TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа информационных технологий и робототехники
Направление подготовки 09.03.02 «Программная инженерия»
Отделение школы (НОЦ) информационных технологий

БАКАЛАВРСКАЯ РАБОТА

Тема работы
Процедурная генерация ландшафтов

УДК 004.92.84:712

Студент

Группа	ФИО	Подпись	Дата
8К51	Федоров Константин Борисович		

Руководитель

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР ТПУ	Фофанов Олег Борисович	к.т.н.		

КОНСУЛЬТАНТЫ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОСГН ШБИП	Подопригора Игнат Валерьевич	к.э.н.		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ООД ШБИП	Винокурова Галина Федоровна	к.т.н.		

ДОПУСТИТЬ К ЗАЩИТЕ:

Руководитель ООП	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР ТПУ	Чердынцев Евгений Сергеевич	к.т.н.		

Томск – 2019 г.

ПЛАНИРУЕМЫЕ РЕЗУЛЬТАТЫ ОБУЧЕНИЯ ПО ООП

Код результата	Результат обучения (выпускник должен быть готов)
Р1	Применять базовые и специальные естественнонаучные и математические знания в области информатики и вычислительной техники, достаточные для комплексной инженерной деятельности.
Р2	Применять базовые и специальные знания в области современных информационных технологий для решения инженерных задач.
Р3	Ставить и решать задачи комплексного анализа, связанные с созданием аппаратно-программных средств информационных и автоматизированных систем, с использованием базовых и специальных знаний, современных аналитических методов и моделей.
Р4	Разрабатывать программные и аппаратные средства (системы, устройства, блоки, программы, базы данных и т. п.) в соответствии с техническим заданием и с использованием средств автоматизации проектирования.
Р5	Проводить теоретические и экспериментальные исследования, включающие поиск и изучение необходимой научно-технической информации, математическое моделирование, проведение эксперимента, анализ и интерпретация полученных данных, в области создания аппаратных и программных средств информационных и автоматизированных систем.
Р6	Внедрять, эксплуатировать и обслуживать современные программно-аппаратные комплексы, обеспечивать их высокую эффективность, соблюдать правила охраны здоровья, безопасность труда, выполнять требования по защите окружающей среды.
Р7	Использовать базовые и специальные знания в области проектного менеджмента для ведения комплексной инженерной деятельности.
Р8	Владеть иностранным языком на уровне, позволяющем работать в иноязычной среде, разрабатывать документацию, презентовать и защищать результаты комплексной инженерной деятельности.
Р9	Эффективно работать индивидуально и в качестве члена группы, состоящей из специалистов различных направлений и квалификаций, демонстрировать ответственность за результаты работы и готовность следовать корпоративной культуре организации.
Р10	Демонстрировать знания правовых, социальных, экономических и культурных аспектов комплексной инженерной деятельности.
Р11	Демонстрировать способность к самостоятельной к самостоятельному обучению в течение всей жизни и непрерывному самосовершенствованию в инженерной профессии.

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа: Инженерная школа информационных технологий и робототехники
Направление подготовки: 09.03.04 «Программная инженерия»
Отделение школы: Отделение информационных технологий

УТВЕРЖДАЮ:
Руководитель ООП
_____Чердынцев Е.С.
(Подпись) (Дата)

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

В форме:

бакалаврской работы

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

Студенту:

Группа	ФИО
8К51	Федорову Константину Борисовичу

Тема работы:

Процедурная генерация ландшафтов	
Утверждена приказом директора (дата, номер)	26.02.2019 г. №1513/с

Срок сдачи студентом выполненной работы:	19.06.2019 г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе *(наименование объекта исследования или проектирования; производительность или нагрузка; режим работы (непрерывный, периодический, циклический и т. д.); вид сырья или материал изделия; требования к продукту, изделию или процессу; особые требования к особенностям функционирования (эксплуатации) объекта или изделия в плане безопасности эксплуатации, влияния на окружающую среду, энергозатратам; экономический анализ и т. д.).*	Объектом исследования данной работы являются методы и алгоритмы процедурной генерации ландшафтов, а также стандарты форматов для хранения файлов конфигурации, изображений, 3D моделей и материалов, а также их программные реализации.

Перечень подлежащих исследованию, проектированию и разработке вопросов *(аналитический обзор по литературным источникам с целью выяснения достижений мировой науки техники в рассматриваемой области; постановка задачи исследования, проектирования, конструирования; содержание процедуры исследования, проектирования, конструирования; обсуждение результатов выполненной работы; наименование дополнительных разделов, подлежащих разработке; заключение по работе).*	1. Индустрия компьютерных игр 2. Структура приложения, реализовывающего компьютерную игру 3. Разработка процедурного генератора ландшафтов 4. Оптимизация в генерировании ландшафтов 5. Вспомогательные инструменты разработки 6. Социальная ответственность 7. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение
Перечень графического материала *(с точным указанием обязательных чертежей)*	1. Демонстрационные изображения алгоритмов 2. Примеры сгенерированных ландшафтов 3. Диаграмма классов в нотации UML 4. Алгебраические преобразования 5. SWOT-анализ 6. Диаграмма Ганта

Консультанты по разделам выпускной квалификационной работы
(с указанием разделов)

Раздел	Консультант
Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	Подопригора Игнат Валерьевич
Социальная ответственность	Винокурова Галина Федоровна

Дата выдачи задания на выполнение выпускной квалификационной работы по линейному графику	26.02.2019 г.

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР ТПУ	Фофанов Олег Борисович	к.т.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К51	Федоров Константин Борисович		

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа: Инженерная школа информационных технологий и робототехники

Направление подготовки: Программная инженерия

Уровень образования: Бакалавр

Отделение школы: Отделение информационных технологий

Период выполнения: осенний / весенний семестр 2018/2019 учебного года

Форма представления работы:

бакалаврская работа

(бакалаврская работа, дипломный проект/работа, магистерская диссертация)

КАЛЕНДАРНЫЙ РЕЙТИНГ-ПЛАН
выполнения выпускной квалификационной работы

Срок сдачи студентом выполненной работы:	19 июня 2019 г.

Дата контроля	Название раздела (модуля) / вид работы (исследования)	Максимальный балл раздела (модуля)
06.05.2019	Раздел 1. Индустрия компьютерных игр	10
08.05.2019	Раздел 2. Структура приложения, реализовывающего компьютерную игру	10
20.05.2019	Раздел 3. Разработка процедурного генератора ландшафтов	20
24.05.2019	Раздел 4. Оптимизация в генерировании ландшафтов	15
	Раздел 5. Вспомогательные инструменты разработки	15
28.05.2019	Раздел 6. Социальная ответственность	15
31.05.2019	Раздел 7. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	15

Составил преподаватель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР ТПУ	Фофанов Олег Борисович	к.т.н.		

СОГЛАСОВАНО:

Руководитель ООП	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР ТПУ	Чердынцев Евгений Сергеевич	к.т.н.		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Студенту:

Группа	ФИО
8К51	Фёдорову Константину Борисовичу

Школа	ИШИТР	Отделение школы (НОЦ)	ОИТ
Уровень образования	Бакалавриат	Направление/специальность	09.03.04 Программная инженерия

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. *Стоимость ресурсов научного исследования (НИ): материально-технических, энергетических, финансовых, информационных и человеческих* 2. *Нормы и нормативы расходования ресурсов* 3. *Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования*	*Амортизационные затраты на спецоборудование – 9030,00 рублей;* *Затраты на основную и дополнительную з/п – 167846,83 + 23500,00 рублей;* *Затраты на отчисление во внебюджетные фонды – 57404,05 рублей;* *Накладные расходы – 41404,94 рублей.* *Итоговая стоимость – 300185,82 рублей*
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. *Оценка коммерческого потенциала, перспективности и альтернатив проведения НИ с позиции ресурсоэффективности и ресурсосбережения*	*Описание потенциальных потребителей;* *Анализ конкурентных технических решений;* *SWOT-анализ.*
2. *Планирование и формирование бюджета научных исследований*	*Структура работ в рамках научного исследования* *Определение трудоемкости выполнения работ и разработка графика проведения научного исследования* *Бюджет проекта*
3. *Определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности исследования*	*Определение интегрального финансового показателя разработки* *Определение интегрального показателя ресурсоэффективности разработки* *Определение интегрального показателя эффективности*
Перечень графического материала *(с точным указанием обязательных чертежей)***:**	
1. *Оценка конкурентоспособности технических решений* 2. *Матрица SWOT* 3. *График проведения и бюджет НИ* 4. *Оценка ресурсной, финансовой и экономической эффективности НИ*	

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент	Подопригора Игнат Валерьевич	к.э.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К51	Фёдоров Константин Борисович		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА «СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ»

Студенту:

Группа	ФИО
8К51	Федорову Константину Борисовичу

Школа	ИШИТР	Отделение (НОЦ)	ОИТ
Уровень образования	Бакалавриат	Направление/специальность	09.03.04 Программная инженерия

Тема ВКР:

Процедурная генерация ландшафтов	
Исходные данные к разделу «Социальная ответственность»:	
1. Характеристика объекта исследования (вещество, материал, прибор, алгоритм, методика, рабочая зона) и области его применения	Объект исследования – набор средств разработки для процедурной генерации ландшафтов Рабочее место – рабочий стол с персональным компьютером в офисном помещении
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Правовые и организационные вопросы обеспечения безопасности: – специальные (характерные при эксплуатации объекта исследования, проектируемой рабочей зоны) правовые нормы трудового законодательства; – организационные мероприятия при компоновке рабочей зоны.	– Рабочее место при выполнении работ сидя регулируется ГОСТом 12.2.032 –78 – Организация рабочих мест с электронно-вычислительными машинами регулируется СанПиНом 2.2.2/2.4.1340 – 03 – Рациональная организация труда в течение рабочего времени предусмотрена Трудовым Кодексом РФ ФЗ-197
2. Производственная безопасность: 2.1. Анализ выявленных вредных и опасных факторов 2.2. Обоснование мероприятий по снижению воздействия	• Недостаточная освещенность рабочей зоны • Повышенный уровень электромагнитных излучений • Отклонение показателей микроклимата • Повышенный уровень шума на рабочем месте • Монотонность труда • Опасность поражения электрическим током
3. Экологическая безопасность:	Анализ негативного воздействия на окружающую природную среду: утилизация компьютеров и другой оргтехники, использованных люминесцентных ламп и мусорных отходов.
4. Безопасность в чрезвычайных ситуациях:	Возможные чрезвычайные ситуации: • Пожар

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
	Винокурова Галина Федоровна	к.т.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К51	Федоров Константин Борисович		

Оглавление

Определения, обозначения и условные сокращения

3D-модель – это объемный образ какого-либо объекта в цифровом виде. Может содержать информацию о вершинах, ребрах, полигонах и прочие данные.

Application User Interface (API) – (рус. программный интерфейс приложения) это описание возможных способов, с помощью которых одна компьютерная программа может взаимодействовать с другой. В состав интерфейса могут входить классы и структуры, процедуры, функции и константы.

BMP – это формат для хранения растровых изображений, разработанный компанией Microsoft. Изображения в данном формате сохраняются без потери данных.

C – это компилируемый язык программирования со статической типизацией. Был разработан в 1969-1973 годах Деннисом Ритчи. Первоначально был создан как специализированный язык для реализации операционной системы UNIX, однако был перенесён на другие платформы. Конструкции языка близко сопоставлены типичным машинным инструкциям, что сделало язык удобным для проектов, которым был свойственен язык ассемблера, например, операционным системам.

C++ – это компилируемый язык программирования со статической типизацией. Отличительной особенностью языка является обратная совместимость с C, а также поддержка таких парадигм программирования, как процедурное программирование, объектно-ориентированное программирование и обобщённое программирование

INI – это текстовый формат файлов конфигурации, предназначенный для хранения настроек приложений. Весь файл представлен списком секций и соответствующих им атрибутам.

MTL – это формат файлов, содержащий информацию о материалах для 3D моделей в формате OBJ. В одном файле могут находиться несколько материалов.

OBJ – это текстовый формат файлов описания трехмерной геометрии, разработанный компанией Wavefront Technologies. Формат файла является открытым и широко распространенным в приложениях для работы с 3D графикой.

Software Development Kit (SDK) – (рус. набор средств разработки) это список инструментов, позволяющий разработчикам создавать приложения с использованием некоторой технологии.

Автоматическое тестирование – это процесс нахождения неисправностей и недочетов в разрабатываемом программном продукте путем эксплуатации автоматической системы. Данный подход увеличивает нагрузку на разработчиков ПО, однако сокращает время на ручное тестирование.

Декаль – это изображение (чаще всего с прозрачными элементами), накладываемое поверх другого, для имитации дополнительных деталей и внесения разнообразия во внешний вид модели, например, царапин, листьев, следов животных и пр.

Игровой движок – это основное программное обеспечение для создания компьютерной игры. Он содержит в себе основную механику создания игрового мира и взаимодействия с ним, не углубляясь в детали каких-либо конкретных проектов.

Игровой ландшафт – это совокупность моделей, текстур и прочих ресурсов, представляющих полную информацию о ландшафте и расположении конкретных структур в игре.

Карта высот – это способ представления трёхмерных объектов на изображении. В данном методе две координаты направлены по ширине и

высоте соответственно, а третья влияет на цвет пикселя – от черного (самая нижняя точка) к белому (самая высокая точка модели). Часто применяется для географических обозначений, но накладывает ограничения на поверхность, так как не отображает полости внутри модели.

Компьютерная игра – это компьютерная программа, предназначенная для организации игрового процесса, связанного с взаимодействием с другими игроками или компьютером.

Макрос – это символьное имя, заменяющее несколько команд языка программирования. Основным отличием от функции является тот факт, что код макроса будет интегрирован в программу столько раз, сколько был вызван, в то время как код функции будет расположен лишь один раз.

Массив – это структура данных, которая хранит набор значений, именуемых элементами массива, идентифицируемых по индексам.

Нормаль – это вектор, определяющий фронтальную сторону полигона и влияющий на освещение модели.

Полигон – это совокупность вершин, ребер и граней, определяющих форму многогранного объекта в трёхмерной графике.

Процедурная генерация – это метод создания какого-либо контента без непосредственного участия человека, благодаря заложенным в систему алгоритмам и методам.

Реиграбельность – это свойство игры, отображающее количество повторных прохождений одной и той же игры.

Статическая библиотека – это набор уже скомпилированных подпрограмм или объектов, подключаемых к исходной программе в виде объектных файлов в виде одного файла.

Текстура – это изображение, накладываемое на 3D модель и отвечающее за цвет модели.

Введение

Создание компьютерных игр в настоящее время стала популярным направлением разработки программного обеспечения. На данный момент существует множество игр разных жанров и масштабов. Новые игры выпускаются практически ежедневно, как крупными компаниями, так и небольшими студиями.

Разработка новой игры является долгим и трудоемким процессом и состоит из множества этапов. Во многих жанрах требуется наличие большого игрового мира, наполненного различными структурами, например, лесами, горами и реками. Зачастую разработка ландшафта занимает большое количество времени. Поэтому данный процесс часто стараются оптимизировать.

Процедурная генерация ландшафтов способна значительно сократить время, затрачиваемое на создание игрового мира. Даже в играх, где не требуется пересоздание карты в новой игровой сессии, данный подход способен значительно упростить работу дизайнеров и 3D художников. Однако этот метод требует вовлечения программистов в процесс создания игрового мира. Процедурная генерация является сложной в реализации, поэтому при всех достоинствах она используется не очень часто. Использование сторонних решений позволит разработчикам значительно сократить время разработки и достичь качественных результатов (приложения 1, 2 и 3).

Данная тема была освещена в статьях, объясняющих алгоритмы генерации ландшафтов[1] или тонкости работы с экспортом 3D моделей[2]. Однако данная работа предназначена для предоставления полной и исчерпывающей информации о проектировании архитектуры решения, технических деталях и выборе средств разработки. Также в работе представлены методы повышения эффективности полученного решения с помощью упрощения вычислений или многопоточной обработки данных.

Объект и методы исследования

Целью работы является разработка методов и алгоритмов для быстрой генерации игровых ландшафтов, представленных 3D моделями, текстурами и прочими файлами для игр с «открытым миром».

Задачи для достижения цели работы:

- Проанализировать процесс создания ландшафтов стандартным способом.
- Выявить этапы, которые можно провести автоматически.
- Провести выбор средств и инструментов.
- Разработать архитектуру приложения.
- Провести разработку набора средств.
- Внедрить систему автоматического тестирования.
- Создать сайт проекта с документацией.

Объектом исследования являются методы и алгоритмы процедурной генерации ландшафтов, а также стандарты форматов для хранения файлов конфигурации, изображений, 3D моделей и материалов.

Предметом исследования является создание набора средств разработки для процедурной генерации ландшафтов, готового к интеграции в сторонние проекты, а также содержащего другие полезные инструменты.

Актуальность исследования состоит в том, вопрос автоматизации создания игровых ландшафтов в настоящее время становится востребованным, но недостаточно проработанным. Из-за этого многие разработчики не решают использовать данный подход, несмотря на его преимущества. Готовое решение, готовое к интеграции, поможет снизить входной порог в данной сфере.

1. Индустрия компьютерных игр.

1.1. Историческая справка

В современном мире информационные технологии получили широкое применение в различных сферах деятельности человека. Компьютер позволил обрабатывать колоссальные объемы информации за небольшое время. Однако использование компьютера с течением времени стало не только производственной необходимостью, но и для развлечений.

Компьютерная игра – это компьютерная программа, предназначенная для организации игрового процесса, связанного с взаимодействием с другими игроками или компьютером.

Часто вместо термина "компьютерная игра" может использоваться "видеоигра", являющийся полностью синонимичным. Это объясняется тем, что в компьютерных играх, как правило, игровая ситуация воспроизводится на экране дисплея или другого экрана.

С 2011 года компьютерные игры официально признаны в США самодостаточным видом искусства.

Первой компьютерной игрой в привычном для современного человека понимании является Tennis for Two[3] – симулятор тенниса, созданный Уильямом Хигинботамом на базе осциллографа. Эта игра была разработана для посетителей дня открытых дверей в Брукхейвенской лаборатории. Игра получила популярность, но просуществовала менее двух лет – в 1959 году установка была разобрана для нужд лаборатории.

С этого момента индустрия игр начала свой рост. В 1970-х годах в Америке стали появляться аркадные автоматы и первые игровые приставки, например, Magnavox Odyssey и Pong. Несмотря на набирающуюся популярность приставок, игры были однотипными и низкокачественными, что повлекло за собой кризис индустрии компьютерных игр 1983 года. Данное

событие повлияло на разработчиков игр, и к 1990-м годам игровая индустрия значительно изменилась – игр стало больше, качество повысилось, а устройствами для запуска все чаще становились персональные компьютеры.

1.2. Современное состояние

В настоящее время игровая индустрия все чаще имеет коммерческий характер. Крупные компании занимаются созданием и публикацией игр. Вместе с тем, у небольших компаний или одиночных разработчиков появилось множество возможностей для реализации своих идей. Таких разработчиков принято называть инди (Indie), образованного от английского слова independent (независимый).

Игровая индустрия разделилась на несколько основных сегментов:

1. PC gaming – игра на персональных компьютерах (ПК). Популярность обусловлена тем, что ПК можно использовать для решения большого количества задач, и поэтому они широко распространены. Однако комплектующие и операционные системы могут сильно варьироваться на разных устройствах, а значит, требуется тщательное тестирование перед релизом. ПК занимает 25% рынка игр.
2. Console gaming – игра на специальных приставках. Основным преимуществом данного способа является одинаковая конфигурация всех приставок одной модели. Это значительно упрощает тестирование игр разработчиком, что влечёт за собой высокое качество разработанного продукта. 28% рынка сосредоточено именно на консолях. Сами консоли, как правило, стоят меньше, чем их комплектующие, зато игры стоят намного дороже, чем для персональных компьютеров.
3. Mobile gaming – игра на смартфонах и телефонах. На 2018 год, это самый прибыльный сегмент — 47% денег заработаны именно на

мобильных играх Это обусловлено широкой распространённостью смартфонов и наличием небольших игровых сессий.

Вместе с развитием индустрии разработки игр активно развиваются сопутствующие области. Основными являются:

- магазины игр;
- магазины инструментов для разработки игр;
- игровые движки;
- издательства компьютерных игр;

Магазины компьютерных игр появились в начале 00-х. Одним из первопроходцев стал Steam, разработанный компанией Valve в 2003 году. Тогда эта система цифровой дистрибуции являлась решением для продажи игр, созданных исключительно самой студией Valve. Однако со временем, в магазине появились игры и других разработчиков. Позже был открыт сервис Steam Greenlight, который позволил небольшим студиям или сольным разработчикам размещать свои игры в магазине, при условии прохождения модерации и пользовательского отбора.

