муниципальное общеобразовательное учреждение

«Лицей № 9

имени заслуженного учителя школы Российской Федерации А.Н. Неверова

Дзержинского района Волгограда»

**Утверждено:**

Директор МОУ Лицей №9

\_\_\_\_\_\_\_\_\_\_\_ Жигульская И.В.

Приказ № \_\_\_ от «\_\_\_\_»\_\_\_\_\_\_\_\_\_ 2024 г.

**ОСТАШЕВСКИЙ ЕВГЕНИЙ ИГОРЕВИЧ**

**10 Б**

**УНИВЕРСАЛЬНАЯ СРЕДА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

(Индивидуальный проект)

Кафедра: Информационные технологии

Научный консультант:

Савина Анна Юрьевна

**Согласовано:**

Зам. директора

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Соколова Е.В.

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Оценка\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

Подпись \_\_\_\_\_\_\_\_\_\_ / Соколова Е.В.

Волгоград, 2023

**Оглавление**

**Введение**3

**Глава I. Теоретическая база универсальной среды разработки программного обеспечения**5

1.1. Фундаментальные основы устройства программного обеспечения5

1.2. Критерии универсальной среды разработки программного обеспечения9

**Вывод к главе I**11

**Глава II. Практическая работа: Создание собственного прототипа универсальной среды разработки программного обеспечения**12

2.1. Постановка цели и задач12

2.2. Структура проекта12

2.3. Начало разработки14

2.4. Первые проблемы в разработке и их решение15

2.5. Тестирование полученного проекта16

2.6. Создание документации проекта17

2.7. Сторонние проекты, использующие нашу среду разработки программного обеспечения18

**Вывод к главе II**19

**Заключение**20

**Список источников литературы и электронных ресурсов**21

**Приложения**22

**Введение**

**Актуальность.** В век информационных технологий разрабатывается бесчисленное множество программного обеспечения для решения различных задач, которое с каждым годом находит применение во все большем количестве отраслей. Создание программы с абсолютного нуля, не имея готовых вспомогательных решений или проекта, который можно взять за основу – довольно тяжелое занятие, требующее больших временных затрат и наличие высококвалифицированных специалистов. Вследствие чего появляются новые инструменты, призванные решить данную проблему. Но может ли рынок предоставить такое универсальное решение, которое можно использовать как в профессиональной сфере, так и для написания простых прикладных программ? Выбранная нами тема является актуальной, поскольку применение такого рода программного обеспечения может поспособствовать облегчению создания компьютерных программ.

**Проблема.** Разработка подобных инструментов производится для создания коммерческих продуктов, для проприетарного использования, вследствие чего правообладатель может запросить внушительную цену за такой инструмент, который ко всему прочему может иметь ограниченный функционал, поскольку изначально был разработан под узкоспециализированную задачу. Одной из немаловажных проблем является высокий порог необходимых знаний, в связи с отсутствием внятной документации у таких проектов.

**Цель.** Создать собственный прототип универсальной среды разработки программного обеспечения с технической документацией и примерами использования.

**Задачи:**

1. Изучить материал по теме исследования.
2. Создать собственный прототип универсальной среды разработки программного обеспечения.
3. Разработать техническую документацию к созданному проекту.

**Объект исследования.** Универсальная среда разработки программного обеспечения.

**Предмет исследования.** Практическое применение универсальной среды разработки программного обеспечения.

**Гипотеза:** Мы предполагаем, что использование данного программного обеспечения облегчит создание компьютерных приложений.

**Методы исследования:**

1. **Методы теоретического уровня:**

* Изучение и анализ литературы и других источников информации

1. **Методы эмпирического уровня:**

* Практическая работа

**Теоретическая значимость:** Будут получены и систематизированы знания по теме программного обеспечения, составленная в ходе исследования документация облегчит понимание устройства созданного нами проекта.

**Практическая значимость:** Разработанный в ходе исследования проект может быть использован для создания прикладных программ.

**Научная новизна** исследования состоит в том, что в ходе разработки проекта будут применены новейшие технологии, не встречающиеся в открытых проектах.

**Глава I. Теоретическая база универсальной среды разработки программного обеспечения**

**1.1. Фундаментальные основы устройства программного обеспечения**

Изучение фундаментальных основ устройства программного обеспечения стоит начать с определения, истории возникновения и дальнейшего распространения программного обеспечения.

Программное обеспечение – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ. [2]

Программный документ – документ, содержащий в зависимости от назначения данные, необходимые для разработки, производства, эксплуатации и сопровождения программы или программного средства. [2]

Программное обеспечение является одним из видов обеспечения вычислительной системы, наряду с техническим (аппаратным), математическим, информационным, лингвистическим, организационным, методическим и правовым обеспечением. [3]

Первая современная теория программного обеспечения была предложена ученым Аланом Тьюрингом, который считается пионером информатики и человеком, положившим начало развитию компьютерных технологий, в его эссе 1935 года «О вычислимых числах с приложением к Entscheidungsproblem (задаче принятия решений)».

В конечном итоге это привело к созданию двух академических областей – информатики и программной инженерии, которые изучают программное обеспечение и его создание. Информатика носит скорее теоретический характер (эссе Тьюринга является примером информатики), тогда как разработка программного обеспечения сосредоточена на более практических задачах.

Аналогом ПО в докомпьютерную эпоху можно считать перфокарты, описывающие узор ткани для программируемой ткацкой машины Жаккарда, а также применявшиеся для записи данных в машинах-табуляторах в конце 19 в. сменные барабаны с записью различных мелодий для музыкальных шкатулок и шарманок. ПО как хранимые программы появляются после работ Дж. фон Неймана, опиравшегося на идеи А. Тьюринга, К. Цузе, а также П. Эккерта и Дж. Мокли. Первые программы были вычислительными.

С середины 50-х г. XX в. начали создавать первые языки программирования высокого уровня. Эти языки были машинонезависимы, то есть не были привязаны к определенному типу ЭВМ. Для каждого из них были разработаны собственные компиляторы.

В отличие от программы, в состав ПО обычно включаются все информационные материалы, необходимые для его функционирования, понимания деталей его работы, поддержки работоспособности и дальнейшего развития при возникновении изменений или дополнений в постановке решаемых задач. В компьютерных системах ПО дополняет аппаратное обеспечение.

По типу решаемых задач и виду программ различают системное, встроенное и прикладное ПО. В вычислительной системе взаимодействие может осуществляться на пользовательском, программном и аппаратном уровнях.

Системное ПО служит для разработки программ или предоставляет среду для работы других программ, решая задачи эффективного выделения им вычислительных ресурсов и доступа к различным устройствам.

К системному ПО относят: операционные системы; системы управления базами данных; драйверы различных устройств; защитное и антивирусное ПО; широко используемые библиотеки программ, а также интерпретаторы, компиляторы, среды разработки программ, инструменты тестирования программ, средства контроля версий и др.

Встроенное ПО размещено в каком-либо устройстве или системе (например, станок, телевизор или автомобиль) и играет важную роль в решении задач этой системы, не взаимодействуя с её пользователями непосредственно (такое взаимодействие осуществляется только через интерфейс самой системы).[[1]](#footnote-1)

Прикладная программа — это любая конкретная программа, способствующая решению какой-либо задачи в пределах данной проблемной области. В свою очередь, прикладное ПО подразделяется на специализированное и универсальное.

К специализированному прикладному относят любое ПО, решающее конкретные практические задачи и имеющее интерфейс для взаимодействия с пользователями: веб-браузеры, текстовые и графические редакторы, электронные таблицы, системы подготовки презентаций, информационные системы для бизнеса, игровое ПО, системы документооборота, управления технологическими процессами, вычислительного моделирования и др.

Универсальное прикладное ПО предназначено для работы с объектами, общими для абсолютного большинства проблемных областей; например символы, числа, графические элементы. [9]

Взаимодействие прикладных программ с процессором, оперативной памятью и другими аппаратными компонентами компьютера осуществляется при этом только через системные программы, т. е. имеет место иерархия компонентов ПО с точки зрения доступа к аппаратным ресурсам.

