РЕФЕРАТ

Расчетно-пояснительная записка 51 с., 16 рис., 4 лист., 6 ист., 1 прил.

КОМПЬЮТЕРНАЯ ГРАФИКА, АЛГОРИТМЫ УДАЛЕНИЯ НЕВИДИМЫХ ЛИНИЙ, Z-БУФЕР, ЗАКРАСКА, ОСВЕЩЕНИЕ, СЦЕНА, НАЛОЖЕНИЕ ТЕКСТУР, ПРОЦЕДУРНОЕ ТЕКСТУРИРОВАНИЕ, ПРОЕКТИВНОЕ ТЕКСТУРИРОВАНИЕ, НЕРОВНОСТЬ

Целью работы является разработка программного обеспечения для наложения текстур на трёхмерные объекты. Для визуализации сцены использовался алгоритм с Z-буфером, а для представления объекта использовалась поверхностная модель, представленная в виде списка ребер.

В процессе работы были проанализированы различные алгоритмы, методы представления, закраски геометрических моделей на сцена. Выбраны технологии решения для поставленной задачи, а также разработаны алгоритмы для их программной реализации. Разработана программа, предназначенная для наложения текстур на трёхмерные объекты.

Проведено исследование быстродействия программы при различном количестве создания объектов на сцене. Также с фиксированным количеством объектов, но с учётом текстуры. Из результатов исследования следует, что время отрисовки сцены увеличивается как при увеличении количества объектов на сцена, так и при выборе текстуры.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире компьютерная графика окружает человека практически везде. Прежде всего стал популярен синтез изображений, так как с помощью него разработчики могут создавать компьютерные игры, спецэффекты в кино, виртуальную реальность. Компьютерная графика используется в науке и промышленности для визуализации и моделирования различных процессов.

В мире компьютерной графики, одним из ключевых аспектов, влияющих на визуальное восприятие и качество изображений, является процесс наложения текстур. Эта техника позволяет придавать поверхностям и объектам на экране уникальный внешний вид, подчеркивать их характер и структуру, а также сделать изображения более реалистичными и привлекательными для наблюдателя.

Целью данной курсовой работы является разработка программного обеспечения для наложения текстур на трёхмерные объекты.

Для достижения поставленной цели необходимо решить следующие задачи:

— провести анализ существующих алгоритмов компьютерной графики, используемых для создания трехмерных сцен;
— выбрать алгоритмы для решения поставленной задачи;
— выбрать язык программирования и среду разработки для реализации поставленной задачи;
— создать программное обеспечение, реализующее выбранные алгоритмы;
— провести замеры временных характеристик разработанного программного обеспечения.

1 Аналитическая часть

В этом разделе будет приведено описание модели трёхмерного объекта. Также проводится анализ существующих алгоритмов построения трехмерных изображений и выбираются алгоритмы для решения поставленной задачи.

1.1 Формализация объектов синтезируемой сцены

Сцена состоит из определенного набора объектов.

1) Стандартный геометрический объект — представляется в виде полигональной сетки. В число рассматриваемых геометрических объектов входят куб, сфера, цилиндр.
2) Источник света, который представляет собой материальную точку, испускающую лучи света во все стороны.
3) Камера характеризуется своим пространственным положением и направлением просмотра.

1.2 Способы описания трехмерных геометрических моделей на сцене

В компьютерной графике для описания трехмерных геометрических объектов существуют три типа моделей: каркасная, поверхностная и объёмная [1].

— *Каркасная модель* — задается информация о вершинах и рёбрах объектов. Это простейший вид моделей, так как задается минимум информации. Однако данный вид представления объектов не всегда корректно передает форму объекта.

— *Поверхностная модель* часто используется в компьютерной графике, кроме информации о вершинах и ребрах, содержит еще информацию о поверхности. Недостатком поверхностной модели является отсутствие информации о том, с какой стороны поверхности находится материал.
— *Твердотельная модель* отличается от поверхностной тем, что в данной модели к информации о поверхностях добавляется информация о том, с какой стороны расположен материал. Это достигается путем указания направления внутренней нормали.

Для решения поставленной задачи не подойдет каркасная модель, так как такая модель не позволяет получить реалистичное изображение. Твердотельная модель также не подойдет, так как по поставленной задачи нет необходимости знать из какого материала будет выполнен тот или иной объект и с какой стороны расположен материал. Поэтому выбор остается лишь поверхностной модели.

1.3 Способы задания поверхностных моделей

Поверхностная модель задается следующими способами [2]:

— *параметрический способ* — заключается в том, что для получения поверхности нужно дополнительно вычислять функцию, зависящую от параметра.
— *полигональная сетка* — характеризуется совокупностью вершин, ребер и граней, определяющих форму объекта в трехмерном пространстве.

1.4 Способ хранения полигональной модели

Рассмотрим существующие способы хранения информации о полигональной сетке.

— *Вершинное представление* — описывает объект множеством вершин, соединенных с другими вершинами. Информация о ребрах и гранях неявно присутствует в представлении из-за чего для восстановления исходного тела необходимо обойти все вершины и составить списки граней.

— *Список граней* — представляет объект не только множеством вершин, но граней.

Стоит отметить, что одним из решающих факторов в выборе способа задания модели в данной работе является скорость выполнения преобразований над объектами сцены. Поэтому наиболее удобным представлением является модель, заданная полигональной сеткой. Такая модель позволит избежать проблем при описании сложных объектов сцены. Способом хранения полигональной сетки был выбран список граней, так как это даст явное описание граней. Этот способ позволит эффективно преобразовывать модели, так как структура будет включать в себя список вершин.

1.5 Анализ алгоритмов удаления невидимых линий и поверхностей

1.5.1 Алгоритм Варнока

Алгоритм Варнока работает в пространстве изображений. В основу алгоритма положен принцип «разделяй и властвуй», состоящий в разбиении

области рисунка на более мелкие подобласти (окна). Единой версии этого алгоритма не существует. Но все модификации основываются на рекурсивном разбиении окна изображения. Для каждого окна определяются многоугольники, которые связаны с ним, и те, у которых легко определить видимость, изображаются. Если нет, то разбиение на подокна продолжается до тех пор, пока либо нельзя будет принять однозначное решение, либо размер окна не станет равным одному пикселю. На каждом этапе разбиения осуществляется определение расположения многоугольников относительно текущего окна. Насколько быстро будет работать данный алгоритм, зависит от сложности сцены. Так как было решено использовать полигональную сетку, то это может затормозить выполнение алгоритма.

Приемущества:

— меньшие затраты по времени в случае области, содержащий мало информации.

Недостатки:

— алгоритм работает рекурсивно;

— алгоритм работает только в пространстве изображений;

— большие затраты по времени в случае области с высоким информационным содержимым.

1.5.2 Алгоритм, использующий Z-буфер

Алгоритм Z-буфера работает в пространстве изображения.

Буфер кадра (регенерации) используется для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения [2]. Для него требуется буфер регенерации, в котором запоминаются значения яркости, а также Z-буфер (буфер глубины), куда можно помещать информацию о координате z для каждого пикселя. Сначала в Z-буфер заносятся минимально

возможные значения z, а буфер регенерации заполняется значениями пикселя, описывающими фон. Затем каждый многоугольник преобразуется в растровую форму и записывается в буфер регенерации, при этом не производится начального упорядочения. В процессе работы глубина (значение координаты Z) каждого нового пикселя, который надо занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесён в Z-буфер. Если это сравнение показывает, что новый пиксель расположен ближе к наблюдателю, чем пиксель, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра. Кроме того, производится корректировка Z-буфера: в него заносится глубина нового пикселя. Если же глубина (значение координаты Z) нового пикселя меньше глубины хранящегося в буфере, то никаких действий производить не надо.

На рисунке 1 представлен пример работы алгоритма Z-буфера.

