 %	4 600
1	2%	4 520

Из полученных данных видно, что нагрузка редактора на ЦП не является большой. Программа работала параллельно с другими службами Windows, из чего можно сделать вывод, что создание новых объектов на сцене имеет минимальное влияние на загруженность ЦП.

Потребление физической памяти изменялось при создании нового объекта. Из данных таблицы можно определить приблизительный размер каждого объекта в системе, вычислив его как среднее арифметическое: узнать количество памяти, занимаемое время объектами и разделить его на три. Таким образом, получаем размер одного объекта в памяти, равное ≈ 69 Кб. Удаление объектов со сцены снижает потребление физической памяти, что означает невозможность утечки памяти.

Тестирование редактора показало, что его работоспособность в рамках системы является приемлемой, а задача по оптимизации программы выполнена.

3.7.3 Тестирование безопасности программы

Тестирование безопасности основывается на вводе некорректных данных или на непредсказуемые действия, выполнение которых не подразумевалось в программе.

Для обнаружения уязвимостей в программе в коде были установлены сообщения, которые оповещают о выполнении некоторых событий – удаление объекта, завершение программы, создание объекта или успешность применения технологии на объекте. При невыполнении стандартных

инструкций программы сообщения не появляются, что указывает на уязвимость.

Рисунок 23 – Вывод сообщения об уничтожении объекта

При тестировании программы было установлено, что ввод некорректных значений в поля координат инициализирует координаты объекта нулевыми значениями. Удаление объекта также приводит к сообщению об очистке памяти при работе деструктора. Применение технологий недоступно без наличия объекта. При наличии объекта технологии успешно применяются и объект появляется на отрендеренной сцене, а сообщение выводится. Закрытие программы разрушает главное окно, затем память очищается и программа полностью закрывается.

Вывод по главе

В данной главе произведено физическое проектирование и разработка программы, реализующая компоненты редактора 3D - моделирования. Проведено тестирование функционала программы, ее производительности и устойчивости к ошибкам.

Заключение

В процессе выполнения бакалаврской работы был проведен анализ теоретических основ, определена концепция программы и реализованы элементы редактора 3D – моделирования. Была выполнена цель ВКР – создать работоспособную программу с компонентами редактора 3D – моделирования с удобным и понятным интерфейсом, реализующим графические технологии. Реализации бакалаврского исследования способствовало решение следующих задач:

– проанализированы существующие 3D – редакторы;

– рассмотрены популярные языки программирования и на основе их анализа был выбран наиболее подходящий;

– рассмотрены среды разработки для выбранного языка программирования и исходя из их функциональных возможностей был выбран наиболее подходящий;

– определена бизнес-модель разработки сцены, на основе которой проектировалась концептуальная модель проекта;

– при анализе полученных данных и проектирования логической модели проекта был реализован интерфейс;

– реализованы графические технологии и внедрены в проект.

Работы над устойчивостью к ошибкам была проведена, результаты тестов показали, что программа может обработать как неверные данные, так и сообщить пользователю о нарушении работы программы в связи с нехваткой памяти. Работа программы показывает, что ее нагрузка на систему не является существенной и работа с объектами не приводит к резким скачкам потребления памяти. Функциональное тестирование продемонстрировало, что требования к программе были реализованы, а реализация графических технологий была продемонстрирована на конкретном примере.

Список используемой литературы

1. . Аллен Э. Типичные ошибки проектирования: Пер. с англ. – СПб.: Питер, 2003. – 224 с.
2. Ашарина, И.В. Основы программирования на языках С и С++: Курс лекций для высших учебных заведений— М.: Гор. линия-Телеком, 2018. — 208 с.
3. Бекишев, Г.А. Элементарное введение в геометрическое программирование - М.: Наука. Главная редакция физико-математической литературы - 2017. - 144 с.
4. Блинов А.О. [и др.] Реинжиниринг бизнес-процессов: учебное пособие для студентов вузов, обучающихся по специальностям экономики и управления — М. : ЮНИТИ-ДАНА, 2015. — 343 с.
5. Буч Г., Рамбо Д., Джекобсон А. Язык UML Руководство пользователя — С-П.: Издательство «Питер», 2010 — 432 с.
6. Буч Градди Максимчук Роберт А., Энгл Майкл У., Янг Бобби Дж., Коналлен Джим, Хьюстон Келли А. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. : Пер с англ. — М.: ООО «И.Д. Вильямс», 2010. — 720 с.
7. Вирт Н. Алгоритмы и структуры данных - М.: Мир, **2016.** - 360 с.
8. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования – СПб.: Питер, 2009. – 366 с.
9. Глазов М.М. , Фирова М.М. Маркетинг предприятия: Анализ и диагностика - М.: АИД, 2009. - 268 с.
10. Довек Ж. Введение в теорию языков программирования — М.: ДМК, 2016. — 134 с.
11. Долженко А.И. Технологии командной разработки программного обеспечения информационных систем - М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 300 с.

12. Джейсон Мак-Колм Смит, Элементарные шаблоны проектирования : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2013. — 304 с.

13. Несвижский В. Программирование аппаратных средств в Windows - СПб.: BHV, 2008. - 528 с.

14. Описание управления бизнес-процессами предприятия на основе методологии IDEF0 [Электронный ресурс]: трудности разработки, рекомендации по совершенствованию построения диаграмм // Фундаментальные исследования. – 2015. – № 8-2. – С. 318-322; URL: http://www.fundamental-research.ru/ru/article/view?id=38893. (дата обращения: 25.04.2020).

15. Роберт, С. Сикорд Безопасное программирование на C и C++ – Москва: РГГУ, 2014. - 496 с.