В настоящее время наблюдается широкий спектр магазинов цифровой дистрибуции. Наиболее популярными из них являются:

- Steam — магазин, курируемый издательством Valve;
- GOG — магазин, курируемый разработчиком CD Project Red;
- Battle.net — магазин, содержащий игры разработчиков Activision Blizzard;
- Origin —магазин издательства EA;
- Uplay — магазин издательства Ubisoft;
- EpicGames Store — магазин, курируемый издательством EpicGames;

Популярность разработки игр повлекла создание рынка инструментов разработки и игровых движков. Часто эти понятия взаимодополняемы: например, библиотека для работы с файлами определенного типа тесно связана с пользовательским интерфейсом какого-либо конкретного игрового движка.

Наиболее известными магазинами плагинов являются:

- Unity Asset Store – магазин плагинов для приложений, разработанных на игровом движке Unity;
- Itch.io – магазин ассетов и плагинов независимо от игрового движка;
- Unreal Engine 4 Marketplace – магазин плагинов для приложений, разработанных на игровом движке Unreal Engine 4;

1.3. Сферы применения и использования

Использование компьютерных игр в образовании является предметом активных обсуждений и исследований.

Видеоигры могут быть полезны в обучении, так как они создают эффект симуляции действия, но при этом не несут опасности. Например, при обучении пилотов в воздушных войсках, их не пускают сразу за штурвал боевой машины, а используют симулятор вождения самолета, чтобы подготовить пилотов к реальным действиям.

Также компьютерные игры используются в медицине для коррекции зрительного аппарата у детей, так как такая форма взаимодействия позволяет удерживать внимание пациентов, что особенно актуально в случае работы с детьми.

1.4. Индустрия компьютерных игр

В современном мире создание видеоигр находится на верхних позициях в индустрии развлечений. По скорости роста за последние годы индустрия видеоигр опередила даже кинопроизводство.

Структура индустрии компьютерных игр состоит из нескольких крупных составляющих, требующих тщательного рассмотрения.

Одним из наиболее крупных элементов данной индустрии можно назвать игровые платформы. Деление игр на консольные, мобильные и игры для ПК является наиболее распространенным. Однако каждый из пунктов можно разделить еще на несколько крупных. Разделение игр по платформам можно представить следующим образом:

- ПК
  - Windows
  - Mac/OS
  - Linux
- Консоли
  - PlayStation
  - Xbox
  - Nintendo
- Мобильные
  - IOS
  - Android
  - Windows Mobile
- Веб игры
- Виртуальная реальность
- Стриминговые сервисы

Еще один этап представлен процессом непосредственной разработки игр. Многие компании используют собственные разработки, а некоторые используют сторонние продукты, например Unreal Engine 4 или Unity.

В качестве следующего этапа можно выделить издание игр. В настоящее время существует множество издательств, которые помогают разработчикам не отвлекаться на юридические или маркетинговые цели. Эти задачи возлагаются на издательства. За эту работу чаще всего издатель берет определенный процент с продаж. Сюда же можно отнести производство и продажу сувенирной продукции, связанной с играми, а также популяризацию игр в СМИ, и рекламу новых продуктов.

2. Структура приложения, реализовывающего компьютерную игру

2.1. Классификация компьютерных игр

В настоящее время существует большое число жанров компьютерных игр. Однако на практике редко встречаются игры, которые относились бы к какому-нибудь одному жанру. В большинстве случаев продукты являются комбинацией 2-3 различных жанров.

Самыми распространенными жанрами компьютерных игр на данный момент являются:

- Экшен – жанр компьютерных игр, в котором геймплей в основном сосредоточен на скорости реакции игрока, а также координации рук. Это один из наиболее популярных жанров на настоящий момент.
    - Шутер от 1 лица – это жанр компьютерных игр, в котором пользователь видит игровой мир глазами протагониста. Основным элементом данного жанра является стрельба из различных видов оружия. Такие игры имеют достаточно низкий порог вхождения, но трудны для достижения высокого уровня мастерства.
    - Шутер от 3 лица. Данный жанр похож на шутер от 1 лица с единственным отличием в точке обзора — игрок видит мир не глазами протагониста, а с позиции, позволяющей увидеть персонажа.
    - Файтинг – это жанр, в котором игрок сражается с противниками, в основном, при помощи ударов ближнего боя. Зачастую, удары могут складываться в сложные комбинации, которые наносят больший урон, нежели в отдельности.

    - Слешер – это жанр игр, геймплей которого сосредоточен на истреблении множества врагов с использованием оружия ближнего боя.
- Аркада – это жанр компьютерных игр, который характеризуется коротким по времени, но в то же время интенсивным игровым процессом.
- Симуляторы – это игры, задача которых состоит в имитации управления каким-либо процессом, аппаратом или транспортным средством.
    - Технические симуляторы
    - Спортивные симуляторы
    - Строительные симуляторы
- Стратегии – это жанр компьютерных игр, где для достижения поставленных целей игрок вынужден применять стратегическое мышление, и оно противопоставлено быстрым действиям и реакции, которые, как правило, не обязательны для успеха в таких играх.
    - Пошаговые
    - В реальном времени
    - С непрямым управлением
- Приключения – это жанр игр, где игрок сталкивается с интерактивной историей. Основной упор в играх данного жанра сделан на повествовании и исследовании мира, ключевая же роль в игровом процессе отведена решению головоломок и задач.
    - Приключенческий боевик
    - Визуальная новелла
    - Ролевые игры

- Логические игры – это название жанра компьютерных игр, где основной целью является решение определенных логических задач, а следовательно, требуют умственных усилий.
    - Квесты
    - Головоломки
    - Карточные
- Текстовые игры можно выделить в отдельный жанр благодаря отсутствию какой-либо графической составляющей в процессе. Все, с чем взаимодействует игрок, выражено с помощью текста. В ряде проектов используется ASCII графика, то есть создание визуальных образов с помощью символов.

Помимо классификации игр по жанрам существует разделение игр по количеству игроков:

- Однопользовательские (1 игрок)
- Многопользовательские (от 2 до приблизительно 20 игроков в одной игровой сессии)
- Массовые многопользовательские (несколько десятков игроков одновременно на одной карте)

Классификация игр по платформам:

- Игры для персональных компьютеров
- Игры для консолей
- Игры для мобильных устройств

2.2. Состав компьютерной игры

Компьютерные игры в большинстве случаев состоят из:

1. Геймплей — это сам процесс взаимодействия с игроком, логика поведения игры в различных ситуациях.

2. Нарратив — это описание мира, его особенностей и истории, а также персонажей. Эта часть компьютерных игр обычно нечасто связана с написанием кода, но требует больших вложений со стороны писателей, дизайнеров и художников.

Процесс разработки компьютерных игр можно разбить на следующие этапы:

1. Разработка идеи. На данном этапе обсуждается основные черты будущего проекта и его отличительные особенности.
2. Описание геймплея. Этот процесс предназначен для детальной проработки интерактивной составляющей игры.
3. Выбор стратегии монетизации. На данном этапе выбирается платная или бесплатная модель монетизации, а также наличие микроплатежей и их размер.
4. Разработка игры. Этот этап заключается в непосредственном создании игры с выполнением всех ранее разработанных требований. Сюда входит не только написание программного кода, но и создание моделей и изображений, написание текстовой информации, перевод на другие языки и другие процессы.
5. Тестирование. Оно позволяет найти ошибки и исправить их еще до релиза.
6. Предрелизная подготовка. Этот этап включает в себя множество подготовительных процессов, таких как размещение игры в магазинах, рекламная кампания и пр.
7. Выпуск.

2.3. Ландшафты в компьютерных играх

Одним из этапов создания компьютерных игр с наличием игрового мира является создание ландшафта. Это трудоёмкий процесс, требующий

слаженной работы специалистов разного профиля. Данный процесс можно разбить на несколько этапов:

1. Выбор расположения основных объектов карты, её основных отличительных черт.
2. Создание основной поверхности мира.
3. Создание объектов, расположенных на карте.
4. Расстановка объектов.
5. Создание текстур.
6. Проверка на совместимость ландшафта с геймплеем.
7. Внесение правок.

Большинство игр имеют неизменяющийся мир, то есть при начале игры заново, все локации будут располагаться на тех же местах, как и в предыдущем прохождении. Это позволяет сэкономить время на разработку игры, но уменьшает реиграбельность — способность к повторным прохождениям игры.

Также существуют и игры, мир в которых создаётся при помощи некоторых алгоритмов. Подход к созданию ландшафтов, при котором мир игры изменяется, называется процедурной генерацией. Такой подход значительно увеличивает реиграбельность, но усложняет процесс создания ландшафтов. Более того, появляется риск, что мир будет построен неверно и это негативно скажется на эмоциональном отклике игроков.

Наиболее важным данный этап становится в играх с «открытым миром». Данный термин обозначает виртуальный мир, который может быть свободно исследован игроком. Данный тип игрового мира противопоставляется играм с менее интерактивным геймплеем, которые часто называют «рельсовыми» или «линейными».

Разработка «открытого мира» в играх является причиной больших трудозатрат, ведь ландшафт в таких проектах должен быть большого размера, а интересные и неповторимые места должны встречаться как можно чаще.

3. Разработка процедурного генератора ландшафтов

3.1. Архитектура приложения

Для решения проблем, связанных с большой продолжительностью создания ландшафтов, было решено создать собственный инструмент для процедурной генерации ландшафтов. Для этого необходимо выделить ключевые моменты разрабатываемого ПО:

1. Инструмент должен быть удобным в использовании и легким в интеграции. Это означает, что ПО должно быть представлено не как отдельное приложение, а как библиотека. Выбранным вариантом готового продукта стала статическая библиотека, так как она не требует наличия дополнительных файлов к готовому стороннему ПО, что упрощает распространение продукта.
2. Инструмент должен быть кросс-платформенным, так как разработчики заинтересованы в охвате наибольшего количества доступных платформ. Из этого условия следует необходимость в отсутствии связи с другими библиотеками или средствами, а сам проект должен быть написан на устоявшемся стандарте распространённого языка.
3. ПО должно иметь высокую производительность. Из этого следует, что язык должен быть компилируемым, что позволит проводить оптимизации автоматически на этапе компиляции. Поэтому в качестве кандидатов были выбраны языки C и C++.
4. API должно быть совместимо с наибольшим числом проектов. Так как большинство игр написано на языках C/C++, а C++ имеет совместимость с C, то API, предоставленное на языке C может использоваться как в проектах, написанных на C и на C++. Значит, язык API должен быть C.
5. Внутренний язык ПО должен иметь высокий функционал и поддерживать парадигмы ООП, так как в разработке потребуются

шаблоны и наследование. В связи с этим внутренним языком ПО был выбран C++ 11 версии стандарта.

Также необходимо выделить функциональные требования для процедурного генератора:

1. Требуется наличие возможности для генерации лесов, полян, гор, рек и водоёмов. Расположение данных биомов должно быть основано на высоте над уровнем моря в следующем порядке (сверху вниз): горы, леса/поляны, водоёмы. Реки должны представлять собой русла, расположенные из верхней точки в нижнюю, пересекая другие биомы.
2. Необходимо наличие широкого спектра настроек для создания необходимого ландшафта, которые можно задавать как через файл, так и с помощью API.
3. Необходимо наличие скриптов на языке Python для экспорта данных в Blender 3D как для сгенерированного ландшафта целиком, так и для части.
4. Необходимо наличие консольного приложения для составления файлов конфигурации и их проверки.
5. Требуется наличие не менее 3 примеров для демонстрации использования разрабатываемого набора средств.
6. Необходим экспорт файлов в форматах OBJ, MTL, BMP, INI.
7. Требуется импорт файлов в форматах BMP и INI.

3.2. Выбор программных средств.

Разрабатываемая статическая библиотека написана на языках C и C++, что позволяет использовать широкий спектр различных инструментов. Во время разработки использовались следующие инструменты:

Для создания файлов проектов:

- CMake. Это кроссплатформенная система, позволяющая автоматизировать создание файлов для различных IDE и сборку программного обеспечения из исходного кода. CMake не занимается непосредственно сборкой, а лишь генерирует файлы управления сборкой из специальных файлов. Данный инструмент необходим для возможности быстрого переноса проекта с компьютера на компьютер независимо от операционной системы, IDE и прочих факторов. Также в CMake встроена утилита CTest, позволяющая проводить автоматическое тестирование.

Для написания кода и прочих файлов:

- Visual Studio 2017. Эта IDE была выбрана в связи с тем, что она обладает большим количеством встроенных инструментов, например, интеллектуальным автодополнением, удобным отладчиком и поддержкой множества форматов файлов.
- Visual Studio Code. Этот текстовый редактор был выбран в качестве облегченного аналога Visual Studio 2017 для тех случаев, когда файл необходимо незначительно отредактировать или просмотреть в текстовом или hex виде.

Для работы с 3D графикой:

- Blender 3D. Этот 3D редактор был выбран из-за свободного распространения, наличия API на языке Python, а также широкого функционала.
- Средство 3D просмотра. Это ПО, предоставляемое в комплекте с Windows 10, решено было использовать как дополнительный инструмент для просмотра моделей, чтобы проверять на корректность сгенерированные модели. В отличии от Blender 3D, эта программа не исправляет некорректные, но близкие к

спецификации, модели. Это позволяет с высокой вероятностью находить ошибки в созданных файлах.

Для работы с 2D графикой:

- GIMP. Данный графический редактор было решено использовать из-за свободной политики распространения, а также большого спектра настроек при создании BMP файлов. Например, данное программное обеспечение позволяет создавать BMP файлы, поддерживающие альфа-канал, отвечающий за прозрачность пикселей в изображении.
- Paint. Аналогично средству 3D просмотра из списка программ для 3D графики, это ПО использовалось как контрольное для просмотра полученных изображений.

3.3. Состав набора средств разработки

После анализа требований был выявлен необходимый состав набора средств разработки. Он включает в себя:

1. Статическая библиотека ядра генератора. Этот компонент ответственен непосредственно за генерацию ландшафта с помощью заданных настроек.
2. Статическая библиотека API генератора. Она позволяет создавать, уничтожать и запускать генератор, а также предоставляет доступ к более чем 100 параметрам кастомизации.
3. Скрипты, написанные на Python для импорта сгенерированного ландшафта в программу работы с 3D графикой Blender 3D в автоматическом режиме. Они позволяют быстро рассмотреть результаты даже без рабочего прототипа игры.
4. Консольное приложение. Оно позволяет использовать генератор не только разработчиками ПО, но и пользователями, далекими от программирования.

5. Подробная документация в виде HTML страниц. В продуктах данного типа наиболее часто встречающейся проблемой является нехватка примеров или документации, так как она сильно влияет на порог вхождения.
6. Готовые примеры приложений для разработчиков., включающее код, тестовые изображения и модели.

Таким образом, данное решение способно обеспечить легкую интеграцию в сторонние проекты и предлагает широкий набор средств для процедурной генерации ландшафтов.

3.4. Общая структура приложения

Основными классами и структурами разрабатываемого ПО являются:

- Класс Generator. Это основной класс, отвечающий за процесс генерации ландшафта. Ведет логирование, назначает приоритет модулям.
- Класс World. Это хранилище информации, необходимой нескольким модулям для чтения и записи. Содержит карты высот, настройки, уровень моря и прочую информацию
- Класс Matrix<T> (шаблонный). Он предназначен для работы с двумерными массивами. Содержит базовые инструкции для корректного изменения размеров, создания и удаления данных.
- Структура WORLD_SETTINGS. Содержит пользовательские настройки, согласно которым будет проводиться генерация. Содержит функции чтения/записи в INI файлы, а также проверку на корректность и методы для получения примерных затрат оперативной памяти и хранилища данных.
- Класс IModule. Он отображает интерфейс модуля, являющегося отдельным шагом генерации.

- Класс MatrixCommon. Состоит из статических функций обработки матриц. Содержит нахождение минимума/максимума, поиска, сдвигов всех элементов, функции генерации шума и прочее.
- Класс Random. Класс предназначен для генерации псевдослучайных чисел, а также часто используемые функции с такими числами.

Диаграмма классов представлена в приложении 4. На ней представлены основные классы, структуры и методы, использующиеся при работе генератора ландшафтов.

Генерация проходит в несколько этапов. За каждый этап отвечает отдельный модуль (являющийся наследником IModule), а возврат из текущего шага в предыдущий невозможен. Такой подход позволяет упростить разработку и предотвратить путаницу.

3.5. Создание структуры для хранения настроек

Структура, содержащая настройки генератора имеет большое количество полей. Их можно задавать как с помощью файлов конфигурации, так и через API. Каждая настройка должна быть проверена на валидность. Дополнительно каждая настройка должна содержать информацию для документации и расположении в файле INI. При разработке генератора список настроек постоянно изменялся, и добавление или удаление каждой настройки влекло за собой большое количество действий. В результате увеличивался риск забыть что-либо добавить. Для решения этой проблемы было решено использовать приём, похожий на популярный подход X-Macro, но с некоторыми изменениями.

В стандартном виде техника X-Macro выглядит следующим образом:

Для создания списка переменных (value1, value2, ...), необходимых в переборе, создается макрос, выглядящий следующим образом:

```
#define VARIABLES \
VAR(value1) \
VAR(value2) \
VAR(...)
```

Стоит отметить, что макрос VAR не определен на данном этапе.

В месте, где требуется обработка списка переменных, размещается определение макроса. Например, для заполнения структуры полями (value1, value2, …) целочисленного типа код будет выглядеть следующим образом:

```
struct Foo
{
#define VAR(name) int name;
VARIABLES
#undef VAR
}
```

В данном фрагменте кода определяется макрос VAR, отвечающий за обработку переменных, используется макрос со списком переменных VARIABLES, после чего макрос VAR лишается определения с целью поддержки последующих примеров. Например, при добавлении функции печати для той же структуры, код будет выглядеть так:

```
struct Foo
{
#define VAR(name) int name;
VARIABLES
#undef VAR

void print()
{
#define VAR(name) printf("%s = %d\n", #name, name);
VARIABLES
#undef VAR
}
}
```

Макросы обрабатываются препроцессором, и после его работы код будет выглядеть таким образом:

```
struct Foo
{
int value1;
int value2;
...
```

```
void print()
{
printf("%s = %d\n", "value1", value1);
printf("%s = %d\n", "value2", value2);
…
}
}
```

У данного подхода можно выделить достоинства и недостатки.

Достоинства:

- при добавлении новой переменной требуется изменение только в одном месте – в макросе, определяющем список переменных;
- отсутствует возможность пропустить добавление логики;
- значительное сокращение строк кода при большом списке;
- легкая модификация кода;

Недостатки:

- список переменных определяется внутри файла с реализацией;
- присутствует дублирование строк кода с вставкой списка и снятия определения макроса;
- доступ только к одному названию в списке;
- отладка макросов является сложным процессом;

Недостаток с отладкой исправить невозможно без дополнительного ПО. Для решения остальных проблем были проведены следующие меры:

Код макроса всего списка и снятия определения макроса элемента списка был помещен в отдельный заголовочный файл. Таким образом, вместо

```
VARIABLES
#undef VAR
```

необходимо писать только

```
#include "variables_list.h"
```

Это решило сразу 2 проблемы - определение макроса списка внутри файла с реализацией и дублирование строк кода.

Для решения проблемы, связанной с одним полем в списке, решено было использовать макрос элемента списка с большим числом аргументов.

В реализации приложения данный подход позволил упростить обработку следующей информации:

- название параметра;
- тип параметра;
- значение по умолчанию;
- диапазон значений;
- комментарии;
- методы API и их реалзацию;
- описание нахождения параметра в INI файле;
- информация для документации;

3.6. Генерация базовой поверхности

Под термином «базовая поверхность» в контексте процедурной генерации ландшафтов следует понимать поверхность с высотами в каждой конкретной точке, рассчитанными по некоторому алгоритму, и представляющую ландшафт без ключевых особенностей вроде рек, озер и гор.

Пользователю предоставляется выбор алгоритма для заполнения, а также расположение ключевых точек относительно карты.

На данный момент в разработанном продукте имеется 3 алгоритма генерации базовой поверхности. Первый и самый простой – плоскость. Он означает, что вся поверхность будет представлена ровным слоем на заданной высоте, без каких-либо колебаний. Второй алгоритм представлен белым шумом. Он позволяет внести хаотические колебания. Данный метод подходит для относительно плоских миров с отсутствием крупных неровностей, но

небольшими отклонениями от заданной высоты. Третий алгоритм является наиболее сложным и часто используемым – алгоритм diamond-square (рис. 1).

Этот алгоритм в стандартной реализации является рекурсивным, однако было решено реализовать его итерационным методом, чтобы избежать переполнения стека при создании карт большого размера.

Рисунок 1. Демонстрация работы алгоритма diamond-square, разбитая по шагам.