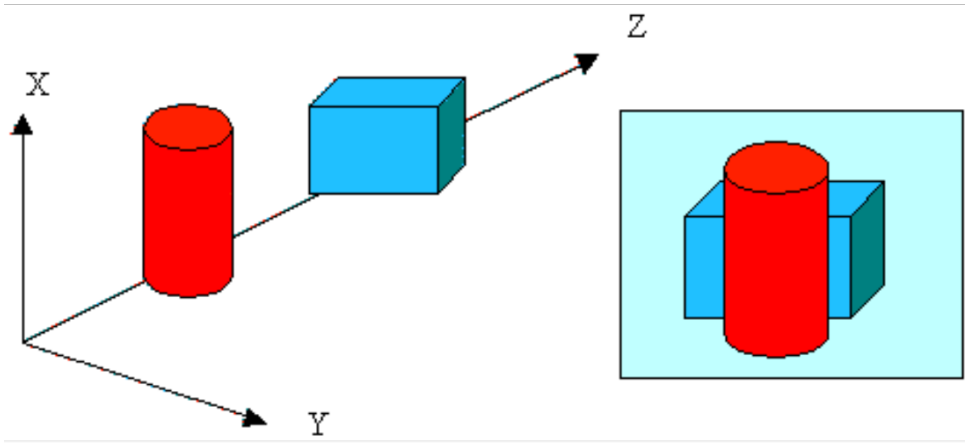

Рисунок 1 – Алгоритм с Z-буфером [3]

Приемущества:

— простота реализации;

— оценка трудоемкости линейна;

— отсуствие сортировок.

Недостатки:

— большой объем требуемой памяти;

— эффекты прозначности и просвечивания тяжело реализовать.

1.5.3 Алгоритм обратной трассировки лучей

Данный алгоритм работает в пространстве изображения. Основная идея заключается в том, что для каждого пиксела на дисплее проводится прямой луч от наблюдателя до элемента сцены [2]. Первое пересечение используется для определения цвета пиксела как функции пересекаемой поверхности элемента. Также проводятся вторичные лучи от точек пересечения до разных источников света для определения освещённости пиксела. Если эти лучи блокируются объектом, то данная точка находится в тени, которую отбрасывает рассматриваемый источник света. Иначе источник света влияет на освещение. Совокупность всех вторичных лучей, которые достигают источника света, определяет качество освещения, которое попадает на объект сцены. Также для более реалистичного изображения необходимо проводить лучи отражения и лучи преломления.

Приемущества:

— высокая реалистичность синтезируемого изображения;

— возможность использования в параллельных вычислительных системах.

Недостатки:

— производительность.

На рисунке 2 представлен пример работы алгоритма обратной трассировки лучей.

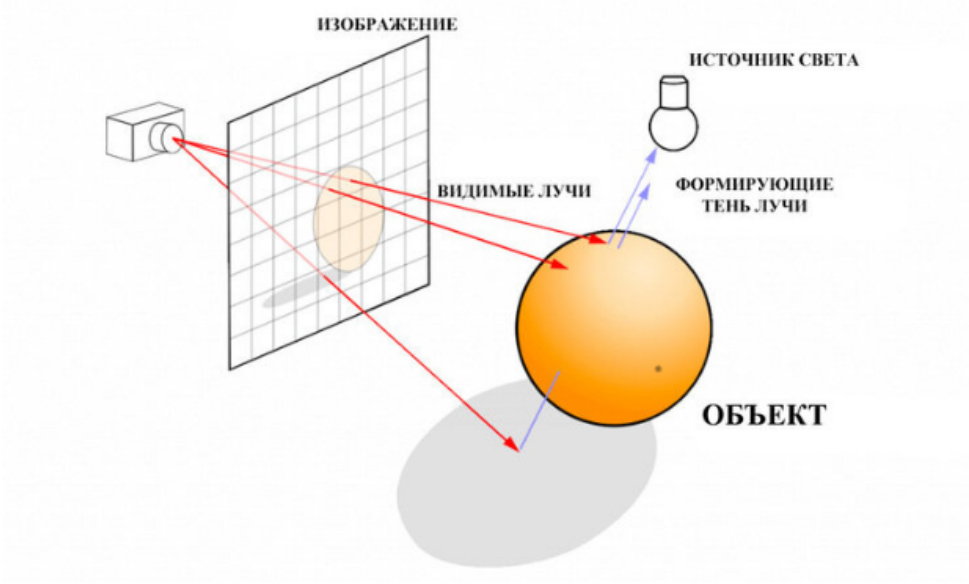

Рисунок 2 – Алгоритм обратной трассировки лучей [4]

1.5.4 Выбор алгоритма удаления невидимых линий и поверхностей

При реализации необходимо обеспечить плавную смену кадров при перемещении камеры, поэтому алгоритм должен иметь минимальную зависимость трудоемкости алгоритма от числа объектов на сцене и использование рекурсивных вызовов.

Таким образом, подходящий алгоритм является алгоритм с Z-буфером, который подходит для решения создания сцены с различным количеством объектов, что позволит иметь динамическую сцену. Данный алгоритм имеет линейную зависимость от числа объектом, что приведет к оптимальной работе программы. Также он простой в реализации и используется в большинстве графических движков, что приведет к быстрой скорости реализации алгоритма.

1.6 Анализ алгоритмов закраски

1.6.1 Простая закраска

Один из самых быстрых алгоритмов. В его основе лежит закон Ламберта, который говорит о том, что интенсивность отражённого света пропорциональна косинусу угла между направлением света и нормалью к поверхности. Данный метод закраски обладает большим быстродействием, однако все пиксели грани имеют одинаковую интенсивность, и сцена выглядит нереалистично. Тем не менее, этот метод крайне прост в реализации и совершенно не требователен к ресурсам.

1.6.2 Закраска по Гуро

Метод Гуро устранить дискретность изменения интенсивности и создать иллюзию гладкой криволинейной поверхности. Он основан на интерполяции интенсивности.

Данный метод отличается от простой закраски тем, что разные точки грани закрашиваются с разными значениями интенсивности. Для это в каждой вершине грани находится вектор нормали и вычисляется значение интенсивности. Затем найденные значения интенсивности интерполируются по всем точкам примыкающих граней.

Закрашивание граней по методу Гуро осуществляется в четыре этапа.

1) Вычисляются нормали к каждой грани.
2) Определяются нормали в вершинах. Нормаль в вершине определяется усреднением нормалей примыкающих граней.
3) На основе нормалей в вершинах вычисляются значения интенсивностей

в вершинах согласно выбранной модели отражения света.

4) Закрашиваются полигоны граней цветом, соответствующим линейной интерполяции значений интенсивности в вершинах.

Метод Гуро применим для небольших граней, расположенных на значительном расстоянии от источника света. Если же размер грани большой, то расстояние от источника света до центра будет меньше, чем до вершин, и центр грани должен выглядеть ярче, чем ребра. Но из-за линейного закона, используемого в интерполяции, метод не позволяет это сделать, поэтому появляются участки с неестественной освещенностью

1.6.3 Закраска по Фонгу

Закраска Фонга по своей идее похожа на закраску Гуро, но ее отличие состоит в том, что в методе Гуро по всем точкам полигона интерполируется значения интенсивностей, а в методе Фонга – вектора нормалей, и с их помощью для каждой точки находится значение интенсивности [2].

Этапы следующие.

1) Определяются нормали к граням.
2) По нормалям к граням определяются нормали в вершинах. В каждой точке закрашиваемой грани определяется интерполированный вектор нормали.
3) По направлению векторов нормали определяется цвет точек грани в соответствии с выбранной моделью отражения света.

1.6.4 Выбор алгоритма закраски

Алгоритм закраски Фонга требует большего числа вычислений по сравнению с другими. Так как в данной курсовой работе будут использоваться полигональные сетки и, желательно, чтобы были сглажены границы многоугольников, то лучше всего подойдёт алгоритм закраски по Гуро.

1.7 Анализ моделей освещения

Все модели освещения делятся на два вида: глобальные и локальные. Глобальные модели учитывают возможности отражения и преломления света от объектов, не являющихся прямыми источниками освещения, поэтому они требуют значительных затрат.

Существуют более простые, локальные модели освещения, которые учитывают только свет от источника. Они вычисляют интенсивность, цвет и дальнейшее распределение света на поверхности, но не учитывают перенос света между объектами.

1.7.1 Модель Ламберта

В этой модели воспроизводится идеальное диффузное освещение. Считается, что свет при попадании на поверхность рассеивается равномерно во все стороны. На рисунке 3 показано, что согласно этой модели, освещенность в точке определяется только плотностью света в точке поверхности, а она линейно зависит от косинуса угла падения. При этом положение наблюдателя не имеет значения, т.к. диффузно отраженный свет рассеивается равномерно по всем направлениям.