16. Система трехмерной видеоигры [Электронный ресурс] - URL: https://patenton.ru/patent/RU2339083C2 (дата обращения: 10.05.2020).

17. Страуструп, Б. Язык программирования C++: Специальное издание Пер. с англ. Н.Н. Мартынов. — М.: БИНОМ, 2017. — 1136 с.

18. Финни, К. 3D-игры. Все о разработке- М.: Бином. Лаборатория знаний, 2011. - 976 с.

19. Campbe{ll Parallel Programming with Microsoft® Visual C++® / Campbell. – Москва: Гостехиздат, 2011. - 784 с.

20. Unity в действии. Мультиплатформенная разработка на C++. - М.: Питер, 2018. - 608 с.

Приложение А

Техническое задание на разработку программы

Техническое задание (далее – ТЗ) включает в себя множество пунктов, которые являются важными для всей разработки. Данное ТЗ разработано по стандартам ГОСТ 19.201-78

Введение

Разрабатываемая программа представляет собой набор элементов 3D – редактора, реализующих графические возможности пакета DirectX.

Назначение разработки

Разрабатываемая программа предназначена для демонстрации объектов, используемых на предприятии в рамках обучения с применением интерактивности.

Требования к функциональным характеристикам

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

Требования к составу и параметрам технических средств

В состав технических средств должен входить персональный компьютер, оснащенный ОС на базе windows 32-битной архитектуры(XP, Vista, Seven) с предустановленным графическим пакетом DirectX.

Минимальный объем оперативной памяти должен составлять 1 ГБ(для windows seven – 2ГБ). Минимальный объем видеопамяти должен составлять 128 МБ.

Требования к надежности

Программа должна освобождать заимствованные ею ресурсы во избежание утечек памяти. При возможных недостачах памяти программа должна уведомлять пользователя о проблеме и пытаться распределить ее.

Функциональные требования

Продолжение Приложения А

Программа должна быть оконной, снабжена меню и возможностью закрытия и сворачивания, а также выполнять следующие основополагающие задачи:

- создание объекта,
- копирование объекта,
- управление объектом.

Этапы разработки

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

- изучение необходимого материала;
- создание главного окна приложения;
- создание интерфейса приложения;
- создание модульной части приложения;
- программирование 3D-среды;
- компиляция приложения;
- испытание приложения.

Данное ТЗ лишь предъявляет требования к программе, которые определены настоящими технологическими возможностями и минимальным набором необходимых взаимодействий с объектом в 3D-среде.