Прикладные программы могут использоваться либо автономно, то есть решать поставленную задачу без помощи других программ, либо в составе программных комплексов или пакетов. [9]

Современное ПО часто имеет очень высокую сложность. На обычном персональном компьютере работает ПО общим размером в сотни миллионов строк кода, написанное на различных языках программирования. Это обусловлено потребностями пользователей в решении большого количества разнообразных задач в рамках одной компьютерной системы и технологиями создания программ, позволяющими строить огромные иерархические программные системы из небольших компонентов.

Сложность ПО приводит к возникновению многочисленных ошибок в силу ограниченной способности разработчиков адекватно воспринимать сложные системы, анализировать их поведение и предсказывать возможные сценарии их развития. Поэтому устранение всех ошибок в практически значимом ПО слишком трудоёмко, вместо этого при его создании обычно пытаются достичь максимально возможного при заданных затратах уровня качества, как можно больше снизить вероятность проявления ошибок и ущерба от них.

Разные виды ПО имеют различные приемлемые уровни качества, часто регулируемые специализированными стандартами в тех областях, где ошибка в работе компьютерной системы может привести к значительному ущербу.

Пользователь получает программное обеспечение вместе с лицензией, которая предоставляет ему право использовать программный продукт при условии выполнения положений о лицензировании. Как правило, эти условия ограничивают возможности пользователя передавать программный продукт другим пользователям, изменять код.

Лицензии на программное обеспечение в целом делятся на две большие группы: несвободные (собственнические, они же проприетарные; и полусвободные) и лицензии свободного и открытого ПО. Их различия сильно влияют на права конечного пользователя в отношении использования программы.

Обычно лицензия на программное обеспечение разрешает получателю использовать одну или несколько копий программы, причём без лицензии такое использование рассматривалось бы в рамках закона как нарушение авторских прав издателя.

Часть программного обеспечения поставляется со свободной лицензией. Такие лицензии позволяют распространять программное обеспечение, а также модифицировать его.

Также существует условно бесплатное программное обеспечение. В этом случае обычно пользователь бесплатно получает демонстрационную версию программного продукта с несколько ограниченными возможностями на определённый испытательный период, а после его окончания обязан или приобрести продукт, или деинсталлировать его.

**1.2. Критерии универсальной среды разработки программного обеспечения**

Под степенью универсальности программного обеспечения мы понимаем степень его применимости в различных вычислительных средах для решения различных задач.

Универсальное программирование основано на идее абстрагирования от конкретных эффективных алгоритмов для получения универсальных алгоритмов, которые можно комбинировать с различными представлениями данных для создания широкого спектра полезного программного обеспечения. [10]

С точки зрения алгоритмического подхода к программированию, универсальность программы зависит от универсальности того алгоритма, который реализуется программой. Универсальные алгоритмы позволяют решать более широкие классы задач, и обычно это свойство алгоритма приобретается за счёт отказа от оптимизации алгоритма для каждого специфического подкласса задач.

Алгоритм обычно тесно связан со структурами данных. При программировании на старых языках часто возникала потребность дублировать одни и те же алгоритмы для различных структур данных. Например, алгоритм обхода элементов такой структуры данных, как дерево, требовал многократного программирования для каждого типа узлов дерева.

Такой метод программирования нельзя признать удачным:

* Он крайне трудоёмок;
* Каждую реализацию алгоритма в коде нужно независимо проверять на наличие ошибок;
* Модификация алгоритма требует переписывания всех его реализаций в программном коде.

Проблему решали путём отказа от строгой типизации структур данных - использовали нетипизированные указатели на хранящиеся в узлах структуры данные. Но и такой подход неудобен, потому что не может уберечь программиста от ошибок неправильной интерпретации данных, скрытых за нетипизированными указателями.

Развитие абстрактных типов данных в языках программирования привело к созданию параметризированных типов данных. При программировании алгоритмов разработчик может воспользоваться неопределённым в момент программирования типом данных, являющимся параметром структуры данных.

Следующий, не менее важный показатель универсальности программного обеспечения – его гибкость. Под гибкостью программного обеспечения понимается организация такой внутренней структуры программы, которая позволяет модифицировать программу с минимальными трудозатратами.

Следует обратить внимание, что гибкость является не одномоментной, а постоянной характеристикой программного решения.

Во-первых, гибкое решение обладает такой структурой, которая обеспечивает минимизацию трудозатрат при любом (в том числе заранее не известном) изменении программы.

Во-вторых, всякое изменение программы должно проводиться таким образом, чтобы не уничтожить гибкость программы.

Гибкое программное решение является долговечным решением. Если в процессе модификации существующей программы разработчик понимает, что дешевле всё переписать "с нуля", нежели разбирать существующие решения, можно констатировать окончание жизненного цикла этой программы.

Назовём такую программу морально устаревшей – не соответствующей новым требованиям к ней, возникшим в процессе эксплуатации. Отныне удовлетворить новые требования может лишь новое программное решение.

Если же всякое новое требование к некой программной системе применимо с незначительными её преобразованиями, жизненный цикл такой программной системы продолжается, вложенные в неё усилия и труд её предыдущих разработчиков не пропадают даром.

Для программных систем, предназначенных для активной работы с пользователями-людьми, для массовых продуктов гибкость зачастую является столь важной характеристикой, что становится приоритетней даже вопросов алгоритмической эффективности, экономичности затрат вычислительных ресурсов.

Достаточно гибкой является структура программного решения, состоящая из тщательно разделенных на разных уровнях абстракции общих и специальных решений. Такие уровни образуют архитектуру программной системы.

**Вывод к главе I**

Мы ознакомились с историей создания программного обеспечения. Определили, что по типу решаемых задач и виду программ различают системное, встроенное и прикладное ПО. В свою очередь, прикладное ПО подразделяется на универсальное и специализированное.

Любое ПО распространяется вместе с лицензией, которая предоставляет пользователю право использовать программный продукт при условии выполнения положений о лицензировании.

Были рассмотрены критерии универсальной среды разработки программного обеспечения, такие как, универсальность и гибкость.

Таким образом, теоретическая часть исследования поспособствовала получению новых знаний по теме и углублению их.

**Глава II. Практическая работа: Создание собственного прототипа универсальной среды разработки программного обеспечения**

**2.1. Постановка цели и задач**

**Цель:** Необходимо разработать собственный прототип универсальной среды разработки программного обеспечения, который бы удовлетворял следующим критериям:

1. Переносимость;
2. Кроссплатформенность;
3. Модульность;
4. Многоуровневость;
5. Гибкость.

**Задачи практической работы**:

1. Проектирование структуры среды разработки;
2. Создание проекта;
3. Тестирование полученных результатов;
4. Разработка технической документации.

**2.2. Структура проекта**

Поскольку проект планируется модульным, необходим общий набор компонентов, которые могли бы быть использованы в любом модуле. Но объединить весь набор компонентов в одну целую библиотеку было бы неправильно по причине того, что для каждого конкретного модуля такой набор компонентов окажется просто избыточным. Да и к тому же это сильно увеличивает время компиляции даже при малейших изменениях внутри этих компонентов.

Было принято решение разбить код на условные уровни, которым соответствует свой набор библиотек и модулей:

* Уровень 0:
* Управление памятью;
* Управление потоками и параллельными вычислениями;
* Система отслеживания ошибок;
* Функции для взаимодействия с операционной системой;
* Определение информации об аппаратном обеспечении;
* Счетчики производительности.
* Уровень 1:
* Собственная библиотека стандартных типов, взамен встроенной в C++ STL;
* Система консольных команд и файлов конфигураций;
* Система событий.
* Уровень 2:
* Базовая файловая система (чтение/запись файлов);
* Асинхронная файловая система (аналогично базовой файловой системе, но вычисления производятся параллельно, не тормозя основной поток);
* Базовая система ввода (обработка нажатий клавиатуры и мыши);
* Контекстная система ввода (включает в себя функционал базовой системы ввода, но имеет возможность использования в разных окнах);
* Система ресурсов (управление всеми типами ресурсов, например текстурами или трехмерными моделями);
* Система схем (хранение информации об объектах и типах данных);
* Система рендера (отрисовка графического изображения);
* Система шейдеров (программ, выполняемых на графическом процессоре);
* Система материалов (свойств поверхностей рендера, использующих шейдеры).
* Уровень 3:
* Система трехмерных моделей и мешей[[2]](#footnote-2).

