Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатики и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

Построение ландшафтных изображений на основе использования фрактальных структур

Студент _____ _____ _____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____
(Подпись, дата) (И.О.Фамилия)

Москва, 2017 г.

Содержание

1 Введение

Задача данного курсового проекта заключается в разработке программы, позволяющей генерировать горный ландшафт с помощью алгоритма построения фракталов, с предоставлением пользователю возможности задавать параметры отрисовки, такие как размер ландшафта, освещение, параметры рельефа и другие.

В зависимости от указанных параметров, пользователь может получить необходимое изображение. Таким образом программа обеспечивает возможность генерирования бесконечного количества разных горных рельефов.

В природе многие формы настолько неправильны и фрагментированы, что в сравнении с евклидовыми фигурами природа демонстрирует необычайно высокий уровень сложности. [1]. Если немного приблизить изображение горного рельефа, мы снова увидим горы. Приблизив картинку еще, мы по-прежнему будем наблюдать горы. То есть мы видим характерное для фракталов свойство самоподобия. [2]. В связи с этим применение фракталов для описания природных форм является полностью обоснованным, так как решение подобной задачи без использования фракталов является малоэффективной и очень сложной задачей. Фактически, благодаря фрактальной графике, найден способ действенной реализации сложных неевклидовых объектов, образы которых весьма похожи на природные.

Данная работа может быть востребована в области дизайна интерьеров, например, для составления фотообоев или проектирования предметов интерьера на этапе графического представления. Так же программу могут использовать разработчики компьютерных игр при прорисовке пейзажей.

Для проектирования был выбран язык программирования C++ и использовались возможности спецификации OpenGL.

2 Теоретическая часть

2.1 Фракталы

Термин «фрактал» был предложен Бенуа Мандельбротом в 1975 году. Выразив желание разработать «геометрию Природы», способную описать многие из неправильных и фрагментированных форм в окружающем мире [1], он определил семейство фигур, которое назвал фракталами.

Мандельброт обозначил контуры фрактальной геометрии, отличной от Евклидовой [1]. Отличие заключалось в отказе от принятого Евклидом по умолчанию требования гладкости. Некоторым объектам присущи шероховатость, пористость или раздробленность, причём многие из них обладают указанными свойствами «в одинаковой степени в любом масштабе». В природе нет недостатка в подобных формах: подсолнух и брокколи, морские раковины, папоротник, снежинки, береговые линии, сталагмиты и сталактиты, молнии, облака и горы.

Люди давно замечали, что некоторые формы демонстрируют повторяющуюся структуру при рассмотрении их «вблизи или издалека». Приближаясь к таким объектам, видно, что изменяются лишь незначительные детали, но форма в целом остаётся почти неизменной. Исходя из этого, фрактал проще всего определить, как геометрическую форму, содержащую в себе повторяющиеся элементы в любом масштабе. [3].

Центральное место в понимании фракталов занимает фрактальная (или хаусдорфова) размерность и симметрия. Во время кризиса в конце XVIII — начале XIX вв. математики осознали, что невозможно достичь истинного понимания неправильности и фрагментации, по-прежнему определяя размерность как число пространственных координат. [1]

Фрактальная размерность была впервые введена как коэффициент, описывающий геометрически сложные формы, для которых детали являются

более важными, чем полный рисунок. Для множеств, описывающих обычные геометрические формы, теоретическая фрактальная размерность равна обычной Евклидовой или топологической размерности. Таким образом, для множеств, описывающих точки, теоретическая фрактальная размерность равна 0; 1 для множеств, описывающих прямую; 2 для множеств, описывающих поверхность; 3 для множеств, описывающих объём. Но это меняется для фрактальных множеств. Если теоретическая фрактальная размерность множества строго превышает топологическую размерность, то считают, что множество имеет фрактальную геометрию. [2]

В отличие от топологической размерности, фрактальный коэффициент может принимать не целочисленное значение, показывая то, что фрактальное множество заполняет пространство не так как его заполняет обычное геометрическое множество. [1]

Итак, Мандельброт помимо интуитивного понятия фракталов дает строго математическое: «Фракталом называется множество, у которого размерность Хаусдорфа-Безиковича больше топологической размерности».

Строго математическое определение размерности Хаусдорфа (Бизекович придал ей окончательный вид) даётся с помощью понятия меры Хаусдорфа. Его суть можно прояснить следующим образом. Пусть отрезок длины A покрывается отрезками длины ε, или квадрат площадью A покрывается квадратами со стороной длины ε, или куб площадью A покрывается кубами со стороной длины ε. Тогда, соответственно, для отрезка $A = N(\varepsilon)$, для квадрата $A = N(\varepsilon)\varepsilon^2$ и для куба $A = N(\varepsilon)\varepsilon^3$, где $N(\varepsilon)$ — обозначает минимальное число отрезков, соответственно, квадратов или кубов, необходимых для покрытия множества A. В приведенных примерах показатель степени в каждом случае совпадает с классической размерностью рассмотренных математических объектов, поэтому можно записать, что $A = N(\varepsilon)\varepsilon^d$, где d — размерность, и, следовательно, при $\varepsilon \to \infty$ имеем, что $N(\varepsilon)$ растёт пропорционально ε^{-d}. Прологарифмировав

последнее равенство для A, получим, что $d = (lnA - lnN(\varepsilon))/ln\varepsilon$. Поэтому хаусдорфова размерность произвольного объекта в n-мерном пространстве определяется по формуле:

$$d = -lim(lnN(\varepsilon))/ln\varepsilon \text{ при } \varepsilon \to 0. [4]$$

Хотя Мандельбротом и было предложено два понятия фракталов, тем не менее точного определения еще не существует. Дело в том, что математическое определение при всей своей правильности и точности слишком ограничительно. Оно исключает многие фракталы, встречающиеся в физике. Первое же определение является скорее философским и не дает точных данных о фракталах.

2.2 Обзор алгоритмов

Для построения фрактальных поверхностей существует несколько способов. Все они имеют свои достоинства и недостатки.

Простой способ построения фрактальной поверхности состоит в параллельном переносе триадной кривой Коха на расстояние l вдоль направления, перпендикулярного ее плоскости. [5] (Кривая Коха является типичным геометрическим фракталом. Процесс её построения выглядит следующим образом: берётся единичный отрезок, разделяется на три равные части и заменяется средний интервал равносторонним треугольником без этого сегмента. В результате образуется ломаная, состоящая из четырёх звеньев длины 1/3. На следующем шаге операция повторяется для каждого из четырёх получившихся звеньев и т. д. Предельная кривая и есть кривая Коха) [6].