Приложение Б

Листинг программного модуля project.cpp

```
#include<windows.h> //блок подключения библиотек
#include<typeinfo>
#include<string>
#include<vector>
#include<d3d9.h>
#include<d3dx9.h>
#include<memory>
#include<shlobj.h>
#pragma comment(lib, "winmm.lib")
#pragma comment(lib, "d3d9.lib")
#pragma comment(lib,"d3dx9.lib")
#include "Triangle_Class.h"
#include "Square_Class.h"
#pragma comment(lib, "winmm.lib") //конец блока

using std::vector; //определение частей пространства имен
using std::tr1::shared_ptr;
using std::string;
using std::fill;

HDC hdc1;  //блок объявления переменных
HINSTANCE h;
PAINTSTRUCT ps;
D3DDISPLAYMODE display;
D3DPRESENT_PARAMETERS param;
HMENU menu,podmenu,podmenu2,m_light,pm1_light,pm2_light,menu_music;
bool Coping_Object=false;
POINT pane;
HWND
hwnd_coord_x,hwnd_coord_y,hwnd_coord_z,hwnd_listbox,hwnd_listbox_camera,music
_list;
D3DXMATRIX Matrix_World, Matrix_Proection,
Matrix_View,Matrix_RotationX,Matrix_RotationY,Matrix_RotationZ;
int cv=0,KEY_INDICATE,mz;
TEXTMETRIC text;
STARTUPINFO zz;
PROCESS_INFORMATION inf;
D3DLIGHT9 Light;
CHOOSECOLOR palitra;
BROWSEINFO folder;
LPITEMIDLIST list_folder;
static COLORREF palitra_user[16];
string path_music,path_music2;
char path_music3[200], bn[200],bn2[200];

bool music=false;
struct List{
      HWND id;
      static double size_y;
      POINTS pl;
    List(HWND id_list=NULL, POINTS& p=POINTS()):id(id_list),pl(p){}
      void Add_Listbox(string stroka,int number);
}LIST_OBJ,LIST_CAMERA;

double List::size_y=0;


void List::Add_Listbox(string stroka,int number){
```

```
       List::size_y = text.tmExternalLeading+text.tmHeight;
   char number_mas[6];
   itoa(number,number_mas,10);
   stroka+=string(" ")+number_mas;
      stroka=stroka.substr(5,strlen(stroka.c_str())-1);
      SendMessage(id,LB_ADDSTRING,0,(LPARAM)stroka.c_str() );
      int list_count=SendMessage(id,LB_GETCOUNT,0,0);
      if(list_count<11)MoveWindow(id,pl.x,pl.y,100,size_y*list_count,true);
}

 struct Camera{
         D3DXVECTOR3 position,view;
             float ugol1,ugol2;
       static unsigned int number_camera,count_camera;
       Camera(D3DXVECTOR3&
ob):position(ob),view(0,0,1),ugol1((float)3.14/2),ugol2((float)3.14/2){count_
camera++;}
       Camera(const Camera&
ob):position(ob.position),view(0,0,1),ugol1((float)3.14/2),ugol2((float)3.14/
2){}
       void operator()(float raz,int index){
             switch(index){
                   case 1:ugol1+=raz;break;
                   case 2:ugol2+=raz;break;
             }
                         view.z = (float)(-10 * sin(ugol1) * cos(ugol2));
                      view.x = (float)(10 * sin(ugol1) * sin(ugol2));
                      view.y = (float)(10 * cos(ugol1));
       }
 };

 vector<Camera>camera_mas;

vector<shared_ptr<object>>objects;
unsigned int Camera::number_camera=0;
unsigned int Camera::count_camera=0;
UINT_PTR CALLBACK _Hook(HWND hwnd,UINT message,WPARAM wparam,LPARAM lparam){
      switch(message){
case WM_INITDIALOG:SetWindowLong(hwnd,GWL_STYLE,WS_BORDER);break;
      }

      return 0;
};

LRESULT CALLBACK _WndProc(HWND hwnd,UINT message,WPARAM wparam,LPARAM
lparam){  //функция обратного вызова основного окна программы
      switch(message){

            case WM_CREATE:   //блок создания окна с описанием его установок
                  D9 = Direct3DCreate9(D3D_SDK_VERSION);
              //блок создания меню
menu=CreateMenu();
                  podmenu=CreateMenu();
             podmenu2=CreateMenu();
                  m_light=CreateMenu();
                  pm1_light=CreateMenu();
                  pm2_light=CreateMenu();
                  menu_music=CreateMenu();
                  AppendMenu(menu,MF_POPUP,(UINT)podmenu,"Work with 3D
object");
```

```
                    AppendMenu(podmenu,MF_POPUP,(UINT)111,"Set coordinates 3D
object");
                    AppendMenu(podmenu,MF_POPUP,(UINT)122,"Set texture");
                    AppendMenu(podmenu,MF_POPUP,(UINT)123,"Set color 3D
object");
                    AppendMenu(menu,MF_POPUP,(UINT)m_light,"Working with
light");
                    AppendMenu(menu,MF_STRING,(UINT)115,"Instruction");
                    AppendMenu(m_light,MF_POPUP,(UINT)pm1_light,"Light
installation");
                    AppendMenu(pm1_light,MF_STRING,(UINT)116,"Inc");
                    AppendMenu(pm1_light,MF_STRING,(UINT)117,"Off");
                    AppendMenu(m_light,MF_POPUP,(UINT)pm2_light,"Light type");
                    AppendMenu(pm2_light,MF_STRING,(UINT)118,"Directional");
                    AppendMenu(pm2_light,MF_STRING,(UINT)119,"Point");
                    AppendMenu(pm2_light,MF_STRING,(UINT)120,"Spotlight");
                    AppendMenu(menu,MF_POPUP,(UINT)menu_music,"Player music");
                    AppendMenu(menu_music,MF_STRING,(UINT)130,"Folder");
                    AppendMenu(menu_music,MF_STRING,(UINT)131,"Open player");


                    SetMenu(hwnd,menu);


                    //конец блока создания меню


          ZeroMemory(&zz,sizeof(STARTUPINFO));
                    zz.cb=sizeof(STARTUPINFO);

      //инициализация переменной для работы с поиском файлов

                    ZeroMemory(&file,sizeof(OPENFILENAME));
     file.lStructSize = sizeof(OPENFILENAME);
     file.lpstrFile = szDirect;
     file.nMaxFile =sizeof(szDirect);
     file.nFilterIndex = 1;
     file.lpstrFileTitle = szFileName;
     file.nMaxFileTitle = sizeof(szFileName);
     file.Flags = OFN_EXPLORER;
       file.lpstrFilter=Filtr;
       file.nFilterIndex=1;

             //иниализация палитры
       palitra.lStructSize=sizeof(CHOOSECOLOR);
       palitra.hwndOwner=hwnd;
       fill(palitra_user,palitra_user+16,RGB(255,255,255));
