Министерство науки и высшего образования 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 «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

КУРСОВАЯ РАБОТА

НА ТЕМУ:

«Генерация ландшафта»

Студент ИУ7-51Б (группа) _____ (подпись, дата) Н. А. Баранов (И.О. Фамилия)

Руководитель курсового проекта _____ (подпись, дата) А. С. Кострицкий (И.О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Генерация ландшафта – это способ создания ландшафта «без участия человека» [1]. Полученные таким образом объекты могут применяться в различных областях. К примеру, сгенерированные ландшафты использовались в научно фантастических фильмах в неземных пейзажах [2]. Также данному методу находится применение в игровой индустрии. С развитием технологий требования к игровым мирам растут, особенно к их размерам и детализации. Для их создания вручную требуется очень много времени и ресурсов. Генерация ландшафта может если не решить, то серьёзно упростить решение этой задачи. У данного метода есть ещё одно преимущество над ручным созданием – возможность реализации полного редактирования игрового мира. Также сгенерированные ландшафты могут применяться при моделировании водной и термической эрозии [3].

Цель работы: разработать программу для генерации ландшафта. Пользователю должны быть доступны следующие возможности: задание, изменение и сохранение параметров генерации (алгоритм генерации, диапазон высот, дальность и шаг отрисовки), задание и изменение положения источника света (источник света находится на бесконечности, положение задаётся углами), управление положением камеры (перенос, поворот). Исследовать зависимость скорости генерации ландшафта от алгоритма генерации.

1) проанализировать и выбрать алгоритмы для создания изображения и генерации ландшафта;
2) спроектировать программу для генерации и отрисовки ландшафта;
3) выбрать средства реализации спроектированной программы;
4) исследовать зависимость скорости генерации ландшафта от алгоритма генерации.

1 Аналитическая часть

В данном разделе приведено описание объектов сцены, а также анализ существующих алгоритмов для решения поставленных задач, в результате которого будут выбираются наиболее подходящие из них.

1.1 Анализ объектов сцены

Исходя из цели курсовой работы, на сцене должны находиться следующие объекты:

– источник света, расположенный на бесконечности. Для задания вектора направления достаточно 2 угла;

– камера. Задается положением в пространстве, а также 3 векторами, задающими систему координат камеры;

– ландшафт. Задается картой высот, так как одним из параметров генерации является шаг отрисовки. В связи с тем, что одним из параметров генерации является дальность прорисовки, а камера может перемещаться в пространстве, ландшафт должен быть бесконечным. Однако в таком случае его невозможно будет сгенерировать, поэтому вместо этого ландшафт разбивается на множество небольших квадратных участков, чтобы генерировать только в пределах прорисовки. В зависимости от траектории камеры некоторые участки должны генерироваться во время работы программы в произвольный момент времени.

1.2 Анализ алгоритмов генерации ландшафта

1.2.1 Алгоритм diamond-square

Алгоритм diamond-square представляет собой расширенную версию алгоритма midpoint displacement, заключающуюся в использовании двумерной плоскости и наличии 2 этапов [4]. Первый этап – «square» – на данном этапе каждый

элемент массива – ромб, для него определяется центральная точка, для которой считается среднее значение из крайних точек и добавляется рандомное смещение. Второй этап – «diamond» – каждому квадрату в массиве определяется срединная точка, которой устанавливается среднее значение угловых точек и добавляется рандомное смещение. На каждом этапе и при каждой итерации случайное отклонение, которое прибавляется к срединным точкам, уменьшается. На выходе у него получается карта высот, точки в ней расположены по сетке. Таким образом, выходит, что вся плоскость покрыта квадратами. Порядок генерации точек можно увидеть на рисунке 1.1.

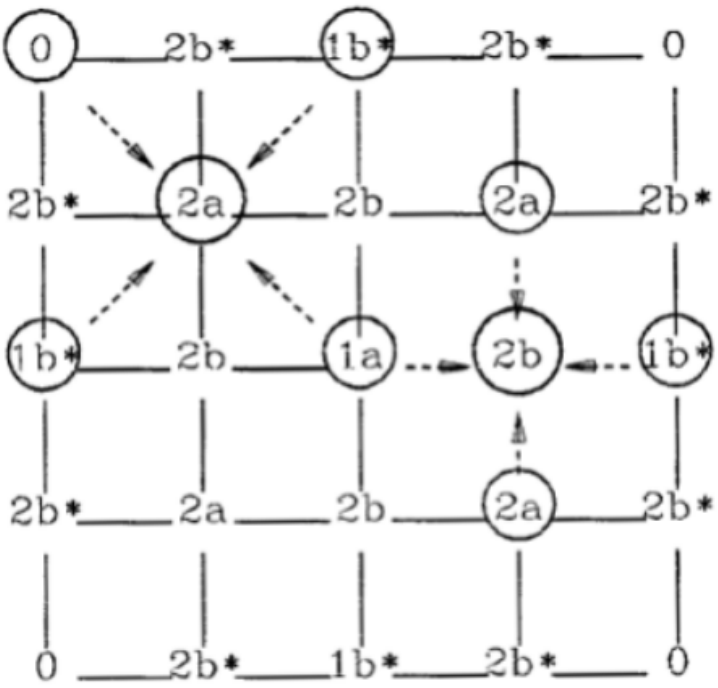

Рисунок 1.1 — Порядок генерации высот в алгоритме diamond-square [5]

Данный алгоритм хорошо подходит для генерации гор [6]. При этом у данного алгоритма есть сложности с генерацией значений на границах квадрата, так как на шаге square не хватает одной точки. Поэтому в данном случае придётся получать значение по двум точкам, из-за чего границы квадратов могут выделяться. Ещё генератор чисел должен выдавать одинаковые значения для одной и той же точки, в противном случае с одним ключом генерации программа будет генерировать разные ландшафты. Задать порядок генерации чисел не получится, он в любом случае будет зависеть от траектории движения камеры. Также сторона квадрата должна иметь размер $2^n + 1$.

1.2.2 Fault алгоритм

Fault алгоритм заключаются в том, что плоскость разделяется на две части случайной прямой [7]. В общем случае, значения высот для точек, оказавшихся с одной стороны от прямой, понижаются, а значения с другой стороны повышаются. Результат алгоритма зависит от количества итераций, которое возможно задать вручную, что продемонстрировано на рисунке 1.2.