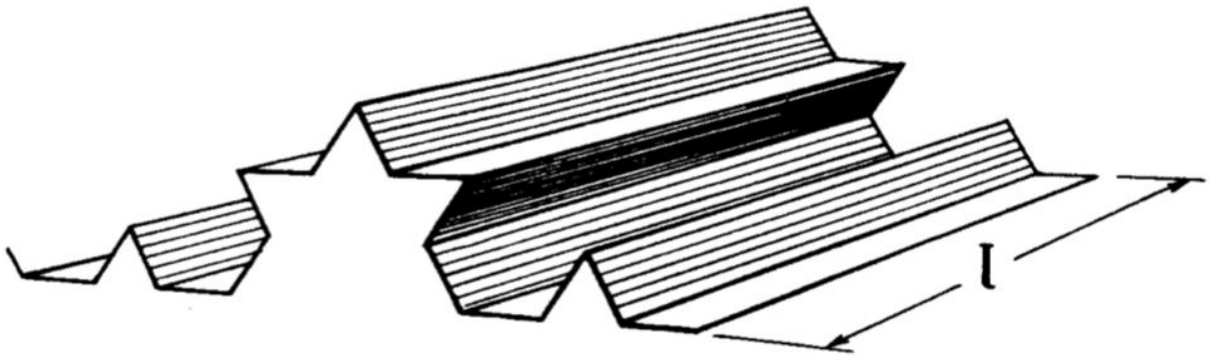

Рисунок 1. Параллельный перенос кривой Коха.

Такая поверхность (рис. 1) не является реалистичной моделью пейзажей, и не подходит для реализации поставленной задачи.

Другой способ построения более приемлемых поверхностей заключается в добавлении к вертикальной координате $z(x, y)$, полученной после параллельного переноса фрактальной кривой, дополнительных слоев с подобными профилями, но повернутых относительно первой поверхности [5]. Поверхности, полученные таким образом также не отвечают заданным требованием реалистичности.

Выбранный для данного проекта метод для построения горного рельефа позволяет при каждом его выполнении построить горный пейзаж или хотя бы линию, разделяющую небо и горы.

Основные отличительные черты данного алгоритма заключаются в следующих утверждениях:

1. Строит линии, имитирующие горный горизонт и почти неотличимые от тех, которые реально наблюдаются в природе
2. Демонстрирует принцип построения фракталов

Этот алгоритм называют методом срединного смещения, или методом случайных сложений, или ромб-квадрат (diamond-square). Последнее название лучше всего передает суть алгоритма, поэтому далее будет использоваться это название.

3 Практическая часть

3.1 Diamond-square

3.1.1 Определение начальных условий

Алгоритм diamond-square основывается на построении карты высот двумерного массива, в котором содержится информация о высоте каждой точки местности.

Этот алгоритм можно применять к сетке любого размера, но удобнее всего использовать квадрат размера $2^m + 1$ [7].

Сначала нужно установить начальное значение $seed$, которое будет стоять в четырех угловых точках массива размера $n \times n$ [2] и повлияет на остальную визуализацию. В реализованном алгоритме, этой величине добавлено случайное приращение в каждой из четырех точек, чтобы увеличить элемент случайности.

Перед тем как приступить к построению карты высот необходимо определить начальное значение $range$, прибавляющееся к срединным точкам и коэффициент неровности $divisor$, который будет уменьшать это значение при каждой итерации [2], и, соответственно, определять, будет ли ландшафт гладким или гористым. Значение, очевидно, должно быть больше 1, чтобы при делении на него, $range$ гарантированно уменьшалось [8] (чем больше $divisor$, тем более гладким будет рельеф; при тестировании программы было выявлено, что значения в диапазоне $(1.5, 2]$ дают наиболее реалистичный рельеф, хотя, в зависимости от потребностей, пользователям могут понадобиться как очень гладкие, так и скалистые рельефы).

3.1.2 Описание алгоритма

Изначально устанавливается начальные значения в углах квадрата размера $n \times n$ (рис. 2).

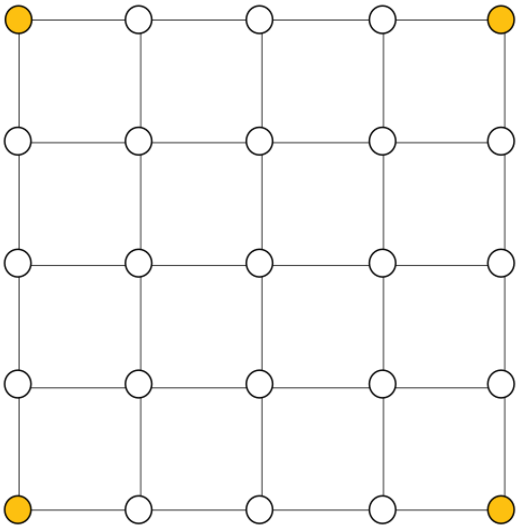

Рисунок 2. Начальные значения в угловых точках.

Далее будут чередоваться этапы diamond (ромб) и square (квадрат).

На шаге «square» происходит нахождение срединной точки, присваивание ей значения, на основе среднего от угловых, плюс случайное число (рис. 3).

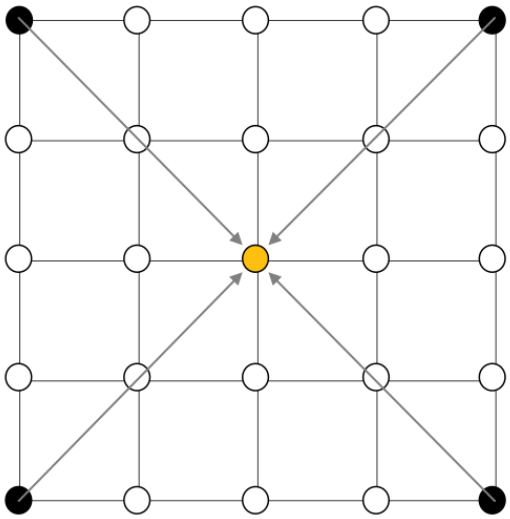

Рисунок 3. Шаг «square».

Второй шаг — «diamond» — определяет высоту точек, лежащих на серединах сторон квадратов (рис. 4). Здесь усредняются четыре точки — «сверху» и «снизу» (если говорить о точках на вертикальной стороне), и пара точек «слева» и «справа», полученных на шаге «square» [9]. Замечу, что эти две высоты, которые были на предыдущем шаге, должны быть уже посчитаны — поэтому обсчет нужно вести «слоями», сначала для всех квадратов выполнить шаг «square» — затем для всех ромбов выполнить шаг «diamond» — и перейти к меньшим квадратам.