       palitra.lpCustColors=palitra_user;
       palitra.lpfnHook=(LPCCHOOKPROC)_Hook;
       palitra.Flags=CC_SHOWHELP|CC_ENABLEHOOK;

       GetModuleFileName(NULL,(LPCH)path_music3,200);
       path_music=path_music3;
       path_music=path_music.substr(0,path_music.length()-18);
       folder.pszDisplayName=path_music3;
       folder.lpszTitle=NULL;
       MessageBox(NULL,path_music.c_str(),"",MB_OK);

//установки параметров сцены
                    ZeroMemory(&display,sizeof(display));
                     ZeroMemory(&param,sizeof(param));
```

```
                    D9->GetAdapterDisplayMode(D3DADAPTER_DEFAULT,&display);
                   param.Windowed = true;
                   param.BackBufferFormat = display.Format;
                   param.BackBufferHeight=800;
                   param.BackBufferWidth=800;
                   param.EnableAutoDepthStencil=TRUE;//позволяет создать буфер
глубины
                   param.AutoDepthStencilFormat=D3DFMT_D16;
                   param.SwapEffect=D3DSWAPEFFECT_DISCARD;
                   D9-
>CreateDevice(D3DADAPTER_DEFAULT,D3DDEVTYPE_HAL,hwnd,D3DCREATE_HARDWARE_VERTE
XPROCESSING,&param,&D9Device);
               D9Device->SetRenderState(D3DRS_CULLMODE,D3DCULL_CW);//отсечение
по часовой стрелке
                   D9Device->SetRenderState(D3DRS_ZENABLE,D3DZB_TRUE);
                   D9Device->SetRenderState(D3DRS_LIGHTING,true);
                   CheckMenuItem(pm1_light,(UINT)116,MF_CHECKED);
                   D9Device->SetRenderState(D3DRS_AMBIENT,0);
                   return 0;

             case WM_SIZE:
               pane.x=LOWORD(lparam);
                   pane.y=HIWORD(lparam);
                   return 0;


             case WM_COMMAND:
                   switch(HIWORD(wparam)){
                 case LBN_DBLCLK:
                        if(LOWORD(wparam)==11){

camera_mas[Camera::number_camera].view=objects[SendMessage(LIST_OBJ.id,LB_GET
CURSEL,0,0)]->coord;
                             MessageBox(NULL,"SSS","SS",MB_OK);
                      SendMessage(hwnd_listbox,LBN_KILLFOCUS,0,0);
                         }
                      break;

                   case BN_CLICKED:
                        switch(LOWORD(wparam)){


//создание окна установки параметров инициализации объекта
                   case
111:if(GetKeyState(VK_NUMLOCK)==0){CreateWindowEx(NULL,"coord","coord",WS_BOR
DER | WS_VISIBLE|WS_SYSMENU,pane.x/2,pane.y/2,250,250,NULL,NULL,h,NULL);
KEY_INDICATE=GetKeyState(VK_NUMLOCK);}break;
                   case 123:if(!objects.empty() &&
ChooseColor(&palitra))objects[object::num_obj]-
>New_Color(palitra.rgbResult);else
MessageBox(NULL,"Error!","Error",MB_OK);break;
                   case 122:if(!objects.empty()){
                         EnableWindow(hwnd,false);
                         objects[object::num_obj]->Open_File();
                         EnableWindow(hwnd,true);
                                }break;
                   case 116:D9Device->SetRenderState(D3DRS_LIGHTING,true);
                              CheckMenuItem(pm1_light,(UINT)116,MF_CHECKED);

CheckMenuItem(pm1_light,(UINT)117,MF_UNCHECKED);break;
                   case 117:D9Device->SetRenderState(D3DRS_LIGHTING,false);
```

```
                              CheckMenuItem(pm1_light,(UINT)117, MF_CHECKED);
                                CheckMenuItem(pm1_light,(UINT)116,
MF_UNCHECKED);break;
                  case 118:Light.Type=D3DLIGHT_DIRECTIONAL;break;
                        case 119:Light.Type=D3DLIGHT_POINT;D9Device-
>LightEnable(1,true);break;
                              case 120:Light.Type=D3DLIGHT_SPOT;D9Device-
>LightEnable(2,true);break;

                              case 130:
                                    if(list_folder=SHBrowseForFolder(&folder))

                                          SHGetPathFromIDList(list_folder,bn);
                                    else MessageBox(NULL,"","MEssage",MB_OK);
                                          path_music2=strcat(bn,"\\");


      MessageBox(NULL,path_music2.c_str(),"",MB_OK);
                                          break;
                              case 131:music=true;
CreateWindowEx(NULL,"coord","Player",WS_BORDER |
WS_VISIBLE|WS_SYSMENU,pane.x/2,pane.y/2,500,500,NULL,NULL,h,NULL);
music=false; break;

                  case 115:if(!CreateProcess(NULL,"Notepad.exe
zx.txt",NULL,NULL,FALSE,NORMAL_PRIORITY_CLASS,NULL,NULL,&zz,&inf))MessageBox(
NULL,"er","er",MB_OK);else{
      EnableWindow(hwnd,false);
      WaitForSingleObject(inf.hProcess,INFINITE);
      EnableWindow(hwnd,true);
      CloseHandle(inf.hProcess);
                        }break;
                        }break;
                  }return 0;

            case WM_KEYDOWN:
                  switch(static_cast<int>(wparam)){

//изменение поворота объекта
            case VK_LEFT:if(!objects.empty())objects[object::num_obj]-
>rotat.y-=0.05;break;
            case VK_RIGHT:if(!objects.empty())objects[object::num_obj]-
>rotat.y+=0.05;break;
            case VK_UP:if(!objects.empty())objects[object::num_obj]-
>rotat.x+=0.05;break;
            case VK_DOWN:if(!objects.empty())objects[object::num_obj]-
>rotat.x-=0.05;break;
            case 87:objects[object::num_obj]->rotat.z-=0.05;break;
            case 83:objects[object::num_obj]->rotat.z+=0.05;break;
            case VK_NUMPAD4:camera_mas[Camera::number_camera](0.02,2);break;
            case VK_NUMPAD6:camera_mas[Camera::number_camera](-0.02,2);break;
            case VK_NUMPAD8:camera_mas[Camera::number_camera](0.02,1);break;
        case VK_NUMPAD2:camera_mas[Camera::number_camera](-0.02,1);break;

            case
VK_NUMPAD5:if(++Camera::number_camera>=camera_mas.size())Camera::number_camer