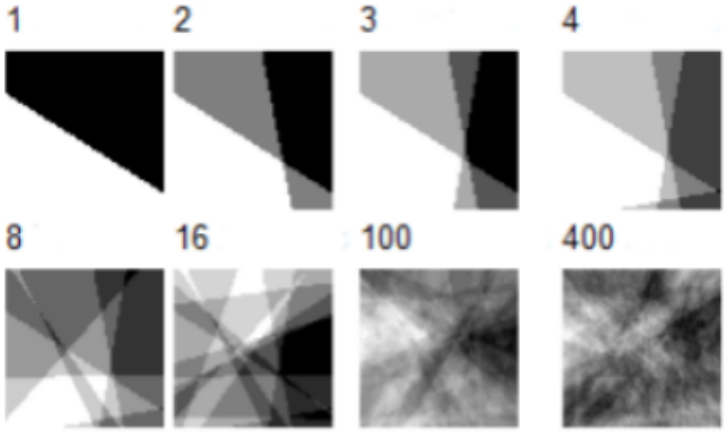

Рисунок 1.2 — Зависимость результата fault алгоритма от количества итераций [5]

Для данного алгоритма также потребуется генератор случайных чисел, который будет выбирать точки внутри квадратов в строго определенно порядке в зависимости от ключа генерации и выбранного квадрата. Один из вариантов – взять генератор случайных чисел от двух точек, значения от которого будут использоваться для создания ключа для генератора чисел для конкретного квадрата, который, в свою очередь, будет использоваться для генерации прямых. Также возникают проблемы со стыковкой соседних квадратов, так как квадраты придётся генерировать независимо друг от друга, и нет никакой гарантии, что на границах высоты совпадут.

1.2.3 Шум Перлина

Шум Перлина – это тип градиентного шума, разработанный Кеном Перлином в 1983 году [8]. Алгоритм имеет множество применений: процедурную

генерацию ландшафта, применение псевдослучайных изменений к переменной и помощь в создании текстур изображений. Чаще всего реализуется в двух, трех или четырех измерениях, но может быть определен для любого количества измерений. Для каждой точки возвращает число в диапазоне от 0 до 1. Результат работы представлен на рисунке 1.3.

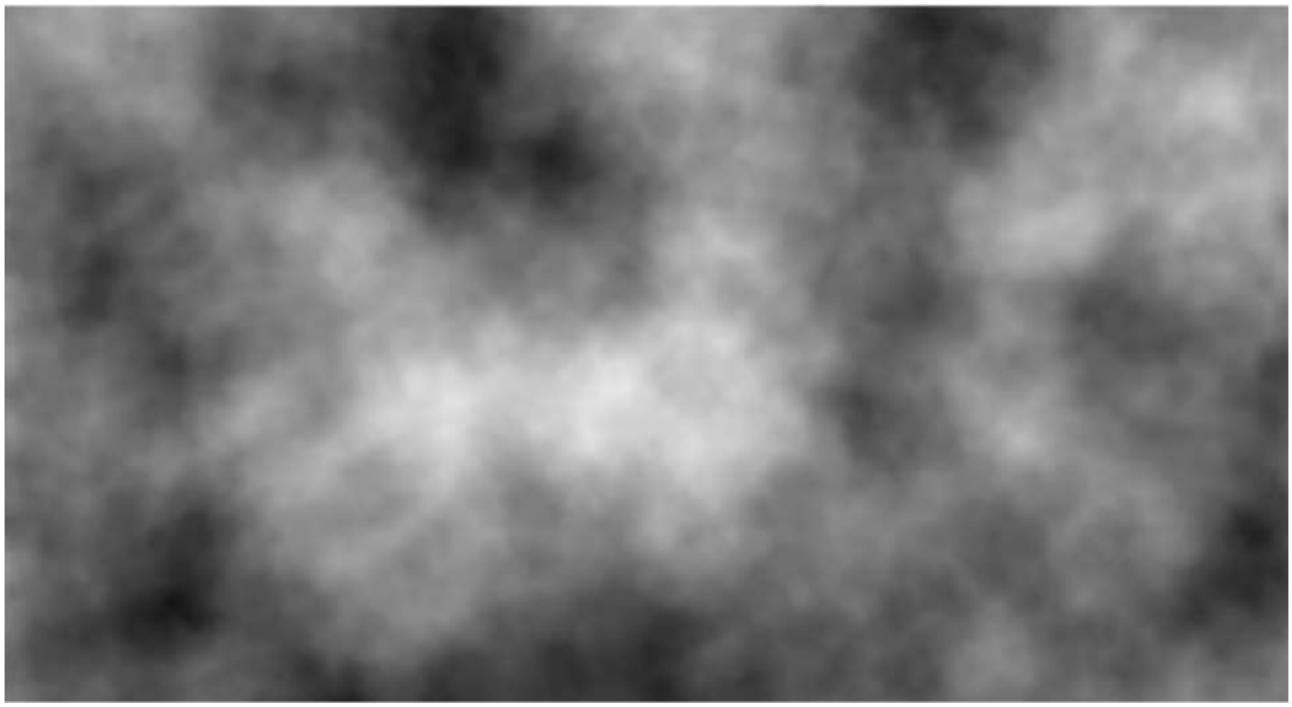

Рисунок 1.3 — Результат работы алгоритма с использованием шума Перлина [8]

Данный алгоритм возвращает одно и то же значение для одной координаты при одном ключе генерации вне зависимости от времени вызова, поэтому ландшафт не будет зависеть от траектории движения камеры. кроме того, благодаря данному свойству значения на границах соседних квадратов будут совпадать. Однако при этом алгоритм не подходит для генерации разных классов ландшафтов, но он может использоваться для, например, размывания различных карт высот при комбинировании с другими алгоритмами [6].

1.2.4 Холмовой алгоритм

Холмовой алгоритм заключается в создании холмов на определенной области [9]. В зависимости от параметров создаваемых холмов можно создать как

скалистую, так и долинистую поверхность. При генерации ландшафт представляется суперпозицией холмовых функций, где каждый холм образуется полусферой [10]. Для достижения реалистичности они иногда генерируются несколькими проходами, высоты холмов на первом проходе соединяются, а карта высот каждого последующего прохода суммируется с картой высот предыдущих. На рисунке reffig:hill показана зависимость результата холмового алгоритма от количества итераций.

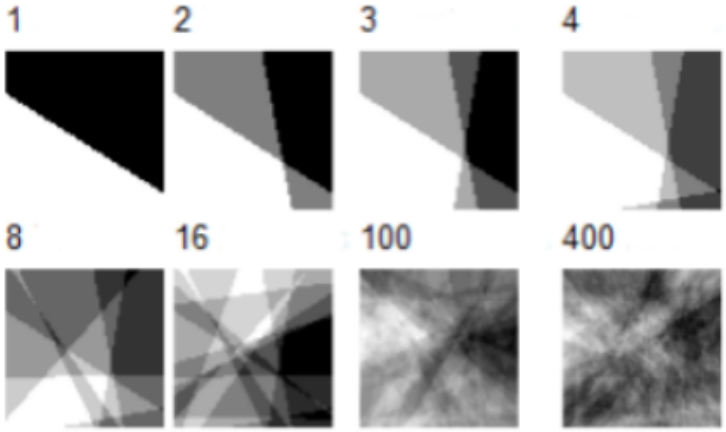

Рисунок 1.4 — Зависимость результата холмового алгоритма от количества итераций [9]

Как и для Fault алгоритма, холмовому алгоритму может потребоваться много итераций для генерации ландшафта. Также в данном случае может потребоваться аналогичная схема с работой генератора случайных чисел. Ещё стоит отметить, что холмы занимают не одну точку на карте и вполне могут попасть на другой квадрат, из-за чего также могут возникнуть проблемы со стыковкой соседних квадратов.