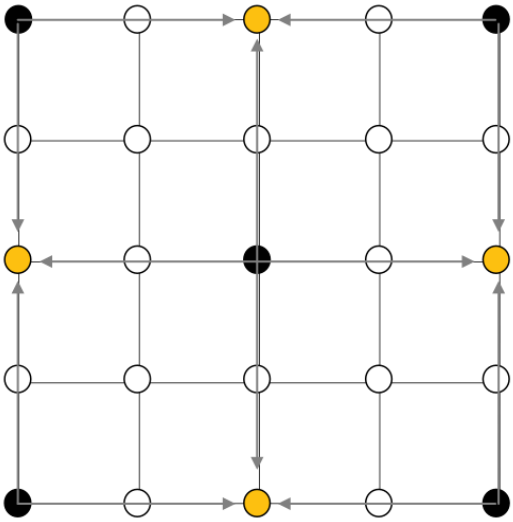

Рисунок 4. Шаг «diamond».

Далее снова повторяется шаг «square» для полученных квадратов (рис. 5).

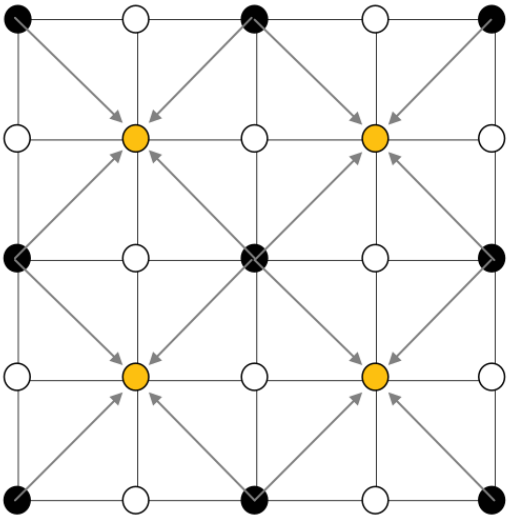

Рисунок 5. Последний шаг «square».

Затем выполняется шаг «diamond», и так до тех пор, пока все точки массива не будут заполнены (рис. 6).

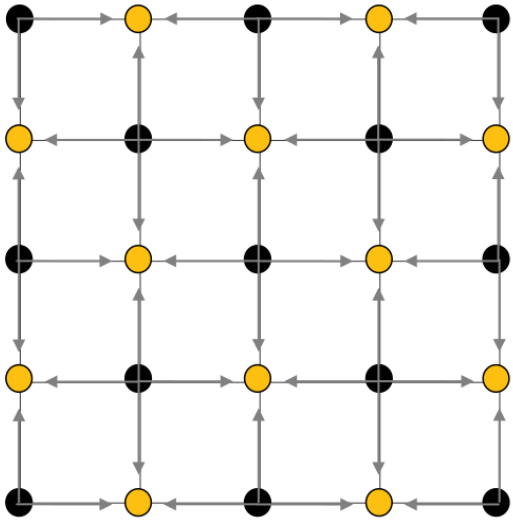

Рисунок 6. Заполненный массив.

На шаге «diamond» может возникнуть ситуация, когда одна из точек выходит за пределы массива. В таких случаях существует несколько вариантов решения, например, принять за эту точку противоположную ей, либо считать ее равной 0 (или 1, или любой другой константе) [2]. В проекте проблема была решена следующим обзором: эта точка не учитывается, то есть берется среднее значение от трех точек.

С помощью алгоритма diamond-square была получена матрица со значениями, соответствующими высотам ландшафта в данных точках сетки.

3.2 Технологии, используемые для реализации

Для реализации проекта был использован язык C++, и спецификация OpenGL версии 3.2.

Характерными особенностями стандарта OpenGL (Open Graphics Library – открытая графическая библиотека) [10], которые обеспечили выбор этого графического стандарта, являются:

- Стабильность. Дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.
- Надежность и переносимость. Приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой операционной системы и организации отображения информации.
- Легкость применения. Стандарт OpenGL имеет продуманную структуру и интуитивно понятный интерфейс, что позволяет с меньшими затратами создавать эффективные приложения, содержащие меньше строк кода, чем с использованием других графических библиотек. Необходимые функции для обеспечения совместимости с различным оборудованием реализованы на уровне библиотеки и значительно упрощают разработку приложений [11].

В проекте были использованы библиотеки:

GLFW — библиотека для создания и открытия окон, создания OpenGL контекста и управления вводом [12]

GLM — библиотека математических вычислений для графических программ [13]

ImGui — функциональный GUI-фреймворк [14]

3.3 Методы придания реалистичности

Прежде чем приступить к описанию построения ландшафта, остановимся на дополнительных методах придания ему реалистичности. В данной программе было реализована имитация водной поверхности и неба.

Вода представляет собой горизонтальный квадрат, который подстраивается под размеры генерируемого рельефа. Ему также добавлена прозрачность (альфа-канал).

o

В качестве неба используется текстура, натянутая на «мир», то есть на условный куб, в котором находится наблюдатель. Для того, чтобы администратор мог встроить желаемый фон, без изменения кода программы, текстура должна представлять собой развертку куба (Приложение 1).

3.4 Генерация ландшафта

3.4.1 Высотная поясность

После того как была получена карта высот, каждой точке необходимо задать цвет, в зависимости от типа природного комплекса, которому она принадлежит.

Для того, чтобы определить тип высотного пояса, использовалась существующая схема природных зон Кавказских гор.

В горах Кавказа отчетливо выражены 4 пояса:

1. Лесостепной/степной
2. Лесной
3. Горная тундра/гольцы
4. Снега и ледники

Средняя высота Кавказских гор составляет 5000 м. [15] Лесостепной пояс заканчивается на высоте примерно 500 м, что составляет 10% общей высоты. Лесной — на высоте 2000 м. (занимает 30%), горная тундра — 2500 м. (10%), а ледники занимают оставшуюся часть.