a=0;

      SendMessage(LIST_CAMERA.id,LB_SETCURSEL,Camera::number_camera,0);break;
            case VK_DELETE:if(!objects.empty()){
                  objects.erase(objects.begin()+object::num_obj);
```

```
                  SendMessage(LIST_OBJ.id,LB_DELETESTRING,object::num_obj,0);
                  int list_count=SendMessage(hwnd_listbox,LB_GETCOUNT,0,0);

      if(list_count<11)MoveWindow(hwnd_listbox,0,0,100,(text.tmInternalLeading
+text.tmHeight)*list_count,true);
                  object::num_obj=0;

                                    }break;
            case VK_RETURN:
                    KEY_INDICATE=GetKeyState(VK_NUMLOCK);
                    if(!objects.empty() &&
KEY_INDICATE==0){Coping_Object=true;CreateWindowEx(NULL,"coord","coordinate
object",WS_BORDER |
WS_VISIBLE|WS_SYSMENU,pane.x/2,pane.y/2,250,250,NULL,NULL,h,NULL);}

if(KEY_INDICATE==1){Coping_Object=true;CreateWindowEx(NULL,"coord","coordinat
e camera",WS_BORDER |
WS_VISIBLE|WS_SYSMENU,pane.x/2,pane.y/2,250,250,NULL,NULL,h,NULL);}

                                    break;

//код фокусирования камеры на объекте
            case VK_SPACE:
                  if(!objects.empty()){

      if(++object::num_obj>=objects.size())object::num_obj=0;

      camera_mas[Camera::number_camera].view=objects[object::num_obj]->coord;

      SendMessage(LIST_OBJ.id,LB_SETCURSEL,object::num_obj,0);
                        }break;

                  }


                  return 0;

            case WM_MOUSEWHEEL:

      if((GET_WHEEL_DELTA_WPARAM(wparam))==true)camera_mas[Camera::number_came
ra].position.z+=0.5;else camera_mas[Camera::number_camera].position.z-
=0.5;break;
                  return 0;

            case WM_DESTROY:PostQuitMessage(0);return 0;
      }
      return DefWindowProc(hwnd, message, wparam, lparam);
};

LRESULT CALLBACK COORDFUNCTION(HWND hwnd, UINT message, WPARAM wparam, LPARAM
lparam){
      switch(message){
            case WM_CREATE:
                  if(!music){hwnd_coord_x=CreateWindow("edit","coordinate
X",WS_CHILD|WS_VISIBLE,100,20,100,20,hwnd,(HMENU)1,h,NULL);
                  hwnd_coord_y=CreateWindow("edit","coordinate
Y",WS_CHILD|WS_VISIBLE,100,70,100,20,hwnd,(HMENU)2,h,NULL);
                  hwnd_coord_z=CreateWindow("edit","coordinate
Z",WS_CHILD|WS_VISIBLE,100,120,100,20,hwnd,(HMENU)3,h,NULL);
```

```
      CreateWindow("button","OK",WS_CHILD|WS_VISIBLE,120,170,50,30,hwnd,(HMENU
                                    )4,h,NULL);
                  if(Coping_Object==false){

      CreateWindow("button",NULL,WS_CHILD|WS_VISIBLE|BS_AUTORADIOBUTTON,2,2,50
,50,hwnd,(HMENU)5,h,NULL);

      CreateWindow("button",NULL,WS_CHILD|WS_VISIBLE|BS_AUTORADIOBUTTON,2,70,5
0,50,hwnd,(HMENU)6,h,NULL);
                  };
                  }else
{music_list=CreateWindow("listbox",NULL,WS_VISIBLE|WS_CHILD,100,100,200,200,h
wnd,(HMENU)222,h,NULL);

CreateWindow("button",NULL,WS_VISIBLE|WS_CHILD,400,400,50,50,hwnd,(HMENU)333,
h,NULL);
                           WIN32_FIND_DATA muz;
                           HANDLE value;

value=FindFirstFile((path_music2+string("\\*.wav")).c_str(),&muz);
                           if(value!=INVALID_HANDLE_VALUE)
                                 do{
                                       if(path_music!=path_music2)

if(!CopyFile((path_music2+string(muz.cFileName)).c_str(),(path_music+string(m
uz.cFileName)).c_str(),true))switch(GetLastError()){
            case ERROR_ACCESS_DENIED:MessageBox(NULL,"XXX","XXX",MB_OK);break;
            case
ERROR_ENCRYPTION_FAILED:MessageBox(NULL,"XXX2","XXX2",MB_OK);break;
                                                  }


SendMessage(music_list,LB_ADDSTRING,0,(LPARAM)muz.cFileName);

                                 }while(FindNextFile(value,&muz));


                  }

                  return 0;



            case WM_COMMAND:
                  switch(HIWORD(wparam)){
            case EN_SETFOCUS:SetWindowText((HWND)lparam,NULL);break;
            case
LBN_DBLCLK:if(LOWORD(wparam)==222)mz=SendMessage(music_list,LB_GETCURSEL,0,0)
;MessageBox(NULL,"VV","VV",MB_OK); break;
            case BN_CLICKED:
                  switch(LOWORD(wparam)){
            case 4:

//код получения переменных из полей ввода координат объекта
                  char buffer[5];
                  int coord_x,coord_y,coord_z;

                        GetWindowText(hwnd_coord_x,buffer,5);
                        coord_x=atoi(buffer);
                        GetWindowText(hwnd_coord_y,buffer,5);
                        coord_y=atoi(buffer);
```

```
                    GetWindowText(hwnd_coord_z,buffer,5);
                        coord_z=atoi(buffer);

//защищенный участок кода от нехватки памяти
                            try{
                                  if(KEY_INDICATE==1){

      camera_mas.push_back(Camera(D3DXVECTOR3(coord_x,coord_y,coord_z)));
                                        LIST_CAMERA.Add_Listbox("class
Camera",Camera::count_camera);
                                  }else{
                            if(Coping_Object==true){

      objects.push_back(shared_ptr<object>(objects[object::num_obj]-
>Factory_Object()));
                            (objects.back())->coord =
D3DXVECTOR3(coord_x,coord_y,coord_z);
                            }else {

      if(cv==0)objects.push_back(shared_ptr<object>(new
Triangle(D3DXVECTOR3(coord_x,coord_y,coord_z))));

      if(cv==1)objects.push_back(shared_ptr<object>(new
Square(D3DXVECTOR3(coord_x,coord_y,coord_z))));
                            };



      LIST_OBJ.Add_Listbox(typeid(*(objects.back())).name(),objects.back()-
>Return_Count_Objects());

                                  };
                            }catch(std::bad_alloc&