1.2.5 Сравнение алгоритмов

В таблице 1.1 представлены результаты сравнения различных алгоритмов генерации ландшафта. В колонке НГ обозначено отсутствие необходимости в генераторе по 2 числам, в колонке ПС – отсутствие проблем со стыковкой соседних квадратов, в колонке ННИ – отсутствие необходимости проводить много итераций.

Таблица 1.1 — Результаты сравнения алгоритмов генерации ландшафта

№	Название	НГ	ПС	НМИ
1	Diamond-square	-	+	+
2	Fault алгоритм	-	-	-
3	Шум Перлина	+	+	+
4	Холмовой алгоритм	-	-	-

В результате сравнения в качестве алгоритмов генерации ландшафта были выбраны алгоритм diamond—square и шум Перлина, так как они способны генерировать бесконечный ландшафт из-за возможности стыковки соседних квадратов.

1.3 Анализ алгоритмов удаления невидимых линий и поверхностей

1.3.1 Алгоритм Робертса

Данный алгоритм работает в объектном пространстве [11].

Сначала идёт подготовка исходных данных. Для каждого тела строится матрица вида:

$$[V] = \begin{bmatrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{bmatrix}, \tag{1.1}$$

a, b, c, d – коэффициенты уравнения плоскости грани. Предполагается, что точки, лежащие внутри тела, дают отрицательное скалярное произведение со всеми уравнениями плоскостей, то есть нормали расположены наружу. Если это не так, то все коэффициенты для уравнения плоскости умножаются на -1.

Следующим шагом является удаление нелицевых граней. Затем происходит удаление ребер, экранируемых другими телами, а также невидимых ребер,

образованных линиями пересечения тел, связанных отношениями взаимного протыкания.

Время работы алгоритма $O(N^2)$, где N – число ребер.

Ландшафт не является выпуклым телом, поэтому для данного алгоритма его придётся изначально разделить на выпуклые тела.

1.3.2 Алгоритм Варнока

Алгоритм Варнока работает в пространстве изображения [12] В этом пространстве рассматривается окно и решается вопрос о том, пусто ли оно или является его содержимое достаточно простым для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое подокна не станет достаточно просты для визуализации или его размер не достигнет требуемого предела разрешения.

Время работы алгоритма $O(CN)$, где C – количество пикселей в окне, а N – число полигонов на сцене. В худшем случае придётся рассматривать каждый пиксель как отдельное окно.

1.3.3 Алгоритм, использующий Z-буфер

Алгоритм предложен Эдом Кэтмулом и представляет собой обобщение буфера кадра [3]. Обычный буфер кадра хранит коды цвета для каждого пиксела в пространстве изображения. Идея алгоритма состоит в том, чтобы для каждого пиксела дополнительно хранить еще и координату Z или глубину. При занесении очередного пиксела в буфер кадра значение его Z-координаты сравнивается с Z-координатой пиксела, который уже находится в буфере. Если Z-координата нового пиксела больше, чем координата старого, т.е. он ближе к наблюдателю, то атрибуты нового пиксела и его Z-координата заносятся в буфер, если нет, то ни чего не делается.

Время работы алгоритма $O(CN)$, где C – количество пикселей в окне, а N – число полигонов на сцене.

Одним из главных минусов является высокое потребление памяти за счёт создания Z-буфера.

1.3.4 Алгоритм, использующий список приоритетов

Данный алгоритм иногда называют алгоритмом художника [11]. Изначально все полигоны сортируются по глубине или приоритету. Тогда можно записать все элементы в буфер кадра поочередно, начиная с элемента, наиболее удаленного от точки наблюдения. Более близкие к наблюдателю элементы будут «затирать» информацию о более далеких элементах в буфере кадра. Эффекты прозрачности можно включить в состав алгоритма путем не полной, а частичной корректировки содержимого буфера кадра с учетом атрибутов прозрачных элементов.

Время работы алгоритма $O(CN)$, где C – количество пикселей в окне, а N – число полигонов на сцене.

Проблема этого алгоритма заключается в том, что иногда приходится отдельно обрабатывать случаи частичного перекрытия интервалов глубины.

1.3.5 Алгоритм обратной трассировки лучей

В данном алгоритме из точки наблюдателя в каждый пиксель экрана выпускается луч [3]. Затем для каждого объекта определяется, пересекается ли он с лучом или нет. Для упрощения вычислений зачастую используют сферическую или прямоугольную оболочку. После этого для пересекающихся объектов вычисляется координата пересечения. В простейшем случае для непрозрачных поверхностей без отражений и преломлений видимой точкой будет точка с максимальным значением Z-координаты. Для более сложных случаев требуется сортировка точек пересечения вдоль луча.

Время работы алгоритма $O(CN)$, где C – количество пикселей в окне, а N – число полигонов на сцене.

Одним из главных минусов является огромное число вычислений, из-за чего он может работать медленнее остальных алгоритмов.

1.3.6 Сравнение алгоритмов

В таблице 1.2 представлены результаты сравнения алгоритмов удаления невидимых линий и поверхностей. В сложности C обозначает количество пикселей в окне, а N – число полигонов на сцене.

Таблица 1.2 — Сравнение алгоритмов удаления невидимых линий и поверхностей

№	Название	Пространство	Сложность	Примечание
1	Робертса	Объектное	$O(N^2)$	Работает только с выпуклыми объектами
2	Варнока	Изображения	$O(CN)$	В худшем случае придётся рассматривать каждый пиксель как окно
3	Z-буффер	Изображения	$O(CN)$	Потребляет много памяти
4	Художника	Изображения	$O(CN)$	Проблема с перекрывающимися интервалами высот
5	Обратной трассировки лучей	Изображения	$O(CN)$	Большой объем вычислений

В результате сравнения в качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм, использующий Z-буффер, так как потребление большего объема памяти наименьшая из проблем в сравнении с другими алгоритмами.

1.4 Анализ алгоритмов закраски

1.4.1 Простая закраска

В данном случае полигон закрашивается полностью одним цветом [3]. Предполагается, что и источник света находится в бесконечности. На изображении могут быть хорошо заметны резкие перепады интенсивности между различно закрашенными многоугольниками, что сильн влияет на реалистичность изображения.

1.4.2 Закраска Гуро

В данном методе нормали в вершинах многоугольников вычисляются как результат усреднения нормалей ко всем полигональным граням, которым принадлежит данная вершина [3]. Далее с помощью билинейной интерполяции вычисляются значения пикселей на сканирующей строке. У данного метода есть недостаток - в результате его применения может появиться эффект полос Маха.

1.4.3 Закраска Фонга

В данном методе значение нормали в вершинах многоугольников вычисляется так же, как и в закраске Гуро [3]. Здесь так же используется билинейная интерполяция для расчета интенсивности в каждом пикселе сканирующей строки, однако вычисляется не сама интенсивность, а вектор нормали в данной точки, с помощью которого уже и вычисляется интенсивность. Данный алгоритм требует больше вычислительных затрат, однако не устраняет эффект полос Маха, хоть и снижает его.