Главная задумка многоуровневой системы заключается в том, чтобы уровни ниже не могли получать доступ к компонентам, которые находятся уровнем выше. Они как бы наслаиваются друг на друга. Например, на уровне 2 мы не можем получить доступ к компонентам уровня 3, но при этом нам доступны уровни 0, 1 и сам уровень 2.

При этом существуют библиотеки, не относящиеся ни к какому уровню и могут быть использованы в любом модуле:

* Фреймворк[[3]](#footnote-3) приложения (отвечает за управление модулями и подсистемами приложения);
* Математическая библиотека (векторные вычисления);
* Библиотека загрузки растровых изображений.

**2.3. Начало разработки**

Для создания проекта мною использовалась программа Visual Studio[[4]](#footnote-4). В связке с ней я использовал CMake[[5]](#footnote-5), чтобы написание кода было возможно не только под Windows, а например под Linux или macOS.

В первую очередь необходимо написать главный CMake-скрипт (файл CMakeLists.txt), который содержит основные настройки проекта, а также список его модулей, библиотек и тестирующих программ. (*Приложение 1*)

Далее необходимо написать точно такой же скрипт, но уже для подпроектов — модулей, библиотек и исполняемых файлов. (*Приложение 3*)

Открываем CMake (*Приложение 4*), указываем необходимый путь к проекту и создаем решение[[6]](#footnote-6), которое мы уже можем открыть в Visual Studio или любой другой удобной нам программе. (*Приложение 5*)

Настройка окончена и мы готовы приступать к написанию кода.

**2.4. Первые проблемы в разработке и их решение**

Было принято решение разрабатывать этот проект с полного нуля, не используя сторонних библиотек, на языке программирования C++. Этот язык обладает всеми необходимыми преимуществами для написания универсального ПО:

1. Переносимость кода, обеспечивающая кроссплатформенность;
2. Высокая производительность;
3. Эффективное использование ресурсов компьютера;
4. Синтаксическая гибкость (полиморфизм, параметризированность, ООП[[7]](#footnote-7));
5. Возможность напрямую взаимодействовать с операционной системой.

Но у С++ есть один существенный минус по сравнению с Java или C#, а именно анонимность типов данных. После компиляции код превращается в набор команд для процессора, не имеющих никакой информации о типах данных. Проще говоря, у нас не выйдет просто так взять и сохранить какой-то объект из памяти на жесткий диск, при этом восстановив его в памяти без потерь.

Для этого необходимо разработать собственную систему, которая бы хранила информацию об объектах и их метаданных. А для того, чтобы эта система могла получать всю необходимую информацию – разработать анализатор кода, действующий по принципу компиляции, который бы считывал код перед компиляцией и создавал необходимые файлы с информацией об объектах.

В открытых источниках нет информации о принципах работы и применении такого подхода, можно лишь сказать, что такой подход действительно существует, но нет достоверных сведений о проектах, использующих его. Поэтому можно сказать, что такое решение является новаторским.

Этой системе мы дали название – «Система Схем» (от англ. Schema System). Сам термин «Схема» (от англ. Schema) используется в веб программировании как раз для описания различных типов данных и объектов.

Для создания компилятора схем (от англ. Schema Compiler) необходим синтаксический анализатор кода С++, который бы извлекал из него необходимую нам информацию об объектах.

Писать собственный анализатор кода для языка С++ это непосильная задача даже для профессионалов своего дела, да и к тому же бессмысленная, когда уже есть готовые решения.

Одним из таких решений является Clang – один из мощнейших компиляторов языка С++ с открытым исходным кодом. Поэтому мы решили использовать модуль синтаксического анализатора именно от компилятора Clang.

Используя информацию, полученную через синтаксический анализатор, компилятор схем может генерировать дополнительные файлы на С++, в которых в автоматическом порядке прописывается информация о типах данных и объектах.

При запуске программы данные из сгенерированных файлов будут переданы в систему схем, которая и будет хранить и обрабатывать всю информацию.

В свою очередь, вспомогательная библиотека системы схем дает нам возможность, используя эти данные, сохранить любой объект, например в виде файла на жестком диске, и восстановить этот объект в памяти без потерь.

Имеется возможность и порождать объекты, зная информацию о их структуре, или изменять значения переменных одним простым действием, прямо во время работы программы.

**2.5. Тестирование полученного проекта**

Для того, чтобы гарантировать полную работоспособность проекта, необходимо как-то проверить его на правильность выполнения. Чтобы выявить сбои в работе, необходима тестирующая система, которая позволяла бы нам писать простые юнит тесты для каждого отдельного компонента.

Мы выяснили, что проще написать её самим, поскольку функционал достаточно примитивен, код уместился всего в 300 строк. (*Приложение 7*)

В задачи тестирующей системы входит только проверка правильности условия, которое обозначает сам программист. Пример такого теста: (*Приложение 9*).

В случае несовпадения с требуемым результатом тест должен быть прекращен. Пример провального запуска теста: (*Приложение 6*).

В конце выполнения любого теста необходимо вывести время, за которое был выполнен сам тест и статус его выполнения.

Таким образом, для каждого модуля был написан свой юнит тест[[8]](#footnote-8). Все тесты успешно проходят проверку. (*Приложение 8*)

Для тех случаев, где необходимо проверить работу графической части, например модуль системы рендера или системы материалов, пишутся отдельные мини-приложения, демонстрирующие функционал конкретного модуля. (*Приложение 10*)

Для тестирования системы ввода используется пустое окно, а результат нажатий на клавиши отображается в консоли разработчика. (*Приложение 11*)

Код таких юнит-тестов уже не использует тестирующую систему, а базируется исключительно на возможностях имеющихся библиотек. (*Приложение 12*)

**2.6. Создание документации проекта**

Хотя в исходном коде проекта присутствуют необходимые комментарии, поясняющие работу конкретных функций (*Приложение 2*), этого все равно недостаточно для создания полной картины представления об устройстве данного проекта.

Мы разработали техническую документацию, которая включает в себя в описание работы всех основных компонентов, доступных другим разработчикам.

В качестве примера имеется небольшой фрагмент документации к математической библиотеке векторных вычислений. (*Приложение 15)*

**2.7. Сторонние проекты, использующие нашу среду разработки программного обеспечения**

Нам удалось заинтересовать своей работой одного из студентов Сочинского Государственного Университета, обучающегося на факультете архитектуры, дизайна и экологии. Для дипломной работы ему была необходима программа для визуализации трехмерных моделей, заготовленных в программе для моделирования. Он, обладая базовыми навыками программирования и не желая использовать готовые решения по причине недостатка их функционала, решил разработать такую программу самостоятельно.

Это отличная возможность проверить наш проект на практике и мы предоставили ему исходный код вместе с технической документацией.

Спустя пару месяцев его работы с нашим проектом, мы решили поинтересоваться, каких результатов он смог достичь за такой довольно маленький промежуток времени. Вот небольшой список функционала полученной программы:

* Загрузка моделей формата Wavefront OBJ[[9]](#footnote-9) и Filmbox FBX[[10]](#footnote-10);
* Загрузка текстур формата TGA, PNG, JPEG, BMP;
* Создание трехмерных сцен;
* Добавление, удаление, дублирование, перемещение, вращение, масштабирование объектов сцены;
* Редактирование свойств материалов в реальном времени;
* Глобальное (солнечное) динамическое освещение с поддержкой каскадных теней;
* Точечные источники света;
* Поддержка пользовательского интерфейса.

С результатом его работы можно ознакомиться в *Приложениях 13 и 14*.

**Вывод к главе II**

В ходе практической части исследования был разработан прототип универсальной среды разработки программного обеспечения.

Созданное ПО имеет подробную техническую документацию и выполняет свое предназначение.

Разработанный нами проект уже успешно применяется на практике.

**Заключение**

Создание программы с полного нуля, не имея под рукой мощного инструмента или готового проекта, который можно взять за основу – это большие временные затраты и необходимость в высококвалифицированных специалистах.

По-настоящему мощные инструменты создаются для разработки коммерческих продуктов, для проприетарного использования.

Правообладатель может запросить внушительную цену за такой инструмент, который ко всему прочему может иметь ограниченный функционал, поскольку изначально был разработан под узкоспециализированную задачу.