ob_error){MessageBox(hwnd,ob_error.what(),"Error!",MB_OK|MB_ICONERROR); }



                  DestroyWindow(hwnd);

                    break;

              case 5:cv=0;break;
              case 6:cv=1;break;
              case 333:
                    memset((void*)bn2,0,200);
                    SendMessage(music_list,LB_GETTEXT,mz,(LPARAM)bn2);
                    if(mz==3)MessageBox(NULL,bn2,"",MB_OK);
                    if(!PlaySound(bn2,NULL,SND_FILENAME | SND_ASYNC |
SND_LOOP))MessageBox(NULL,"QQ","",MB_OK);break;
                    }
                    break;
                    }
                    return 0;

              case WM_DESTROY:Coping_Object=false; PostQuitMessage(0);
EnableWindow(FindWindow("Directx","Directx"),true); return 0;
      }
              return DefWindowProc(hwnd, message, wparam, lparam);
};
```

```
//главная исполняемая функция
int WINAPI WinMain(HINSTANCE h1, HINSTANCE h2, LPSTR s1, int ch1){
      h=h1;
tagMSG ms;
ZeroMemory(&ms,sizeof(tagMSG));
      WNDCLASSEX classwindow,classwindowcoord;
//инициализация окна
      classwindow.cbSize = sizeof(WNDCLASSEX);
      classwindow.lpfnWndProc = _WndProc;
      classwindow.style = CS_HREDRAW;
      classwindow.cbClsExtra = 0;
      classwindow.cbWndExtra = 0;
      classwindow.hInstance = h1;
      classwindow.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
      classwindow.hIcon = LoadIcon(NULL, IDI_APPLICATION);
      classwindow.hIconSm = LoadIcon(NULL, IDI_APPLICATION);//Стандартный
значок окна
      classwindow.lpszClassName = "Directx";
      classwindow.lpszMenuName = 0;
      classwindow.hCursor = LoadCursor(NULL, IDC_ARROW);
      classwindowcoord=classwindow;
      classwindowcoord.lpszClassName = "coord";
      classwindowcoord.lpfnWndProc = COORDFUNCTION;
//регистрация окна
      if(!RegisterClassEx(&classwindow))MessageBox(NULL,"Регистрация окна не
завершена", "Ошибка", MB_OK);
      if(!RegisterClassEx(&classwindowcoord))MessageBox(NULL,"Регистрация окна
не завершена", "Ошибка", MB_OK);

   CreateWindowEx(NULL,"Directx",
"Directx",WS_VISIBLE|WS_THICKFRAME|WS_SYSMENU|WS_MINIMIZEBOX,0,0,1000,1000,NU
LL,NULL,h1,NULL);
   hdc1=GetDC(FindWindow("Directx", "Directx"));
                GetTextMetrics(hdc1,&text);
                ReleaseDC(FindWindow("Directx", "Directx"),hdc1);
      POINTS point;

LIST_OBJ=List(CreateWindow("listbox",NULL,WS_VISIBLE|WS_CHILD|LBS_NOTIFY|WS_V
SCROLL,point.x=0,point.y=0,0,0,FindWindow("Directx",
"Directx"),(HMENU)11,h,NULL),point);

LIST_CAMERA=List(CreateWindow("listbox",NULL,WS_VISIBLE|WS_CHILD|LBS_NOTIFY|W
S_VSCROLL,point.x=pane.x-100,point.y=0,0,0,FindWindow("Directx",
"Directx"),(HMENU)12,h,NULL),point);
      camera_mas.push_back(Camera(D3DXVECTOR3(0,0,-5)));
      LIST_CAMERA.Add_Listbox("class Camera",Camera::count_camera);
      SendMessage(LIST_CAMERA.id,LB_SETCURSEL,0,0);
InvalidateRect(FindWindow("Directx", "Directx"),NULL,true);
ZeroMemory(&Light,sizeof(D3DLIGHT9));
//установка освещения
Light.Type=D3DLIGHT_DIRECTIONAL;
Light.Diffuse.r=1;
Light.Diffuse.g=1;
Light.Diffuse.b=1;
Light.Range=100;
D9Device->LightEnable(0,true);
//главный цикл окна
while(ms.message!=WM_QUIT | ms.hwnd!=FindWindow("Directx","Directx")){
           if(PeekMessage(&ms,NULL,0,0,PM_REMOVE)!=0){
           TranslateMessage(&ms);//транслирует сообщение в функцию обратного
вызова
```

```
            DispatchMessage(&ms);//удаляет из очереди сообщений
            }else{

            D9Device-
>Clear(0,NULL,D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,D3DCOLOR_XRGB(0,0,0),1,0);
//начало отрисовки сцены
            D9Device->BeginScene();
            if(!objects.empty()){

//перебор объектов иклом с последующим изменением состояния каждого
                for(vector<shared_ptr<object>>::iterator
it=objects.begin();it!=objects.end();it++){
                D3DXMatrixTranslation(&Matrix_World,(*it)->coord.x,(*it)-
>coord.y,(*it)->coord.z);
                        D3DXMatrixRotationY(&Matrix_RotationY,(*it)->rotat.y);
                        D3DXMatrixRotationX(&Matrix_RotationX,(*it)->rotat.x);
                D3DXMatrixRotationZ(&Matrix_RotationZ,(*it)->rotat.z);
            D9Device-
>SetTransform(D3DTS_WORLD,&(Matrix_RotationZ*Matrix_RotationX*Matrix_Rotation
Y*Matrix_World));
            //блок работы с камерой

            D3DXMatrixLookAtLH(&Matrix_View,
                               &camera_mas[Camera::number_camera].position,

&camera_mas[Camera::number_camera].view,
                                    &D3DXVECTOR3(0,1,0));
            D9Device->SetTransform(D3DTS_VIEW,&Matrix_View);
            D3DXMatrixPerspectiveFovLH(&Matrix_Proection,D3DX_PI/4,1,1,1000);
            D9Device->SetTransform(D3DTS_PROJECTION,&Matrix_Proection);
      //установка материала
D9Device->SetMaterial(&(objects[object::num_obj]->Material));
            //установка потока индексных данных и установка текстур
        D9Device->SetStreamSource(0,(*it)->D3Vertex,0,sizeof(vertex));
            D9Device->SetFVF(D3DFVF_NORMAL|D3DFVF_XYZ|D3DFVF_TEX1);
            D9Device->SetIndices((*it)->Return_Index());
            D9Device->SetTexture(0,(*it)->textura);

            D9Device->SetTextureStageState(0,D3DTSS_COLORARG1,D3DTA_TEXTURE);
            D9Device->SetTextureStageState(0,D3DTSS_COLOROP,D3DTOP_MODULATE);
            D9Device->DrawIndexedPrimitive(D3DPT_TRIANGLELIST,0,0,(*it)-
>Count_Vertex(),0,(*it)->Count_Vertex()/3);
                        }
                }

//конец рендеринга и представление сцены
            D9Device->EndScene();
            D9Device->Present(NULL,NULL,NULL,NULL);
            };
      }
//освобождение захваченной памяти
D9Device->Release();
D9->Release();
      return 0;
};
```