1.4.4 Сравнение алгоритмов

Среди рассмотренных алгоритмов простая закраска наиболее быстрая, так как не требует никаких вычислений, и допустима, так как по условию единственный источник света находится на бесконечности, однако изображение недостаточно реалистично. Поэтому в качестве алгоритма закраски был выбран метод Гуро из-за большей скорости работы в сравнении с методом Фонга.

1.5 Выводы

В данном разделе были формализованы объекты сцены, а также произведён поиск и выбор подходящих алгоритмов для решения поставленной задачи. Для генерации ландшафта было принято решение использовать алгоритм

diamond-square и шум Перлина. Для удаления невидимых линий был выбран алгоритм Z-буфера, а для закраски граней – метод Гуро.

2 Конструкторская часть

2.1 Используемые типы и структуры данных

Для работы программы потребуются следующие структуры данных:

– четырёхмерный вектор, состоящий из 4 чисел с плавающей точкой. Требуется для преобразования координат точки для последующей отрисовки. Четвёртая координата нужна для получения перспективной проекции;

– трехмерный вектор, состоящий из 3 чисел с плавающей точкой. Требуется для вычисления нормалей, а также для хранения точек;

– матрица размером 4 на 4, состоящая из 16 чисел с плавающей точкой. Требуется для перевода точек в нужный базис;

– камера. Состоит из позиции (трехмерный вектор), базиса (3 трехмерных вектора), фокусного расстояния (число с плавающей точкой) и дальности прорисовки (число с плавающей точкой);

– источник света, состоящий из 2 углов (2 числа с плавающей точкой);

– ландшафт. Состоит из карты высот (двумерный словарь точек, ключами являются числа с плавающей точкой) и похожей карты квадратов. Каждый из квадратов, в свою очередь, является двумерным квадратным массивом точек;

– сцена, содержащая в себе все остальные объекты.

2.2 Структура программы

Функциональная схема работы программы приведена в приложении А.

Также для снижения сложности модификации было принято решение использовать парадигму объектно-ориентированного программирования и патерны проектирования. Диаграмма классов представлена на рисунке 2.1

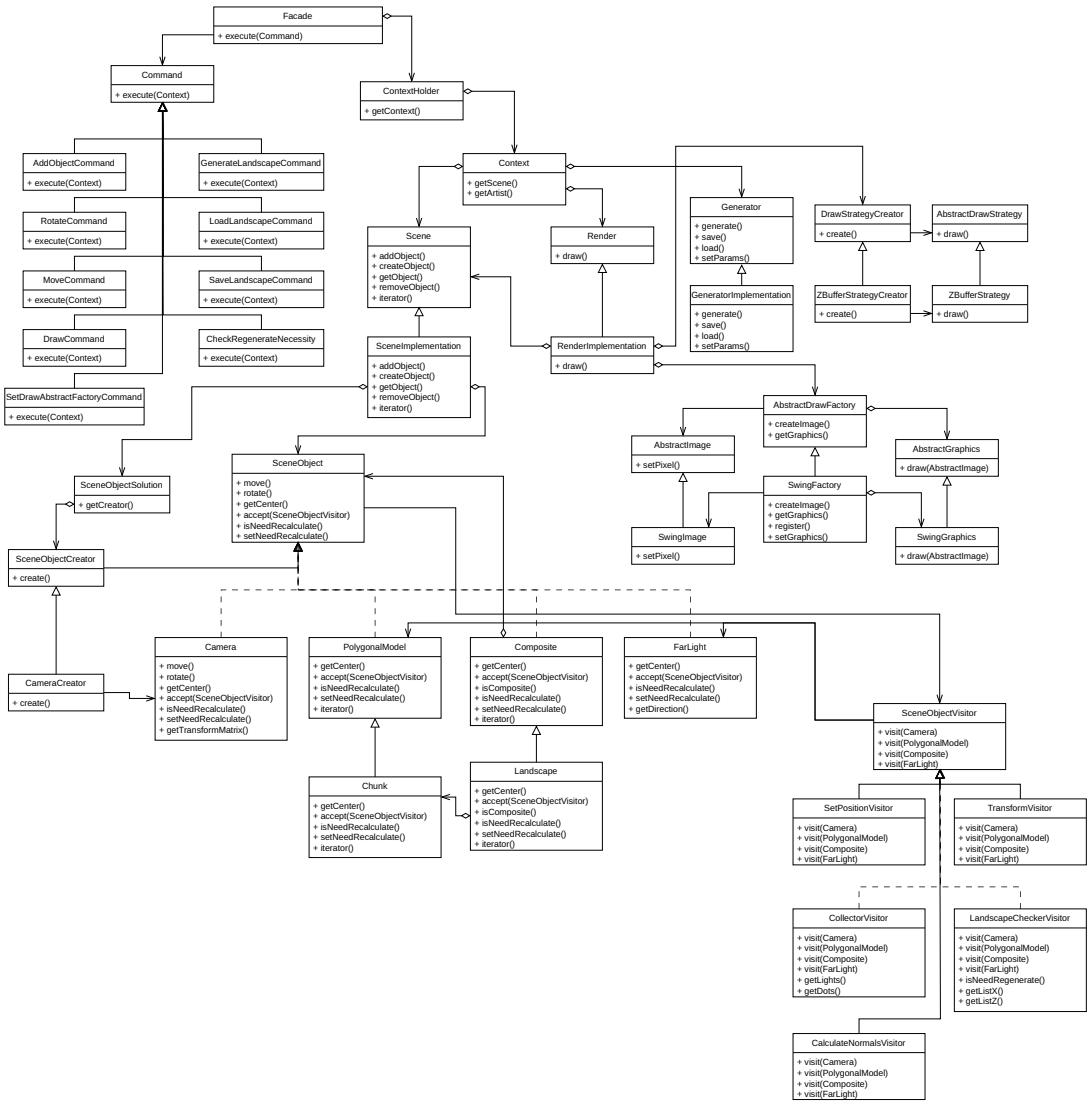

Рисунок 2.1 — Диаграмма классов

2.3 Схемы алгоритмов генерации ландшафта

На вход каждому алгоритму подаётся диапазон высот, дальность и шаг отрисовки, а также положение камеры. Сторону квадрата примем за 1024 из-за ограничений алгоритма diamond-square.

2.3.1 Алгоритм diamond-square

Схема алгоритма представлена на рисунке 2.2.