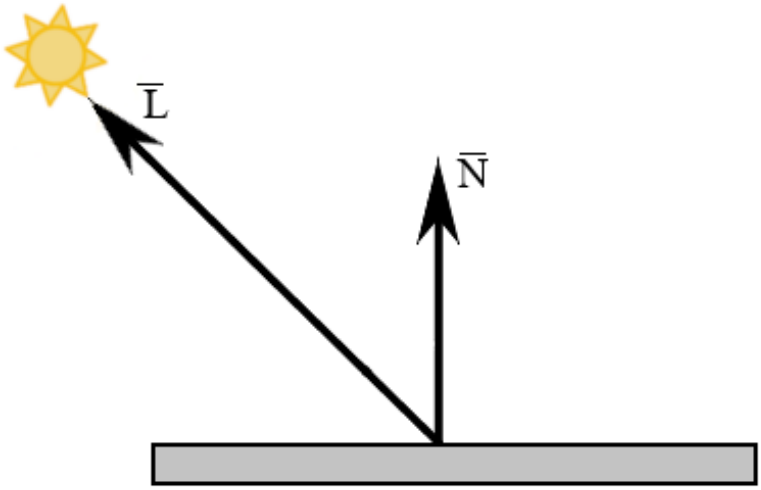

Рисунок 3 – Модель освещения Ламберта

Пусть:

— $\overrightarrow{L}$ — вектор от точки до источника;

— $\overrightarrow{N}$ — вектор нормали;

— I — результирующая интенсивность света в точке;

— I_0 — интенсивность источника;

— K_d — коэффициент диффузного освещения.

Формула расчета интенсивности имеет следующий вид:

$$I = I_0 \cdot K_d \cdot cos(\overrightarrow{L}, \overrightarrow{N}) = I_0 \cdot K_d \cdot (\overrightarrow{L}, \overrightarrow{N}) \quad (1)$$

Из формулы (3) следует главный недостаток модели Ламберта – одинаковая интенсивность во всех точках, принадлежащих одной грани.

Модель Ламберта является одной из самых простых моделей освещения. Данная модель очень часто используется в комбинации других моделей, практически в любой другой модели освещения можно выделить диффузную составляющую.

1.7.2 Модель Фонга

Модель Фонга – классическая модель освещения. Модель представляет собой комбинацию диффузной составляющей (модели Ламберта) и зеркальной составляющей и работает таким образом, что кроме равномерного освещения на материале может еще появляться блик. Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части как показано на рисунке 4.

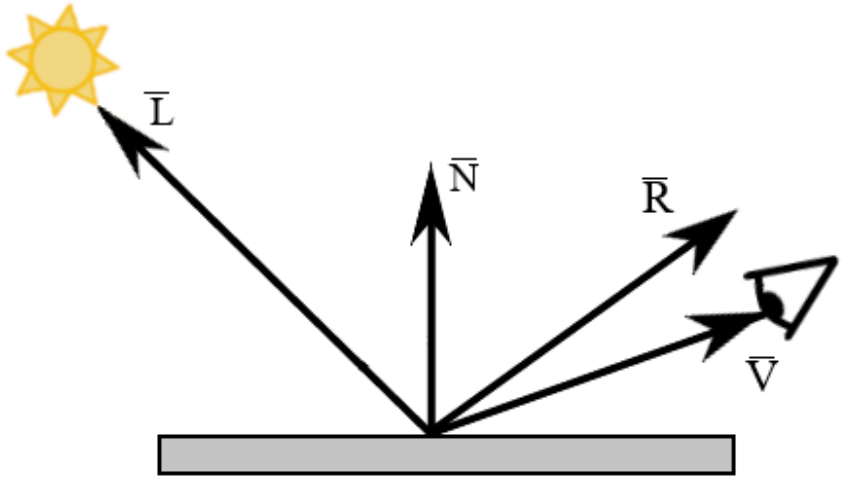

Рисунок 4 – Модель освещения Фонга

Отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча. Также в модели освещения Фонга используется понятие рассеянного освещения – это константа, которая прибавляется к интенсивности в точке для придания сцене большей реалистичности.

Таким образом, согласно модели Фонга интенсивность к точке складывается будет рассчитываться по формуле:

$$I = I_a + I_d + I_s, \tag{2}$$

где I_a — диффузная составляющая, I_d — рассеянная составляющая, I_s — зеркальная составляющая.

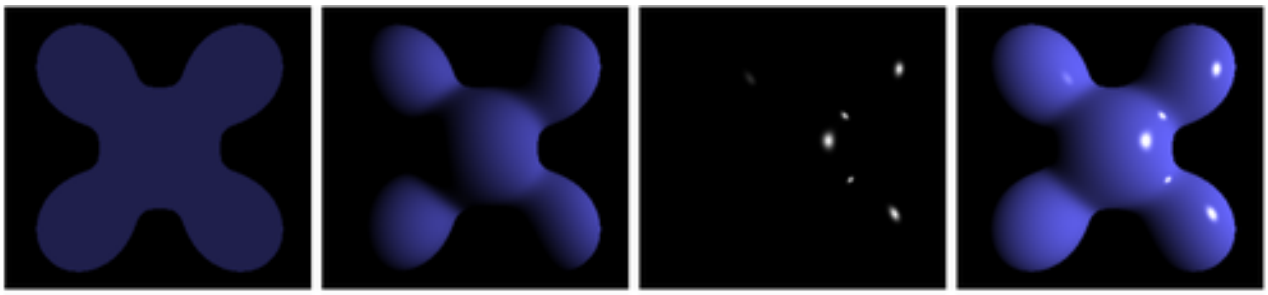

Рисунок 5 – Составляющие модели Фонга (слева направо: рассеянная, диффузная, зеркальная, суммарная)

Пусть:

— $\overrightarrow{R}$ — вектор отраженного луча;

— $\overrightarrow{V}$ — вектор от точки до наблюдателя;

— I_p — интенсивность рассеянного освещения;

— K_3 — коэффициент зеркального освещения;

— K_a — коэффициент рассеянного освещения;

— α — коэффициент блеска.

Формула для расчета интенсивности для модели Фонга имеет вид:

$$\begin{aligned} I = I_p \cdot K_a + I_0 \cdot K_d \cdot cos(\overrightarrow{L}, \overrightarrow{N}) + I_0 \cdot K_3 \cdot cos^{\alpha}(\overrightarrow{R}, \overrightarrow{V}) = \\ = I_p \cdot K_a + I_0 \cdot K_d \cdot (\overrightarrow{L}, \overrightarrow{N}) + I_0 \cdot K_3 \cdot (\overrightarrow{R}, \overrightarrow{V})^{\alpha} \end{aligned} \tag{3}$$

1.7.3 Модель Блинна-Фонга

Есть большое сходство с моделью Фонга, только она исключает расчёт отражённого луча, что сильно упрощает вычисления, тем самым, экономится время работы. Но существенной разницы нет. В этой модели используется медианный вектор, который является единичным и находится посередине между вектором, указывающим направление обзора, и вектором направле-

ния освещения. Чем ближе такой вектор к нормали поверхности, тем больше будет вклад зеркальной компоненты.

Благодаря тому, что измеряется угол между нормалью и медианным вектором (а не между вектором направления наблюдения и вектором отражения, как в модели Фонга), не будет проблемы с резкой границей области зеркального отражения.

1.7.4 Выбор модели освещения

В качестве модели освещения была выбрана модель Блинна-Фонга, она позволяет изобразить более реалистичное изображение, чем модель Ламберта, так как учитывает зеркальную составляющую и не требует дополнительных расчётов. Также такая модель позволяет исключить расчёт отражённого луча, что сильно упрощает вычисления, тем самым, экономится время работы.

1.8 Перспективно-корректное текстурирование

Пусть u, v – координаты текстуры, которые требуется найти для решения задачи наложения текстур на объект трёхмерной сцены.

Метод перспективно-корректного текстурирования основан на приближении u и v кусочно-линейными функциями. При отрисовке каждая сканирующая строка разбивается на части, в начале и конце каждого куска вычисляются точные значения u и v, а внутри каждой части применяется линейная интерполяция.

Пусть s_x и s_y – координаты, принадлежащие проекции текстурируемого треугольника. Тогда значения $\frac{1}{z}$, $\frac{u}{z}$ и $\frac{v}{z}$ линейно зависят от s_x и s_y. Сле-

довательно, для каждой вершины достаточно вычислить значения $\frac{1}{z}$, $\frac{u}{z}$ и $\frac{v}{z}$ и линейно их интерполировать.

Вывод

Подведя итог и проанализировав все вышеописанные алгоритмы, можно сделать вывод, что наилучшим алгоритмом для решения задачи будет алгоритм, использующий Z-буфер, модель Фонга-Блинна в сочетании с закраской Гуро. В качестве алгоритма текстиризации был выбран алгоритм перспективно-корректное текстурирование.