Найденное процентное соотношение применяется к генерируемому рельефу, причем начинаться природные зоны в данной реализации будут с установленного уровня воды. Для этого находится высота рельефа (от максимального значения отнимается минимальное), после чего определяются четыре полосы закраски путем задания их верхних границ, зависящих от найденного выше их процентного содержания для конкретной высоты рельефа:

- WATER – 30% по умолчанию
- STEPPE (степь) – 10%
- FOREST (лес) – 30%
- MOUNTAIN (каменистый склон) – 10%
- GLACIER (ледник) – оставшаяся часть

3.4.2 Полигональная сетка

Карта высот представляет собой сетку — квадраты. Для рисования полигональной сетки необходимо генерировать треугольники. Все треугольники содержат в себе помимо вершин дополнительную информацию о своем цвете и нормали. Эта информация пригодится позже. Класс Triangle содержит в себе следующе атрибуты:

- Векторы вершин a, b, c
- Вектор цвета

Также он предоставляет методы setWater(), setBeach(), setForest(), setMountain(), setGlacier() для задания треугольнику цвета в зависимости от его принадлежности тому или иному типу, и метод getNormal(), возвращающий вектор нормали к плоскости треугольника, чтобы корректно настроить освещение ландшафта.

Для генерации рельефа в классе Terrain заполняется вектор типа Triangle (по два треугольника на квадрат).

3.5 Визуализация

Чтобы придать ландшафту реалистичный объем, необходимо задать параметры освещения. По умолчанию в программе заданы следующие параметры:

- Позиция источника света lightPos
- Рассеянный свет lightColorAmbient
- Зеркальное отражение lightColorSpecular
- Диффузное отражение lightColorDiffuse

Помимо освещения, изначально задан естественный цвет воды:

waterColor = glm::vec4(0.28f, 0.48f, 0.8f, 0.6f) (вектор соответствует голубому цвету).

Основным этапом визуализации является проецирование изображения на плоскость. В программе применяется перспективная проекция, матрица проекции задается следующим образом:

```
glm::mat4 projection = glm::perspective (
80.0f, (float)window_width / window_height, 0.3f, 1700.0f),
```

где

- первый параметр — вертикальное поле зрения в радианах,
- второй — отношение сторон (рассчитывается путем деления ширины области просмотра на её высоту)
- третий — ближняя плоскость отсечения
- четвертый — дальняя плоскость отсечения.

glm::perspective создает усеченную пирамиду, которая определяет видимое пространство, а все, что находится за его пределами и не попадет в объем пространства отсечения будет обрезано [16].

Затем задается матрица модели (единичная матрица):

```
glm::mat4 model = glm::mat4(1.0f)
```

и трансформируется с учетом заданных параметров масштабирования ландшафта scaleTerrain (1.0, 1.0, 1.0) — по умолчанию):

model = glm::scale(model, scaleTerrain).

Итоговая матрица mpv (ModelViewProjection), которая является результатом перемножения наших трех матриц задается следующим образом:

glm::mat4 mvp = projection * view * model.

Далее с помощью glMatrixMode(GL_PROJECTION) программой определяется, что будет изменяться матрица проекции, а затем в функцию glLoadMatrixf передается матрица mpv, которая заменит матрицу по умолчанию [17].

3.6 Камера

В OpenGL камеру можно сымитировать, перемещая все объекты сцены в направлении противоположном движению наблюдателя, и тем самым создать иллюзию, что движется наблюдатель.

Для однозначного математического описания камеры, необходимо ее положение в мировом пространстве, направление в котором она смотрит, вектор указывающий правое направление, и вектор указывающий направление вверх [18].

Положение камеры — это вектор, содержащий координаты камеры в мировом пространстве [11], поэтому прежде всего задается начальное положение камеры: glm::vec3(0, 0, 0), которое будет изменяться при изменения положения курсора мыши и нажатии соответствующих клавиш пользователем, то есть координаты позиции будут увеличиваться или уменьшаться на величину direction * deltaTimeSec * speed [19], где

- direction — направление, куда смотрит камера,
- speed — некая константная величина,
- deltaTimeSec — величина, введенная для того, чтобы избежать проблем, связанных с тем, что на разных компьютерах с отличающейся скоростью рендеринга камера будет двигаться с разной скоростью (возможно

слишком быстро или медленно). Для решения этой проблемы используется эта величина, равная времени, затраченному на визуализацию последнего выведенного кадра. Вычисляется эта величина так:

```
auto currentTime = std::chrono::high_resolution_clock::now();
float DeltaTimeMs = std::chrono::duration_cast<std::chrono::milliseconds>
    (currentTime - prevTime) ;
float deltaTimeSec = deltaTimeMs / 1000.0f.
```

В результате умножения на deltaTimeSec, когда вывод кадра занимает много времени и значение deltaTimeSec большое, то и скорость, умноженная на эту переменную, станет больше, что сбалансирует общую производительность. При использовании этого подхода мощность компьютера перестает влиять на поведение программы и уже не имеет значения, на медленном или очень быстром компьютере запускается программа, скорость движения камеры в любом случае будет скорректировано, и у всех пользователей будет одинаковый результат.

Чтобы посчитать вектор, который будет представлять направление в Мировом Пространстве, в которое смотрит камера (вектор direction) сферические координаты преобразуются в декартовы [20] по известному алгоритму:

```
glm::vec3 direction(cos(verticalAngleRad) * sin(horizontalAngleRad),
                    sin(verticalAngleRad),
                    cos(verticalAngleRad) * cos(horizontalAngleRad)),
```

где

- verticalAngleRad — это угол, характеризующий величину наклона вверх или вниз
- horizontalAngleRad — представляет собой величину поворота влево или вправо

Углы вычисляются следующим образом:

horizontalAngleRad += mouseSpeedRad * mouseDeltaX

verticalAngleRad += mouseSpeedRad * mouseDeltaY,

где mouseDeltaX— величина, на которую изменилось положение мыши по горизонтали (mouseDeltaY — по вертикали), а mouseSpeedRad — коэффициент, используемый для того, чтобы ускорить или замедлить вращение.
Чтобы правильно настроить его, необходимо было протестировать программу на различных значениях.

Еще одно необходимое значение — это вектор, указывающий в правую сторону и всегда горизонтальный (значение по оси Y всегда равно 0):

glm::vec3 right = glm::vec3(sin(horizontalAngleRad - 3.14f / 2.0f), 0,
cos(horizontalAngleRad - 3.14f / 2.0f)).

Вектор «вверх» это вектор, перпендикулярный векторам «вправо» и «направление»:

glm::vec3 up = glm::cross(right, direction).

Для задания матрицы камеры используется glm::lookAt(position, position + direction, up) [21], где первый и третий параметр были описаны выше, а второй параметр определяет координату цели.

3.7 Режим закраски

Чтобы расширить возможности программы была добавлена функция выбора режима отрисовки:

- полигональная сетка (glPolygonMode(GL_FRONT_AND_BACK, GL_LINE))
- закрашенными полигонами (glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)) [18]

Это может пригодиться в зависимости от потребностей пользователя.