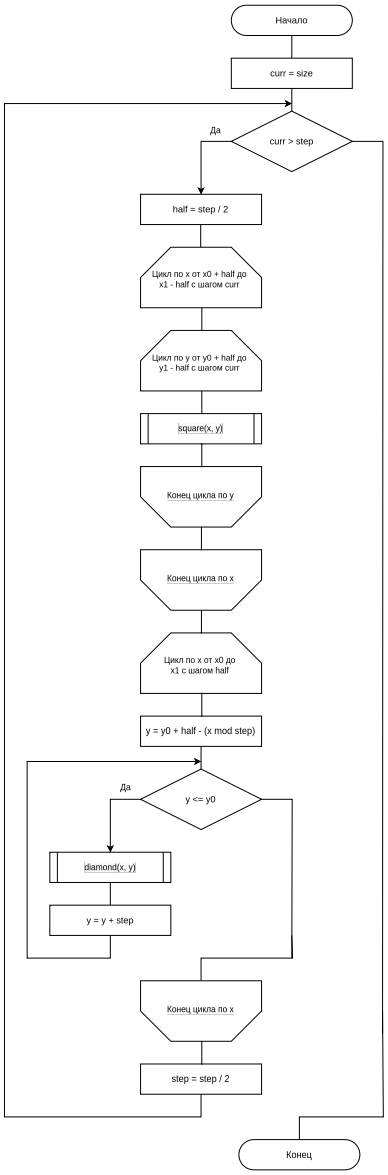

Рисунок 2.2 — Схема алгоритма diamond-square

На шаге $square$ значение в точке (x, y) берется как среднее между значе-

ниями в точках $(x-1, y-1)$, $(x-1, y+1)$, $(x+1, y-1)$, $(x+1, y+1)$. На шаге $diamond$ значение в точке (x, y) берется как среднее между значениями в точках $(x-1, y)$, $(x, y-1)$, $(x+1, y)$, $(x, y+1)$. Также на каждом из шагов к полученному значению прибавляется случайное число, после чего идёт проверка на принадлежность диапазону высот.

2.3.2 Шум Перлина

В отличие от алгоритма diamond-square, в шуме Перлина отсутствует зависимость между точками, поэтому параметры нужны только для настройки генератора. Схема алгоритма поиска значения в точке представлена на рисунке 2.3.

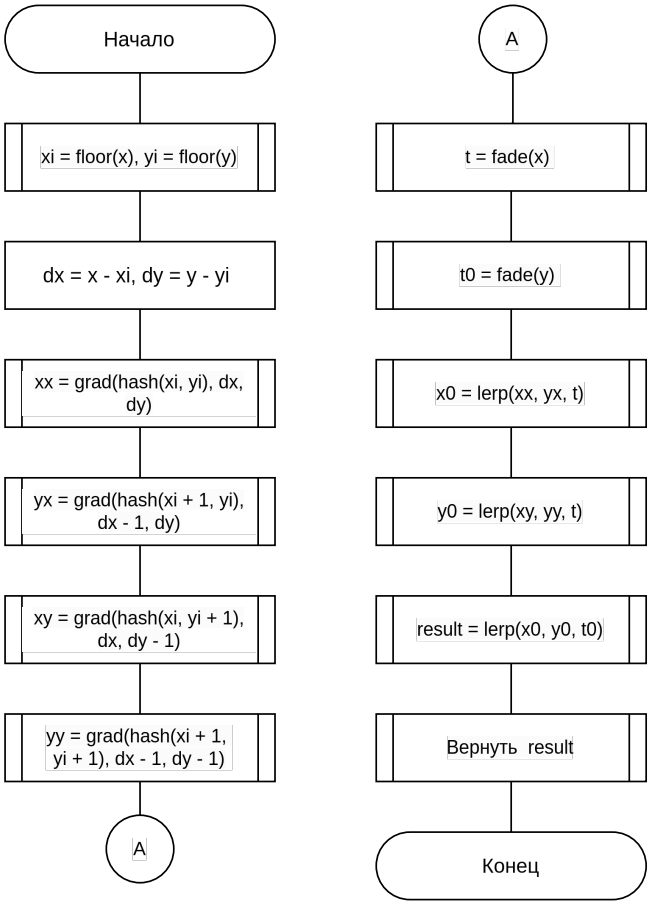

Рисунок 2.3 — Схема алгоритма поиска значения для шума Перлина

Здесь функция $grad$ ищет градиент вектора, $hash$ – значение хеш-функции в точке, $lerp$ – линейная интерполяция, а $fade$ вычисляется по формуле 2.1:

$$fade(t) = t * t * t * (t * (t * 6 - 15) + 10) \tag{2.1}$$

2.4 Построение кадра

В первую очередь для каждой точки вычисляется интенсивность света I по закону Ламберта (формула 2.2):

$$I = I_0 \cos \phi \tag{2.2}$$

где I_0 – интенсивность освещения от источника света, а ϕ – угол между нормалью и вектором направления света.

Далее с использованием паттерна Посетитель обходятся все объекты сцены. Посетитель игнорирует источник света и камеру, а ландшафт воспринимает как паттерн Компоновщик, из-за чего обходит все квадраты, принадлежащие ландшафту. На рисунке 2.4 изображена схема алгоритма преобразования координат перед отрисовкой для каждого квадрата. На вход подаются матрицы преобразования координат $camera\_matrix$ и $frustum\_matrix$, а также фокусное расстояние $focus$.

Матрица $camera\_matrix$ получается по формуле 2.3:

$$camera\_matrix = \begin{pmatrix} VX_x & VX_y & VX_z & 0 \\ VY_x & VY_y & VY_z & 0 \\ VZ_x & VZ_y & VZ_z & focus \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

где VX, VY и VZ – векторы базиса камеры, $focus$ - фокусное расстояние камеры.

Матрица $frustum\_matrix$ получается по формуле 2.4:

$$frustum\_matrix = \begin{pmatrix} -f & 0 & \frac{width}{2} & 0 \\ 0 & -f & \frac{height}{2} & 0 \\ 0 & 0 & \frac{v+f}{v} & -\frac{f(v+f)}{v} \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.4}$$

где f – фокусное расстояние камеры, v – дальность прорисовки, $width$ - ширина экрана, $height$ - высота экрана.