2 Конструкторская часть

В данном разделе представлены требования к программному обеспечению, рассмотрены алгоритмы, выбранные для построения сцены

2.1 Требования к программному обеспечению

Программа должна предоставлять графический интерфейс с функционалом:

- задать спектральные характеристики добавляемого объекта;
- изменить положение объекта, положение камеры, положение источника света в пространстве;
- выбрать функции при процедурном текстурировании, рисунки из библиотеки при проективном текстурировании;
- моделировать неровную поверхность.

2.2 Аппроксимация трёхмерных объектов

В ходе работы программы активно используется полигональная сетка. Это совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трёхмерной компьютерной графике. Недостатком этого метода является его приблизительность. Правда, видимые недочёты можно исправить, сделав разбиение более детальным, но это приведёт к дополнительным затратам по памяти и временным затратам.

Примеры использования полигональной сетки приведены на рисунок 6.

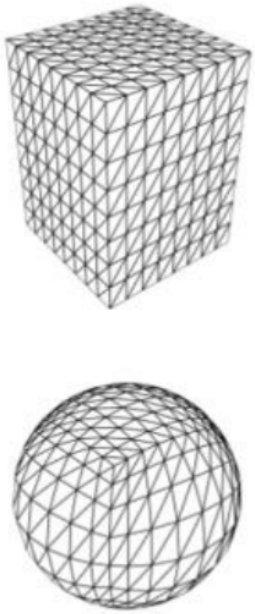

Рисунок 6 – Примеры использования полигональной сетки

В рамках текущей задачи в качестве граней выбраны треугольники. Построение полигональной сетки осуществляется следующим образом:

1) Куб : зная число разбиений по осям, промежуточные точки вычисляются через ее предыдущую точку и шаги, которые задаются пользователем:

$$x = x + step_x, y = y + step_y, z = z_0 \tag{4}$$

$$x = x_0, y = y + step_y, z = z + step_z \tag{5}$$

$$x = x + step_x, y = y_0, z = z + step_z \tag{6}$$

где step_x — шаг разбиения по оси x, step_y — шаг разбиения по оси y, step_z — шаг разбиения по оси z.

2) Цилиндр — зная высота h и радиус r, координаты вычисляются через цилиндрический параметр:

$$x = r * cos(\alpha), y = r * sin(\alpha), z = z_0 \tag{7}$$

где $0 \leqslant \alpha \leqslant 360$ (α в градусах), $-h \leqslant z_0 \leqslant h$.

3) Сфера — зная радиус r, координаты вычисляются через сферический параметр.

$$x = r * sin(\alpha) * cos(\beta), y = r * sin(\alpha) * sin(\beta), z = r * cos(\alpha) \tag{8}$$

где $0 \leqslant \alpha \leqslant 180$, $0 \leqslant \beta \leqslant 360$ (α, β в градусах).

Далее все полученные точки соединяются в треугольники.

2.3 Общий алгоритм построения изображения

Алгоритм генерации изображения представлен на рисунке 7.

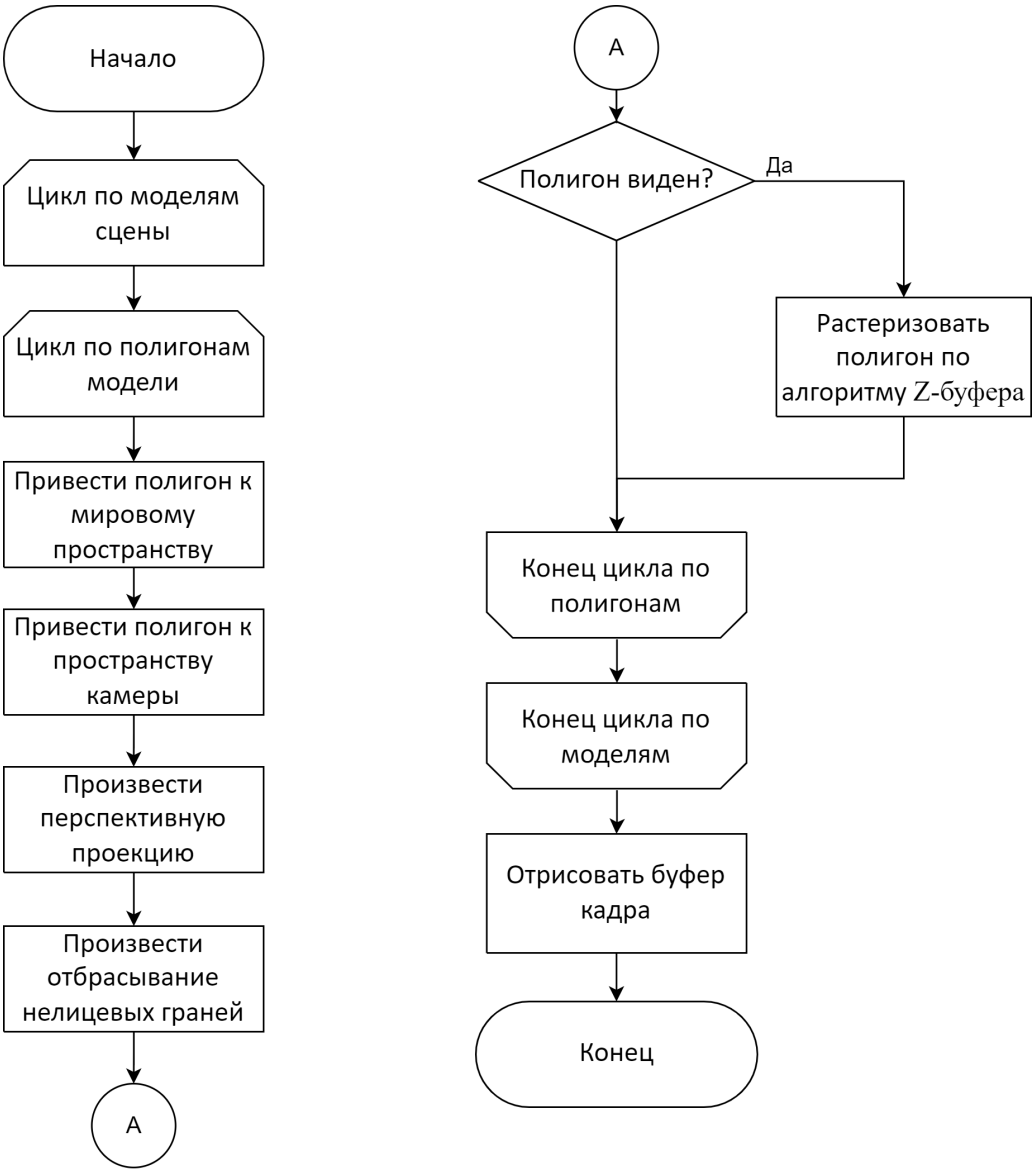

Рисунок 7 – Общая схема алгоритма синтеза изображения

2.4 Алгоритм Z-буфера

Для растеризации треугольного полигона сначала необходимо найти ограничивающий его прямоугольник, в котором этот полигон содержится. Это делается для того, чтобы не тратить время на растеризацию пикселей, не являющихся частью полигона. Затем для каждого пиксела ограничивающего прямоугольника находятся его барицентрические координаты относительно вершин полигона.

Затем производится сравнение значения глубины точки со значением глубины из Z-буфера. Если глубина пикселя меньше, значит он лежит ближе к камере и должен быть растеризован. Происходит вычисление интенсивности пикселя, его значение заносится в буфер кадра, а в Z-буфер заносится значение глубины пиксела.

Полная схема алгоритма Z-буфера представлена на рисунке 8.