Одной из немаловажных проблем является высокий порог необходимых знаний, в связи с отсутствием внятной документации у таких проектов.

Поэтому, для решения этой проблемы в ходе практической работы был разработан собственный прототип универсальной среды разработки программного обеспечения.

При разработке проекта применялись новейшие технологии. Была создана собственная система описания типов данных и объектов, которая значительно упрощает написание кода. Нет достоверных сведений об аналогах этой технологии, об их принципе работы или применении, так что можно считать такое решение новаторским.

За счет многоуровневой модульной структуры проекта, любой человек может без особых проблем вносить изменения в различные участки кода или создавать новые модули, расширяющие функционал.

Первым сторонним ПО, созданным при помощи нашего проекта является программа для визуализации трехмерной графики, автором которой является один из студентов Сочинского Государственного Университета, обучающегося на факультете архитектуры, дизайна и экологии.

Таким образом, разработанный нами проект уже применяется на практике.

**Список источников литературы и электронных ресурсов**

**Литературные источники:**

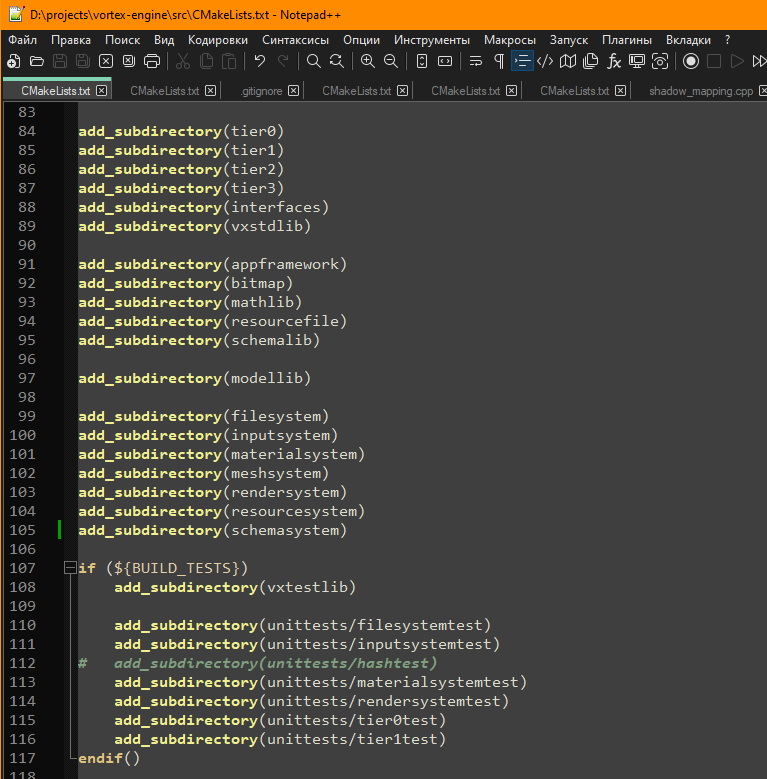
1. Батоврин В. К. Толковый словарь по системной и программной инженерии. М.: ДМК Пресс, 2012, С. 280.
2. ГОСТ 19781-90 Обеспечение систем обработки информации программное. Термины и определения, 1990.
3. ГОСТ 34.003-90 Информационная технология. Автоматизированные системы. Термины и определения, 1990.
4. ISO/IEC/IEEE 24765:2010 Systems and software engineering — Vocabulary, 2010.
5. СТО НОСТРОЙ 2.15.9-2011: Инженерные сети зданий и сооружений внутренние. Устройство систем распределенного управления. Монтаж, испытания и наладка. Требования, правила и методы контроля, 2011.
6. Федеральный закон N 99-ФЗ "О лицензировании отдельных видов деятельности", 2011.
7. Першиков В. И., Савинков В. М. Толковый словарь по информатике / Рецензенты: канд. физ.-мат. наук А. С. Марков и д-р физ.-мат. наук И. В. Поттосин. — М.: Финансы и статистика, 1991.
8. Философский словарь инженера — М.: Издательский дом МЭИ, 2016.
9. Кубанский государственный технологический университет. ЛЕКЦИЯ № 19. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ЕГО ОСНОВНЫЕ ХАРАКТЕРИСТИКИ, 2019.
10. Массер, Дэвид Р.; Степанов, Александр А. Универсальное программирование, 2009.

**Электронные ресурсы:**

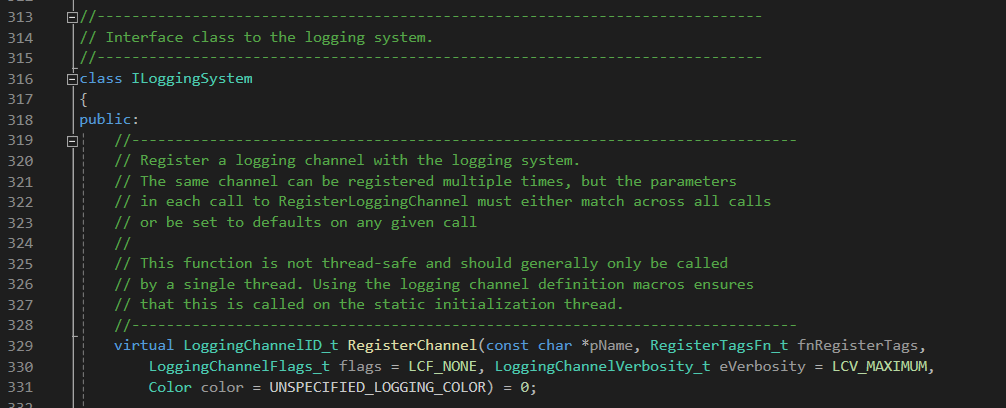
1. <https://ru.wikipedia.org>
2. <https://bigenc.ru>
3. <https://proglib.io>
4. <https://shelek.com>