3.8 Пользовательское меню

Для реализации графического пользовательского интерфейса были использованы возможности ImGui — графический интерфейс немедленного режима, то есть способ создания графического интерфейса, который включает в себя создание и рисование виджетов в каждом кадре [22].

Для использования ImGui в билд проекта необходимо добавить следующие файлы:

1. imgui.cpp
2. imgui_draw.cpp
3. imgui_impl_glfw_gl2.cpp
4. imgui_demo.cpp

Чтобы отобразить/скрыть меню на экране по желанию пользователя, к нажатию определенной кнопки привязана булевая переменная show_menu. Если она истинна, происходит рендеринг меню.

3.8.1 Инструментарий ImGui

Ниже приведен краткий обзор инструкций, используемых в проекте.

1. Описание окна меню с его возможностями происходит в блоке

```
ImGui::Begin("Menu", &show_menu);
…
ImGui::End();
```

2. Размер окна задается функцией

```
ImGui::SetWindowSize(ImVec2(500.0f, 400.0f));
```

3. ImGui::CollapsingHeader(…) — меню с выплывающим подсписком (дерево). Пример на рис. 7.

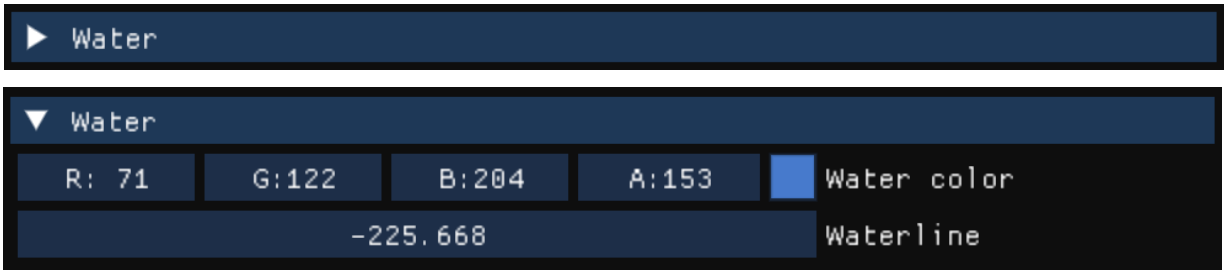

Рисунок 7. Меню с выплывающим подсписком.

4. ImGui::Checkbox(…) — флажок

   Пример на рис. 8.

Рисунок 8. Установка флажка.

5. ImGui::ColorEdit4(…) — функция для задания компонентов цвета (RGBA)
6. ImGui::Button(…) — кнопка
7. ImGui::DragFloat(), ImGui::DragInt() — поля для введения значений типа float и int

Представленный выше инструментарий позволяет создать исчерпывающее для данного проекта меню с приятным интерфейсом.

3.8.2 Описание пользовательского интерфейса

После запуска программы перед пользователем всплывает окно размера 1024 * 768, в котором отображается графическое представление созданных структур.

Чтобы двигать камеру и перемещаться по карте используется мышь и клавиши «W», «A», «S», «D» для движения вперед, влево, вниз и вправо соответственно.

Для открытия меню, пользователь должен нажать клавишу «M», после чего появляется возможность управлять параметрами изображения (Приложение 2).

Ниже представлены возможности всех пунктов меню.

1. «Lighting»
   1.1. «Light» — включение/выключение света
   1.2. «Light position» — позиция источника света
   1.3. «Light ambient color» — рассеянный свет
   1.4. «Light specular color» — зеркальное отражение
   1.5. «Light diffuse color» — диффузное отражение
2. «Water»
   2.1. «Water color» — цвет воды
   2.2. «Waterline» — уровень воды в процентах
3. «Terrain»
   3.1. «Terrain scale» — масштабирование ландшафта по трем осям
4. «Diamond Square»
   4.1. «Size» — размер стороны карты (задается значение степени, в которую будет возведена двойка)
   4.2. «Range» — случайное значение, прибавляющееся к срединным точкам при реализации алгоритма «diamond-square»
   4.3. «Divisor» — число *n*: при каждой итерации цикла в алгоритме «diamond-square» величина *range* уменьшается в *n* раз
   4.4. «Length» — параметр масштабирования при создании полигонов
   4.5. «Generate» — генерация нового ландшафта
5. «Wireframe» — включение/выключение режима отрисовки полигональной сетки

Чтобы выйти из меню необходимо так же нажать клавишу «M».

Тестирование

В результате тестирования программы было выявлено, что для размеров карты оптимальными будут значения 2^8 — 2^9, так как при меньших значениях рельеф не отвечает заданным условиям реалистичности, демонстрируя малое разнообразие форм. При больших значениях потребуется достаточное количество времени для создания реберного архива и рендеринга сцены, так как число граней растет экспоненциально [8].

Значение $divisor$ по умолчанию установлено на значении 2, так как при тестировании программы было оказалось, что значения в диапазоне $(1.5, 2]$ дают наиболее реалистичный и неоднородный рельеф.

Изменение параметров света позволило добиться разнообразных реалистичных результатов, например, имитация заката или солнечного дня (Приложение 3).

Возможные комбинации, полученные при изменении параметров масштабирования или параметров, необходимых при реализации алгоритма, могут давать довольно непредсказуемый результат, поэтому при пользовании программой следует соблюдать пропорции, при которых рельеф оказывается реалистичным. Пропорции эти для себя определяет сам пользователь в зависимости от потребностей.

Во время проектирования необходимо было определить коэффициент, используемый для того, чтобы ускорить или замедлить вращение камеры при перемещении курсора. Было обнаружено, что значение 0.0005 обеспечивает оптимальную скорость вращения. При больших значениях, камера отвечала малейшим возмущениям, связанным с перемещением курсора, что давало крайне неудобный результат. При меньших значениях — наоборот, движение курсора практически не давало результата по вращению камеры.

4 Заключение

В ходе работы был реализован на практике алгоритм построения фрактальных поверхностей «diamond-square», или алгоритм серединного смещения. Были произведены доработки алгоритма, направленные на увеличение элемента случайности в рельефе.

Для корректной закраски ландшафта была изучена география существующей системы гор и рассчитаны высоты цветовых областей для предоставления данных программе.

Было разработано меню, с помощью которого пользователь может изменять и настраивать параметры освещения, отображения и построения ландшафта.