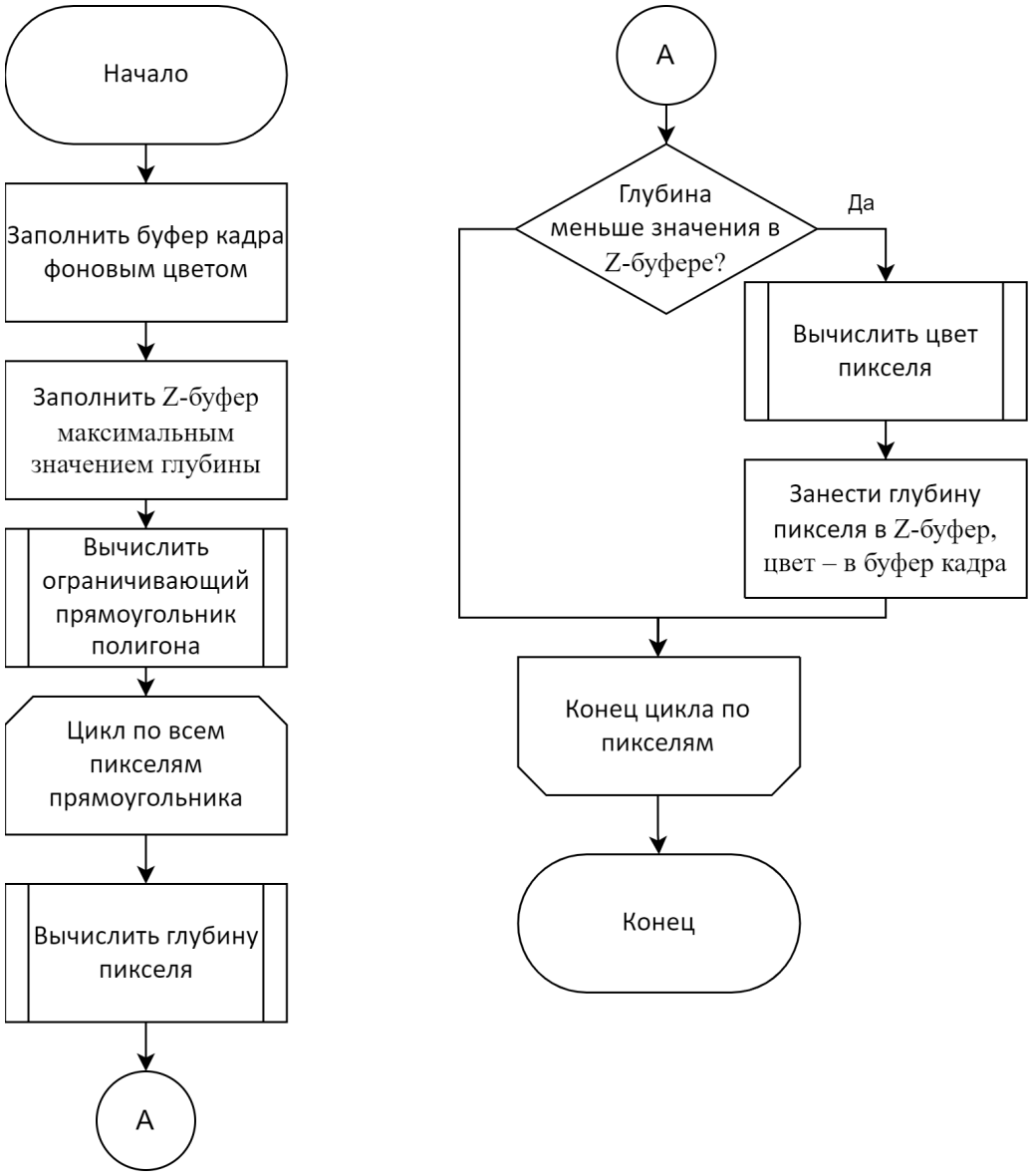

Рисунок 8 – Схема алгоритма Z-буфера

2.5 Описание трёхмерных преобразований

2.5.1 Способ хранения декартовых координат

Координаты можно хранить в форме вектор-столбца [x, y, z]. Однако в этом случае неудобно применять преобразования, так такой вектор нельзя умножить на соответствующие матрицы трансформации размерности четыре. Целесообразнее использовать вектор-столбцы размерности четыре [x, y, z, w], где w для точки равно одному. Преобразования координат выполняются умножением слева преобразуемого вектора-столбца на соответствующую матрицу линейного оператора.

2.5.2 Аффинные преобразования

В представленном алгоритме синтеза изображения первым этапом преобразования полигона пред его растеризацией является переход модели в мировое пространство. Такое действие осуществляется с помощью матриц аффинных преобразований. В данном курсовом проекте над объектами возможно произвести следующие операции.

— Поворот вокруг координатных осей описывается углом α и осью вращения. Матрица поворота имеет вид:

  – вокруг оси OX:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & sin\alpha & 0 \\ 0 & -sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (9)$$

– вокруг оси OY:

$$\begin{pmatrix} cos\alpha & 0 & -sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ sin\alpha & 0 & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{10}$$

– вокруг оси OZ:

$$\begin{pmatrix} cos\alpha & sin\alpha & 0 & 0 \\ -sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{11}$$

— Перенос в трехмерном пространстве задается значения вдоль координатных осей OX, OY, OZ — dx, dy, dz соответственно. Матрица переноса имеет вид:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix} \tag{12}$$

2.5.3 Преобразования трёхмерной сцены в пространство камеры

Чтобы привести трёхмерную сцену к пространству камеру, нужно умножить каждую вершину всех полигональных моделей на матрицу камеры. Сама же камера задаётся аргументами: положение камеры в мировом пространстве, вектор наблюдения взгляда, направление верха камеры. Пусть: α – координаты точки в пространстве, на которую смотрит камера, β – вектор, который указывает, куда смотрит верх камеры, ψ– ортогональный вектор к векторам направления взгляда и вектору направления.

Тогда матрица будет выглядеть так:

$$A = \begin{pmatrix} \alpha_x & \beta_x & \psi_x & 0 \\ \alpha_y & \beta_y & \psi_y & 0 \\ \alpha_z & \beta_z & \psi_z & 0 \\ -(P * \alpha) & -(P * \beta) & -(P * \psi) & 1 \end{pmatrix}$$

2.5.4 Матрица перспективной проекции

После перехода в пространство камеры нужно умножить каждую вершину всех полигонов на матрицу проекции. Эта матрица преобразует заданный диапазон усечённой пирамиды в пространство отсечения, и изменяет w-компоненту так, что чем дальше от наблюдателя находится вершина, тем больше возрастает w. После преобразования координат в пространство отсечения, координаты x и y попадают в промежуток $[-w, w]$, а вершина z – [-0, w]. Всё, что находится вне диапазона, отсекается.

Пусть AR – отношение ширины изображения к его высоте, α – угол обзора камеры, Z_n – координата z ближайшей к камере плоскости отсечения пирамиды видимости, Z_f – координата z дальней от камеры плоскости отсечения пирамиды видимости. Тогда матрица перспективной проекции принимает вид:

$$A = \begin{pmatrix} \frac{cot(\frac{\alpha}{2})}{AR} & 0 & 0 & 0 \\ 0 & cot(\frac{\alpha}{2}) & 0 & 0 \\ 0 & 0 & \frac{Z_f \times Z_n}{Z_f - Z_n} & 1 \\ 0) & 0 & \frac{Z_f}{Z_f - Z_n} & 0 \end{pmatrix}$$

Следующий этап – спроецировать все координаты на одну плоскость, разделив всё на координату z. После умножения вектора координат на матрицу перспективной проекции, реальная координата z заносится в w-компоненту,

так что вместо деления на z делят на w.

2.6 Процедурные текстуры

Процедурные текстуры — это тектсуры, описываемые математическими формулами. Такие текстуры не занимают в памяти места, а создаются, например, пиксельным шейдером.

Преимуществом процедурных текстур является неограниченный уровень детализации каждой тектсуры (пикселизации не будет, так как текстура всегда генерируется под необходимый для её отображения размер). Недостатком процедурных текстура можно назвать тот случай, когда подобранная функция является сложной, так как зависит от большого числа переменных.

Необходимо найти функцию C(X,Y,Z), определяющую для каждой точки поверхности цвет таким образом, чтобы он соответствовал цвету моделируемого материала (наносимого рисунка). Такой подход не требует больших затрат памяти и хорошо работает с поверхностями любой сложности. Но поскольку используемая функция зависит от большого количества параметров, то возникают сложности в подборе этой функции, с другой стороны, изменение этих параметров позволяет легко изменять свойства текстуры.

2.7 Проективные текстуры

Суть моделирования проективных тектсур заключается в том, что текстуры проецируются на поверхность параллельным переносом (плосоке проецирование), либо цилиндрическим и сферическим методами. Недостатки проективных текстур - большой объем памяти для хранения образцов текстур, небольшая гибкость и трудность текстурирования объектов сложной

формы.

Задается в некоторой (обычно двумерной) системе координат рисунок, подлежащий нанесению на трехмерную поверхность. Тогда для наложения заданного рисунка на поверхность необходимо найти функцию отображения или, другими словами, произвести преобразование координат. Проективным текстурам присущи определенные недостатки:

— для хранения используемых изображений требуется достаточно большой объем памяти;
— они не обладают гибкостью;
— возможны большие сложности при подборе способа проектирования при нанесении рисунка на объекты сложной формы.