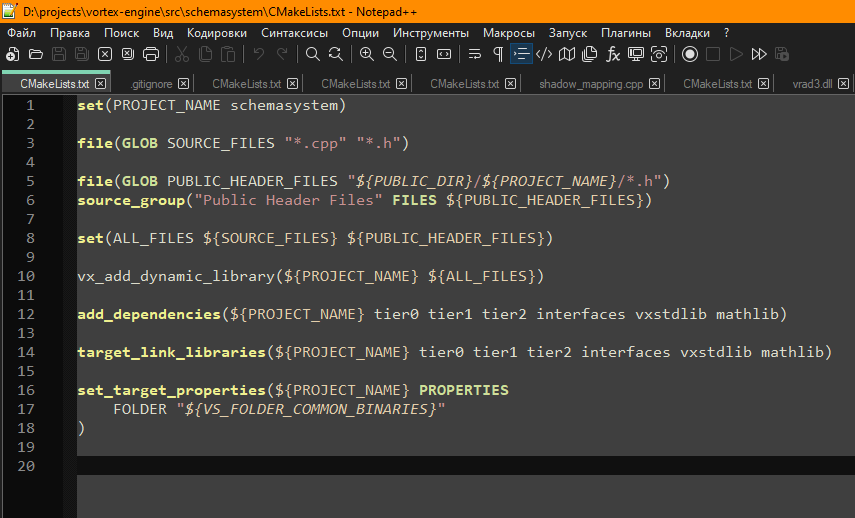
**Приложения**

*Приложение 1:*

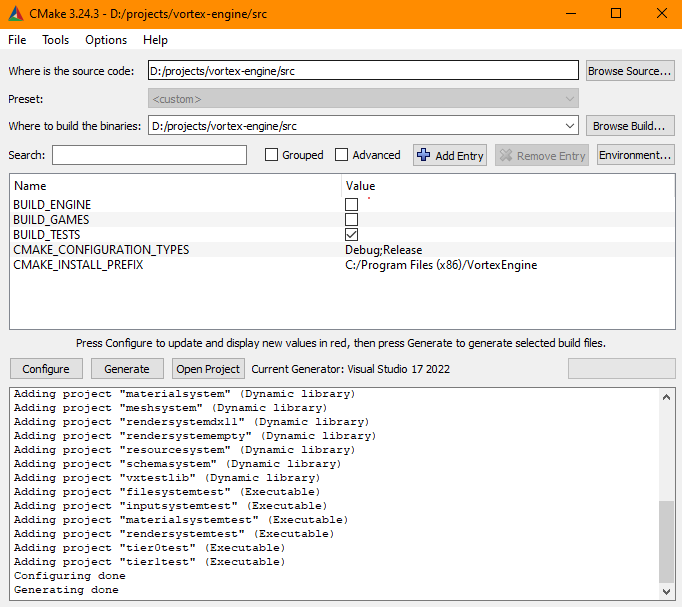


*Приложение 2:*

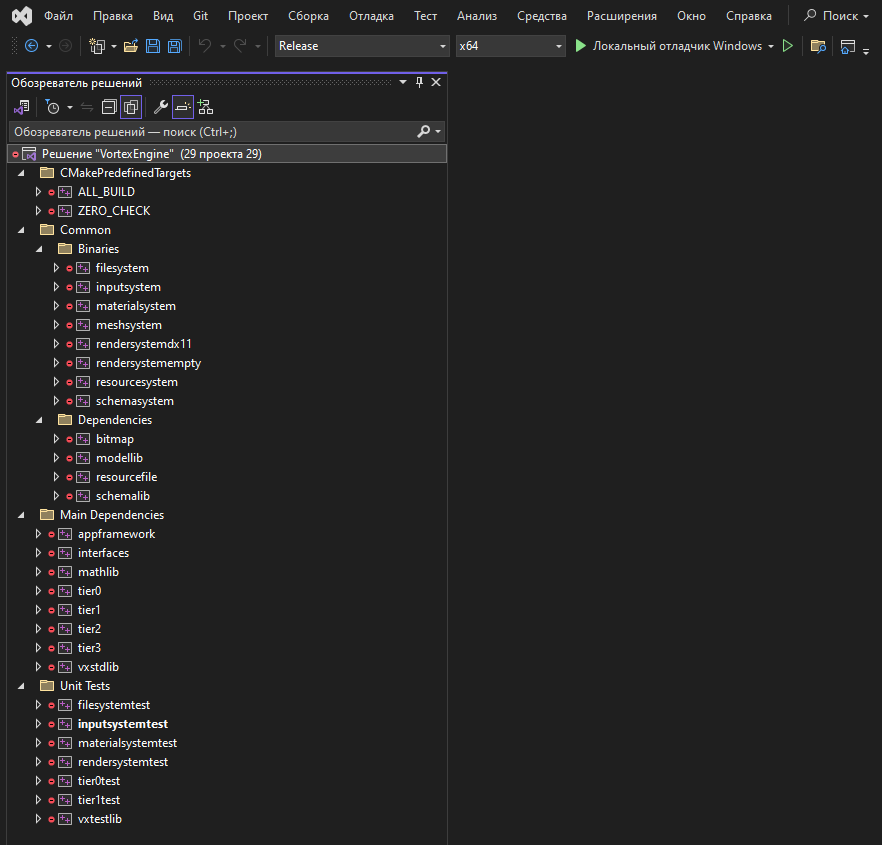
**

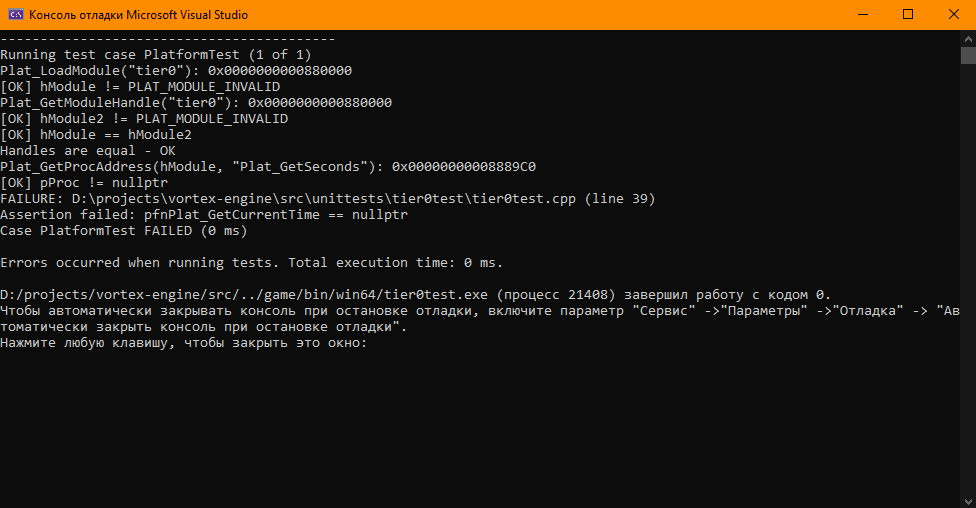
*Приложение 3: *

*Приложение 4:*

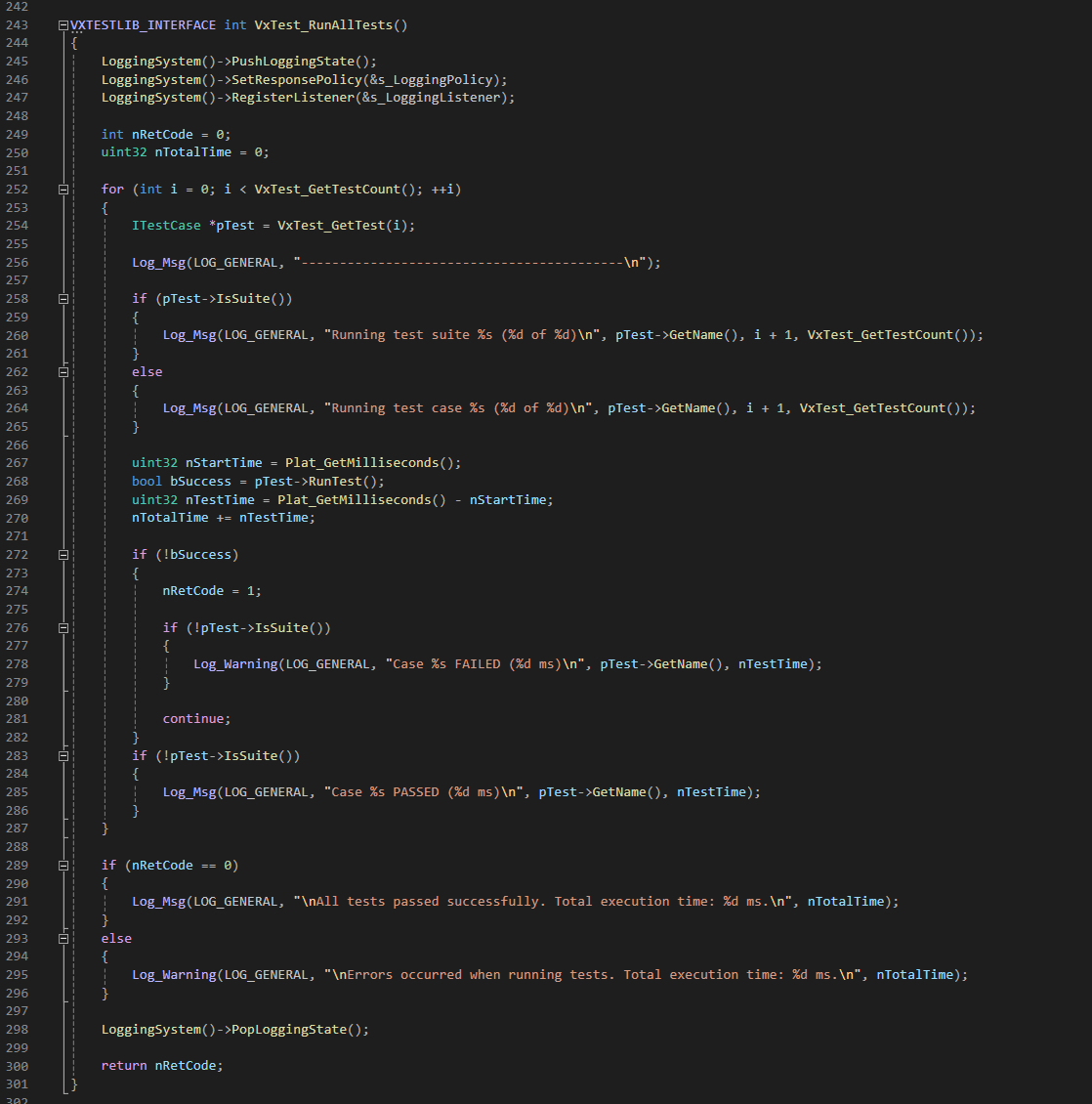


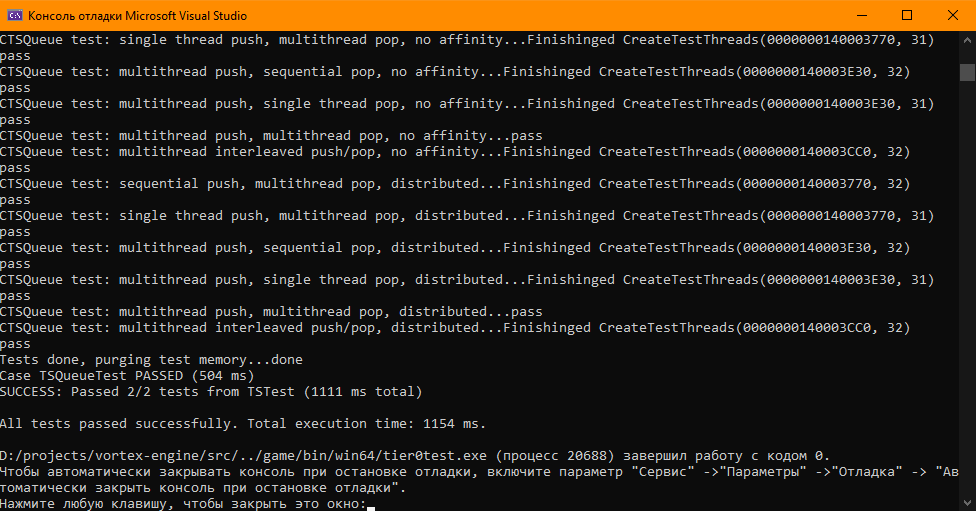
*Приложение 5:*



*Приложение 6: *

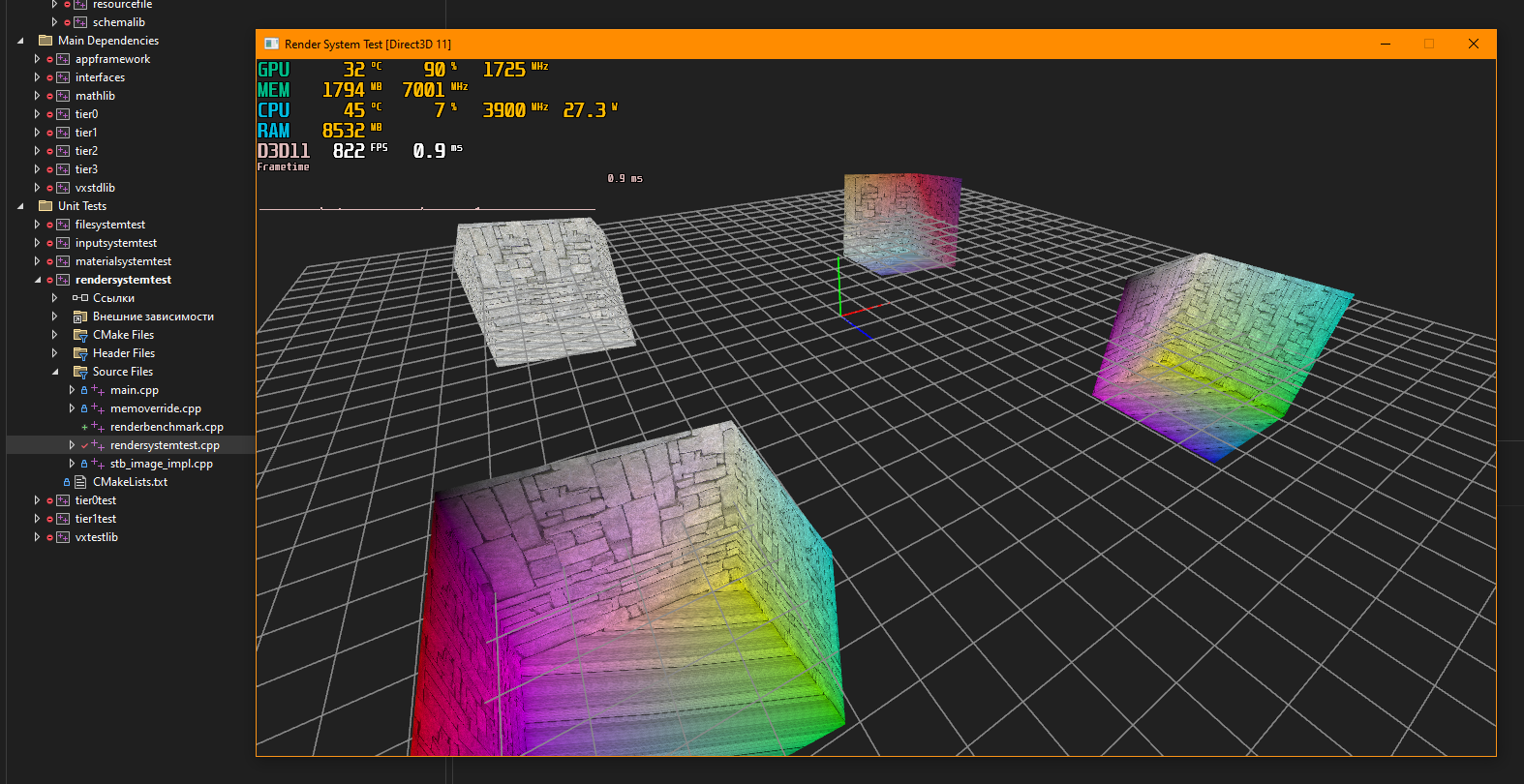
*Приложение 7:*



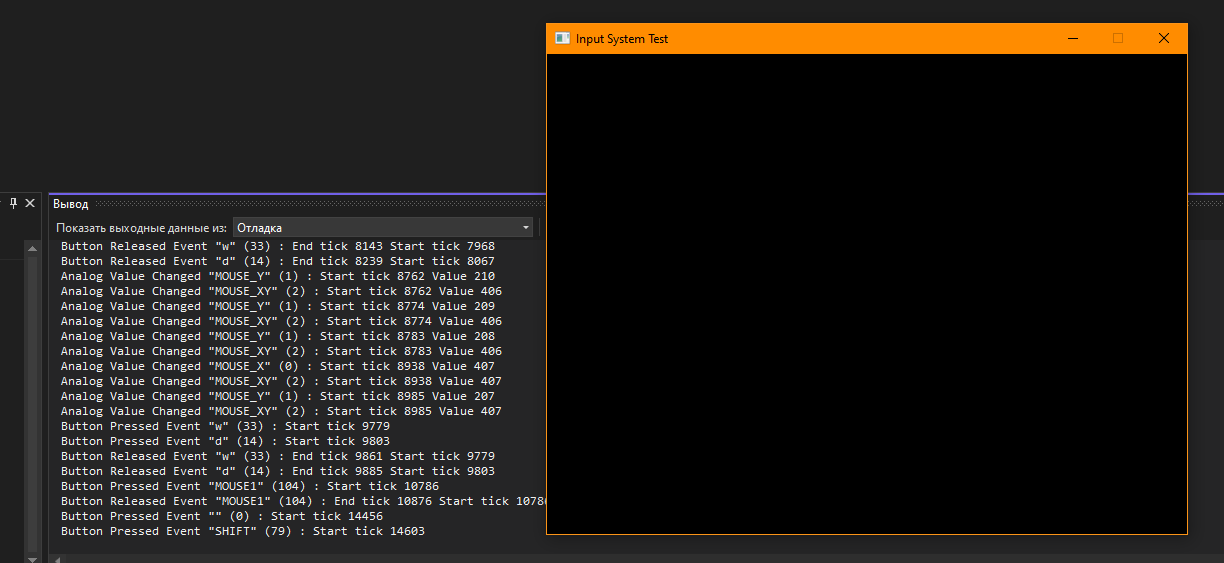
*Приложение 8: *

*Приложение 9:*

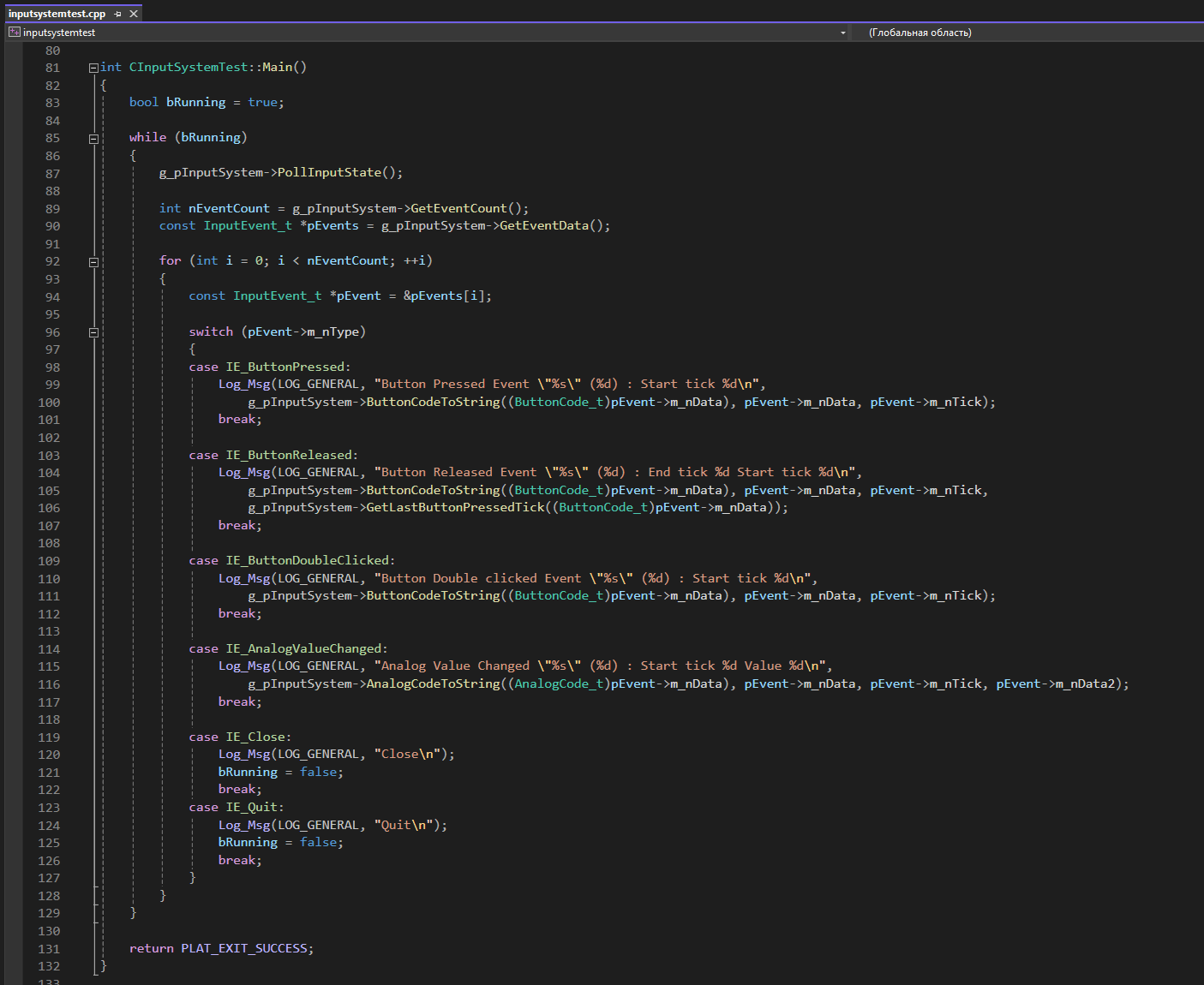


*Приложение 10: *

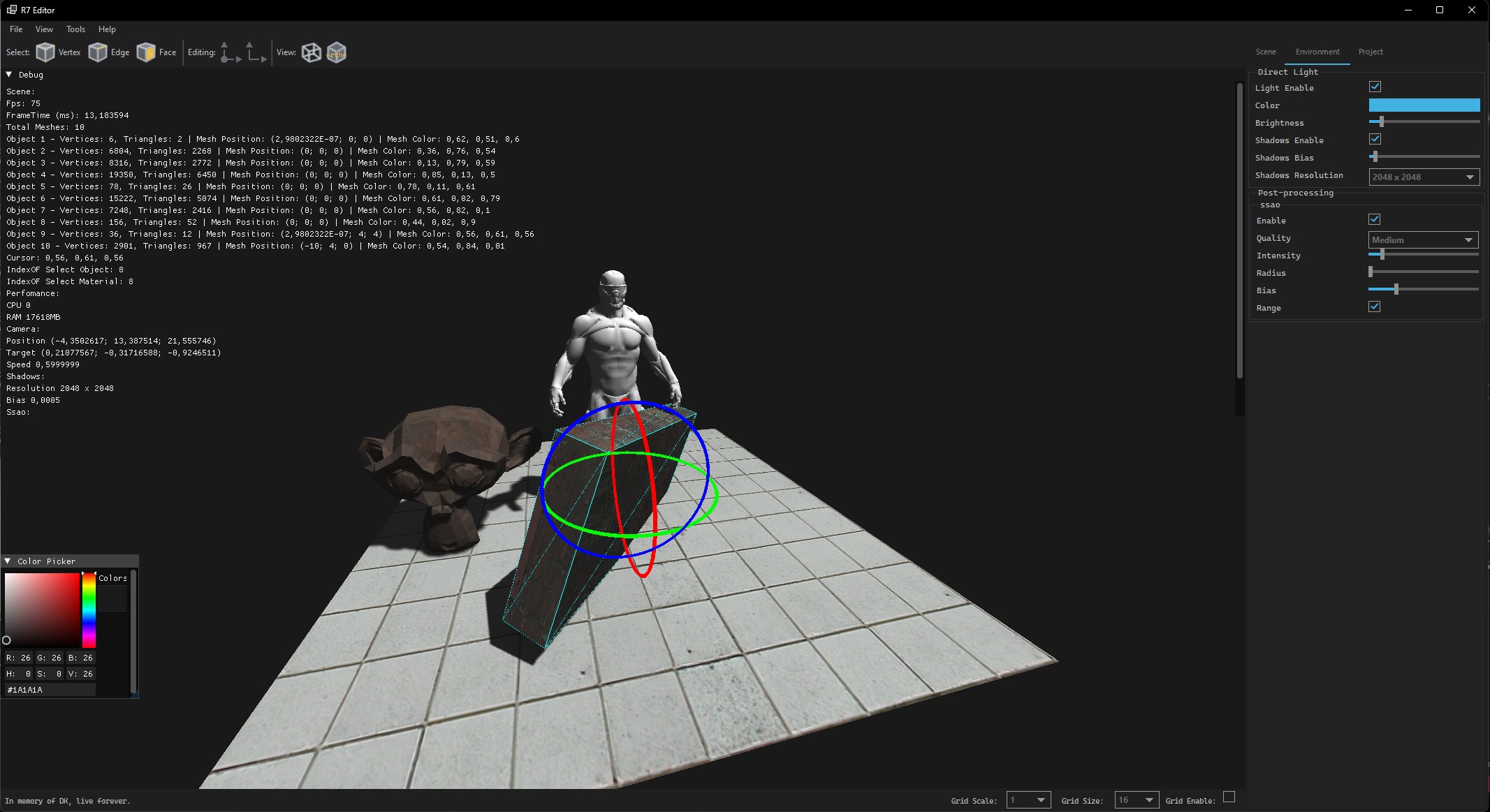
*Приложение 11:*



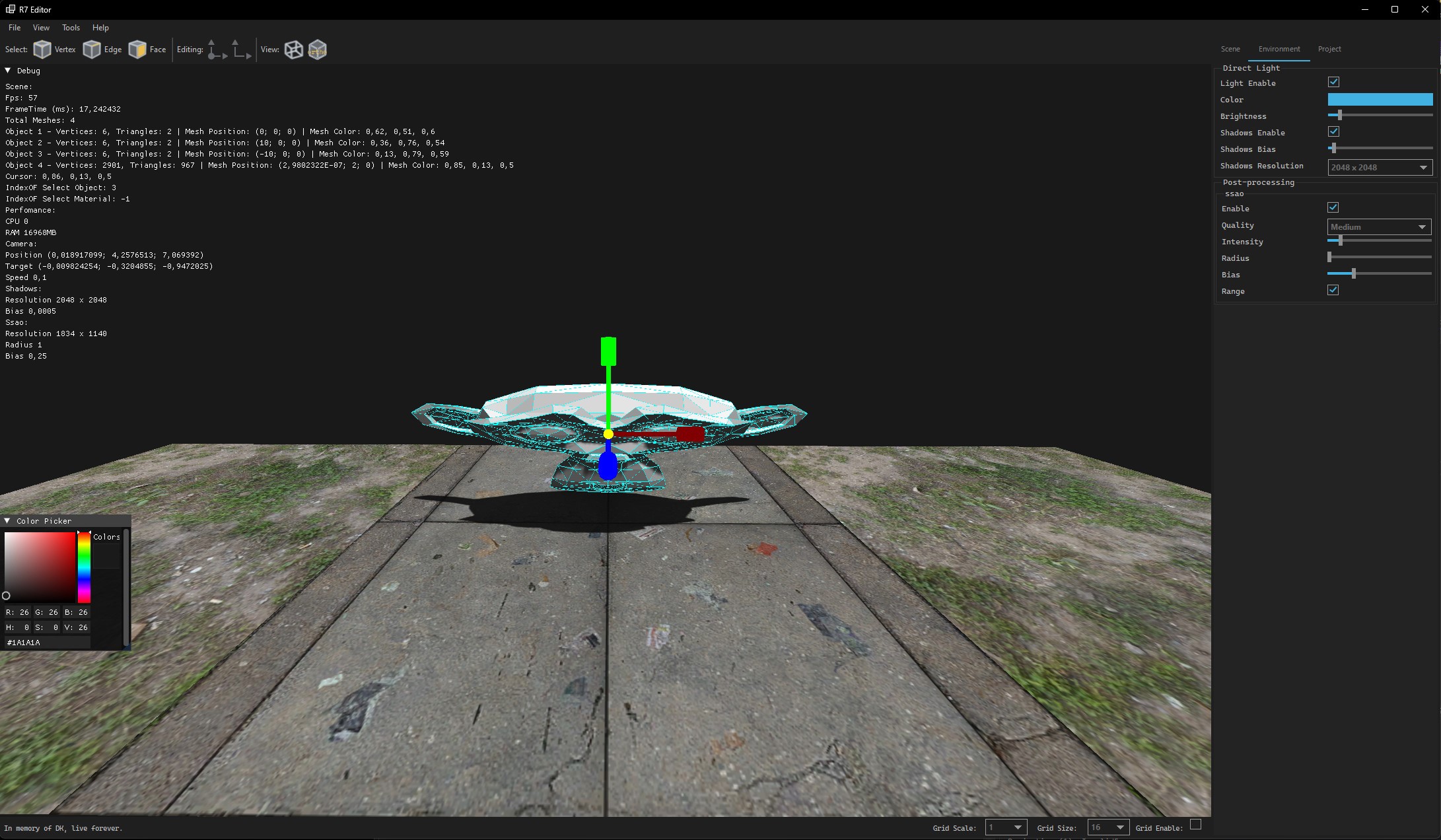
*Приложение 12:*



*Приложение 13:*



*Приложение 14:*



*Приложение 15:*

**2.2. Математическая библиотека (векторные вычисления)**

**Общие сведения**

Библиотека: mathlib

Заголовочный файл: /public/mathlib/mathlib.h

#include “mathlib/mathlib.h”

**Определения типов**

**class Vector;**

**Функции**

|  |  |  |
| --- | --- | --- |
| Возвращаемое значение | Имя | Описание |
| **void** | **VectorClear**  **(**  **Vector &vSrc**  **)** | Очищает вектор. Задает всем компонентам нулевое значение. |
| **void** | **VectorCopy**  **(**  **const Vector &vSrc,**  **Vector &vDest**  **)** | Производит копирование значения одного вектора в другой вектор. |