Алгоритм[4] состоит из двух шагов: первый – «square» – определяет высоту центральной точки в квадратной рамке с помощью усреднения высот на углах добавления случайной величины. Причем очень важно, чтобы величина прямо зависела от размера текущей рамки, иначе вся карта превратится в шум. На втором шаге – «diamond» – алгоритм рассчитывает высоту всех точек рамки, которые лежат на серединах её сторон. Здесь усредняются не только две точки, между которыми и заключена новая, но и пара точек на прямой, перпендикулярной первым двум. Эти две точки были получены на шаге «square». Таким образом, эти две высоты должны быть посчитаны до начала выполнения второго шага. Из-за этого алгоритм работает поэтапно. Преимуществом данного подхода является малая вычислительная сложность, недостатком является невозможность распараллеливания вычислений на несколько ядер[5]. Следует отметить, что в стандартный алгоритм также были внесены изменения, влияющие на периодичность шума. Эта модификация позволяет создавать более сложные шумы, являющиеся наложением сразу нескольких других шумов.

Работа с базовой поверхностью происходит во время работы с картой высот, все значения которой нормализованы[6], то есть являются числами с плавающей точкой и не выходят за диапазон [0, 1].

3.7. Генерация гор

Создание гор – это второй этап генерации ландшафта. Пользователь может задать произвольное число гор. Каждая структура описывается настройками, задающими размеры и высоту горы, форму склонов, расположение на карте. Также существует возможность создания вулканов.

Для создания гор требуется отдельная карта высот с информацией, связанной исключительно с горами. Именно на эту карту накладываются все горы, после чего карта высот мира накладывается на карту высот гор в режиме «Максимум». Этот режим обрабатывает два двумерных массива и заполняет каждый элемент выходного массива максимальными значениями соответствующих элементов исходных массивов.

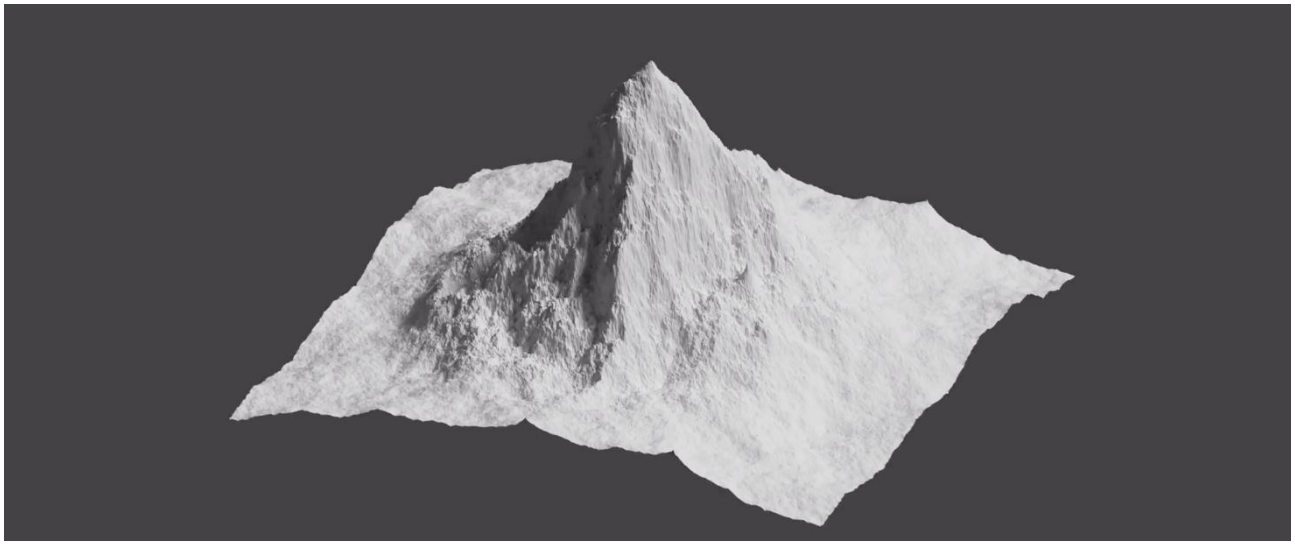


Рисунок 2. Демонстрация сгенерированной горы без текстур.

3.8. Генерация водоемов и рек

Водоемы и реки – это объект моделирования третьего шага генерации.

Создание водоемов сводится к нахождению уровня моря на карте, исходя из заданного соотношения суши к воде. Формулировка задачи сводится

к нахождению такого значения, при котором соотношение суши к воде, заданное пользователем, наиболее близко к соотношению количества элементов со значениями меньше параметра к количеству остальных элементов.

Так как все значения карты высот находятся в интервале [0, 1], то данную задачу можно решить с помощью алгоритма бинарного поиска. Сначала находится соотношение для параметра 0.5, после чего интервалом становится нижняя или верхняя половина предыдущего интервала (решение зависит от того, больше или меньше полученное соотношение чем требуемое). Таким образом, за достаточно небольшое количество итераций можно получить результат с высокой точностью.

После нахождения уровня моря, все области ниже него помечаются как водоемы.

Создание рек начинается зразу после разметки водоемов. Выбираются две точки — точка начала реки и точка конца. Эти точки должны соответствовать нескольким требованиям: начало реки должно находиться выше конца, а конец реки всегда находится в водоёме. После этого между точками строится прямая линия и серединный перпендикуляр к ней. На перпендикуляре находится точка с наименьшей высотой над уровнем моря, эта точка помещается в список точек реки, после чего данная операция повторяется для уже полученных отрезков до тех пор, пока все точки списка не будут находится на непрерывной кривой. Таким образом, получается извилистое русло, которое направляется из начала в конец под наибольшим углом, что моделирует воздействие гравитации на реки. После этого русло опускается на некоторую глубину, одновременно удаляя возможные подъемы на пути реки. Наглядную демонстрацию данного процесса можно рассмотреть на рис. 3.

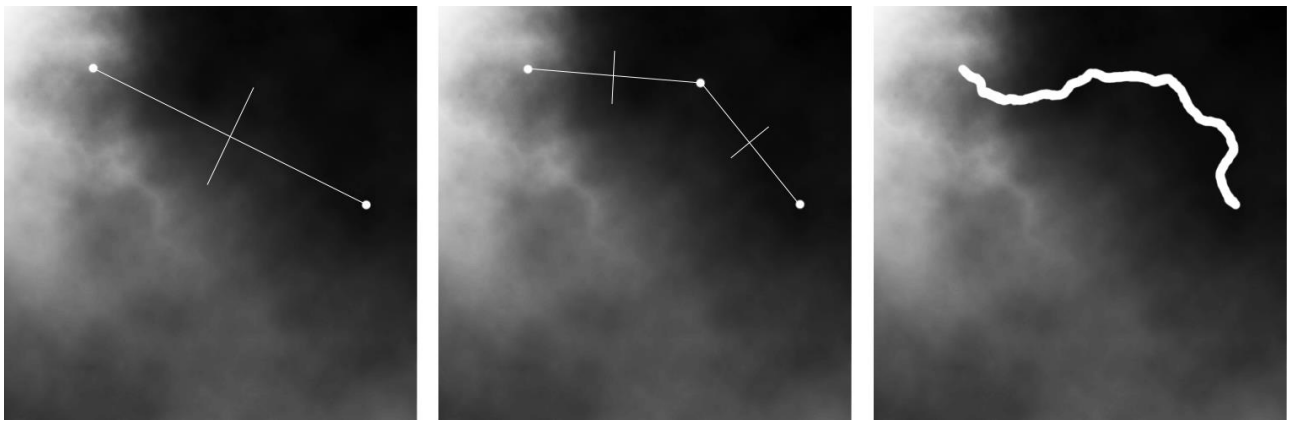


Рисунок 3. Генерация русла реки. Слева – первый шаг, по центру – второй шаг, справа – итоговый вариант русла.

3.9. Генерация текстур

Генерация ландшафта тесно связана связана с работой с текстурами. Однако простого сохранения текстур недостаточно для получения красивого результата. Если два соседних места должны быть заполнены разными текстурами, то на их границе необходимо произвести смешивание. Простое смешивание линейным способом не даст нужного эффекта: две текстуры будут плавно переходить друг на друга, полностью игнорируя особенности фактуры. Для обхода этого недостатка было решено использовать смешивание не только по месту нахождения пикселя, но и по его яркости[8]. Это поможет выбирать тот цвет, который находился бы на верхней текстуре. Это было сделано исходя из того, что на текстурах чаще всего яркие места обозначают более близкое отношение к источнику света, а значит находятся выше. Таким образом, для каждой текстуры создаётся карта прозрачности, а полученные текстуры накладываются в соответствии с этой картой. Пример сгенерированной текстуры представлен на рис. 4.

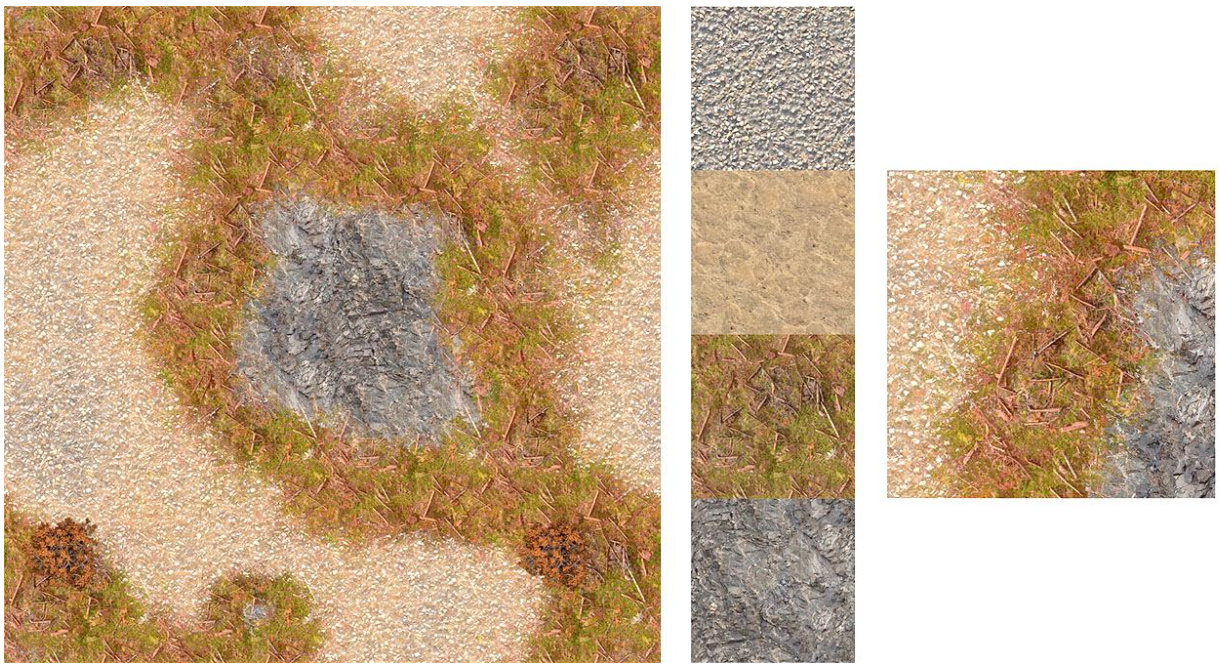


Рисунок 4. Участок текстуры, сгенерированной на основе 4 изображений. Стоит отметить, что на текстуре нет ярко выраженных повторяющихся элементов, являющихся негативным моментом при использовании бесшовных текстур.

Достаточно часто в играх используются карты. В данном проекте есть возможность по созданию миникарты. Она отображает текстуру ландшафта, однако позволяет дополнительно накладывать на неё тени и фильтры.

Качество текстуры во многом определяется ее разрешением, то есть количеством пикселей. При больших картах хранить одно большое изображение не представляется возможным. Поэтому текстура ландшафта генерируется частями – чанками, что позволяет подгружать только необходимые на данный момент времени изображения. Несмотря на разбитие на отдельные участки, соседние изображения не содержат швов и образуют цельную текстуру. К тому же, создание 3D моделей также производится по чанкам, что позволяет рассматривать чанк как отдельную самостоятельную единицу ландшафта.

Рисунок 5. Демонстрация плавного перехода текстуры на стыке трёх чанков.

Высокая реалистичность картинки может быть достигнута не только при помощи текстур и векторов нормалей. Существует несколько различных карт, накладываемых на модель, которые позволяют улучшить визуальное отображение объекта. Одним из таких изображений является карта нормалей[9]. Она позволяет хранить информацию о векторах нормалей не для вершин и плоскостей, а для каждого пикселя текстуры, тем самым значительно увеличивая детализацию освещения.

Карта нормалей представляет собой изображение в голубых тонах, так как синей компоненте цвета соответствует направление вектора по высоте, а плоскость направлена лицевой стороной вверх. На красную и зеленую компоненты цвета влияет направление вектора нормали в данной точке по остальным осям. Таким образом, зная относительную высоту текущей и соседних точек, можно определить эти компоненты цвета. Однако такие данные не предоставляются генератору. Это значит, что эту информацию необходимо каким-либо образом получить.

В реальном мире источники света, как правило, располагаются сверху – например, Солнце, фонари или звездное небо. Входными данными для генератора являются в том числе и текстуры, созданные с учетом такого освещения. Логично предположить, что в чем ярче пиксель, тем выше находится эта часть изображения. Данная оценка может быть не справедлива для ряда случаев, однако при отсутствии прочих данных данный тезис может быть оправдан.

Исходя из этого, после генерации текстуры для каждого пикселя вычисляется его яркость. Полученная карта высот и используется для создания карты нормалей. Пример карты нормалей представлен на рис. 6.

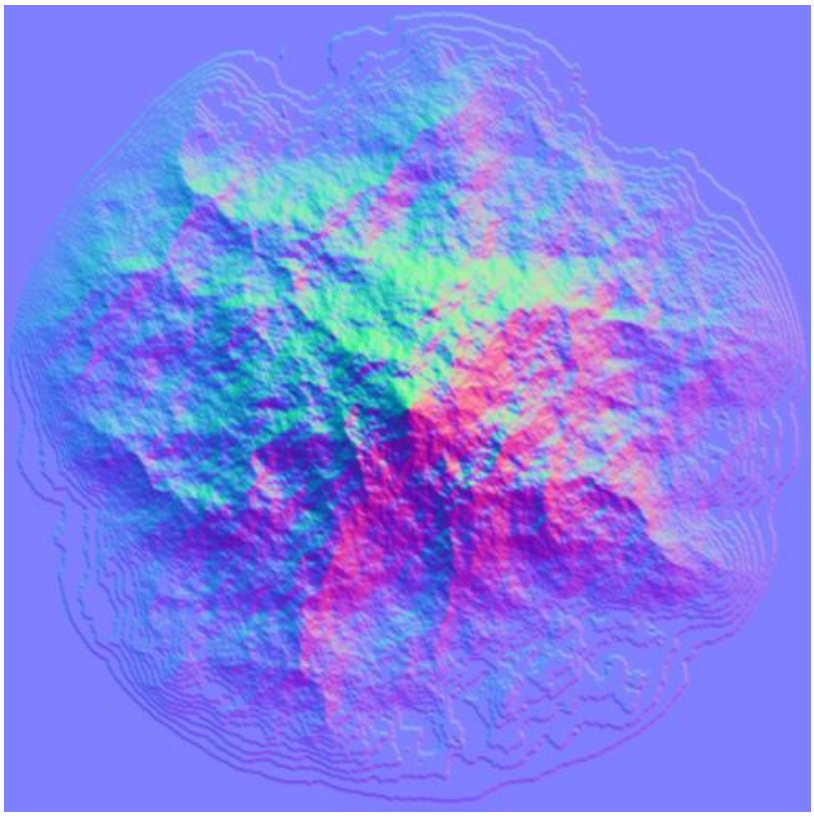


Рисунок 6. Карта нормалей для текстуры горы.

3.10. Генерация локаций

Следующим этапом генерации является создание локаций. Он необходим для последующей расстановки моделей, созданных человеком. Большая часть локаций размечается исходя из высоты, однако некоторые

локации требуют дополнительной обработки. Например, поляны и леса находятся на одном уровне, но должны быть четко разделены. Для этого был использованы алгоритмы diamond-square и бинарный поиск.

Изначально генерируется шум, после чего для него с помощью бинарного поиска находится соотношение лесов и полян, аналогично поиску уровня моря. Стоит отметить, что берутся не все значения, а лишь пригодные для размещения лесов или полян. После нахождения требуемого уровня, лесами помечаются все места, где значения шума выше найденного порога, а полянами становятся все остальные.

3.11. Расположение сторонних моделей

Без расстановки моделей, созданных человеком, игровой мир будет выглядеть пустым и неестественным. Расстановка сторонних моделей полностью случайным образом будет выглядеть искусственно. Следовательно, стоит размещать модели исходя из локаций, определенных на предыдущем этапе.

Для каждой локации составляется список возможных моделей. Каждый объект описывается широким спектром настроек, например максимальным и минимальным количеством, минимальным расстоянием до других объектов, необходимостью стоять на ровной поверхности и прочих.

Заполнение пустых мест не является сложным в реализации, однако следует придерживаться нескольких основных правил. Во-первых, стоит устанавливать различную вероятность появления разным моделям. Во-вторых, не рекомендуется устанавливать заполнение локаций полностью. Все эти замечания помогают создать наиболее реалистичный вид.

3.12. Экспорт моделей

Работа с генерацией ландшафта ведется с помощью карт высот. Однако игры используют 3D модели для хранении информации о ландшафте. Это подразумевает потребность в экспорте данных геометрии в каком-либо

распространенном формате. Для этих целей был выбран формат OBJ и сопутствующий формат MTL, предназначенный для хранения информации о материалах.

Карту можно сохранить двумя способами – целиком или отдельными частями (чанками). Первый способ подходит для маленьких ландшафтов, а второй предназначен для предотвращения загрузки большой модели в оперативную память. Совместно с этим выполняется сохранение файлов материалов для данных моделей.

Сложность данного этапа заключается в трудностях работы с файлами форматов OBJ и MTL, а также необходимостью учитывать многие параметры, например, создавались ли текстуры, необходимо ли создавать новый материал, есть ли информация о данных освещения и прочее.

3.13. Шаблоны проектирования

В процессе создания статической библиотеки классов, отвечающей за процедурную генерацию, были выявлена потребность в расширении данного инструмента. Так как ядро написано на языке C++ и были широко использованы классы, появилась потребность во внедрении шаблонов проектирования, также называемых паттернами[7]. Стоит отметить, что не всегда их стоит применять в первоначальном виде. В частных случаях некоторые модификации паттернов позволят упростить читабельность кода.

3.13.1. Посредник

Основной структурой, работа с которой ведется на протяжении всего процесса генерации, является структура World, хранящая карту высот, вспомогательные массивы и прочую полезную информацию. Так как генератор должен обеспечивать высокую кастомизацию, то требуется написание нескольких модулей, модифицирующих информацию о мире последовательно. Не имеет смысла связывать классы модулей непосредственно друг с другом, так как затруднит написание и поддержку

кода. С данной проблемой может справиться паттерн «Посредник» (или «Медиатор»).

Для этих целей был создан главный класс библиотеки ядра – Generator. Он инициализирует структуру World, являющуюся его мембером. Затем он последовательно вызывает функции модулей, ответственные за генерацию и удаляет их при завершении работы. Таким образом, структура проекта изменилась со сложно связанного списка на главный объект и множество независимых модулей с общим интерфейсом. В дальнейшем это позволило легко интегрировать дополнительные модули, например модуль расстановки сторонних моделей и модуль разметки локаций.

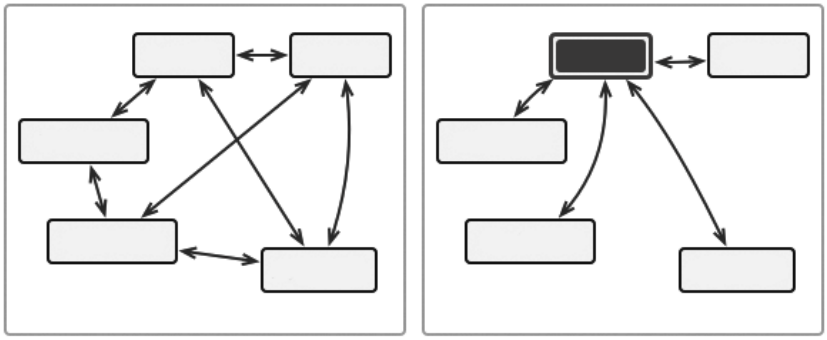


Рисунок 7. Исходная связанность классов внутри библиотеки ядра генератора (слева) и полученная в результате рефакторинга (справа)

3.13.2. Прототип

В процессе генерации ландшафта с помощью разработанных алгоритмом требуется хранение двумерных массивов. Например, цвета текстуры в каждом конкретном месте и карта высот мира, являются по сути двумерными массивами. Организация такого массива может быть выполнена двумя способами: с помощью одного линейного массива, хранящего двумерные данные построчно или с использованием массива указателей на отдельные строки.

Выбор подходящего варианта представлен в таблице 3.1.