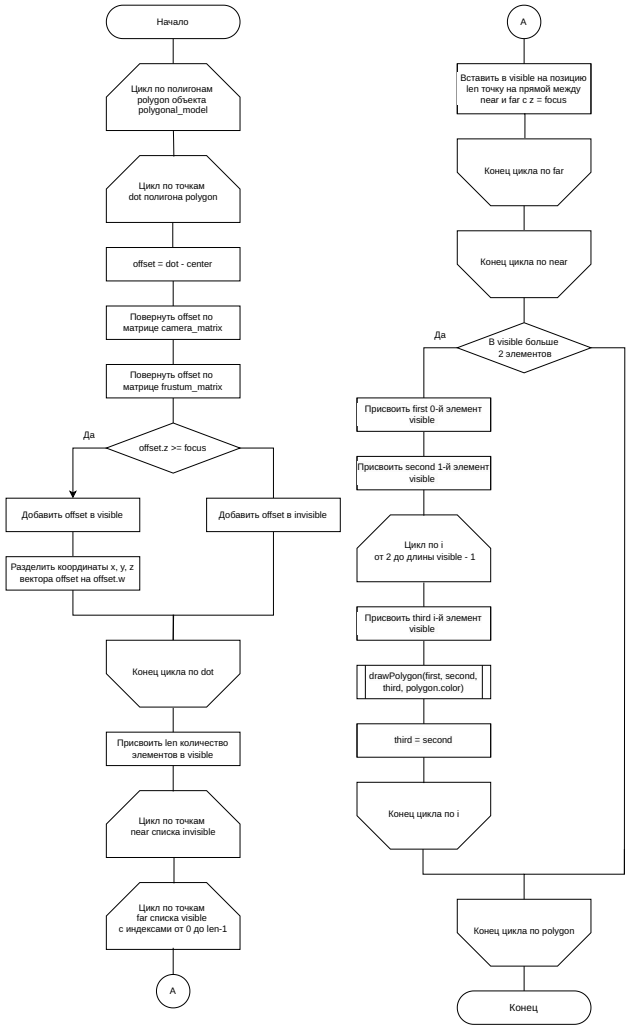

Рисунок 2.4 — Подготовка данных к отрисовке

На рисунке 2.5 изображена схема алгоритма отрисовки полигона. На вход алгоритму передаются 3 преобразованные вершины полигона A, B, и C, интенсивность света в них, цвет полигона $color$, Z-буфер $buffer$.

Координата z произвольной точки D на плоскости ABC вычисляется по формуле 2.5:

$$D_z = \alpha A_z + \beta B_z + \gamma C_z \tag{2.5}$$

где α, β и γ – барицентрические координаты точки D на плоскости ABC.

Интенсивность света I в точке D вычисляется по формуле 2.6:

$$D_I = \alpha A_I + \beta B_I + \gamma C_I \tag{2.6}$$

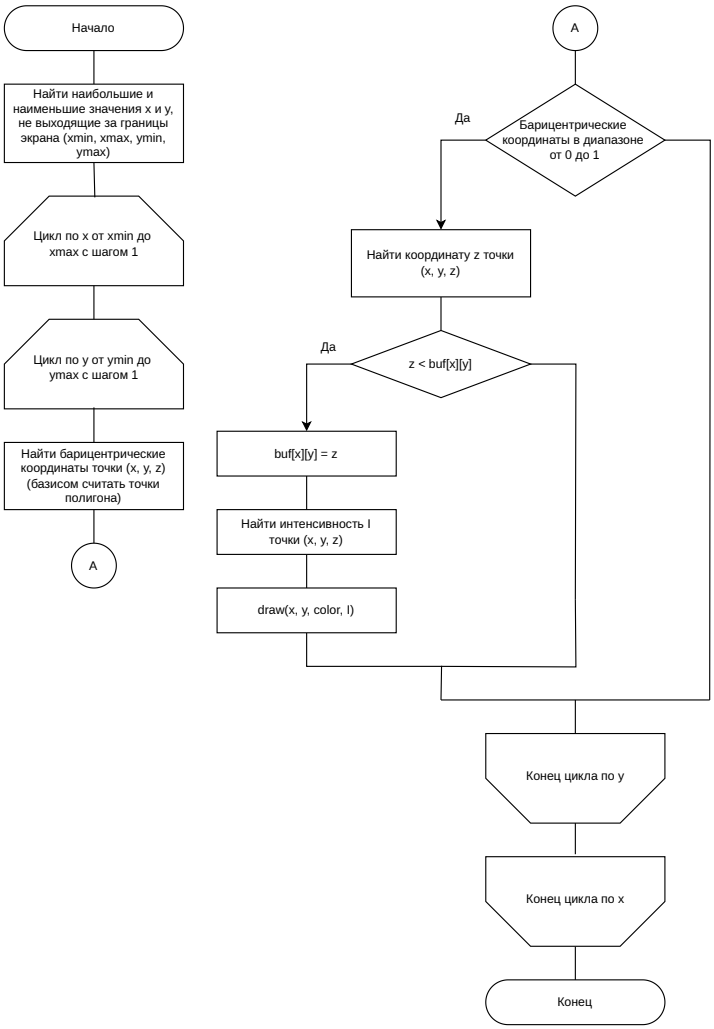

Рисунок 2.5 — Отрисовка треугольного полигона

2.5 Выводы

В данном разделе были выбраны структуры данных, а также построены схемы алгоритмов, функциональная схема работы программы и диаграмма классов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. В. Барсуков А. Исследование популярных алгоритмов процедурной генерации ландшафтов на возможность усовершенствования. Альманах научных работ молодых учёных XLIX научной и учебно-методической конференции Университета ИТМО. Том 3, 2020. — С. 30.

2. Херн Дональд Бейкер Паулин М. Компьютерная графика и стандарт OpenGL. 2005. — С. 247–254.

3. Основные алгоритмы компьютерной графики [Электронный ресурс]. Режим доступа: `http://bourabai.bladeweb.org/graphics/02.htm` (Дата обращения: 21.09.2024).

4. А.Д. Мультан Р.И. Мазур. Процедурная генерация текстур на основе алгоритма diamond-square. 54-я научная конференция аспирантов, магистрантов и студентов БГУИР, 2018. — С. 66–67.

5. Р.М. Гимранова. Методы генерации ландшафта карт в разработке игр на примере Unreal Engine 4. 2020. — С. 21–22.

6. И.В. Шишлянников. Анализ и применение алгоритмов генерации и моделирования ландшафта. 78-я научная конференция аспирантов, магистрантов и студентов Белорусского государственного университета, 2021. — С. 107–111.

7. Кинтонова А.Ж. Габдрешов Г.Е. Кисманова А.А. Сансызбай К.М. Проектирование ландшафта с помощью алгоритмов генерации. Вестник КазАТК No 1 (124), 2023. — С. 182–194.

8. Е.А. Ульянов. Реализация простого алгоритма шума Перлина на C++. 2024.

9. С.В. Колесников. О решении задачи генерации ландшафтов. Решетневские чтения, 2013. — С. 214–215.

10. А.С. Аникеев. Генерация и моделирование 2D ландшафта по контрольным значениям с использованием Unity на языке программирования C♯. Инженерный вестник Дона №4, 2020.

11. А.А. Головонин. Базовые алгоритмы компьютерной графики. Проблемы качества графической подготовки студентов в техническом вузе: традиции и инновации, 2016. — С. 13–30.

12. В.А. Провкин. Построение реалистичных изображений при помощи алгоритма Варнока и сравнение эффективности различных его реализаций. 2017. — С. 9–10.