| **bool** | **VectorsAreEqual**  **(**  **const Vector &vSrc1,**  **const Vector &vSrc2,**  **float flTolerance = 0.0f**  **)** | Проверяет, являются ли два входных вектора одинаковыми, с учетом заданной погрешности. |
| **void** | **VectorAdd**  **(**  **const Vector &vSrc1,**  **const Vector &vSrc2,**  **Vector &vDest**  **)** | Сумма двух векторов. |
| **void** | **VectorSubtract**  **(**  **const Vector &vSrc1,**  **const Vector &vSrc2,**  **Vector &vDest**  **)** | Вычитание двух векторов. |
| **void** | **VectorMultiply**  **(**  **const Vector &vSrc1,**  **float flSrc2,**  **Vector &vDest**  **)** | Умножение вектора на число. |
| **void** | **VectorMultiply**  **(**  **const Vector &vSrc1,**  **const Vector &vSrc2,**  **Vector &vDest**  **)** | Умножение вектора на другой вектор. |
| **void** | **VectorDivide**  **(**  **const Vector &vSrc1,**  **float flSrc2,**  **Vector &vDest**  **)** | Деление вектора на число. |
| **void** | **VectorDivide**  **(**  **const Vector &vSrc1,**  **const Vector &vSrc2,**  **Vector &vDest**  **)** | Деление вектора на другой вектор. |
| **float** | **VectorLength**  **(**  **const Vector &vSrc**  **)** | Возвращает длину вектора. |
| **float** | **VectorNormalize**  **(**  **Vector &vSrc**  **)** | Нормализует значение компонентов вектора. Возвращает длину вектора. |