В дальнейшем возможны доработки, направленные на повышение реалистичности изображения, путем дополнительных преобразований алгоритма, оснащение шумом закраски рельефа, то есть обеспечение более плавного перехода между границами высотных поясов и добавление элемента случайности. Также возможна работа по повышению реалистичности воды, например, генерация волн с помощью фрактальных алгоритмов.

Возможны изменения пользовательского меню, направленные на предоставление дополнительных возможностей в настройке параметров, например, изменении текстуры фона на пользовательскую, либо, наоборот, скрытие некоторых возможностей от пользователя, с целью предотвращения нежелательных результатов работы программы (например, задание слишком больших размеров карты, на рендеринг которых могут потребоваться часы работы).

5 Список литературы

1. Мандельброт Б. Фрактальная геометрия природы. — Москва: Институт компьютерных исследований, 2002. — 656 с.
2. Кроновер Р. М. Фракталы и хаос в динамических системах. Основы теории. Москва: Постмаркет, 2000. — 352 с.
3. Деменок С. Л. Просто фрактал. 3-е издание — СПб.: ООО «Страта», 2016. — 224 с.
4. Еровенко В. А. Концепция фрактала Мандельброта с математической и философской точек зрения // Математические структуры и моделирование. — 2015. — №4(36). — С. 29 — 34.
5. Feder J. Fractals. Plenum Press, New York, 1988.
6. Максименко-Шейко К. В., Толок А. В., Шейко Т. И. R-функции как аппарат в приложениях фрактальной геометрии // Прикладная математика. — №6(30). — 2010.
7. Boxley. P. Terrain generation with the diamond square algorithm [Электронный ресурс] // URL: http://www.paulboxley.com/blog/2011/03/terrain-generation-mark-one
8. Mann Z. Terrain Generator [Электронный ресурс] // URL: https://sdm.scad.edu/faculty/mkesson/vsfx419/wip/best/winter12/jonathan_mann/terrain.html
9. Beard D. Terrain Generation — Diamond Square Algorithm [Электронный ресурс] // URL: https://danielbeard.wordpress.com/2010/08/07/terrain-generation-and-smoothing/
10. Программные методы создания изображений. библиотека opengl ее возможности [Электронный ресурс] // URL: http://csaa.ru/programmnye-metody-sozdanija-izobrazhenij/

11. Графическая библиотека OpenGL [Электронный ресурс] // URL: https://rsdn.org/article/opengl/ogltut2.xml
12. GLFW [Электронный ресурс] // URL: http://www.glfw.org/index.html
13. OpenGL Mathematics [Электронный ресурс] // URL: https://glm.g-truc.net/0.9.8/index.html
14. Daler E. Using ImGui with modern C++ and STL for creating awesome game dev tools. Part 2. Some tips and tricks. [Электронный ресурс] // URL: https://eliasdaler.github.io/using-imgui-with-sfml-pt2/
15. Мильков Ф.Н. Природные зоны СССР — Москва: Мысль, 1977 — 293 с.
16. Рябинин К. В. Вычислительная геометрия и алгоритмы компьютерной графики. Работа с 3D-графикой средствами OpenGL: учеб. пособие / К. В. Рябинин; Перм. гос. нац. исслед. ун-т. — Пермь, 2017. — 100 с.: ил.
17. Учебное пособие по курсу "Компьютерная визуализация". Порт вывода и система координат [Электронный ресурс] // URL: http://aco.ifmo.ru/el_books/computer_visualization/lectures/4.html
18. Shreiner D. OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1 / Dave Shreiner; the Khronos OpenGL ARB Working Group — 7th ed. — 2009.
19. Tutorial 6: Keyboard and Mouse [Электронный ресурс] // URL: http://opengl-tutorial.blogspot.ru/p/6.html?m=1
20. Hearn D., Baker M. Computer Graphics with OpenGL (3rd Edition). Prentice Hall. — 2003.
21. Tutorial 6: Matrices [Электронный ресурс] // URL: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/
22. Daler E. Using ImGui with SFML for creating awesome game dev tools. Part 1. [Электронный ресурс] // URL: https://eliasdaler.github.io/using-imgui-with-sfml-pt1/