Приложение А: Функциональная схема работы программы

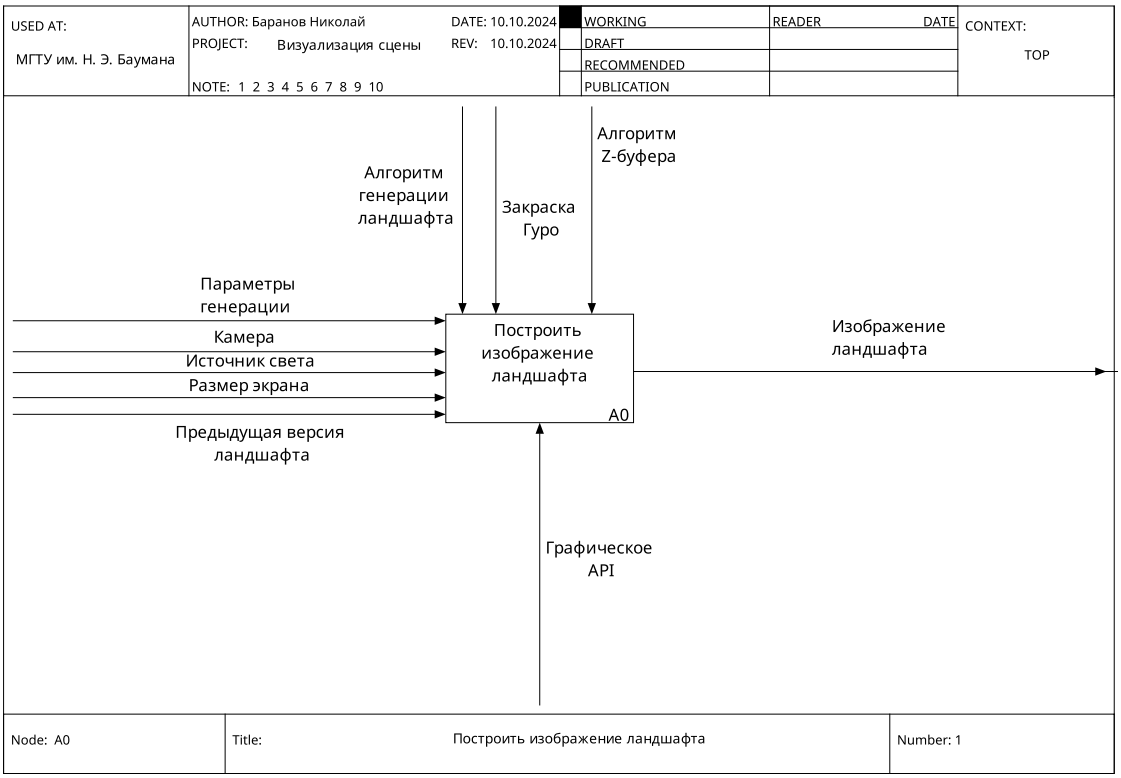

Рисунок А.1 — Верхний уровень функциональной схемы

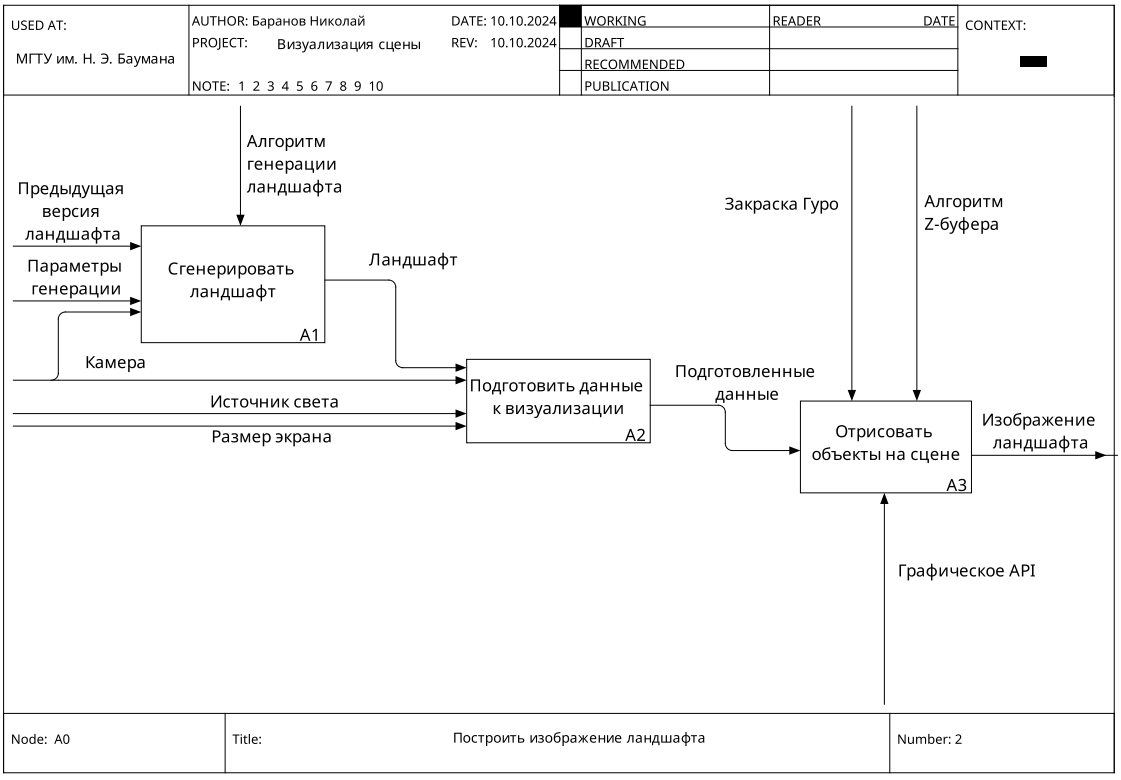

Рисунок А.2 — Декомпозиция уровня А0