| **void** | **VectorAbs**  **(**  **const Vector &vSrc,**  **Vector &vDest**  **)** | Возвращает абсолютное значение (модуль) вектора. |
| **float** | **DotProduct**  **(**  **const Vector &vSrc1,**  **const Vector &vSrc2**  **)** | Скалярное произведение двух векторов. |
| **void** | **CrossProduct**  **(**  **const Vector &vSrc1,**  **const Vector &vSrc2,**  **Vector &vDest**  **)** | Выполняет перекрестное произведение двух векторов. Получает вектор, который перпендикулярен двум входным векторам. |

1. https://bigenc.ru [↑](#footnote-ref-1)
2. Меш (от англ. Mesh) – полигональная сетка, совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трёхмерной компьютерной графике и объёмном моделировании. [↑](#footnote-ref-2)
3. Фреймворк – это набор правил, шаблонов и инструментов, которые используются для построения продуктов или процессов как в программировании, так и в других областях. [↑](#footnote-ref-3)
4. Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. [↑](#footnote-ref-4)
5. CMake (от англ. cross-platform make — «кроссплатформенный make») — кроссплатформенное программное средство автоматизации сборки программного обеспечения из исходного кода.

   Не занимается непосредственно сборкой, а лишь генерирует файлы сборки из предварительно написанного файла сценария CMakeLists.txt и предоставляет простой единый интерфейс управления. [↑](#footnote-ref-5)
6. Решения — это просто контейнеры, используемые Visual Studio для упорядочения одного проекта или нескольких связанных. Когда вы открываете решение в среде Visual Studio, все содержащиеся в нем проекты загружаются автоматически. [↑](#footnote-ref-6)
7. Объе́ктно ориенти́рованное программи́рование (сокр. ООП) — методология или стиль программирования на основе описания типов/моделей предметной области и их взаимодействия, представленных порождением из прототипов или как экземпляры классов, которые образуют иерархию наследования. [↑](#footnote-ref-7)
8. Юнит-тест (от англ. unit test), или модульный тест, — это программа, которая проверяет работу небольшой части кода. Разработчики регулярно обновляют сайты и приложения, добавляют фичи, рефакторят код и вносят правки, а затем проверяют, как всё работает. [↑](#footnote-ref-8)
9. OBJ — формат файлов описания геометрии, разработанный в Wavefront Technologies для их анимационного пакета Advanced Visualizer. [↑](#footnote-ref-9)
10. FBX, FilmBox — ехнология и формат файлов разработан Kaydara, в настоящее время принадлежит и продолжает разрабатываться компанией Autodesk. [↑](#footnote-ref-10)