Таблица 3.1 – сравнение методов хранения двумерного массива

Критерий	Линейный массив	Массив указателей на массивы
Обращение к элементу по индексам x, y массива a на языке C	a[x+y*width] Данная конструкция сложна для чтения, а также высока вероятность ошибки по невнимательности.	a[x][y] Лаконичная конструкция.
Сложность аллокации памяти для массивов больших размеров	Да Большие линейные участки в памяти ищутся достаточно долго, к тому же есть вероятность не найти такого участка вообще.	Нет Несмотря на то, что общий объем памяти немного больше, чем в линейном массиве, так как требуется хранение массива указателей на строки, каждая строка массива представлена небольшим отдельным участком памяти. Это значительно ускоряет процесс поиска нужного объема.
Необходимость в дополнительной структуре данных	Нет	Да, так как требуются методы для выделения и освобождения памяти.

Для хранения двумерных массивов был спроектирован шаблонный класс Matrix<T>, хранящий данные о количестве строк и столбцов в таблице и самих данных.

Такая организация структуры данных стала причиной проблемы копирования объекта, а написание кода копирования каждый раз ухудшило бы читабельность кода. К тому же, в проекте существуют и другие структуры с возможностью копирования. Это стало причиной использования паттерна «Прототип». Это озачает, что копирование объекта в новый было предоставлено самому классу, например Matrix<T>. Это позволило не нагружать код однотипными операциями и упростить устранение ошибок.

3.13.3. Фасад

Код класса Generator представлен несколькими методами, которые требуют строгой очередности. Если пользователю придётся самому инициализировать эти объекты и следить за правильным порядком зависимостей, то логика библиотеки генератора тесно переплетется с логикой пользователя. Такой код сложно поддерживать.

Для решения данной проблемы было решено использовать паттерн «Фасад». Вся логика инициализации, проверок, генерации и деинициализации была помещена в один метод. Однако отличием от классического шаблона стало то, что эти методы были помещены в другую библиотеку с API, интерфейс для которой был написан на языке C, а не C++. Таким образом, было решено 2 проблемы: упрощение взаимодействия с пользователем и упрощение интеграции в сторонние проекты, так как приложения на языке C могут поддерживаться как со стороны приложений, написанных на C, так и со стороны приложений, написанных на C++. Недостатком данного подхода стало увеличение количества библиотек. Однако это позволит в дальнейшем создать другие библиотеки API для интеграции в программные продукты, написанные в Unreal Engine и прочие продукты.

3.14. Создание API и документации

Для набора средств разработки важен не только широкий функционал, но и удобный доступ к нему, полностью описанный в документации. Поэтому был разработан API к библиотеке генератора, написанный на языке C.

Изначально были созданы файлы заголовков со специфическими типами данных, а также функции работы с генератором. Эти файлы получились небольшими, но они предоставляют весь необходимый функционал.

Еще один файл заголовков отвечает за работу с настройками генератора. Заполнение его в ручном режиме не обоснованно – количество параметров значительно больше 100, и добавление новых параметров повлечет необходимость в обновлении этих файлов в ручном режиме. Метод X-Macro, использующийся во внутреннем представлении структуры настроек, позволил создать приложение, создающее этот файл автоматически при наличии изменений. Это позволило в значительной мере упростить разработку API, а также избежать ошибок, связанных с человеческим фактором.

3.15. Тестирование

Тестирование ПО является важным аспектом разработки. Сложные системы должны быть избавлены от грубых ошибок. Тестирование в ручном режиме является эффективным, но долгим процессом. Поэтому было решено использовать автоматическое тестирование. Для этого была выбрана утилита CTest, находящаяся в составе используемой системы CMake.

Код тестового приложения занимает более 20% всего исходного кода проекта.

В процессе написания тестов рассматривалось влияние различных параметров генерации и их влияние друг на друга. Проверки функций библиотеки заключались в трёх основных условиях:

- при параметрах, заданных верно, генерация должна проходить успешно;
- при параметрах, заданных неверно, генерация должна автоматически останавливаться в штатном режиме;
- для случаев с успешной генерацией требуется проверка выходных данных;

Основной проблемой при разработке тестового приложения стало большое количество настроек, которые влияют на выходные данные таким образом, что определение исходных настроек является сложной задачей. Например, тип склона гор может иметь несколько различных значений. Для таких случаев было решено использовать хеширование, чтобы удостовериться, что настройка влияет на результат. Качественные характеристики результата может быть оценено исключительно субъективно.

Также сложным моментом в написании автоматических тестов было наличие параметров, связанных с другими. Например, размер 1 чанка должен быть степенью числа 2, больше 2, а также меньше или размеру карты. Это потребовало тщательного анализа всех параметров, выявления зависимостей и рассмотрения всех возможных случаев.

Для тестирования некоторых параметров потребовались нестандартные решения. Например, тестирование вывода логов на экран во время выполнения – оно потребовало перенаправления стандартного вывода в файл с последующей проверкой размера файла.

3.16. Работа с изображениями

Сохранение и загрузка текстур требуют возможности работы с файлами изображений. Таким образом, требовалось определить формат, в котором будут храниться входные и выходные изображения.

Было рассмотрено несколько вариантов:

- RAW-форматы. Эти форматы хранят только данные изображения, и не содержат каких-либо дополнительных данных, например, ширину или высоту, что вносит трудности при работе с данными изображениями. Также данные в таком формате занимают много места на жёстком диске. Стоит отметить, что при сохранении изображений в данный формат нет никаких потерь качества.
- BMP. Данный формат схож с RAW, однако имеет несколько заголовков перед пиксельными данными, в которых хранится наиболее используемая информация. Размер таких файлов зачастую даже больше, чем RAW, но работа с ним намного проще. Этот формат также не вносит потери в исходные данные.
- JPEG. Этот формат значительно выигрывает в размере фалов, однако это достигается путем исключения малозначительных данных, что вносит потери в исходные данные. Кроме того, реализация поддержки данного формата занимает достаточно много времени, которое можно потратить на разработку основного функционала системы.

Таким образом, было решено использовать широко распространённый формат BMP. Этот формат поддерживается многим ПО, а также не вносит потерь. Для устранения недостатка в размере файлов можно конвертировать данный формат в любой удобный пользователю SDK формат, но уже собственными средствами пользователя.

Структура BMP файла является достаточно простой[10]. Она состоит из следующих частей:

1. Заголовок файла
2. Блок информации изображения
3. Палитра цветов (присутствует не всегда)
4. Пиксельные данные

Заголовок файла (BITMAPFILEHEADER) содержит общую информацию о самом файле. Это тип файла, его размер и расположение начала пиксельных данных. Следующий за ним блок информации изображения может быть нескольких типов (BITMAPCOREHEADER, BITMAPINFOHEADER, BITMAPV4HEADER и BITMAPV5HEADER), однако все они содержат ключевые параметры изображения: ширину, высоту, глубину цвета и пр.. Палитра цветов находится после блока BITMAPINFO и присутствует только в изображениях с глубиной цвета меньше или равной 8 бит/пиксель. В таком случае пиксельные данные, расположенные в конце файла, содержат не значения цветов, а номер цвета в таблице, значительно уменьшая размер файла.

Для оптимизации работы с файлом BMP было решено загружать файл в оперативную память целиком, чтобы избежать множественных чтений с диска, которые являются достаточно долгой операцией.

3.17. Работа с 3D моделями

Алгоритм генерации ландшафта реализован с помощью двумерных массивов, заполненных градиентным шумом. Поэтому оптимальным решением будет экспорт полигональной сетки, которая при виде сверху будет представлена множеством квадратов, а интенсивность шума в конкретной точке будет влиять на вертикальную составляющую каждой конкретной точки. Это позволит легко перенести данные с карты высот в 3D модель[11], а также поможет избежать трудностей с координатами текстур. Благодаря формированию полигональной сетки в виде квадратных клеток, текстурные координаты будут равномерно распределены по всей плоскости фигуры. Стоит отметить, что каждый квадрат разделен на 2 треугольника, чтобы избежать возможных артефактов.

Важным этапом в создании 3D моделей являются нормали. Эти вектора предназначены для хранения информации об отражении света от поверхности

объекта. Нормали бывают применены к двумя способами: к полигонам (плоскостям) и к вершинам. В первом способе границы полигонов четко видны, что нехарактерно для реальных ландшафтов. При привязке нормалей к вершинам освещение плавно переходит между полигонами, что создает более плавную картинку. Таким образом, требуется реализация второго способа.

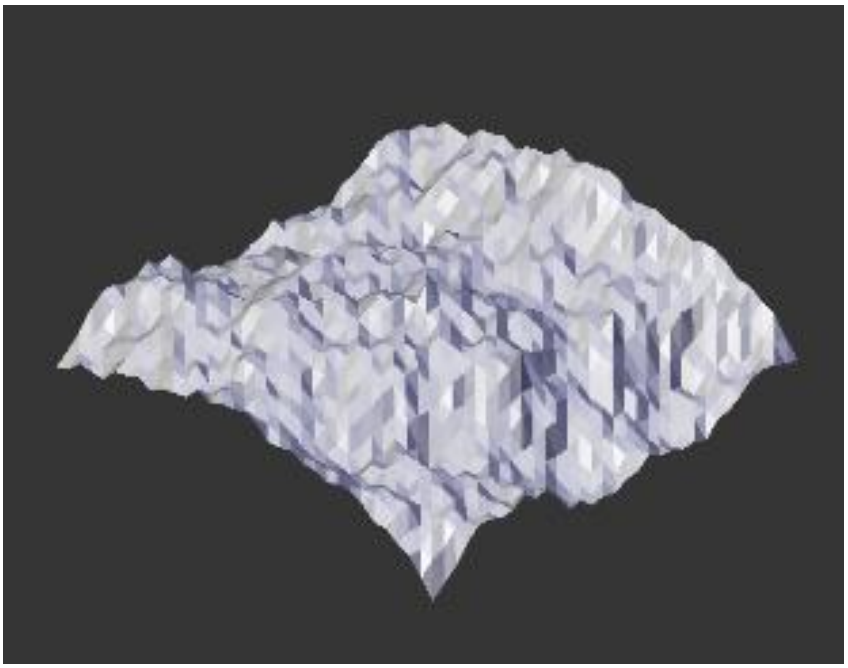


Рисунок 8. 3D модель с нормалями, привязанными к полигонам

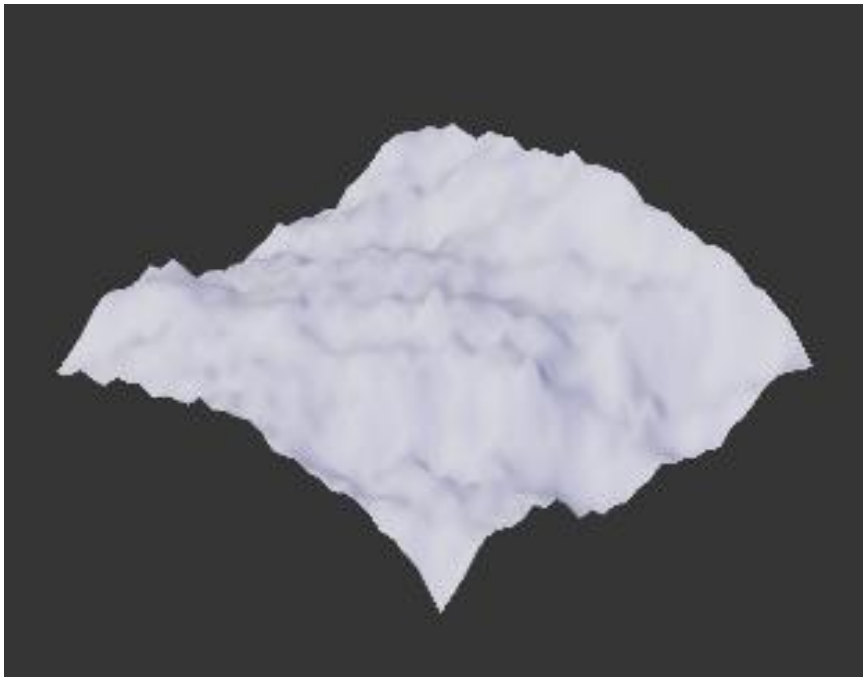


Рисунок 9. 3D модель с нормалями, привязанными к вершинам

Чтобы привязать нормали к вершинам, стоит рассчитать их для каждой точки. В этом заключается основная проблема, так как нормаль можно рассчитать только по трём точкам. Для решения данной проблемы было решено рассчитывать нормали для всех прилегающих к точке плоскостей, а затем находить их среднее значение.

Информация о текстурных координатах также крайне важна при работе с 3D моделями. Эти данные привязаны к полигонам. Благодаря формированию полигональной сетки в виде квадратных клеток, текстурные координаты будут равномерно распределены по всей плоскости фигуры.

После определения требований встал вопрос о выборе формата файлов. В итоге были выбраны два формата: OBJ для моделей и MTL для материалов. Эта связка была выбрана в связи с тем, что данные форматы общеприняты, спецификации к ним находятся в общем доступе, а реализация поддержки не является трудной. Недостатком данных форматов является их текстовый вид, а следовательно и большой объем файлов. Также, как и при работе с изображениями, при необходимости данный формат может быть легко преобразован пользователем в необходимый.

После экспорта моделей их можно использовать в любом месте. Стоит отметить, что не все программные средства способны правильно обрабатывать все данные. Например, часть продуктов просмотра 3D объектов не способна считывать нормали, а высчитывает их сама. Для таких случаев стоит предусмотрена опция отключения карт нормалей, чтобы не занимать лишнее дисковое пространство. Также такие продукты накладывают ограничения на порядок обхода вершин в полигонах – от этого зависит, какое из двух направлений примет вектор нормали.

3.18. Работа с INI файлами

Для ряда случаев настройка генератора через API не является удобной. Например, при переносе настроек из одного проекта в другой возникает

необходимость копирования кода. Также иногда возникает необходимость просмотреть настройки, с которыми был запущен генератор. Для таких случаев была добавлена поддержка формата INI. Настройки могут быть считаны из файла и применены к генератору. Также настройки могут быть сохранены в файл для дальнейшего использования.

Существует несколько распространённых форматов файлов для хранения настроек: INI, XML, JSON. Формат INI был выбран исходя из следующих соображений:

XML данные легко просматриваются в виде таблицы, однако занимают большее количество места, а их парсинг является нетривиальной задачей. Также изменение данных в формате XML не является удобным. JSON является удобным форматом для представления данных для человека, однако он изначально придумывался для языка JavaScript, и реализация его поддержки в языках со строгой типизацией является непростой задачей. INI же является одним из простейших форматов, который легко просматривается человеком.

Из недостатков стоит отметить, что четкой спецификации у данного формата нет. Это значит, что разные программы могут обрабатывать данные по-разному.

Структура файла представляет собой несколько секций, начинающихся с названия в квадратных скобках и списком параметров для этой секции. Также предусмотрена возможность комментирования части информации.

Для работы с файлами INI были разработаны два класса: IniReader и IniWriter для записи и чтения соответственно. Они содержат список секций, который может пополняться. Таким образом, работа с файлами конфигурации вынесена в отдельные классы и не имеет сильной связанности с остальным кодом.

4. Оптимизация в проектировании ландшафтов

4.1. Оптимизации путем многопоточной обработки данных.

Наиболее сложным этапом генерации ландшафта является создание текстур. Даже при средних настройках количество текстур достаточно велико, и над каждым пикселем требуется воспроизвести достаточно большое количество операций. Таким образом, для этого этапа требовалось создание многопоточного режима.

Так как текстуры создаются чанками, то вся карта была разделена на полосы равной (или почти равной) ширины. Количество полос равно количеству потоков, заданному пользователем. В каждой полосе должно умещаться целое число чанков. Далее каждый поток последовательно обрабатывает чанки из своей полосы, после чего при необходимости рисует их на миникарте. После работы потоков сохраняется миникарта (если таковая имеется) и этап завершается.

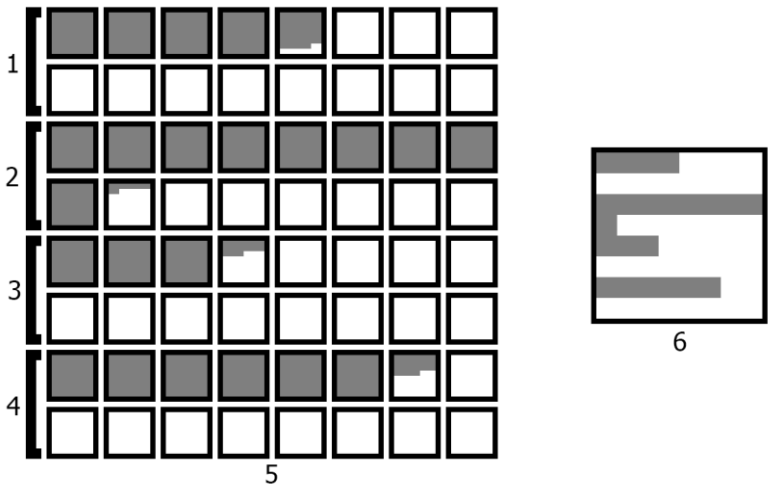


Рисунок 10. Наглядное представление многопоточной обработки текстур. Карта (5) разбита на 64 чанка, все текстуры обрабатываются 4 потоками, каждый из которых обрабатывает исключительно свой блок (1), (2), (3) или (4). На миникарте (6) данные появляются только при полном завершении чанка.

Стоит отметить, что увеличение числа потоков повлечет за собой рост количества требуемой оперативной памяти, а задание числа потоков более числа логических ядер процессора не даст прироста.

4.2. Оптимизации путем математических преобразований.

Вычисление нормалей во время создания 3D модели требует многочисленных вычислений, так как для каждой вершины требуется расчет 4 векторов нормалей с последующим их сложением и нормализацией (рис. 11).

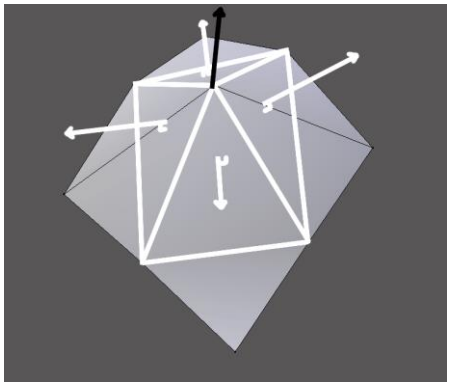


Рисунок 11. Обоснование расчета вектора нормали к вершине с помощью суммы соседних нормалей.

Однако при рассмотрении структуры 3D модели становится ясно, что данные вычисления можно легко преобразовать в более простой вид. Это связано с тем, что расстояния между вершинами по координатам X и Y всегда одинаковы и равны ширине ребра.

Рассмотрим два свойства векторных произведений:

$[\vec{a}, \vec{b}] = -[\vec{b}, \vec{a}]$ – антикоммуникативность

$[\vec{a} + \vec{b}, \vec{c}] = [\vec{a}, \vec{c}] + [\vec{b}, \vec{c}]$ - дистрибутивность по сложению

Для расчета вектора нормали в заданной точке следует обозначить 4 вектора, направленных из неё по сторонам полигонов и идущих в соседние вершины (рис. 12).

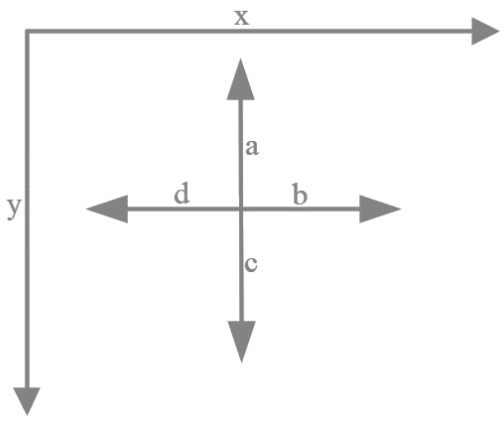


Рисунок 12. Обозначения для расчета вектора нормали к вершине (вид сверху).

Преобразование формулы будет выглядеть следующим образом:

$$[\vec{b},\vec{a}] + [\vec{c},\vec{b}] + [\vec{d},\vec{c}] + [\vec{a},\vec{d}] =$$

$$= [\vec{c},\vec{b}] - [\vec{a},\vec{b}] + [\vec{a},\vec{d}] - [\vec{d},\vec{c}] =$$

$$= [\vec{c}-\vec{a},\vec{b}] + [\vec{a}-\vec{c},\vec{d}] =$$

$$= [\vec{a}-\vec{c},\vec{d}] - [\vec{a}-\vec{c},\vec{b}] =$$

$$= [\vec{b},\vec{a}-\vec{c}] - [\vec{d},\vec{a}-\vec{c}] =$$

$$= [\vec{b}-\vec{d},\vec{a}-\vec{c}].$$

Таким образом, путем использования свойств векторных произведений, число векторных произведений было сокращено с четырех до одного, что значительно сказалось на производительности.

5. Вспомогательные инструменты разработки

5.1. Создание сайта проекта

Для любого проекта важно наличие документации и примеров использования. Данные моменты было решено оформить в виде HTML страниц. Для объединения этой информации в одно целое был создан сайт проекта. На нем можно найти информацию о спецификациях форматов файлов, примерах использования набора средств разработки, а также полная документация по всем доступным в API методам.