6 Приложение 1. Представление текстуры

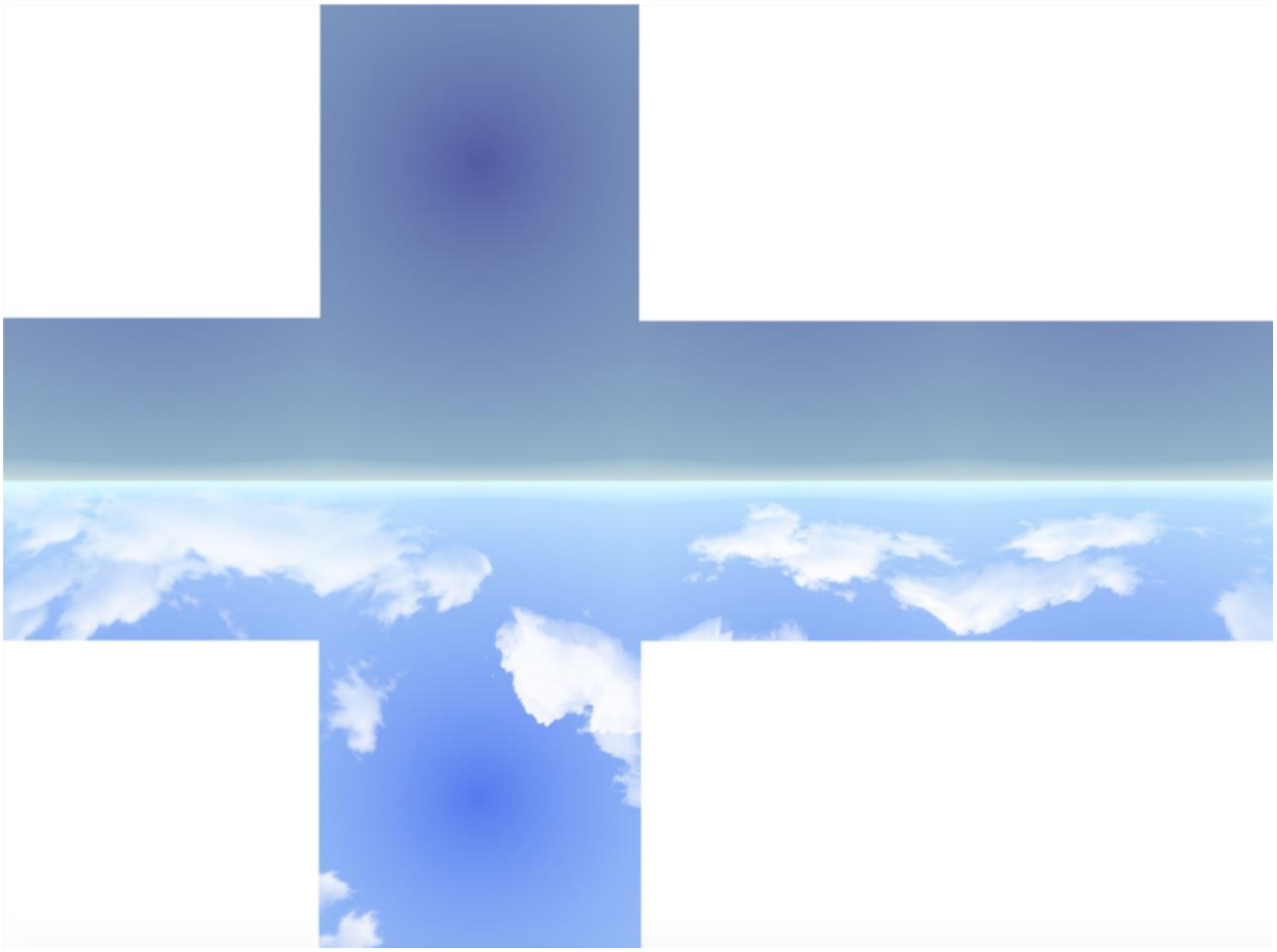

Рисунок 1. Развертка куба.

7 Приложение 2. Интерфейс

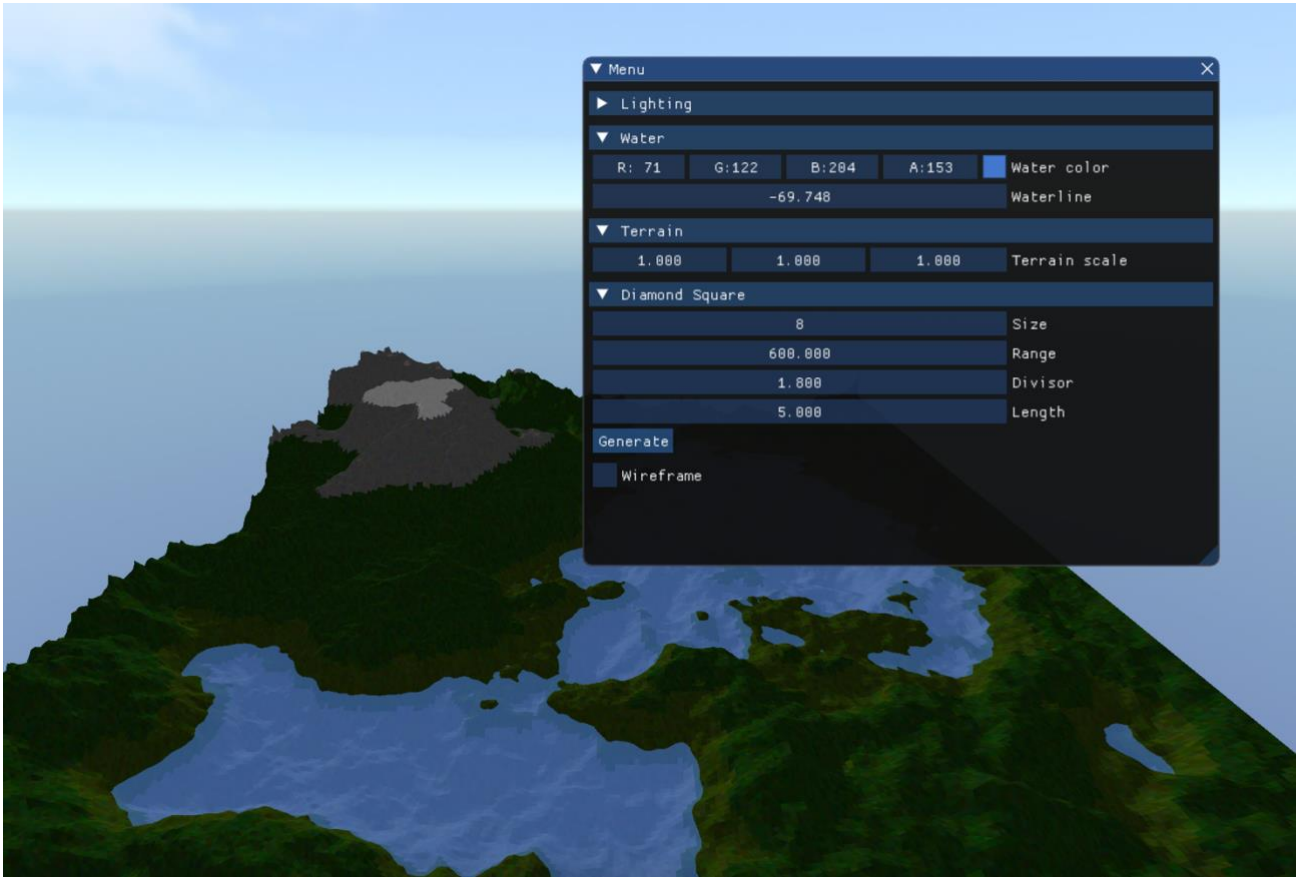

Рисунок 1. Интерфейс приложения.