2.8 Моделирование неровностей

Для создания шероховатой поверхности можно создать новую поверхность путем внесения возмущения в направлении нормали в точках поверхности [2].

В данной работе тела аппроксимируются треугольниками, затем треугольники растеризуются с помощью барицентрического координата. Поэтому возмущение вносится в нормаль каждой вершины треугольника.

Нормали поверхности, состоящей из трех точек A, B, C определяется векторным произведением:

$$\overrightarrow{n}_{ABC} = [\overrightarrow{AB}, \overrightarrow{AC}] \tag{13}$$

$$\overrightarrow{n}_{A} = \overrightarrow{n}_{B} = \overrightarrow{n}_{C} = \overrightarrow{n}_{ABC} \tag{14}$$

Вектор возмущение вычисляется случайным образом, затем сложится с вектором нормали в каждой вершине. Новая нормаль вычисляется следу-

ющим образом:

$$\overrightarrow{n}' = (sin(random()), cos(random()), sin(2 * random())) \tag{15}$$

$$\overrightarrow{n}_A = \overrightarrow{n}_A + \overrightarrow{n}' \tag{16}$$

Результирующий вектор после сложения нормализуется и этот вектор записывается в каждую вершину в качестве новой нормали.

$$\overrightarrow{n}_A = \frac{\overrightarrow{n}_A}{|\overrightarrow{n}_A|} \tag{17}$$

Полная схема алгоритма моделирования неровностей представлена на рисунке 9.

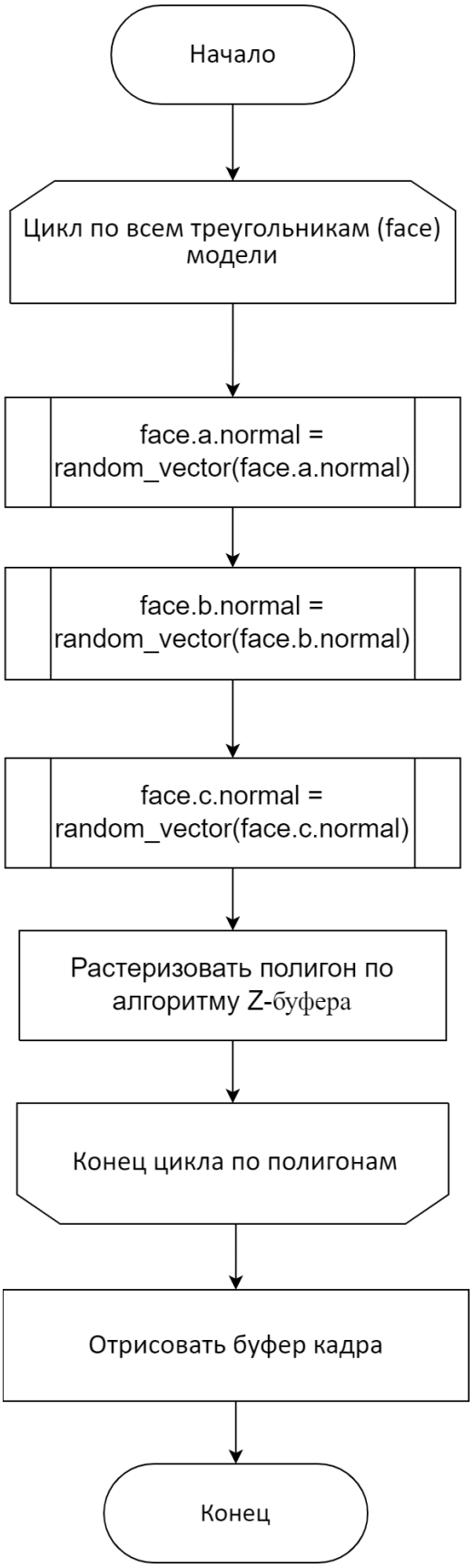

Рисунок 9 – Схема алгоритма моделирования неровностей

Вывод

Были описаны требования к программному обеспечению, алгоритмы для построения сцены в пространстве изображения, изменения положения объекта в пространстве, построение камеры и ее проекций. Также описаны процедурные текстуры, проективные текстуры и алгоритм моделирования неровностей.

3 Технологическая часть

В данной части рассматривается выбор средств реализации, описывается структура классов программы и приводится интерфейс программного обеспечения.

3.1 Средства реализации

Для написания данной курсовой работы был выбран язык C++ [5]. Выбор данного языка программирования обусловлен следующим образом:

— C++ — объектно-ориентированный язык, а именно такая методология программирования была выбрана для разработки программы;

— в данном языке имеется большое количество библиотек и шаблонов, позволяющих не тратить время на изобретение готовых конструкций.;

— обладает высокими показателями вычислительной производительности, а так как требуется быстродействие задач генерации реалистичных изображений, то язык C++ необходим.

В качестве среды разработки был использован Qt Creator [6]. Данный выбор обусловлен следующими факторами:

— в QT Creator есть возможность быстрого создания интерфейса с помощью расширения QT Design;

— QT Creator обладает всем необходимым функционалом для написания, профилирования и отладки программ.

3.2 Описание структуры программы

На рисунке 10 представлена схема разработанных классов.

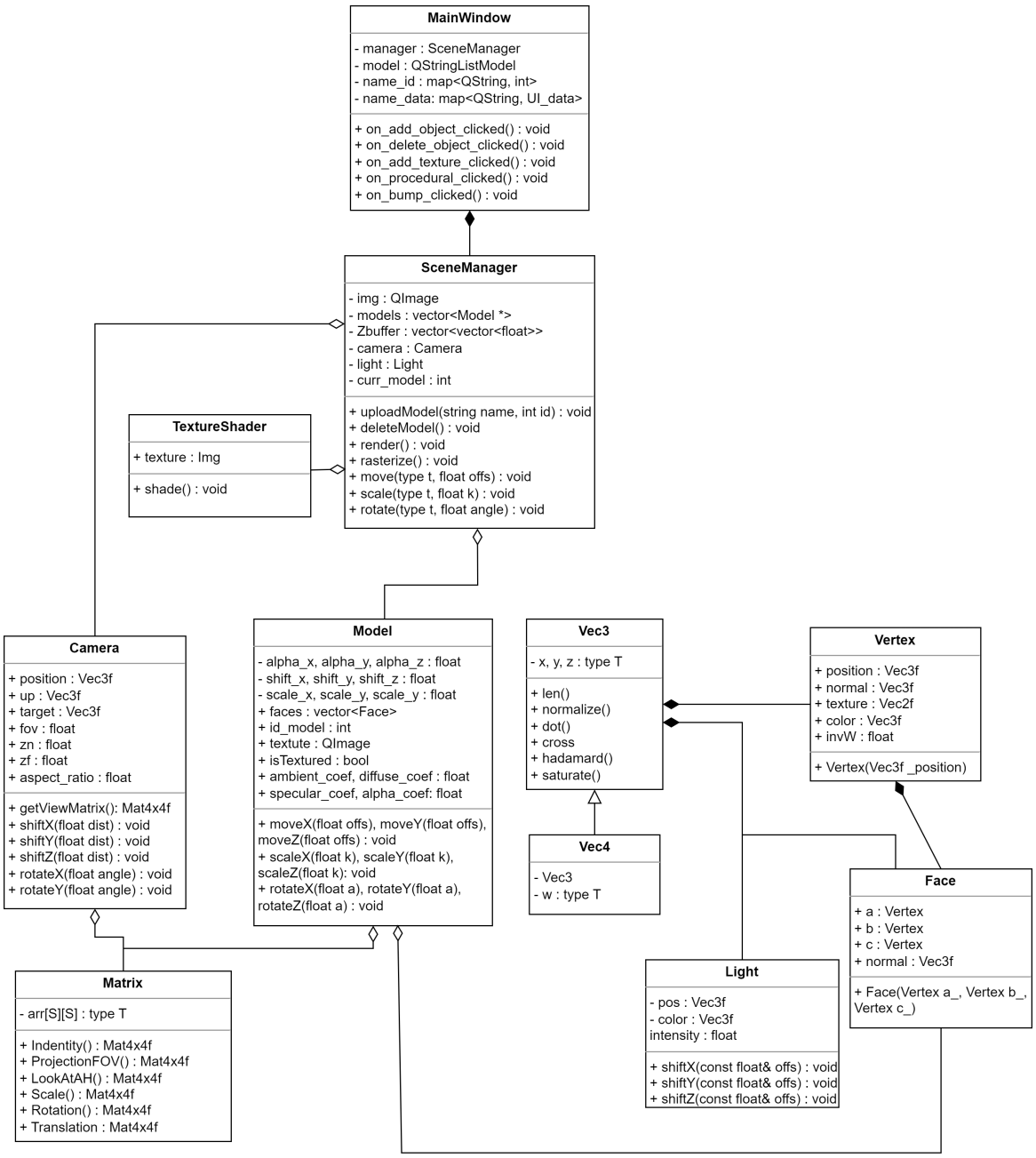

Рисунок 10 – Схема классов программы

В программе реализованы следующие классы:

— class SceneManager — хранит сцену, описывает методы взаимодействия со сценой и объектами на ней;

— class Model — описывает представление трёхмерного объекта и методы работы с ним;

— class Light — описывает источники света и методы взаимодействия с ним;

— class Camera — описывает камеру и методы взаимодействия с ней;

— class Vector — реализация векторов размерности 3 и 4;

— class Matrix — описывает матрицы и методы взаимодействия с ними.