Так как проект значительно изменялся в стадии разработки, то список параметров часто дополнялся или менялся со временем. Редактирование страницы со списком параметров в ручном режиме является плохим решением, так как на каждую правку в коде приходится делать соответствующую правку в HTML коде. Очевидно, что создание такой страницы следует автоматизировать. Код проекта написан на C++, а настройки реализованы с помощью приёма, базирующемся на X-Macro, поэтому данный подход можно использовать и внутри генератора страницы настроек.

Все страницы сайта были разделены на 3 логические категории: контент главной страницы, контент, связанный с кодом, и основной контент. Для каждого типа был разработан собственный стиль.

Для быстрого нахождения в сайт был внедрен «живой поиск» по параметрам генерации, работающий даже в оффлайн режиме.

Контент главной страницы не имеет отступов по краям и занимает всю ширину страницы полностью. Основной текст на главной странице заключен в секции: темные и светлые. Они состоят из 2 колонок: текста и картинки к нему. В верхней части главной страницы находится видео. Однако видео может некорректно воспроизводиться в браузерах Internet Explorer и MS Edge, а также на мобильных устройствах. По этой причине в страницу был внедрен

скрипт, определяющий неподдерживающие видео устройства и заменяющий на них видео на статичную картинку.

Основной и связанный с кодом контент оформлен в виде страницы со светлым фоном и тёмным текстом, так как такое сочетание наиболее удобно для представления формализованной информации. Ссылки могут быть оформлены в 2 вариантах. В стандартном варианте весь текст отображается в верхнем регистре, сама ссылка является блочным элементом и занимает всю доступную ширину. Во втором варианте ссылка находится внутри текста и занимает минимально возможное пространство. Первый вариант ссылок представлен для навигации в крупных разделов, а второй — для ссылок на небольшие страницы или сторонние ресурсы.

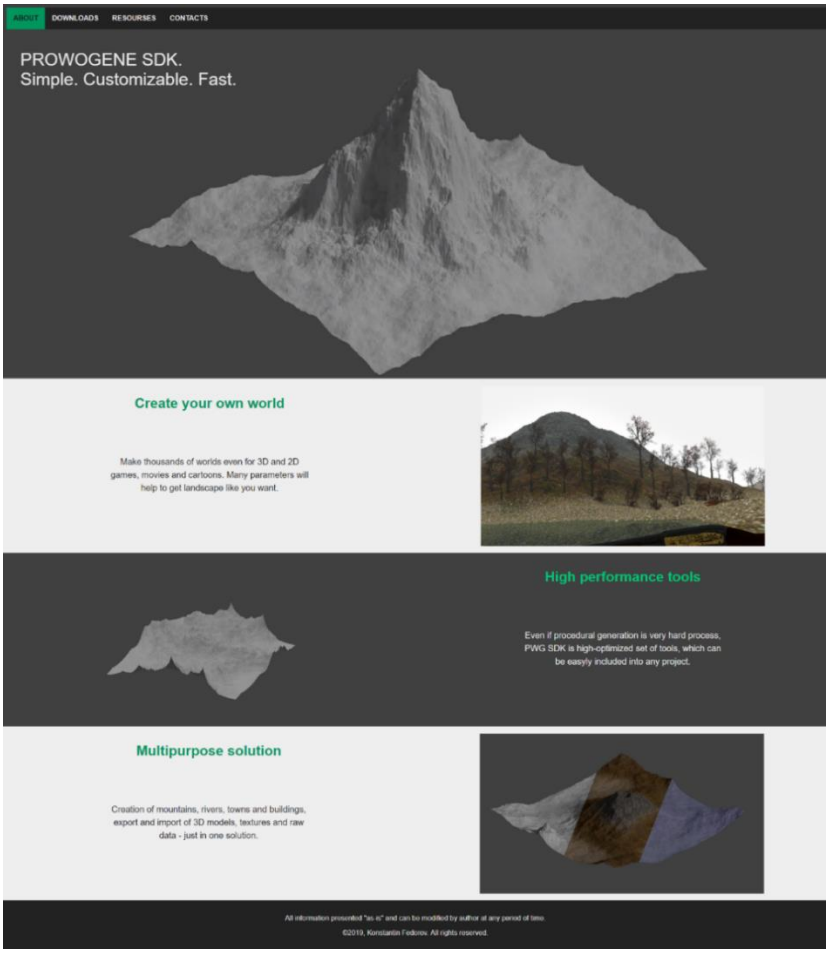


Рисунок 13. Главная страница сайта

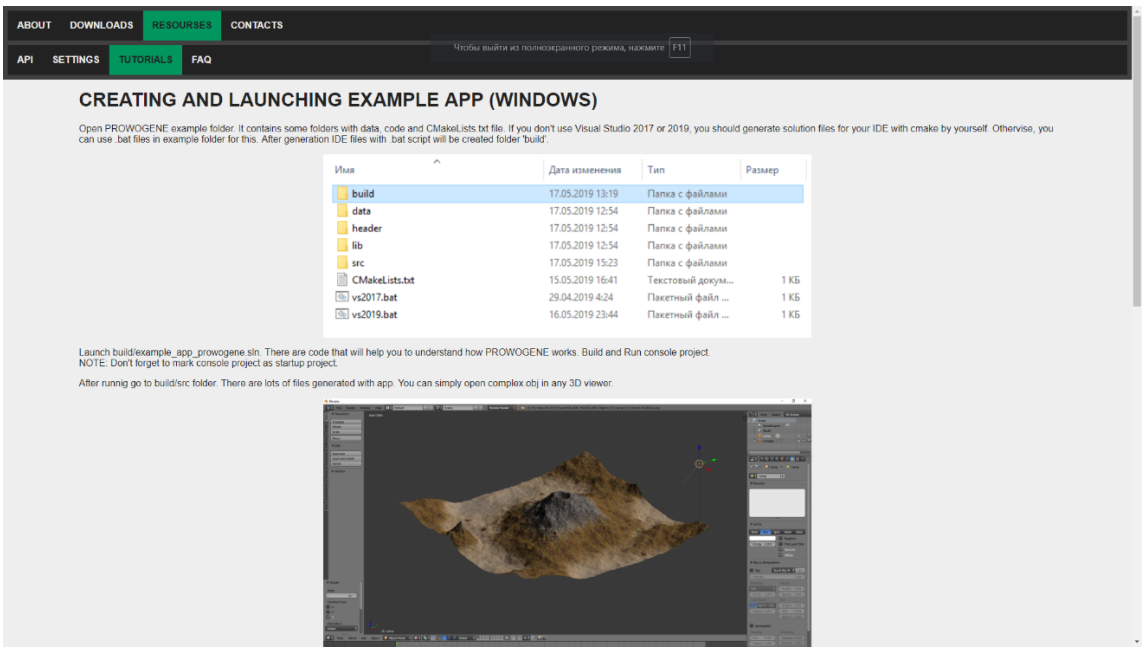


Рисунок 14. Пример страницы с обучающими материалами

Дизайн данного сайта также поддерживает мобильную версию. Все горизонтально расположенные в полной версии блоки позиционируются вертикально. Меню и подменю разделяются небольшой полосой. Также в мобильной версии видео заменено на картинки. Шрифты и кнопки увеличены для удобства пользователей мобильных устройств.

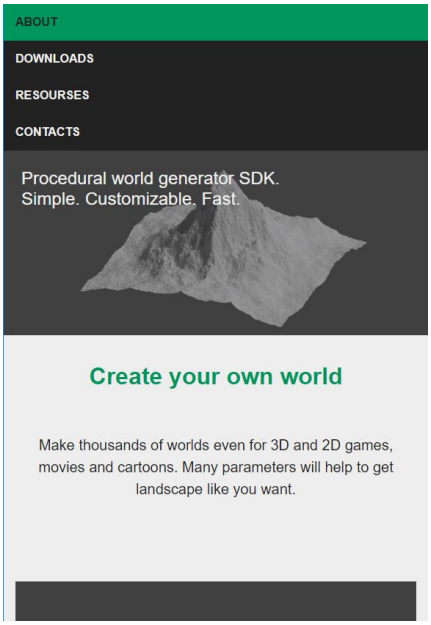


Рисунок 15. Демонстрация мобильной версии сайта

5.2. Создание скриптов для Blender 3D

Для быстрого просмотра результатов генерации необходима интеграция с каким-либо инструментов 3D визуализации. Blender 3D является свободно распространяемой программой для работы с 3D моделями[12]. К тому же он имеет удобный для разработчиков API[13], что послужило основным преимуществом при выборе приложения.

Для автоматической загрузки моделей в рабочую область программы Blender 3D было написано 2 скрипта – для ландшафта, созданного в виде цельной модели и для мира, сгенерированнного чанками. Скрипт автоматически загружает модели ландшафта, текстуры и материалы, а также сторонние модели и располагает их на нужных местах.

В качестве входных данных скриптам требуются лишь полные пути до некоторых папок и файлов.

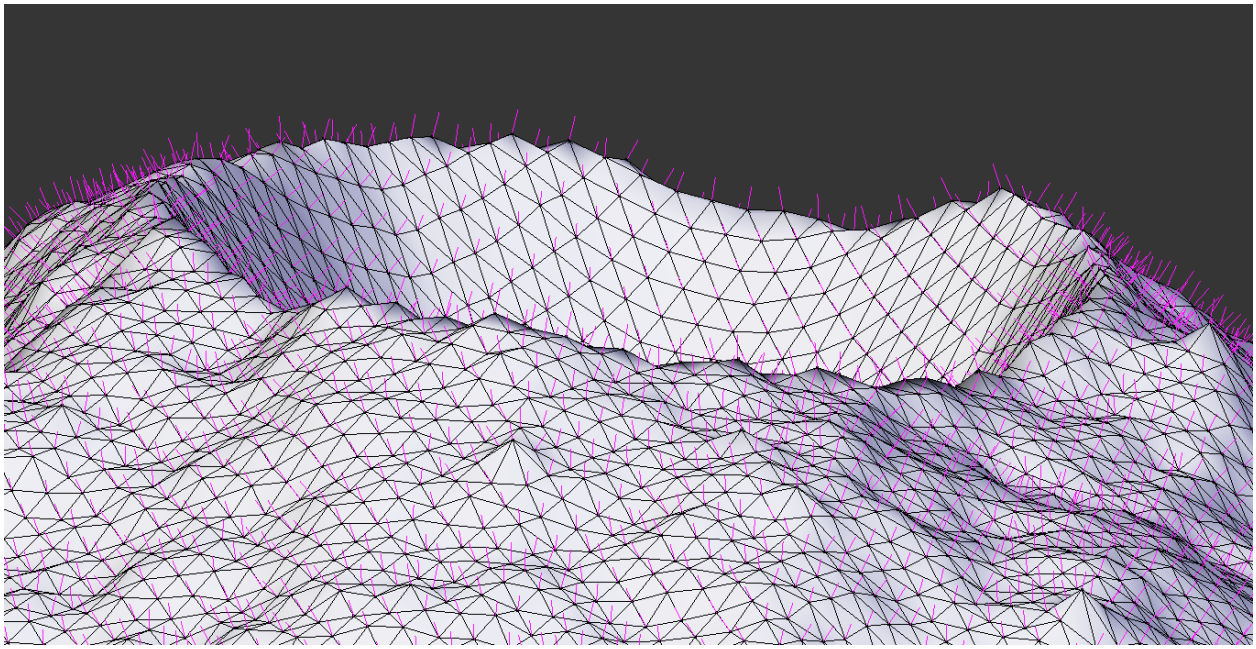


Рисунок 16. Импортированная в Blender 3D модель с отображенными векторами нормалей.

6. Социальная ответственность

Введение

Процесс профессиональной деятельности разработчика программного обеспечения связан с воздействием широкого спектра производственных факторов. Необходимо предотвратить возможные негативные последствия для здоровья работника, приняв ряд мер по обеспечению безопасности трудовой деятельности.

В данном разделе проведен комплексный анализ факторов труда, способных повлечь негативные последствия для жизни и здоровья работника. Обозначен список мер организационного, правового, технического и режимного характера, для комплексной защиты разработчика.

Выпускная квалификационная работа по созданию набора средств разработки для процедурной генерации ландшафтов выполнялась в процессе прохождения преддипломной практики. Рабочим местом послужил офис со следующими характеристиками:

- ширина помещения – 8,0 м, длина – 8,0 м, высота – 3,0 м;
- площадь рабочего помещения – 64,0 м^2,
- объем помещения – 192,0 м^3;
- в помещении установлен кондиционер, а также вытяжные вентиляционные отверстия, дверь и окно.

Рабочее помещение рассчитано на 9 рабочих мест, из которых фактически занято лишь 7. В среднем на одного сотрудника приходится около 9,14 м^2 площади и примерно 27,43 м^3 объема помещения. Данное рабочее помещение соответствует санитарным нормам, в которых регламентирована минимальные площадь и объем на одного работника: на каждого сотрудника должно приходиться минимум 6 м^2 площади и 24 м^3 объема помещения.

6.1. Правовые и организационные вопросы обеспечения безопасности

При организации рабочего места с ПК требуется учитывать требования безопасности[14]. Они подробно описаны в ГОСТ 12.2.032-78 «ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования» и СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы».

Эти требования направлены на обеспечение удобной и безопасной обстановки на рабочем месте. Например, сотрудники, работающие за ПК, подвергаются серьезным зрительным нагрузкам, что является причиной наличия требований к расположению источников света, а также рабочего места и мониторов.

Согласно требованиям, при организации работы с ПК должны соблюдаться условия, представленные в таблице 6.1.

Таблица 6.1 – Требования к рабочему месту для разработчиков ПО.

Параметр	Условие соответствия норме	Характер требования (обязательный или рекомендательный)	Соответствие требованиям рассматриваемого рабочего места
Площадь помещения	Не менее 6м² на одного сотрудника	Обяз.	Полностью соответствует
Объем помещения	Не менее 24м^3 на одного сотрудника	Обяз.	Полностью соответствует
Конструкция рабочей мебели	Предусмотрена возможность регулировки по росту и/или весу	Обяз.	Полностью соответствует
	Наличие мягкой обивки и спинки	Обяз.	Полностью соответствует

Естественные источники света	Расположение сбоку, лучше слева (для левшей – справа)	Реком.	Соответствует не полностью. Источники естественного света расположены спереди, однако искусственные находятся слева от рабочего места
	Оборудование устройствами регулировки (шторы/жалюзи/…)	Обяз.	Полностью соответствует
Расположение рабочего места	Ориентированность спиной к стене, расположенность не в углу.	Реком.	Соответствует не полностью: рабочее место расположено спиной к стене, но в углу помещения.
Расстояние от ПК до стен	Более 1м от стены	Обяз.	Полностью соответствует
Расположение комплектующих ПК	Перед работником	Обяз.	Полностью соответствует

Высота рабочего стола	680-800 мм от уровня пола	Обяз.	Полностью соответствует
Расположение монитора.	900-1280 мм над уровнем пола, 600-700 мм от работника, на 20 градусов ниже уровня глаз	Обяз.	Полностью соответствует
	Направление просмотра – 90 градусов (максимальное отклонение до 75)	Обяз.	Полностью соответствует

В соответствии с Трудовым кодексом РФ 197-ФЗ предусмотрены точные временные рамки рабочей смены, а также перерывов. Длительность рабочей смены в дневной период времени (с 6 часов утра до 10 вечера) должна составлять не более 8 часов. Также во время рабочего дня должны быть предусмотрены небольшие перерывы и обеденный перерыв. Обеденный перерыв не должен быть менее 40 минут, его время может быть скользящим в течение рабочей смены. Остальные же перерывы должны быть строго регламентированы в соответствии с одним из вариантов:

- не менее 20 минут после 1-2 часов работы;
- не менее 30 минут после 2 часов работы;

При выполнении выпускной квалификационной работы не было зафиксировано нарушений правовых или организационных норм. Все обязательные требования были выполнены в полном объеме. Требования с

рекомендательным характером были выполнены с небольшими отклонениями, не сказавшимися на здоровье выполняющего работу.

6.2. Производственная безопасность

При анализе рабочего процесса было выявлено, что химические и биологические факторы не оказывают влияния на состояние здоровья разработчика программного обеспечения. Список наиболее опасных производственных факторов для программиста представлен в таблице 6.2.

Таблица 6.2. – Возможные опасные и вредные факторы

Факторы (ГОСТ 12.0.003-2015)	Этапы работ			Нормативные документы
	Разработка	Изготовление	Эксплуатация	
1.Недостаточная освещенность рабочей зоны	+		+	СП 52.13330.2016 «Естественное и искусственное освещение».
2. Повышенный уровень электромагнитных излучений	+	+	+	СанПиН 2.2.4.1191-03 «Электромагнитные поля в производственных условиях»
3.Отклонение показателей микроклимата	+	+	+	СанПиН 2.2.4.548-96 «Гигиенические требования к микроклимату производственных помещений»
4.Превышение уровня шума	+	+	+	ГОСТ 12.1.003-83 Система стандартов безопасности труда (ССБТ). Шум.
5.Монотонность труда	+		+	
6. Опасность поражения электрическим током	+	+	+	ГОСТ Р 12.1.019-2009 ССБТ. Электробезопасность.

Приведенный список требует более тщательного рассмотрения, а рабочее место на соответствие установленным санитарным нормам.

6.3. Анализ опасных и вредных производственных факторов и обоснование мероприятий по снижению их воздействия

Недостаточная освещенность рабочей зоны

Так как работа программиста заключается в длительном контакте с монитором компьютера, а также проходит исключительно в помещении, то проблема вреда зрению особенно актуальна. Одним из основных негативных факторов, влияющих на здоровье сотрудников, является плохая освещенность рабочего места. При недостаточном количестве света зрение работников подвергается риску, а производительность сотрудника падает. Неправильная организация света может быть следствием наличия плохо освещенных зон, слишком ярких источников света, а также мерцания освещения. Допускается установка светильников местного освещения для подсветки документов. Местное освещение не должно создавать бликов на поверхности экрана и увеличивать освещенность экрана более 300 лк.

В рабочих помещениях, рассчитанных на работу за ПК требуется наличие естественного и искусственного освещения. Нормативные показатели естественного, искусственного и совмещенного освещения представлены в таблице 6.3. Данные этой таблицы взяты из СП 52.13330.2016 «Естественное и искусственное освещение»[15].

Таблица 6.3. Нормируемые показатели естественного, искусственного и совмещенного освещения

	Рабочая поверхность и плоскость нормирования КЕО и освещенности и высота плоскости над полом, м	Естественное освещение		Совмещенное освещение		Искусственное освещение				
		КЕО ен, %		КЕО ен, %		освещенность, лк			показатель дискомфота М, не более	коэффициент пульсации освещен-ности, Кп, % не более
		при верхнем или комбини-рованном освещении	при боковом освещении	при верхнем или комбини-рованном освещении	при боковом освещени и	при комбини-рованном освещении		при общем освещении		
						всего	от общего			
Помещения для работы с дисплеями	Г-0,8 Экран монитора: В-1,2	3,5 -	1,2 -	2,1 -	0,7 -	500 -	300 -	400 200	15 -	10 -

На рассматриваемом рабочем месте задействованы как естественные, так и искусственные источники света, таким образом, освещение имеет смешанный характер.

Естественное освещение представлено оконным проемом 2х8 м в наружной стене. Искусственный свет в помещении представлен 12 светильниками типа ЛПО 36. В каждом светильнике установлено 4 люминесцентные лампы типа ЛБ-40. Для достижения наилучшего эффекта, светильники должны быть расположены линиями параллельно линии взгляда работника. В рабочем помещении светильники установлены двумя способами, но ближайший к каждому рабочему месту светильник всегда установлен согласно рекомендации. Защитный угол светильников должен составляет не менее 40 градусов, как и требуется.

Важным условием при организации рабочего места является равномерная освещенность всего поля зрения. Это означает, что и уровень освещенности помещения, и яркость экрана монитора должны быть соотносимы. Яркий свет на периферии приводит к утомляемости. То же самое наблюдается и при темном окружении и высокой яркости дисплея.

Приведем расчет освещенности рабочего пространства.

Размеры помещения 8х8х3 м, световой поток используемых ламп равен 900 лк. Норма освещенности рабочего места программиста находится в диапазоне от 300 до 500 лк. Коэффициент запаса следует считать равным 1,2, так как запыленность помещения значительно меньше 1 мг/м^3.

Требуется определить коэффициент отражения для поверхностей пола, стен и потолка. Их можно определить, исходя из таблицы отношений цвета к коэффициенту отражения света.

Таблица 6.4. Отношение цвета к коэффициенту отражения света

Цвет	Коэффициент отражения света
Белый	65 – 80
Кремовый	55 – 70
Коричневый	20 – 50

Потолок в помещении белого цвета, стены – светло-желтого, а пол светло-коричневого, а значит, индекс отражения для потолка равен 80, для стен – 70, а для пола – 40.