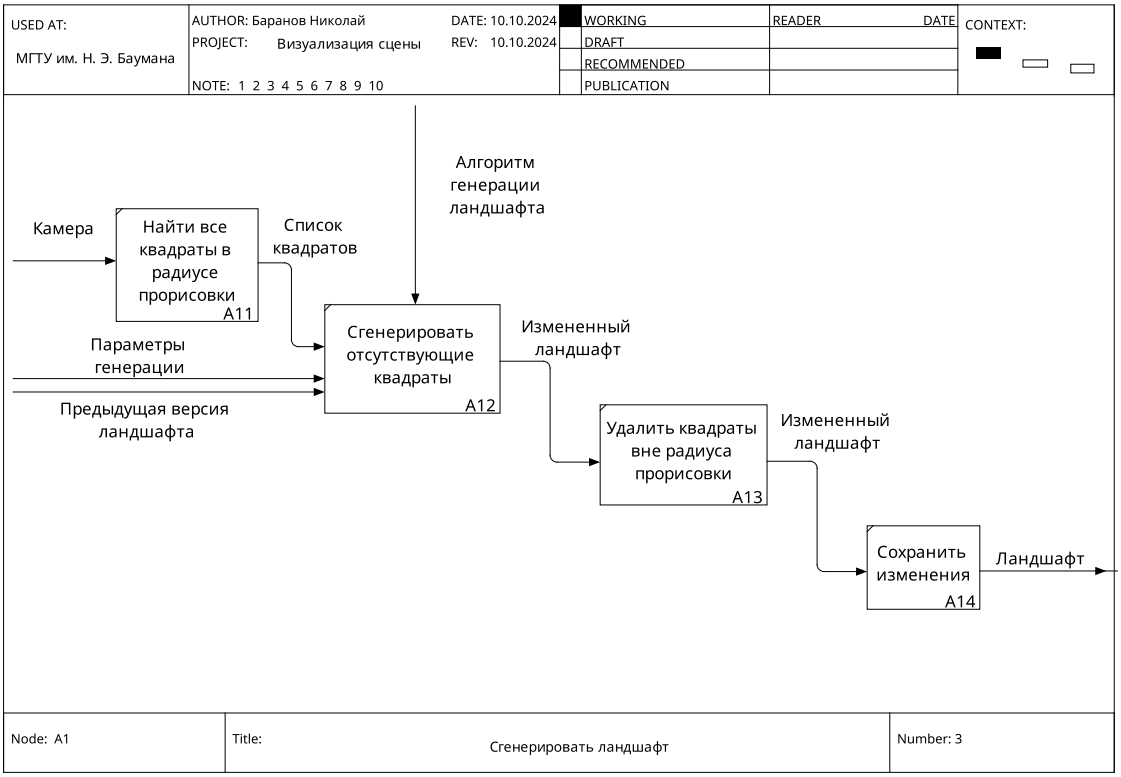

Рисунок А.3 — Декомпозиция уровня А1

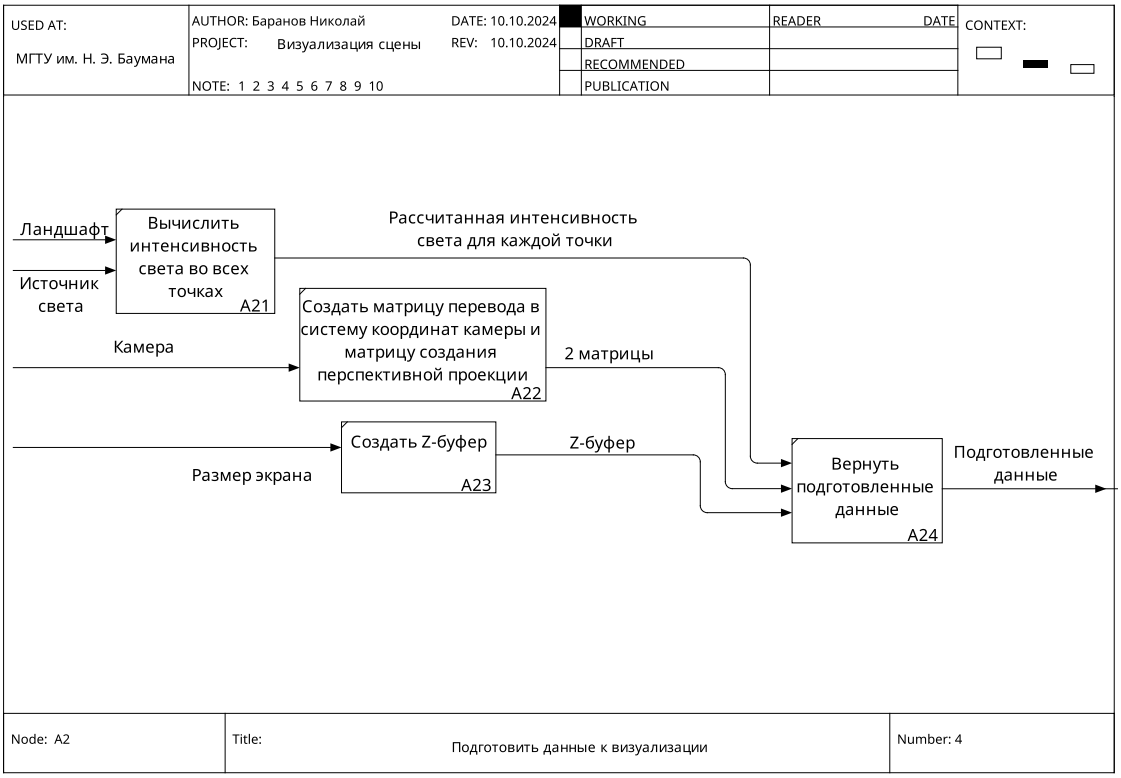

Рисунок А.4 — Декомпозиция уровня А2

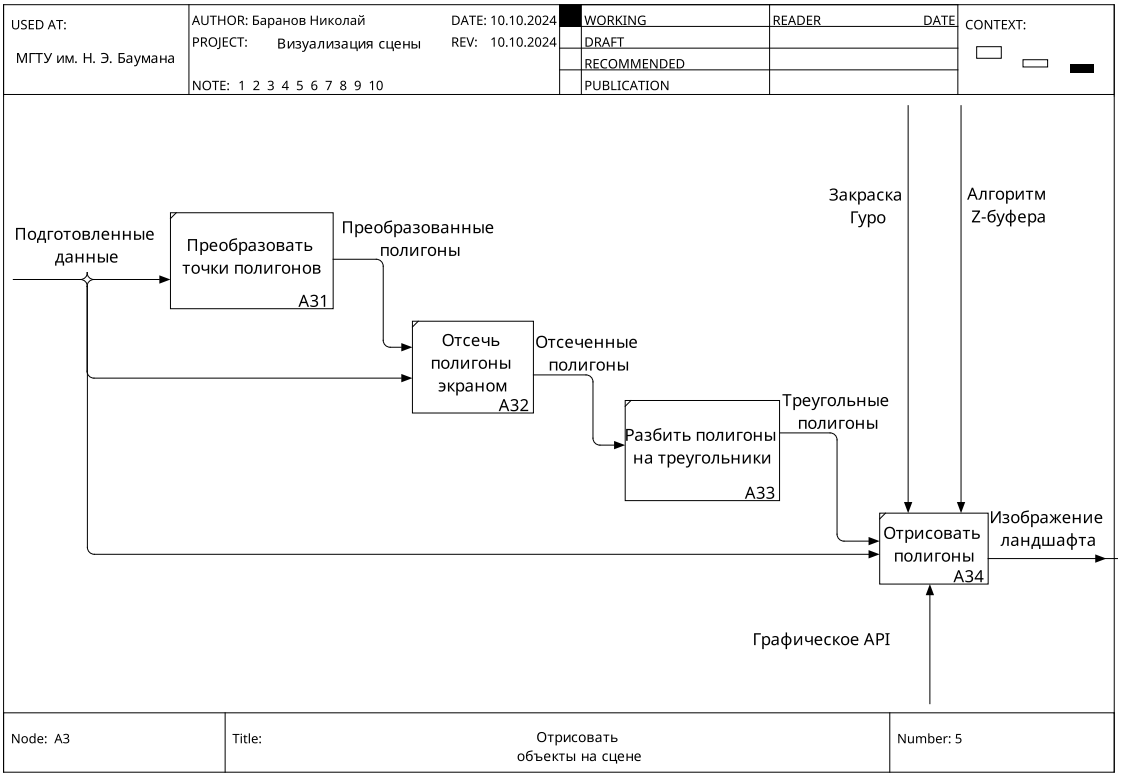

Рисунок А.5 — Декомпозиция уровня А3