— class TextureShader — содержит функции для интерполяции значения текстурных координат в конкретном пикселе;

3.3 Реализация алгоритмов

Листинг 1 – Алгоритм растеризации каждого объекта

```
1     for (auto& face: m.faces)
2     {
3         auto a = vertex_shader->shade(face.a, objToWorld,
              rotation_matrix, projectMatrix, viewMatrix, light, m, camera);
4         auto b = vertex_shader->shade(face.b, objToWorld,
              rotation_matrix, projectMatrix, viewMatrix, light, m, camera);
5         auto c = vertex_shader->shade(face.c, objToWorld,
              rotation_matrix, projectMatrix, viewMatrix, light, m, camera);
6         rasterizeTriangle(a, b, c);
7     }
8 }
9
10 void SceneManager::render()
11 {
12     this->img.fill(Qt::black);
13     for (auto &row : this->Zbuffer)
14         std::fill(row.begin(), row.end(), numeric_limits<float>::max());
```

```
15      for (auto& m : models)
16      {
```

Листинг 2 – Алгоритм растеризации треугольника

```
1       auto p2 = b.position;
2       auto p3 = c.position;
3
4       float sx = floor(min(min(p1.x, p2.x), p3.x));
5       float ex = ceil(max(max(p1.x, p2.x), p3.x));
6
7       float sy = floor(min(min(p1.y, p2.y), p3.y));
8       float ey = ceil(max(max(p1.y, p2.y), p3.y));
9
10      for (int y = static_cast<int>(sy); y < static_cast<int>(ey); y++)
11      {
12          for (int x = static_cast<int>(sx); x < static_cast<int>(ex); x++)
13          {
14              Vec3f bary = getBarycentric(p1, p2, p3,
                    Vec3f(static_cast<float>(x), static_cast<float>(y)));
15              if ( (bary.x > 0.0f || fabs(bary.x) < eps) && (bary.x < 1.0f
                    || fabs(bary.x - 1.0f) < eps) &&
16                   (bary.y > 0.0f || fabs(bary.y) < eps) && (bary.y < 1.0f
                        || fabs(bary.y - 1.0f) < eps) &&
17                   (bary.z > 0.0f || fabs(bary.z) < eps) && (bary.z < 1.0f
                        || fabs(bary.z - 1.0f) < eps))
18              {
19                  auto interpolated = baryCentricInterpolation(p1, p2, p3,
                        bary);
20                  interpolated.x = x;
21                  interpolated.y = y;
22                  if (testDepth(interpolated))
23                  {
24                      auto pixel_color = pixel_shader->shade(a, b, c, bary)
                            * 255.f;
25                      img.setPixelColor(x, y, qRgb(pixel_color.x,
                            pixel_color.y, pixel_color.z));
26                  }
27              }
28          }
29      }
```

```
30 }
31
32 void SceneManager::rasterize(Model& m)
33 {
34     auto projectMatrix = camera.projectionMatrix;
35     auto viewMatrix = camera.viewMatrix();
36     auto objToWorld = m.objToWorld();
37     auto rotation_matrix = m.rotation_matrix;
```

Листинг 3 – Алгоритм преобразования вершин в экранное пространство

```
1  Vertex VertexShader::shade(const Vertex &a, const Mat4x4f& objToWorld,
       const Mat4x4f& rotation, const Mat4x4f &projection, const Mat4x4f&
       camView, Light& light, Model& m, Camera& cam)
2  {
3      Vec4f new_pos(a.position);
4      new_pos = new_pos * objToWorld * camView * projection;
5      Vec4f new_normal(a.normal);
6      new_normal = new_normal * rotation;
7
8      Vertex output = a;
9      output.position = Vec3f(new_pos.x, new_pos.y, new_pos.z);
10     output.normal = Vec3f(new_normal.x, new_normal.y,
           new_normal.z).normalize();
11     if (fabs(new_pos.w) < eps)
12         new_pos.w = 1;
13     output.invW = 1 / new_pos.w;
14     output.position *= output.invW;
15     auto diffuse_comp = std::max(0.f,
           Vec3f::dot(output.normal.normalize(), (light.pos -
           output.position).normalize())) * m.diffuse_coef;
16     Vec3f half = ((light.pos - output.position).normalize() +
           (cam.position - output.position).normalize()).normalize();
17     auto specular_comp =
           std::pow(std::max(Vec3f::dot(output.normal.normalize(), half),
           0.f), m.alpha_coef) * m.specular_coef;
18     auto c = (light.color * light.intensity * (diffuse_comp +
           specular_comp) + ambient * m.diffuse_coef).saturate();
19     output.color = output.color.hadamard(c).saturate();
20
21     return output;
```

```
22 }
23
24 Vec3f ColorShader::shade(const Vertex &a, const Vertex &b, const Vertex
       &c, const Vec3f& bary)
```

Листинг 4 – Алгоритм текстуризации

```
1  Vec3f TextureShader::shade(const Vertex &a, const Vertex &b, const Vertex
       &c, const Vec3f &bary){
2
3      auto face_color = baryCentricInterpolation(a.color, b.color, c.color,
           bary);
4
5      float pixel_u = interPolateCord(a.texture.x , b.texture.x,
           c.texture.x, bary);
6      float pixel_v = interPolateCord(a.texture.y , b.texture.y,
           c.texture.y, bary);
7
8      int x = std::floor(pixel_u * (texture.width()) - 1);
9      int y = std::floor(pixel_v * (texture.height() - 1));
10
11     if (x < 0) x = 0;
12     if (y < 0) y = 0;
13
14
15     auto color = texture.pixelColor(x, y);
16     auto red = (float)color.red();
17     auto green = (float)color.green();
18     auto blue = (float)color.blue();
19     Vec3f pixel_color = Vec3f{red / 255.f, green/ 255.f , blue /255.f};
20
21     auto t = pixel_color.hadamard(face_color).saturate();
22
23     return t;
24 }
```

3.4 Описание интерфейса программного обеспечения

На рисунке 11 представлен стартовый экран программы. Он предоставляет пользователю возможность добавить объект, изменить положение источника света и ёго интенсивность.

Рисунок 11 – Пользовательский интерфейс

Для добавления объекта модели её нужно выбрать в выпадающем меню в левом верхнем углу, а затем нажать на кнопку добавления. Модель появится на графике, а также в текстовом поле. Для наложения текстур на объект, нужно выбрать объект из списки, а затем нажать либо на кнопку «Процедурные» либо на кнопку «Проективные». Для моделирования неровностей нажать на кнопку «Моделировать неровности». Управление камерой осуществляется при помощи клавиатуры кнопками W, A, S, D, H, K, U, J.

3.5 Демонстрация работы программы

На рисунках 12–14 представлены результаты работы программы.