Индекс помещения рассчитывается по формуле:

$$И_П = \frac{S}{((h_1 - h_2) \times (a + b)}$$

где $И_П$ – индекс помещения; S – площадь; h_1 – высота потолков; h_2 – высота рабочего стола; a – длина помещения; b – ширина помещения.

Для представленного рабочего места он равен:

$$И_П = \frac{64}{((3 - 0{,}8) \times (8 + 8)} = 1{,}72$$

По полученному индексу помещения можно определить, что коэффициент использования помещения U равен 80.

Освещенности рассчитывается по следующей формуле:

$$E = \frac{К_{св} \times К_л \times СП_л \times U}{S \times k_з \times 100}$$

где $К_{СВ}$ – количество светильников; $К_Л$ – количество лампочек в светильнике; $СП_Л$ – световой поток лампочки; U – коэффициент использования; S – площадь; $k_З$ – коэффициент запаса.

Для представленного рабочего места освещенность равна:

$$E = \frac{12 \times 4 \times 900 \times 80}{64 \times 1{,}2 \times 100} = 450$$

Получено значение освещенности в 450 лк, а значит, освещение рабочего места соответствует нормативным значениям. Таким образом, место спроектировано специально для работы с ПК в течение длительного промежутка времени.

Повышенный уровень электромагнитных излучений

Основным предметом взаимодействия с разработчиком во время выполнения трудовых обязательств является персональный компьютер. Он подвергает работника электромагнитному излучению. Электромагнитное

излучение ПК является сложным по спектральному составу, и способно изменяться в диапазоне частот от 0 Гц до 1 ГГц. Оно состоит из двух составляющих: электрической (Е) и магнитной (Н). Исследования показали, что в организме человека под влиянием электромагнитного излучения монитора происходят изменения обмена веществ.

По установленным нормам время пребывания работника в рабочей зоне вычисляется по формуле:

$$T = (50 / E) – 2$$

На рабочем месте уровень напряженности электрических полей не превышает значения 4 кВ/м, из чего следует, что время пребывания в рабочей зоне может составлять до 10,5 часов. Стандартная смена длится 8 часов, что менее рассчитанного максимума, а значит, уровень электромагнитных излучений на рабочем месте в норме.

Отклонение показателей микроклимата

Микроклимат рабочего помещения – это климат внутренней среды рабочего помещения во время нахождения сотрудников. Микроклимат можно определить с помощью определенного списка показателей, действующих на организм работника. К таким показателям относятся: температура воздуха и поверхностей, относительная влажность воздуха, скорость движения воздуха и интенсивность теплового облучения.

Нормативные показатели микроклимата представлены в СанПиН 2.2.4.548-96 «Гигиенические требования к микроклимату производственных помещений»[16]. Эти нормы устанавливаются в зависимости от времени года, характера трудового процесса и характера производственного помещения. Допустимые параметры микроклимата для категории работ, к которой относится работа программиста, приведены в таблицах 6.5 и 6.6.

Таблица 6.5. Оптимальные величины показателей микроклимата на рабочих местах производственных помещений

Период года	Температура воздуха, °С	Температура поверхностей, °С	Относительная влажность воздуха, %	Скорость движения воздуха, м/с
Холодный	22 – 24	21 – 25	60 – 40	0,1
Теплый	23 – 25	22 – 26	60 – 40	0,1

Таблица 6.6. Допустимые величины показателей микроклимата на рабочих местах производственных помещений

Период года	Температура воздуха, °С		Температура поверхностей, °С	Относительная влажность воздуха, %	Скорость движения воздуха, м/с	
	диапазон ниже оптимальных величин	диапазон выше оптимальных величин			для диапазона температур воздуха ниже оптимальных величин, не более	для диапазона температур воздуха выше оптимальных величин, не более
Холодный	20,0 – 21,9	24,1 – 25,0	19,0 – 26,0	15 – 75	0,1	0,1
Теплый	21,0 – 22,9	25,1 – 28,0	20,0 – 29,0	15 – 75	0,1	0,2

Значения показателей, полученные на рабочем месте:

- температура воздуха 22,5 °С – оптимальное значение;
- температура поверхностей 23 °С – оптимальное значение;
- относительная влажность воздуха 60% – оптимальное значение;
- скорость движения воздуха 0,1 м/с – оптимальное значение.

Все измеренные показатели не только удовлетворяют санитарным нормам, но и находятся в пределах оптимальных значений.

Повышенный уровень шума на рабочем месте

Под понятием «шум» подразумевается совокупность звуков, неблагоприятно воздействующих на организм человека и мешающих его работе и отдыху. В рабочем помещении не должно быть источников шума, так как это негативно влияет на здоровье рабочих. Данное утверждение регламентируется ГОСТ 12.1.003-83[17]. При трудовой деятельности, связанной с ПК, уровень шума не должен превышать 50 дБА. Источники звуков выше данного порога должны находиться в другом помещении.

В рассматриваемом рабочем помещении источниками шума являются персональные компьютеры и кондиционер. Однако уровень шума не превышает нормы и составляет 45 дБА. Это достигается благодаря пластиковым окнам, не пропускающим звуки с улицы в помещение, и компьютерам с тихим охлаждением.

Монотонность труда

Работа программиста тесно связана с малоподвижной, однотипной деятельностью, что является показателем монотонного труда. Этот негативный фактор играет важную роль в эмоциональном состоянии. При большом количестве монотонной работы, у сотрудника могут наблюдаться следующие негативные последствия:

- бессонница;
- депрессия;

- сонливость;
- повышенная утомляемость.

Для снижения уровня монотонности работы разработчика программного обеспечения следует принимать следующие меры:

- во время рабочего дня делать регулярные перерывы;
- вне рабочего времени стоит проводить время на свежем воздухе;
- стараться организовывать свое рабочее время таким образом, чтобы не заниматься одной задачей на протяжении длительного времени;
- развитие уровня корпоративной культуры путем проведения совместных мероприятий.

Опасность поражения электрическим током

Работа программиста происходит в непосредственной близости от электрических сетей и приборов, поэтому работник должен с осторожностью обращаться с электропроводкой и компьютером, а также помнить об опасности поражения электрическим током.

Покрытие полов следует делать из однослойного линолеума, что снизит величины зарядов статического электричества. Несмотря на то, что эти величины безопасны для здоровья человека, вычислительная техника подвергается опасности при воздействии зарядов такого рода.

Также стоит уделить внимание на возможное поражение электрическим током. Факторами, повышающими вероятность удара током, являются:

- повышенная влажность (более 75%);
- высокая температура воздуха и поверхностей (более 35 °С);
- наличие токопроводящей пыли;
- неверная проектировка рабочего места;
- отсутствие защитных конструкций для проводов;
- наличие посторонних предметов на электроприборах

Несмотря на соблюдение правил электробезопасности, существует риск поражения током при соприкосновении с предметами под напряжением в штатном режиме или при возникновении неполадки.

Рассматриваемое рабочее место не относится к помещениям повышенной опасности поражения электрическим током. Для предотвращения возможных опасных ситуаций обязательны следующие меры предосторожности:

- работа за ПК должна проводиться исключительно сухими руками;
- на столе не должно быть легко проливающихся напитков и еды;
- перед началом трудовой активности следует убедиться в исправности приборов, кабелей и розеток;
- при обнаружении неисправностей следует немедленно сообщить ответственному лицу, не делая никаких самостоятельных исправлений;
- рабочее место не должно быть загромождено лишними предметами.

6.4. Экологическая безопасность

Одной из наиболее обсуждаемых проблем экологии в современном мире является загрязнение литосферы твердыми отходами, которые накапливаются на свалках, в отвалах и являются опасными источниками загрязнения земной поверхности. Наиболее частым видом мусора является бытовой. Он в основном состоит из бумаги, металла, древесины, стекла и пластика.

Основными электроприборами при работе над разработкой программного обеспечения можно назвать компьютерную технику и люминесцентные лампы. По истечению срока службы они становятся бытовым мусором. Утилизацию бытовых отходов можно подробно рассмотреть на примере люминесцентных ламп. В источниках света такого типа содержится ртуть в газообразном состоянии, что может стать причиной попадания ртути в окружающую среду при неправильной утилизации.

Попадание паров ртути в дыхательную систему человека может привести к тяжелому вреду здоровью.

При выходе из строя ртутьсодержащей лампы её замену должно осуществлять лицо, ответственное за сбор и хранение ламп. Перегоревшие лампы должны быть сданы исключительно на полигон токсичных отходов для захоронения. Смешивать отработанные люминесцентные лампы с прочим бытовым мусором запрещено.

Утилизация таких отходов, как бумажная макулатура, отходы от продуктов питания и личной гигиены, производится через сбор с помощью урны и утилизацию. Часть отходов после сортировки отправляют на переработку через специализированные компании.

Правильно организованная утилизация отходов уменьшает ущерб как окружающей среде, так и человеку. Более того, переработка позволяет сэкономить ресурсы на производстве новой продукции.

6.5. Безопасность в чрезвычайных ситуациях

Чрезвычайной ситуацией называют обстановку на определенной территории, которая сложилась в результате аварии, опасного природного явления или другого бедствия, и которая может стать причиной человеческих жертв, и/или материальных потерь. Наиболее вероятным из возможных ЧС для представленного помещения является пожар.

Наиболее частыми причинами возникновения пожара можно назвать короткое замыкание, перегрузку сетей, с последующим нагревом токоведущих частей и неисправность оборудования;

Профилактические методы борьбы с пожарами в помещении предусматривают:

- организационные методы: надлежащее содержание помещений, инструктаж персонала;
- технические методы: соблюдение противопожарных правил;

- эксплуатационные методы: своевременная профилактика, ремонт оборудования.

Также для уменьшения ущерба от возможного пожара следует:

- регулярно проводить инструктаж персонала;
- обеспечить свободный подход к оборудованию;
- поддерживать в исправности изоляцию проводников;
- соблюдать меры противопожарной безопасности при устройстве электросетей, водоснабжения и воздухопроводов.

Для быстрого детектирования пожара в рабочем помещении следует произвести установку систем противопожарной сигнализации, реагирующих на дым и температуру. Также необходимо провести установку огнетушителей.

В рассматриваемом рабочем помещении имеется план эвакуации, установлена система противопожарной сигнализации, а также 2 огнетушителя типа ОУ-2.

Выводы

Подводя итоги, можно отметить, что нарушений по организации рабочего процесса при выполнении ВКР выявлено не было, а все необходимые требования и нормы безопасности соблюдены. Необходимые рабочие условия обоснованы законодательно. Рассматриваемое в разделе рабочее место оборудовано в соответствии с правилами производственной и экологической безопасности, а также безопасности в чрезвычайных ситуациях. Рассмотрены наиболее опасные для жизни и здоровья человека ситуации, а также выявлены меры по их предупреждению.

7. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение

7.1. Оценка коммерческого потенциала и перспективности проведения научных исследований с позиции ресурсоэффективности и ресурсосбережения

7.1.1. Потенциальные потребители результатов исследования.

Компьютерная техника получила широкую распространенность достаточно недавно, однако прочно вошла в жизнь человека. Многие специальности задействуют персональные компьютеры (ПК) в своей профессиональной деятельности. Однако со временем вычислительные устройства стали не только инструментом, но и средством для развлечений. Индустрия компьютерных игр в настоящее время стала крупным источником финансов. Например, в 2018 году объем данного рынка составил 134.9 млрд долларов. Это влечет за собой усложнение программ данного вида. Таким образом, новые компьютерные игры должны конкурировать на широком рынке, что неизбежно влечет за собой увеличение сроков разработки. Из-за этого разработчики принимают меры по автоматизации некоторых процессов разработки. Одним из таких процессов является создание ландшафтов не в ручном режиме, а с помощью алгоритмических методов. Такой способ называется процедурной генерацией ландшафтов. Исходя из популярности данной тематики, но её малой популярности было решено создать свой набор средств разработки для процедурной генерации ландшафтов. Данное средство для разработки игр будет использоваться программистами, занимающимися созданием компьютерных игр.

7.1.2. Анализ конкурентных технических решений.

Некоторые компании используют свои наработки по процедурной генерации исключительно для внутреннего пользования, например в играх No Man's Sky или Minecraft. Другие же разработчики создают средства для

последующей продажи на торговых площадках. Так как рассматриваемый продукт принадлежит к проектам для продажи, то аналоги должны быть представлены также этой категорией.

Ориентировочная цена собственного разрабатываемого решения на 1 единицу товара — 80 долларов.

Сравнение будет проводиться с несколькими похожими проектами с платным доступом.

- Mantle Environment System — платный (440 долларов) набор средств для автоматического создания городов в стиле low poly. Интегрирован в игровой движок Unity.
- Vegetation Studio Pro — платный (129 долларов) набор средств для процедурной генерации естественных объектов. Требует ручной работы, но упрощает монотонную деятельность. Интегрирован в игровой движок Unity.
- World Creator Professional — платный (189 долларов) набор средств для процедурной генерации естественных объектов. Требует ручной работы, не расставляет сторонние объекты, но создает поверхность ландшафта высокого качества. Интегрирован в игровой движок Unity.

7.1.3. Оценка конкурентоспособности проекта

Для оценки конкурентоспособности следует составить оценочную карту.

Её смысл заключается в выделении наиболее важных аспектов продукта, и сравнении с аналогами на рынке. Оценки для каждого параметра выставляются по 10-балльной шкале. Также оценивается влияние каждой характеристики на продукт в целом. Этот метод позволяет получить оценку продукта в целом, а также сравнить различные продукты по отдельным критериям. Это помогает как найти слабые места в новом продукте, так и его наиболее сильные стороны.

Таблица 7.1 – Оценочная карта для сравнения конкурентных технических решений (разработок)

№ п/п	Продукт	Факторы конкурентоспособности							Итог
		Кроссплатфор-менность	Сходство результатов с реальными объектами	Создание текстур	Создание природных объектов (горы, реки, озера, леса и пр.)	Возможность использования в проектах различных жанров	Степень вовлеченности разработчика в процесс создания ландшафта	Цена	
1	Mantle Environment System	9	3	2	2	6	9	1	4,2
2	Vegetation Studio Pro	9	8	6	7	8	3	5	6,75
3	World Creator Professional	9	8	6	9	8	5	4	7,1
4	PROWOGENE SDK (свой продукт)	7	7	9	8	7	9	9	7,95
	bi	3	4	3	3	2	2	3	20
	wi	0,15	0,2	0,15	0,15	0,1	0,1	0,15	1,0

Из таблицы видно, что многие уже существующие средства разработки более кросплатформенны и дают более реалистичные результаты, однако для этого требуется ручная доработка получившихся ландшафтов, что противоречит самому смыслу процедурной генерации. Несмотря на более слабые позиции в некоторых аспектах, проект получит наибольшую оценку из всех кандидатов благодаря простоте использования и качественной симуляции реальных объектов и их текстур.

7.1.4. SWOT анализ

Для выявления достоинств и недостатков продукта с целью улучшения качества следует провести SWOT анализ[19].

SWOT-анализ – это один из методов стратегического планирования, который заключается в выявлении ключевых факторов внутренней и внешней среды. Название произошло от четырех категорий, на которые разделяются факторы: сильные стороны (Strengths), слабые стороны (Weaknesses), возможности (Opportunities) и угрозы (Threats).

Таблица 7.2 – SWOT анализ.

		Внутренние факторы	
Внешние факторы		**Сильные стороны** 1. Отсутствие сторонних разработок в проекте. 2. Целостность архитектуры проекта из-за одиночной разработки. 3. Использование новых алгоритмов генерации.	**Слабые стороны** 1. Повторное создание уже существующих решений. 2. Высокая вероятность в застое проекта из-за одиночной разработки. 4. Отсутствие самостоятельной технической поддержки.
	Возможности 1. Малое количество продуктов данного рода.	Сильные стороны и возможности показывают, что данное решение может быть полезной новинкой для	Исходя из слабых сторон и возможностей, можно сделать вывод о том, что данный продукт будет востребован, но подвержен рискам застоя или

	2. Популярность направления. 3. Рост производительности вычислительной техники.	многих компаний, причем разработка и внедрение такой системы могут пройти достаточно мягко.	столкновения с трудноразрешимыми проблемами.
	Угрозы 1. Вероятность создания аналогичного продукта крупной компанией. 2. Повышение интереса к небольшим играм.	Сильные стороны и угрозы показывают, что, несмотря на полезность разработки, существует вероятность спада интереса к данному продукту. На данный момент это лишь потенциально возможная ситуация.	При рассмотрении слабых сторон и угроз, видно, что наибольшие трудности связаны с малым масштабом разработок. На данном этапе это не является критичным, однако необходимо предпринять некоторые меры для успешности проекта в будущем.

Для улучшения качества продукта можно принять некоторые меры. Во-первых, найти дополнительных разработчиков. Это покроет риски возникновения трудноразрешимых ситуаций, а также увеличит скорость разработки. Кроме того, это позволит создавать собственные вспомогательные средства. Во-вторых, внедрение автоматического тестирования способно заменить отсутствие технической поддержи, так как этот метод быстрее, чем использование отдельного человека. Однако стоит отметить, что работа с пользователями плохо подвергается автоматизации.

Для улучшения качества и скорости работы программы стоит задуматься о методах машинного обучения. Это позволит укрепить позиции на рынке, но потребует значительных временных затрат на внедрение.

Существенной угрозой, на которую практически невозможно повлиять, является появление крупного игрока на рынке. Данный сценарий маловероятен, но возможен. В случае реального появления данной проблемы продукту стоит переквалифицироваться в более узкую область, получая конкурентное преимущество хотя бы в одном из аспектов.

7.2. Планирование научно-исследовательских работ.

7.2.1. Структура работ в рамках научного исследования.

Таблица 3 – Структура работ в рамках научного исследования.

№ работы	Наименование работы	Исполнители работы
1	Выбор научного руководителя бакалаврской работы	Федоров Константин Борисович
2	Составление и утверждение темы бакалаврской работы	Фофанов Олег Борисович, Федоров Константин Борисович
3	Составление календарного плана-графика выполнения бакалаврской работы	Фофанов Олег Борисович
4	Подбор и изучение литературы по теме бакалаврской работы	Федоров Константин Борисович
5	Анализ предметной области	Федоров Константин Борисович
6	Проектирование набора средств разработки	Федоров Константин Борисович
7	Разработка продукта	Федоров Константин Борисович
8	Тестирование продукта	Федоров Константин Борисович
9	Согласование выполненной работы с научным руководителем	Фофанов Олег Борисович, Федоров Константин Борисович
10	Выполнение других частей работы (финансовый менеджмент, социальная ответственность)	Федоров Константин Борисович
11	Подведение итогов, оформление работы	Федоров Константин Борисович

7.2.2. Разработка графика проведения научного исследования.

Согласно производственному календарю (для 6-дневной рабочей недели) в 2019 году 365 календарных дней, 299 рабочих дней, 66 выходных/праздничных дней. Таким образом, коэффициент календарности на 2019 год равен:

Ткал = Ткал / (Ткал — Твых — Тпр) = 1.22

После расчета коэффициента календарности можно составить таблицу временных показателей проведения научного исследования и диаграмму Ганта.

Таблица 4 – Временные показатели проведения научного исследования.