8 Приложение 3. Освещение ландшафта

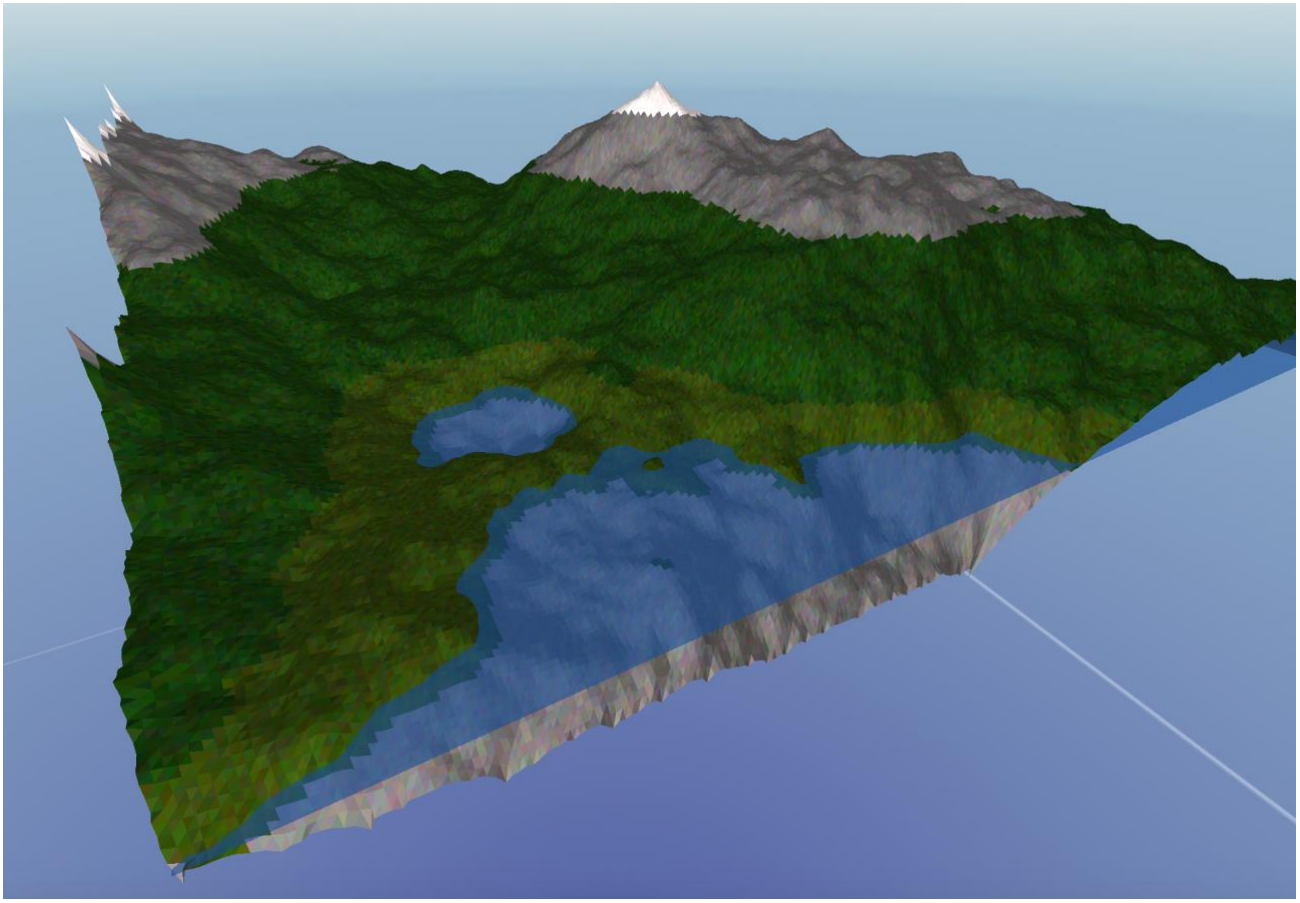

Рисунок 1. Ландшафт с дневным освещением.

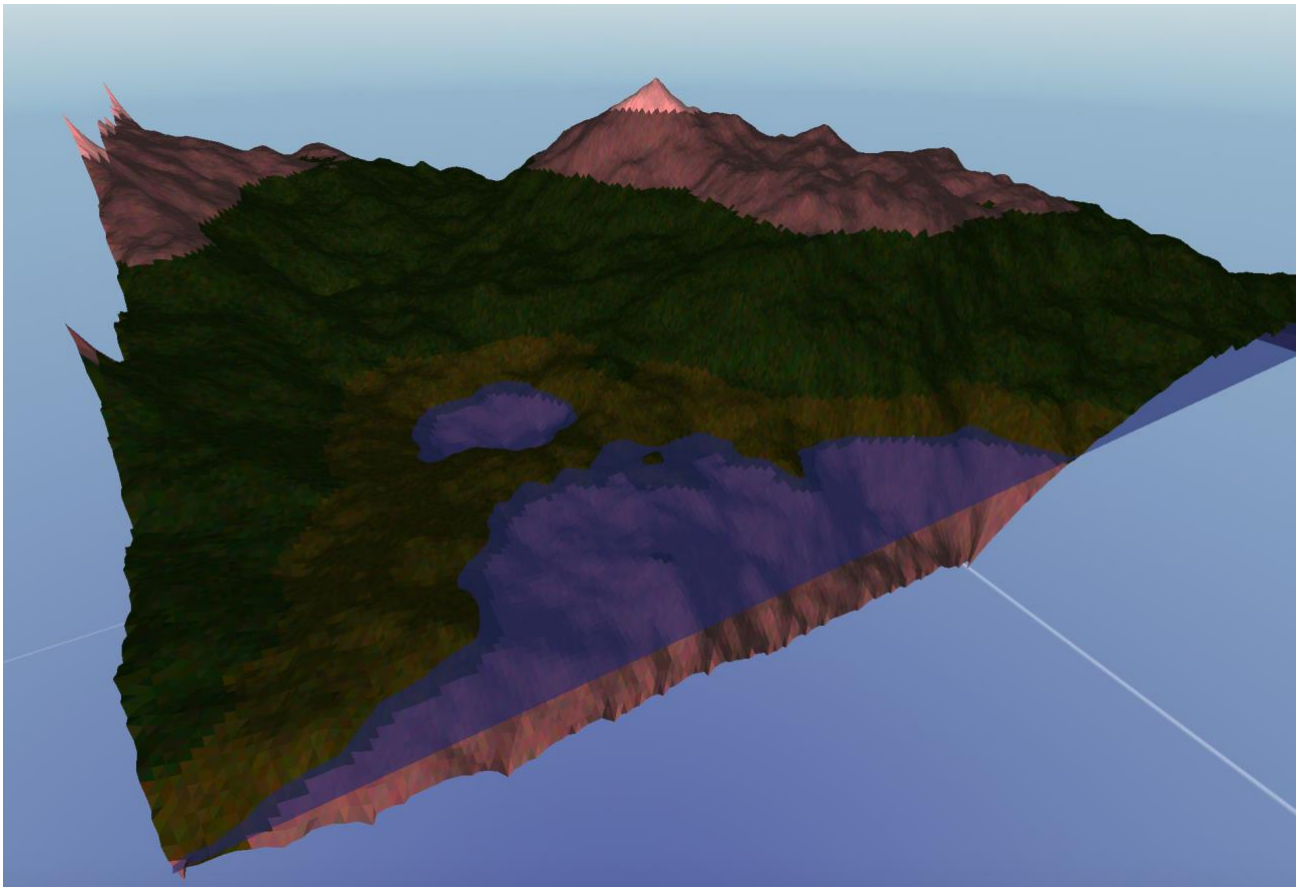

Рисунок 2. Ландшафт с вечерним освещением.