Приложение В

Листинг программного модуля Object.h

```
#ifndef __OBJECT_CLASS_H_  //данные макросы позволяют избежать повторного
подключения
#define __OBJECT_CLASS_H_

extern IDirect3D9* D9 = NULL;
extern IDirect3DDevice9* D9Device = NULL;

OPENFILENAME file;//инициализация переменных
char szDirect[200];
    char szFileName[200];
      char Filtr[]="Image\0*.JPEG;*.JPG;*.PNG\0\0";
      char Filtr2[]="Music\0*.WAV\0\0";
//структура вершин
struct vertex{
      float x,y,z;
      float nx,ny,nz;
      float tu,tv;
      vertex(float a, float b, float c, float nx2, float ny2, float nz2,float
tu2, float tv2):x(a),y(b),z(c),nx(nx2),ny(ny2),nz(nz2),tu(tu2),tv(tv2){}
};


class object{
public:D3DXVECTOR3 coord,rotat;
         static int num_obj;
IDirect3DVertexBuffer9* D3Vertex;
D3DMATERIAL9 Material;
IDirect3DTexture9* textura;
char path_file[200];

//конструктор объекта
object(const D3DXVECTOR3& ob,  vertex* ob2,int
size):coord(ob),rotat(0,0,0),textura(NULL){
               void *x;
                D9Device-
>CreateVertexBuffer(sizeof(vertex)*size,0,D3DFVF_NORMAL|D3DFVF_XYZ|D3DFVF_TEX
1,D3DPOOL_DEFAULT,&D3Vertex,NULL);
           D3Vertex->Lock(0,sizeof(vertex)*size,(void**)&x,0);
           memcpy(x,ob2,sizeof(vertex)*size);
           D3Vertex->Unlock();
               ZeroMemory(&Material,sizeof(D3DMATERIAL9));
               Material.Diffuse.r=Material.Ambient.r=1;
                Material.Diffuse.g=Material.Ambient.g=1;
                   Material.Diffuse.b=Material.Ambient.b=1;
                    Material.Diffuse.a=Material.Ambient.a=0;

 }

//конструктор копирования
         object( const object& ob,int size):rotat(ob.rotat),textura(NULL){
               void *x,*x2;
 D9Device-
>CreateVertexBuffer(sizeof(vertex)*size,0,D3DFVF_NORMAL|D3DFVF_XYZ|D3DFVF_TEX
1,D3DPOOL_DEFAULT,&D3Vertex,NULL);
             D3Vertex->Lock(0,sizeof(vertex)*size,(void**)&x,0);
                  ob.D3Vertex->Lock(0,sizeof(vertex)*size,(void**)&x2,0);
           memcpy(x,x2,sizeof(vertex)*size);
               ob.D3Vertex->Unlock();
```

```
            D3Vertex->Unlock();
                Material=ob.Material;
               if(ob.textura!=NULL){

memcpy((void*)path_file,(void*)ob.path_file,sizeof(char)*200);
                    D3DXCreateTextureFromFile(D9Device,path_file,&textura);
                }

          }

//метод изменения цвета
          void New_Color(COLORREF& rgb){

Material.Diffuse.r=Material.Ambient.r=static_cast<int>(GetRValue(rgb)*255);

Material.Diffuse.g=Material.Ambient.g=static_cast<int>(GetGValue(rgb)*255);

Material.Diffuse.b=Material.Ambient.b=static_cast<int>(GetBValue(rgb)*255);
                     Material.Diffuse.a=Material.Ambient.a=0;
          }

//метод открытия файла
          void Open_File(){
               GetOpenFileName(&file);
               D3DXCreateTextureFromFile(D9Device,file.lpstrFile,&textura);
               if(textura==NULL)MessageBox(NULL,"Not loaded
texture","Error",MB_OK);else
               memcpy((void*)path_file,(void*)file.lpstrFile,sizeof(char)*200);
          };


          virtual object* Factory_Object()=0;//виртуальные абстрактные методы
          virtual int Count_Vertex()=0;
          virtual IDirect3DIndexBuffer9* Return_Index()=0;
          virtual int Return_Count_Objects()=0;

//деструктор класса
          virtual ~object(){D3Vertex->Release();
                            if(textura!=NULL)textura->Release();
                                      MessageBox(NULL,"Clear"," ",MB_OK);
          }
};

int object::num_obj=0;

                                      #endif
```