Наименование работы	Исполнители работы	Трудоемкость работ, чел-дни			Длительность работ, дни	
		Tmin	Tmax	Тож	Тр	Тк
Выбор научного руководителя бакалаврской работы	Федоров К.Б.	1	2	1,4	1	1
Составление и утверждение темы бакалаврской работы	Фофанов О.Б., Федоров К.Б.	1	2	1,4	1	1
Составление календарного плана-графика выполнения бакалаврской работы	Фофанов О.Б.	1	2	1,4	1	1
Подбор и изучение литературы по теме бакалаврской работы	Федоров К.Б.	5	10	7	7	9
Анализ предметной области	Федоров К.Б.	3	5	3,8	4	5
Проектирование набора средств разработки	Федоров К.Б.	3	5	7	7	9
Разработка продукта	Федоров К.Б.	35	40	37	37	45
Тестирование продукта	Федоров К.Б.	5	10	7	7	9
Согласование выполненной работы с научным руководителем	Фофанов О.Б., Федоров К.Б.	3	4	3,4	2	2
Выполнение других частей работы	Федоров К.Б.	5	10	7	7	9
Подведение итогов, оформление работы	Федоров К.Б.	10	15	12	12	15

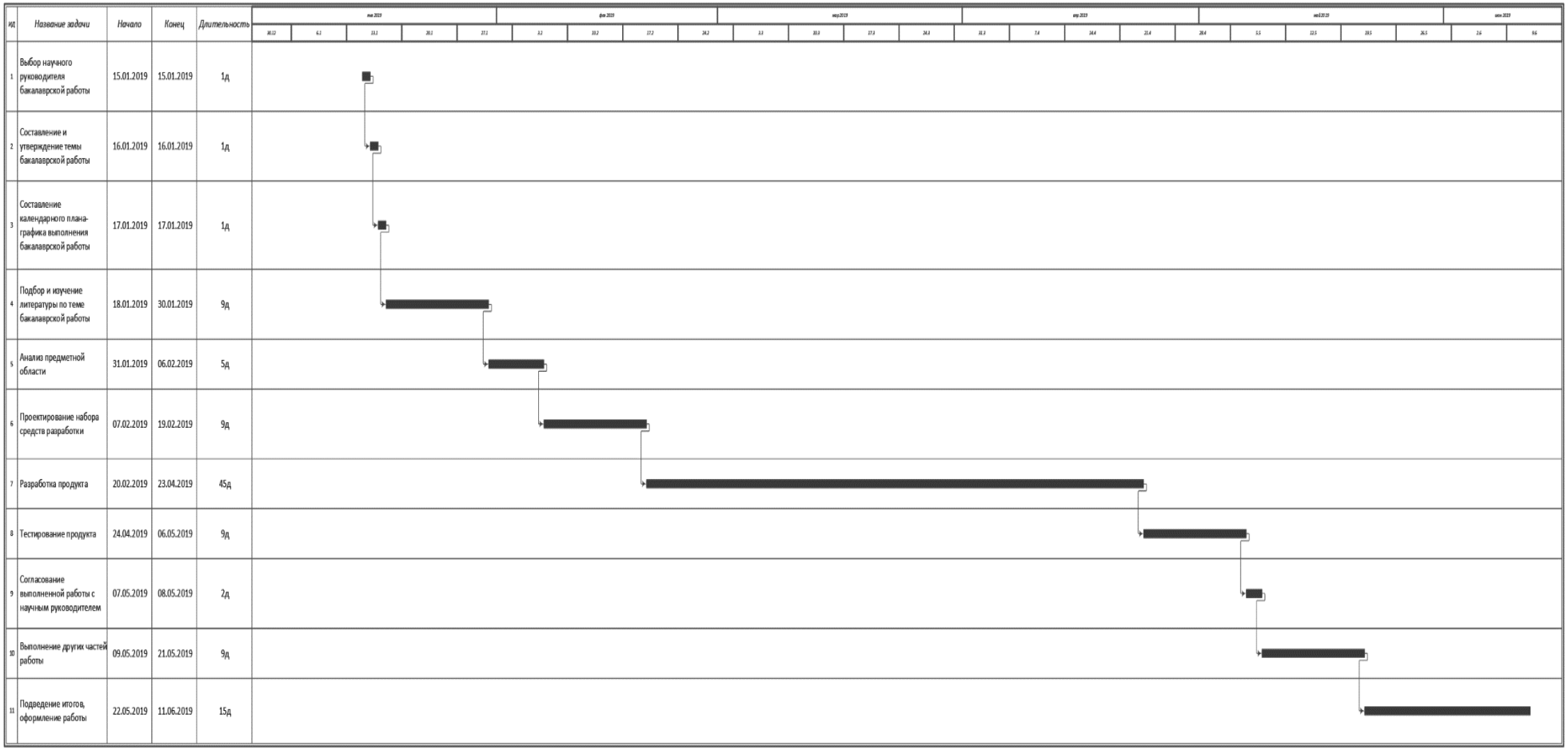


Рисунок 17. Диаграмма Ганта

7.3. Бюджет научно-технического исследования

7.3.1. Расчет материальных затрат научно-технического исследования

В связи с наличием всего необходимого для написания ВКР оборудования и программного обеспечения, закупка дополнительных средств не потребовалась. Единственным объектом закупок стали канцелярские принадлежности на общую сумму 1000 рублей. Тем не менее, требуется рассчитать амортизацию основных средств или нематериальных активов.

Для написания ВКР использовался компьютер с периферийными устройствами на общую сумму 65000 рублей. Так как срок полезного использования для офисных машин (код 330.28.23.23) составляет 2-3 года, то можно принять его за 3 года. Написание ВКР занимает примерно 5 месяцев.

Норма амортизации:

Ан = 1 / n * 100% = 1 / 3 * 100% = 33,33%

Годовые амортизационные отчисления:

Аг = 65000 * 0,33 = 21667 рублей

Ежемесячные амортизационные отчисления:

Ам = 21667 / 12 = 1806 рублей

Итоговая сумма амортизации основных средств:

А = 1806 * 5 = 9030 рублей

Все используемое ПО имеет бесплатную модель распространения для всех пользователей или для студентов. Таким образом, амортизацию для ПО рассчитывать не нужно.

7.3.2. Основная заработная плата исполнителей темы

Затраты на заработную плату:

Зп = Зосн + Здоп

Здесь Зосн – основная заработная плата, руб; Здоп – дополнительная заработная плата, руб.

Заработная плата основная:

Зосн = Здн * Тр * (1 + Кпр + Кд) * Кр

Здесь Здн – среднедневная заработная плата, руб; Кпр – премиальный коэффициент (0,3); Кд – коэффициент доплат и надбавок (0,2-0,5); Кр – районный коэффициент (для Томска равен 1,3); Тр – продолжительность работ, выполняемых работником, раб. дни.

Среднедневная заработная плата:

Здн = Зм * М / Fд

Здесь Зм – месячный оклад работника, руб.; М – количество месяцев работы без отпуска в течение года; Fд – действительный годовой фонд рабочего времени персонала, раб. дн. М следует принять за 10,4 месяца, так как рабочая неделя длится 6 дней, а отпуск – 48 дней.

Таблица 5 – Баланс рабочего времени (для 6-дневной недели)

Показатели рабочего времени	Дни
Календарные дни	365
Нерабочие дни (праздники/выходные)	66
Потери рабочего времени (отпуск/невыходы по болезни)	56
Действительный годовой фонд рабочего времени	243

Таким образом, в 2019 году действительный годовой фонд рабочего времени составляет 243 дня. Исходя из всех найденных показателей можно составить таблицу расчета основной заработной платы. Стоит отметить, что зарплата студента в месяц равняется 21760, а руководителя – 33664 рублей.

Таблица 6 – Расчет основной заработной платы

Исполнители	Здн, руб	Кпр	Кд	Кр	Тр	Зосн
Фёдоров Константин Борисович	931,29	0,3	0,2	1,3	85	154361,32
Фофанов Олег Борисович	1440,76	0,3	0,5	1,3	4	13485,51
					Итого	167846,83

7.3.3. Формирование бюджета научно-исследовательского проекта

После нахождения всех составляющих, можно оценить общий бюджет научно-исследовательского проекта. Также полезно определить удельный вес каждого наименования затрат, так как это позволит наглядно увидеть наиболее и наименее затратные позиции.

Существует 3 варианта имполнения исследования. Первый способ (Исп. 1) – использование оборудования и ПО, которое уже имеется в наличии, или бесплатных средств. Второй способ способ (Исп. 2) – дополнительно приобрести программы для профессиональной работы с изображениями. Третий способ способ (Исп. 3) заключается во внедрении автоматической системы для работы с исходным кодом программ, а также автоматической сборкой на удалённой машине.

С точки зрения финансовых вложений, варианты исполнения будут отличаться лишь затратами на специальное оборудование. Для варианта Исп. 2 необходима покупка подписки на пакет графических приложений Adobe Creative Cloud за 644 рубля в месяц (3220 рублей на весь срок написания ВКР). Для Исп. 3 требуется аренда сервера для хранения кода, а также исполняемых файлов. Системой контроля версий может послужить бесплатный Tortoise SVN/Таким образом, третий вариант потребует от 3000 рублей в месяц (15000 рублей всего).

Таблица 7 – Расчет бюджета затрат НТИ

Наименование	Исп. 1		Исп. 2		Исп. 3	
	Сумма, руб.	Уд. вес, %	Сумма, руб.	Уд. вес, %	Сумма, руб.	Уд. вес, %
Материальные затраты	1000,00	0,3	1000,00	0,3	1000,00	0,3
Затраты на специальное оборудование	0,00	0,0	3220,00	1,1	15000,00	4,9
Затраты на амортизацию	9030,00	3,1	9030,00	3,0	9030,00	2,8
Затраты на основную заработную плату	167846,83	55,9	167846,83	55,2	167846,83	52,6
Затраты на дополни-тельную заработную плату	23500,00	7,8	23500,00	7,7	23500,00	7,5
Страховые взносы	57404,05	19,1	57404,05	18,9	57404,05	18,1
Накладные расходы	41404,94	13,8	42370,94	13,8	45904,94	13,8
Общий бюджет	300185,82	100,0	304371,82	100,0	319685,82	100,0

Как видно из таблицы, более половины средств приходится на заработную плату. Наименее существенные расходы связаны с материальными закупками, что связано с характером работы за компьютером.

7.4. Определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности исследования

7.4.1. Интегральный финансовый показатель

Для нахождения интегрального показателя эффективности следует выделить самый дорогой вариант исполнения и его стоимость, после чего для каждого варианта произвести расчет по формуле:

$$I_{фин} = Ф_р / Ф_{max}$$

Здесь $I_{фин}$ – это интегральный финансовый показатель, $Ф_р$ – стоимость текущего варианта, $Ф_{max}$ – стоимость максимального варианта.

Результаты для 3 вариантов представлены в таблице 8.

Таблица 8 – Интегральные финансовые показатели

Исп. 1	Исп. 2	Исп. 3
0,94	0,95	1,0

Таким образом, самым дешевым вариантом является Исп. 1, а самым дорогостоящим – Исп. 3. Стоит отметить, что различие в ценах невелико и не превышает нескольких процентов. Таким образом, следует рассмотреть другие показатели.

7.4.2. Интегральный показатель ресурсоэффективности

Расчет интегрального показателя ресурсоэффективности состоит в том, что для каждого критерия ресурсоэффективности выбирается весовой коэффициент, а критерию проекта присваивается оценка по пятибалльной шкале. После этого весовой коэффициент умножается на оценку, а сумма этих произведений будет равна интегральному показателю ресурсоэффективности.

Таблица 9 – Сравнительная оценка характеристик исполнения проекта

Критерии	Весовой коэффициент параметра	Оценка Исп. 1	Оценка Исп. 2	Оценка Исп. 3
Способствует производительности труда	0,10	4	5	4
Удобство в эксплуатации	0,15	4	4	5
Помехоустойчивость	0,15	4	4	4
Энергосбережение	0,20	4	4	3
Надёжность	0,25	4	3	3
Материалоемкость	0,15	5	5	4
ИТОГО	1,00			

Интегральный показатель ресурсоэффективности:

$I_{p1} = 0,1 * 4 + 0,15 * 4 + 0,15 * 4 + 0,20 * 4 + 0,25 * 4 + 0,15 * 5 = 4,15$

$I_{p2} = 0,1 * 5 + 0,15 * 4 + 0,15 * 4 + 0,20 * 4 + 0,25 * 3 + 0,15 * 5 = 4$

$I_{p3} = 0,1 * 4 + 0,15 * 5 + 0,15 * 4 + 0,20 * 3 + 0,25 * 3 + 0,15 * 4 = 3,7$

7.4.3. Определение интегрального показателя эффективности

Интегральный показатель эффективности разработки может быть рассчитан по формуле:

$I = I_p / I_ф$

Таким образом, для всех вариантов исполнения:

$I1 = 4,15 / 0,94 = 4,41$

$I2 = 4,00 / 0,95 = 4,21$

$I2 = 3,70 / 1,00 = 3,7$

Основные результаты работы

Результатами выполнения выпускной квалификационной работы бакалавра являются:

1. Набор средств разработки, включающий в себя:
   - 1.1. Статическую библиотеку с реализацией алгоритмами процедурной генерации ландшафтов.
   - 1.2. Статическую библиотеку с API для генератора, написанную на языке C.
   - 1.3. Скрипты, написанные на Python для интеграции решения с Blender 3D.
   - 1.4. Консольное приложение для использования генератора без написания программ.
   - 1.5. Подробная документация в виде HTML страниц. В продуктах данного типа наиболее часто встречающейся проблемой является нехватка примеров или документации, так как она сильно влияет на порог вхождения.
   - 1.6. Исходный код программ-примеров для ознакомления с API.
2. Сайт проекта, содержащий:
   - 2.1. Информацию о проекте.
   - 2.2. Полную документацию по методам генератора.
   - 2.3. Спецификации используемых форматов файлов.
   - 2.4. Обучающие материалы.
3. Приложение для автоматического тестирования, содержащее ~70 тестов.
4. Скрипты и прочие файлы для автоматической подготовки проекта для релиза (сборка, тестирование и подготовка файлов проекта)

Заключение

В результате выполнения выпускной квалификационной работы был разработан набор средств разработки для процедурной генерации ландшафтов, позволяющий сократить время на создание игрового мира, а также обеспечить возможность разнообразия повторных прохождений.

Архитектура полученного решения была разработана с учетом последующего расширения функционала. В процессе разработки были созданы высокопроизводительные библиотеки классов, консольное приложение и прочие инструменты для пользователя. Проект представляет собой комплексное решение для автоматического создания природных объектов в виде 3D моделей, текстур и файлов с настройками.

Значительная роль была отведена повышению производительности решения. В проекте реализована многопоточная обработка данных, а вычисления были по возможности упрощены.

Важная роль была отведена контролю над устранением ошибок и снижению вероятности их появления в будущем. Для этого была внедрена система автоматического тестирования.

Также для удобства пользователей был разработан сайт, содержащий руководства по быстрому началу работы с набором средств разработки, полную документацию, а также спецификации используемых форматов файлов.

В данной работе также был проведен финансовый анализ разработанного решения, а также соблюдение правовых и социальных норм во время выполнения проекта.

Список используемой литературы

1. Фёдоров К. Б. Процедурная генерация ландшафтов / науч. рук. О. Б. Фофанов // Сборник избранных статей XXIII Международной научно-технической конференции студентов, аспирантов и молодых учёных «Научная сессия ТУСУР – 2018», 2018 год, том 1, с 188-190.
2. Фёдоров К. Б. Процедурная генерация ландшафтов / К. Б. Фёдоров ; науч. рук. О. Б. Фофанов // Сборник трудов XVI Международной научно-практической конференции студентов, аспирантов и молодых учёных «Молодежь и современные информационные технологии», 2018 г, Изд-во ТПУ, 2018.
3. Retromodo: Tennis for Two, the World's First Graphical Videogame [Электронный ресурс]. – Режим доступа: https://gizmodo.com/retromodo-tennis-for-two-the-worlds-first-graphical-v-5080541 (дата обращения: 05.04.2019).
4. Алгоритм «diamond-square» для построения фрактальных ландшафтов [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/111538/ (дата обращения: 23.11.2018).
5. Процедурная генерация планетарных карт [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/313420/ (дата обращения: 20.11.2018).
6. Генерация ландшафта как в MineCraft [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/128368/ (дата обращения: 20.11.2018).
7. Паттерны/шаблоны проектирования [Электронный ресурс]. – Режим доступа: https://refactoring.guru/ru/design-patterns (дата обращения: 01.03.2019).
8. Процедурное текстурирование: генерация текстуры булыжника [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/182346/ (дата обращения: 03.01.2019).

9. Особенности применения карт нормалей [Электронный ресурс]. – Режим доступа: https://render.ru/ru/articles/post/10701 (дата обращения: 26.02.2019).

10. Описание формата BMP | jenyay.net [Электронный ресурс]. – Режим доступа: https://jenyay.net/Programming/Bmp (дата обращения: 03.02.2019).

11. Спецификация формата OBJ [Электронный ресурс]. — Режим доступа: http://paulbourke.net/dataformats/obj (дата обращения: 21.03.2019)

12. Простой Blender. Часть 1 [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/272519/ (дата обращения: 03.03.2019).

13. Blender 3D API [Электронный ресурс]. – Режим доступа: https://docs.blender.org/api/ (дата обращения: 03.03.2019).

14. ГОСТ 12.0.003-74 ССБТ. Опасные и вредные производственные факторы. Классификация. – М.: Информационно-издательский центр Минздрава России, 1974.

15. СанПиН 2.2.1/2.1.1.1278–03. Гигиенические требования к естественному, искусственному и совмещённому освещению жилых и общественных зданий.

16. СанПиН 2.2.4.548–96. Гигиенические требования к микроклимату производственных помещений – М.: Информационно-издательский центр Минздрава России, 2003..

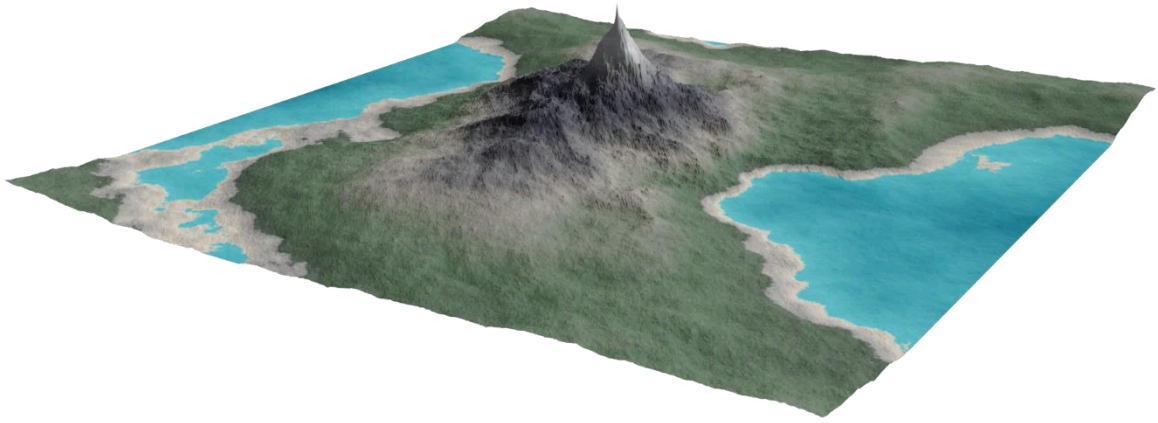
17. СНиП 2.2.4/2.1.8.562-96 Шум на рабочих местах, в помещениях жилых, общественных зданий и на территории жилой застройки. – М.: Информационно-издательский центр Минздрава России, 1996.

18. ГОСТ Р 12.1.019-2009 ССБТ. Электробезопасность. Общие требования и номенклатура видов защиты.

19. SWOT-анализ [Электронный ресурс]. – Режим доступа: https://www.e-xecutive.ru/wiki/index.php/SWOT-анализ (дата обращения: 21.01.2019)

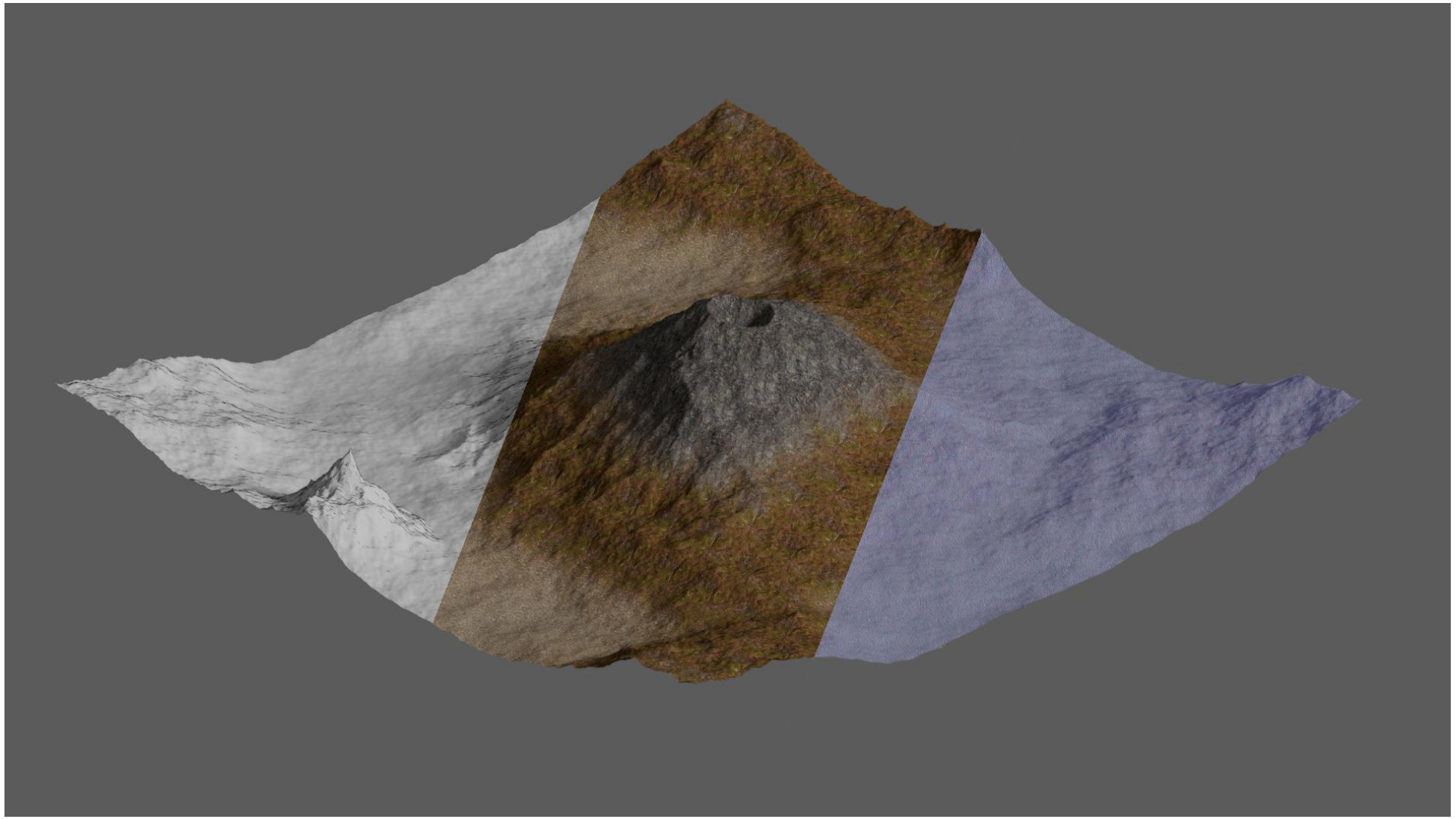
Приложения

Приложение 1. Результат генерации тропического ландшафта

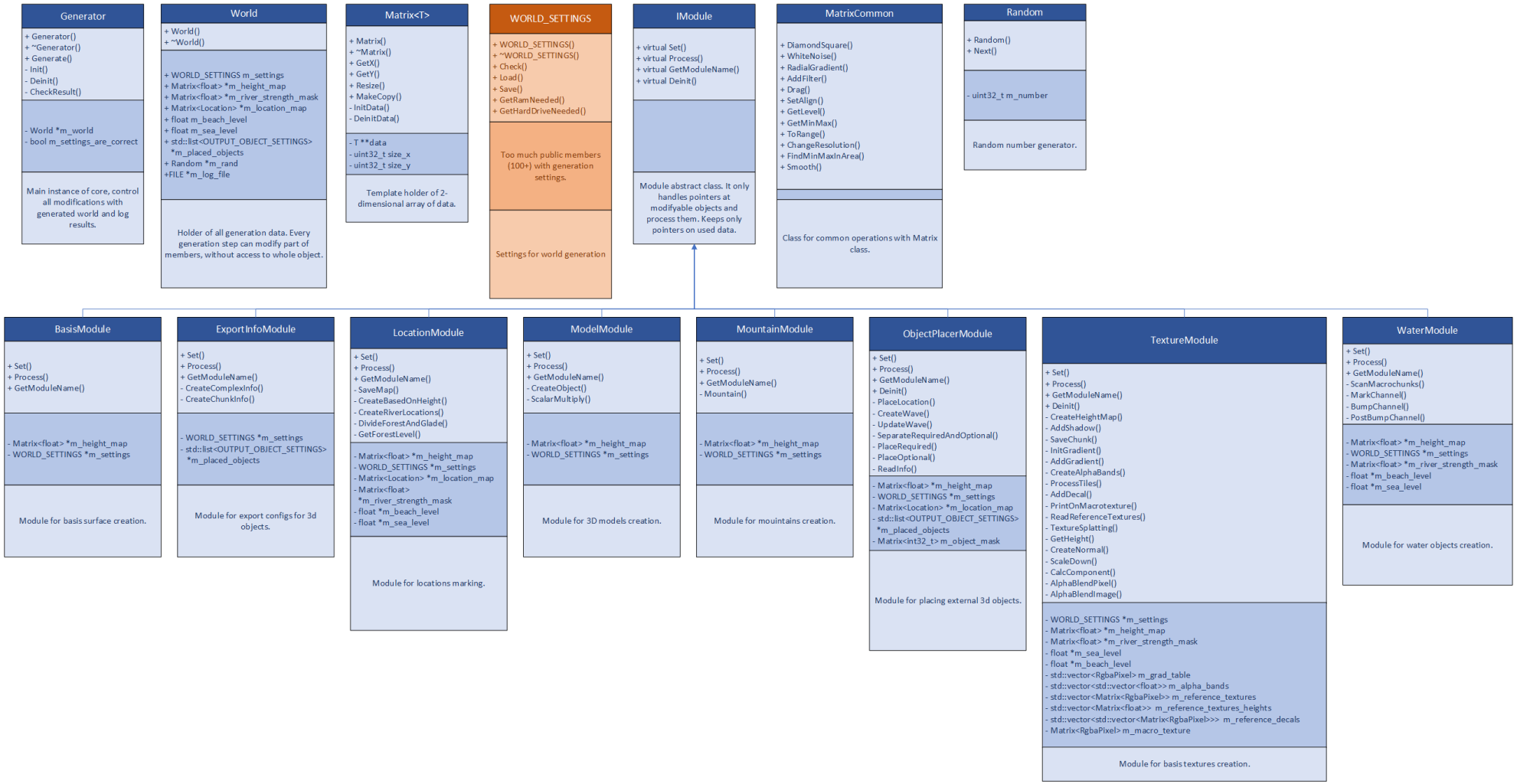


Приложение 2. Пример сгенерированного осеннего ландшафта с видом от первого лица

Приложение 3. Демонстрация создания 3D модели ландшафта, текстуры и карты нормалей



Приложение 4. Диаграмма классов



Приложение 5. Код модуля, отвечающего за генерацию водоёмов и рек

```
////////////////////////////////////////////////////////////////////////////////
//                                                                            //
// water_module.cpp                                                           //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//                                                                            //
// Implementation of water_module.h                                           //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////

#define _USE_MATH_DEFINES

#include <string>
#include <algorithm>
#include <cmath>

#include "matrix_common.h"
#include "advanced_log.h"

#include "water_module.h"

void WaterModule::Set(World * world)
{
    m_height_map = world->m_height_map;
    m_settings = &world->m_settings;
    m_river_strength_mask = world->m_river_strength_mask;
    m_sea_level = &world->m_sea_level;
    m_beach_level = &world->m_beach_level;
      m_log_file = world->m_log_file;
}

bool WaterModule::Process()
{
    bool success = true;

    if    (m_river_strength_mask->GetX()    !=    m_settings->general_size    ||
m_river_strength_mask->GetY() != m_settings->general_size)
        m_river_strength_mask->Resize(m_settings->general_size,      m_settings-
>general_size);

    for (uint32_t i = 0; i < m_river_strength_mask->GetX(); i++)
        for (uint32_t j = 0; j < m_river_strength_mask->GetY(); j++)
            m_river_strength_mask->data[i][j] = 0;

    // We need to calculate sea level always.
      PROWOGENE_LOG("    Calculating water level...             ");
    *m_sea_level = 0.0f;
    success  =  MatrixCommon::GetLevel(*m_sea_level,  *m_height_map,  m_settings-
>general_water_ratio, m_settings->system_binary_search_iterations);
    if (!success)
        return success;
      PROWOGENE_LOG("Done\n");

      PROWOGENE_LOG("    Calculating beach level...             ");
    *m_beach_level = 0.0f;
    success = MatrixCommon::GetLevel(*m_beach_level, *m_height_map, m_settings-
>general_water_ratio     +     m_settings->general_beach_ratio,     m_settings-
>system_binary_search_iterations);
    if (!success)
        return success;
      PROWOGENE_LOG("Done\n");
```

```
    int size = m_settings->general_size;
    if (m_settings->rivers_count == 0)
        return true;

      PROWOGENE_LOG("    Preprocess map for river generation... ");
    uint32_t chunks_count = size / m_settings->general_chunk_size;
    Matrix<Point<float>>    *min    =    new    Matrix<Point<float>>(chunks_count,
chunks_count);
    Matrix<Point<float>>    *max    =    new    Matrix<Point<float>>(chunks_count,
chunks_count);

    // Find min and max points in macrochunks.
    success = ScanMacrochunks(min, max, m_height_map, chunks_count);
    if (!success)
        return success;
      PROWOGENE_LOG("Done\n");

    // Mask for marking rivers
    Matrix<uint8_t> *river_mask = new Matrix<uint8_t>(m_settings->general_size,
m_settings->general_size);

    // foreach river
    for (uint32_t i = 0; i < m_settings->rivers_count; i++)
    {
            PROWOGENE_LOG("    Generating river #%d \n", i);
        // set values to 0.
        for (uint32_t i = 0; i < river_mask->GetX(); i++)
            for (uint32_t j = 0; j < river_mask->GetY(); j++)
                river_mask->data[i][j] = 0;

            PROWOGENE_LOG("         Marking possible end points of the river...
");
        std::vector<Point<float>> last_points;
        for (uint32_t x = 0; x < min->GetX(); x++)
            for (uint32_t y = 0; y < min->GetY(); y++)
            {
                Point<float> p = min->data[x][y];
                if (p.value < *m_sea_level)
                    last_points.push_back(p);
            }
            PROWOGENE_LOG("Done\n");

            if (last_points.size() == 0)
            {
                  PROWOGENE_LOG("        No more place for end of the river.\n");
                  return true;
            }

        // Find bottom and top keypoints of river
            PROWOGENE_LOG("       Choosing top and bottom points of the river...
");
        Random rand(m_settings->general_seed);
        Point<float>      bottom_point      =      last_points[rand.Next()      %
last_points.size()];
        Point<float> top_point = bottom_point;
        {
            int64_t river_size = (int64_t)m_settings->rivers_max_length[i];

            int search_window_left = (int)std::max((int64_t)0,
                (int64_t)bottom_point.x          /          (int64_t)m_settings-
>general_chunk_size -
                river_size);
            int search_window_right = (int)std::min((int64_t)chunks_count,
```

```
                (int64_t)bottom_point.x          /          (int64_t)m_settings-
>general_chunk_size +
                river_size);
            int search_window_top = (int)std::max((int64_t)0,
                (int64_t)bottom_point.y          /          (int64_t)m_settings-
>general_chunk_size -
                river_size);
            int search_window_bottom = (int)std::min((int64_t)chunks_count,
                (int64_t)bottom_point.y          /          (int64_t)m_settings-
>general_chunk_size +
                river_size);
            for (int x = search_window_left; x < search_window_right; x++)
                for (int y = search_window_top; y < search_window_bottom; y++)
                {
                    if (max->data[x][y].value > top_point.value &&
                        max->data[x][y].value           <           m_settings-
>rivers_max_head_height[i])
                        top_point = max->data[x][y];
                }
        }
            PROWOGENE_LOG("Done\n");

        // Find middle points
            PROWOGENE_LOG("       Creating river's channel...                ");
        MarkChannel(top_point, bottom_point, i, river_mask);
            PROWOGENE_LOG("Done\n");

        // Set boldness to channel
            PROWOGENE_LOG("       Set boldness to the channel...             ");
        for (uint32_t w = 2; w < m_settings->rivers_channel_width[i]; w++)
        {
            for (int x = 0; x < (int)river_mask->GetX(); x++)
                for (int y = 0; y < (int)river_mask->GetY(); y++)
                    if (river_mask->data[x][y] == w - 1)
                    {
                        // Founded previous val, mark neighbours
                        int left = std::max(0, x - 1);
                        int right = std::min((int)river_mask->GetX(), x + 2);
                        int top = std::max(0, y - 1);
                        int bottom = std::min((int)river_mask->GetY(), y + 2);
                        for (int k = left; k < right; k++)
                            for (int l = top; l < bottom; l++)
                                if (river_mask->data[k][l] == 0)
                                    river_mask->data[k][l] = w;
                    }
        }
            PROWOGENE_LOG("Done\n");

        // Bump channel
            PROWOGENE_LOG("       Bump channel on surface...                 ");
        BumpChannel(river_mask, i);

        // Post-bump channel because river couldn't go higher
        PostBumpChannel(top_point.x,   top_point.y,   river_mask,   m_height_map-
>data[top_point.x][top_point.y], i);
            PROWOGENE_LOG("Done\n");
    }

      PROWOGENE_LOG("    Smooth surface...                        ");
    success       =       MatrixCommon::Smooth(*m_height_map,       m_settings-
>rivers_smooth_radius);
    if (!success)
        return success;
    success = MatrixCommon::ToRange(*m_height_map, 0.0f, 1.0f);
```

```
    if (!success)
        return success;
      PROWOGENE_LOG("Done\n");

    delete river_mask;
    delete max;
    delete min;

    return true;
}

const char * WaterModule::GetModuleName()
{
    return "WaterModule";
}

bool           WaterModule::ScanMacrochunks(Matrix<Point<float>>*           min,
Matrix<Point<float>>*     max,      Matrix<float>      *height_map,      uint32_t
macro_chuncks_count)
{
    // Find min and max points in macrochunks.
    uint32_t macro_chunck_size = height_map->GetX() / macro_chuncks_count;
    for (uint32_t x = 0; x < macro_chuncks_count; x++)
        for (uint32_t y = 0; y < macro_chuncks_count; y++)
        {
            min->data[x][y].value = height_map->data[x * macro_chunck_size][y *
macro_chunck_size];
            max->data[x][y].value = height_map->data[x * macro_chunck_size][y *
macro_chunck_size];
            max->data[x][y].x = x * macro_chunck_size;
            max->data[x][y].y = y * macro_chunck_size;
            min->data[x][y].x = x * macro_chunck_size;
            min->data[x][y].y = y * macro_chunck_size;
            bool success = MatrixCommon::FindMinMaxInArea(
                min->data[x][y],
                max->data[x][y],
                *height_map,
                x * macro_chunck_size,
                y * macro_chunck_size,
                macro_chunck_size,
                macro_chunck_size
            );
            if (!success)
                return success;
        }
    return true;
}

void WaterModule::MarkChannel(const Point<float> &top_point, const Point<float>
&bottom_point, const int river_id, Matrix<uint8_t> *mask)
{
    uint64_t dx = std::abs((int64_t)top_point.x - (int64_t)bottom_point.x);
    uint64_t dy = std::abs((int64_t)top_point.y - (int64_t)bottom_point.y);

    if (dx > 1 || dy > 1)
    {
        int x = (top_point.x + bottom_point.x) / 2;
        int y = (top_point.y + bottom_point.y) / 2;
        float min = m_height_map->data[x][y];
        Point<float> middle_point;
        middle_point.x = x;
        middle_point.y = y;
        if (dx >= dy)
        {
```

```
            middle_point.x = x;
            uint32_t radius = (uint32_t)std::roundf(-0.5f + (float)m_settings-
>rivers_distortion[river_id] * dx);
            int top = std::max(0, (int)(y - radius));
            int bottom = std::min((int)m_settings->general_size, (int)(y +
radius));
            for (int i = top; i < bottom; i++)
            {
                if (m_height_map->data[x][i] < min)
                {
                    min = m_height_map->data[x][i];
                    middle_point.y = i;
                }
            }
        }
        else
        {
            middle_point.y = y;
            uint32_t radius = (uint32_t)std::roundf(-0.5f + (float)m_settings-
>rivers_distortion[river_id] * dy);
            int left = std::max(0, (int)(x - radius));
            int right = std::min((int)m_settings->general_size, (int)(x +
radius));
            for (int i = left; i < right; i++)
            {
                if (m_height_map->data[i][y] < min)
                {
                    min = m_height_map->data[i][y];
                    middle_point.x = i;
                }
            }
        }
        // Mark river point and go deeper
        mask->data[middle_point.x][middle_point.y] = 1; // center of channel
        MarkChannel(top_point, middle_point, river_id, mask);
        MarkChannel(middle_point, bottom_point, river_id, mask);
    }
}

void WaterModule::BumpChannel(Matrix<uint8_t> *river_mask, int river_id)
{
    // Create bump palette
    float *bump_palette = new float[m_settings->rivers_channel_width[river_id]];
    float max_bump_val = 0.0f;
    for (uint32_t bump = 0; bump < m_settings->rivers_channel_width[river_id];
bump++)
    {
        bump_palette[bump] = (std::cosf((float)M_PI * (float)bump /
            (float)m_settings->rivers_channel_width[river_id]) + 1.0f) *
            m_settings->rivers_channel_depth[river_id];
        max_bump_val = std::max(max_bump_val, bump_palette[bump]);
    }

    // Bump channel and fill biomes with river biome
    for (uint32_t x = 0; x < river_mask->GetX(); x++)
        for (uint32_t y = 0; y < river_mask->GetY(); y++)
            if (river_mask->data[x][y] != 0)
            {
                float add_koef = m_height_map->data[x][y] * m_settings-
>rivers_bump_add_coef[river_id];
                if (m_height_map->data[x][y] - bump_palette[river_mask-
>data[x][y] - 1] * add_koef > 0.0f)
                    m_height_map->data[x][y] -= bump_palette[river_mask-
>data[x][y] - 1] * add_koef;
```

```
                else
                    m_height_map->data[x][y] = 0;

                m_river_strength_mask->data[x][y]                                   =
std::max(m_river_strength_mask->data[x][y],
                    bump_palette[river_mask->data[x][y] - 1] * 255.0f * (1.0f /
max_bump_val));
            }

    delete bump_palette;
}

void WaterModule::PostBumpChannel(uint32_t x, uint32_t y, Matrix<uint8_t> *mask,
float prev_height, int river_id)
{
    float height = m_height_map->data[x][y];
    if (height > prev_height)
    {
        int  left  =  std::max((int32_t)0,  (int32_t)x  -  (int32_t)m_settings-
>rivers_channel_width[river_id] / 2);
        int   right   =   std::min(mask->GetX(),   x   +   (int32_t)m_settings-
>rivers_channel_width[river_id] / 2 + 1);
        int  top  =  std::max((int32_t)0,  (int32_t)y  -  (int32_t)m_settings-
>rivers_channel_width[river_id] / 2);
        int   bottom   =   std::min(mask->GetY(),   y   +   (int32_t)m_settings-
>rivers_channel_width[river_id] / 2 + 1);
        for (int k = left; k < right; k++)
            for (int l = top; l < bottom; l++)
            {
                if (m_height_map->data[k][l] - (height - prev_height) > 0.0f)
                {
                    m_height_map->data[k][l] -= height - prev_height;
                    height = prev_height;
                }
                else
                {
                    m_height_map->data[k][l] = 0.0f;
                    height = 0.0f;
                }
            }
    }

    mask->data[x][y] = 0;
    int left = std::max((int32_t)0, (int32_t)x - 1);
    int right = std::min((int32_t)mask->GetX(), (int32_t)x + 2);
    int top = std::max((int32_t)0, (int32_t)y - 1);
    int bottom = std::min((int32_t)mask->GetY(), (int32_t)y + 2);
    for (int k = left; k < right; k++)
        for (int l = top; l < bottom; l++)
        {
            if (mask->data[k][l] == 1)
                PostBumpChannel(k, l, mask, height, river_id);
        }
}
```

Приложение 6. Фрагменты кода API

1. Основные методы

```
#ifndef __PROWOGENE_LIB_H__
#define __PROWOGENE_LIB_H__

#include "prowogene_settings.h"

#ifdef __cplusplus
extern "C" {
#endif

int prowogene_generate_location(const char *config);
int prowogene_generate_location_2(prowogene_settings settings);
size_t prowogene_get_ram_needed(prowogene_settings settings);
size_t           prowogene_get_hard_drive_needed(prowogene_settings
settings);

#ifdef __cplusplus
}
#endif

#endif // __PROWOGENE_LIB_H__
```

2. Внутренние типы

```
#ifndef __PROWOGENE_TYPES_H__
#define __PROWOGENE_TYPES_H__

// Type of gradient.
typedef enum
{
    prowogene_gradient_linear     = 0,    // Linear distribution.
    prowogene_gradient_sinusoidal   =   1,           //   Sinusoidal
distribution.
    prowogene_gradient_quadric    = 2     // Quadric distribution.
} prowogene_gradient;

// Align value in array.
typedef enum
{
    prowogene_align_default = 0,    // No align.
    prowogene_align_middle   = 1,     // Choosen value in center of
array.
    prowogene_align_corner  = 2,    // Choosen value in 0 indexes.
    prowogene_align_random   = 3       // Align will be generated
randomly.
} prowogene_align;
```

```
// Type of value in array.
typedef enum
{
    prowogene_keypoint_default = 0,    // Value in [0][0] position.
    prowogene_keypoint_minimal = 1,    // Minimal value in array.
    prowogene_keypoint_maximal = 2     // Maximal value in array.
} prowogene_keypoint;

// Type of surface base form.
typedef enum
{
    prowogene_surface_white_noise    = 0,    // White noise.
    prowogene_surface_diamond_square  =  1,       //  Gradient  noise
generated with diamond-square algorythm.
    prowogene_surface_radial         = 2,    // Radial gradient.
    prowogene_surface_flat           = 3     // Surface without any
details.
} prowogene_surface;

// Distortion of surface.
typedef enum
{
    prowogene_distortion_none    = 0,    // All values stay "as is".
    prowogene_distortion_quadric = 1       // All values will bee
raised to the power of 2.
} prowogene_distortion;
#endif // __PROWOGENE_TYPES_H__
```