Приложение Г

Листинг программного модуля Triangle.h

```
#include"Object_Class.h"

class Triangle:public object{
      public:static vertex Vertex_Buffer_Triangle[];
      static unsigned short Index_Buffer_Triangle[];
      static IDirect3DIndexBuffer9* D3Index;
      static int countvertex,count_objects,count;
           Triangle(D3DXVECTOR3& ob):object(ob,Vertex_Buffer_Triangle,12){
                     count_objects++;
                     if(count==0){
                           void* x;
                           D9Device->CreateIndexBuffer(sizeof(unsigned
short)*countvertex,0,D3DFMT_INDEX16,D3DPOOL_DEFAULT,&D3Index,NULL);
                  D3Index->Lock(0,sizeof(unsigned short)*countvertex,&x,0);
                  memcpy(x,Index_Buffer_Triangle,sizeof(unsigned
short)*countvertex);
                  D3Index->Unlock();
                     }
                     count++;
               }
             Triangle(Triangle& ob):object(ob,12){count_objects++; count++;}
             object* Factory_Object(){return new Triangle(*this);}
               int Count_Vertex(){return countvertex;}
               IDirect3DIndexBuffer9* Return_Index(){return D3Index;}
               int Return_Count_Objects(){return count_objects;}
               ~Triangle(){count--; if(count==0)D3Index->Release();}

};

int Triangle::countvertex=12;
int Triangle::count_objects=0;
int Triangle::count=0;
IDirect3DIndexBuffer9* Triangle::D3Index=NULL;

vertex Triangle::Vertex_Buffer_Triangle[]={
      vertex(-1,-1,0,0,0,-1,0,0),
       vertex(1,-1,0,0,0,-1,0,0),
       vertex(0,1,0.5,0,0,-1,0,0),

      vertex(0,1,0.5,-1,0,1,0,0),
     vertex(0,-1,1,-1,0,1,0,0),
       vertex(-1,-1,0,-1,0,1,0,0),

      vertex(1,-1,0,1,0,1,0,0),
      vertex(0,-1,1,1,0,1,0,0),
        vertex(0,1,0.5,1,0,1,0,0),

      vertex(-1,-1,0,0,-1,0,0,0),
      vertex(1,-1,0,0,-1,0,0,0),
      vertex(0,-1,1,0,-1,0,0,0)
 };

unsigned short Triangle::Index_Buffer_Triangle[]={
      0,1,2,
      3,4,5,
      6,7,8,
    9,11,10
                                          };
```

Приложение Д

Листинг программного модуля Square.h

```
#include "Object_Class.h"


class Square:public object{
      public:static vertex Vertex_Buffer_Square[];
      static unsigned short Index_Buffer_Square[];
      static IDirect3DIndexBuffer9* D3Index;
      static int countvertex,count_objects, count;
       Square(D3DXVECTOR3& ob):object(ob,Vertex_Buffer_Square,24){
              count_objects++;
              if(count==0){
        D9Device->CreateIndexBuffer(sizeof(unsigned
short)*countvertex,0,D3DFMT_INDEX16,D3DPOOL_DEFAULT,&D3Index,NULL);
           void* x2;
                D3Index->Lock(0,sizeof(unsigned short)*countvertex,&x2,0);
                memcpy(x2,Index_Buffer_Square,sizeof(unsigned
short)*countvertex);
                D3Index->Unlock();
              }
        count++;
        }
        Square(Square& ob):object(ob,24){count_objects++; count++;}
        object* Factory_Object(){return new Square(*this);}
        int Count_Vertex(){return countvertex;}
        IDirect3DIndexBuffer9* Return_Index(){return D3Index;}
        int Return_Count_Objects(){return count_objects;}
        ~Square(){count--; if(count==0)D3Index->Release();}
};

int Square::countvertex=36;
int Square::count_objects=0;
int Square::count=0;
IDirect3DIndexBuffer9* Square::D3Index=NULL;

vertex Square::Vertex_Buffer_Square[]={
      vertex(-1,1,-1,0,0,-1,0,0),
      vertex(-1,1,-1,-1,0,0,1,0),
      vertex(-1,1,-1,0,1,0,0,0),
      vertex(1,1,-1,0,0,-1,1,0),
      vertex(1,1,-1,1,0,0,0,0),
      vertex(1,1,-1,0,1,0,0,0),
      vertex(1,-1,-1,0,0,-1,1,1),
      vertex(1,-1,-1,1,0,0,0,0),
      vertex(1,-1,-1,0,-1,0,0,0),
      vertex(-1,-1,-1,0,0,-1,0,1),
      vertex(-1,-1,-1,-1,0,0,1,1),
      vertex(-1,-1,-1,0,-1,0,0,0),//11
      vertex(-1,1,1,0,0,1,0,0),
      vertex(-1,1,1,-1,0,0,0,0),
      vertex(-1,1,1,0,1,0,0,0),
      vertex(1,1,1,0,0,1,0,0),
      vertex(1,1,1,1,0,0,0,0),
      vertex(1,1,1,0,1,0,0,0),
      vertex(-1,-1,1,0,0,1,0,0),
      vertex(-1,-1,1,-1,0,0,0,1),
      vertex(-1,-1,1,0,-1,0,0,0),
      vertex(1,-1,1,0,0,1,0,0),
      vertex(1,-1,1,1,0,0,0,0),
```

```
      vertex(1,-1,1,0,-1,0,0,0)
};

unsigned short Square::Index_Buffer_Square[]={9,6,3,3,0,9,
10,1,13,13,19,10,
18,12,15,15,21,18,
       22,16,4,4,7,22,
       14,2,5,5,17,14,
      23,8,11,11,20,23
                                                              };
```