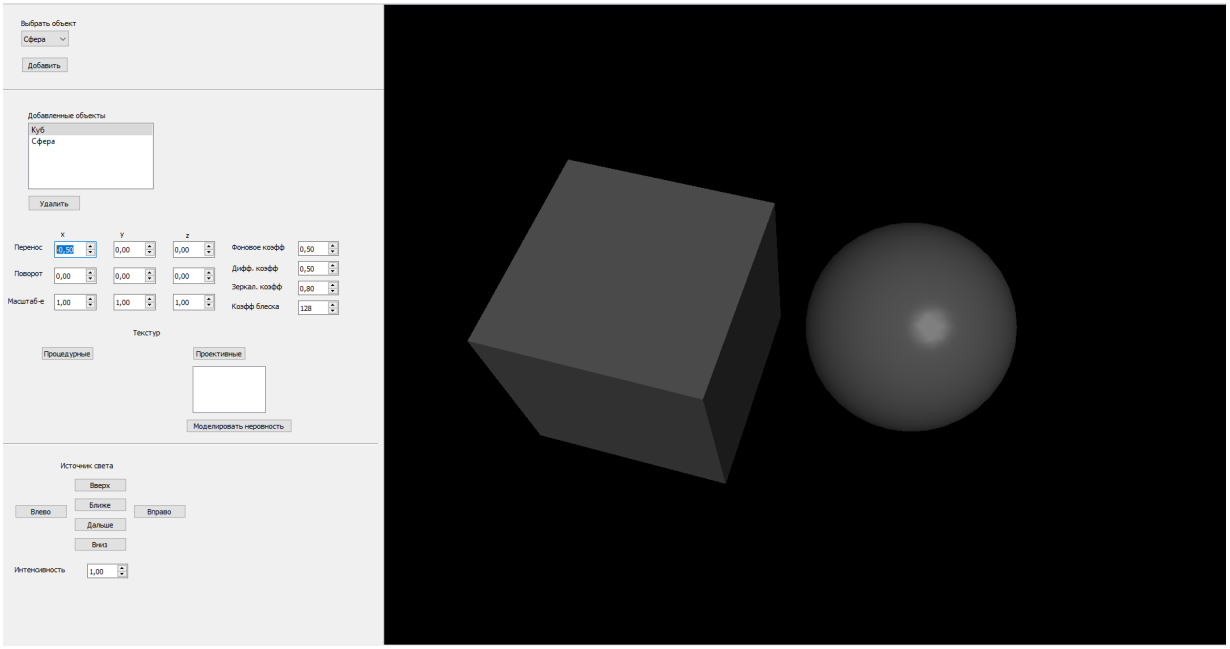

Рисунок 12 – Результат работы программы без текстуры

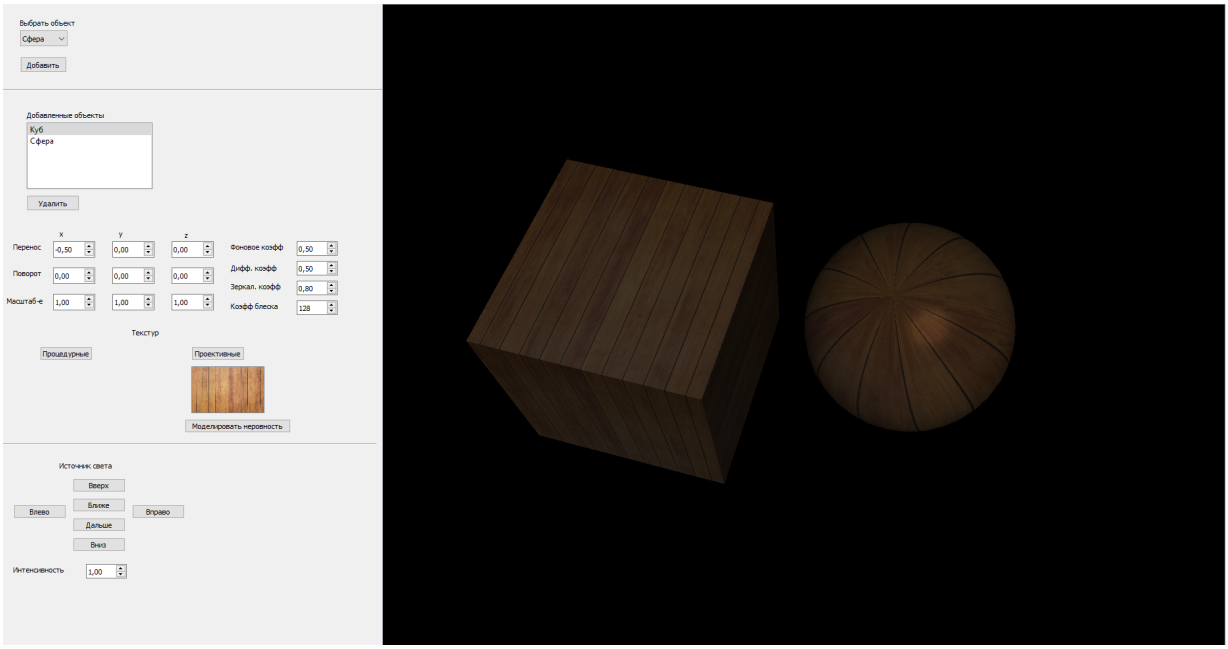

Рисунок 13 – Результат работы программы при наложении текстуры

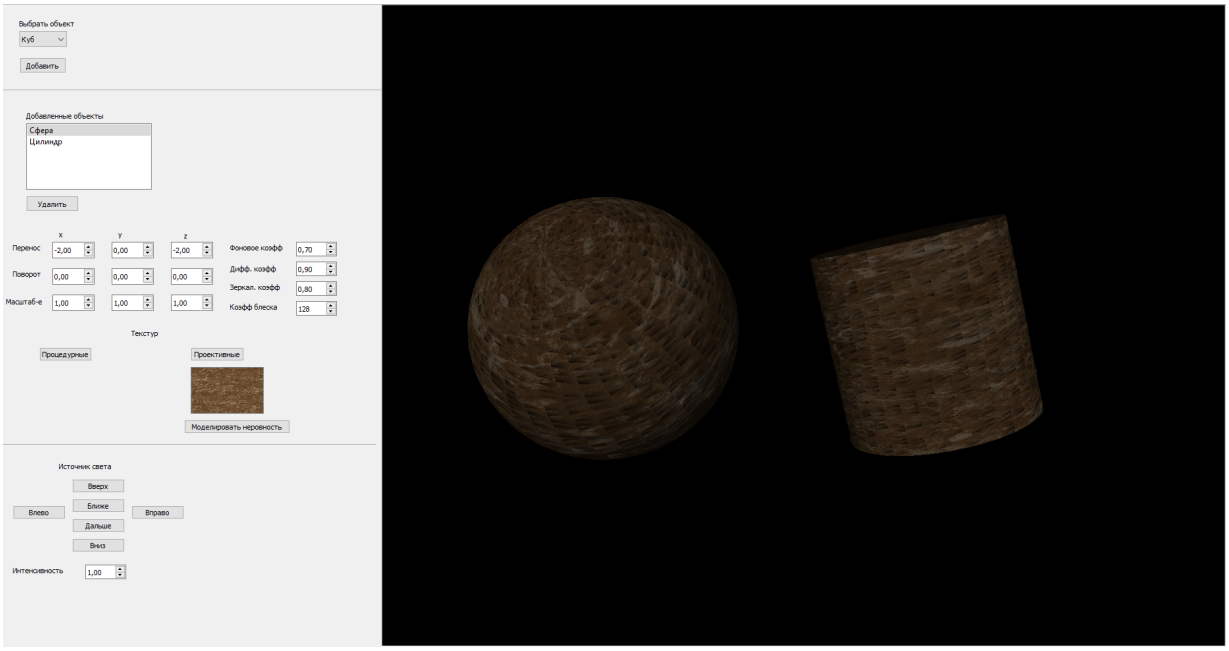

Рисунок 14 – Результат работы программы при моделировании неровностей

Вывод

Было приведено описание структура программы, выбраны средства реализации программного обеспечения, приведены листинги кода, продемонстрирован интерфейс программы и представлены результаты работы программы.

4 Исследовательская часть

В данном разделе приведены технические характеристики устройства, на котором проводилось измерение времени работы программного обеспечения, а также результаты замеров времени.

4.1 Постановка исследования

Целью исследования является проведение анализа скорости работы алгоритма генерации изображения с использованием алгоритма Z-буфером.

4.1.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование.

— операционная система Window 10 Home Single Language;
— память 8 Гб;
— процессор 11th Gen Intel(R) Core(TM) i7-1165G7 2.80 ГГц, 4 ядра.

4.1.2 Результаты исследования

Для исследования зависимости времени тренинга изображения от числа объектов на сцене, использовались объекты без текстуры. Количество объектов менялось на сцене от 50 до 500 с шагом 50, были рассмотрены случаи для куба, цилиндра, сферы. Результаты проведенного исследования представлены.

Как видно из графика, время визуализации сцены зависит от количе-

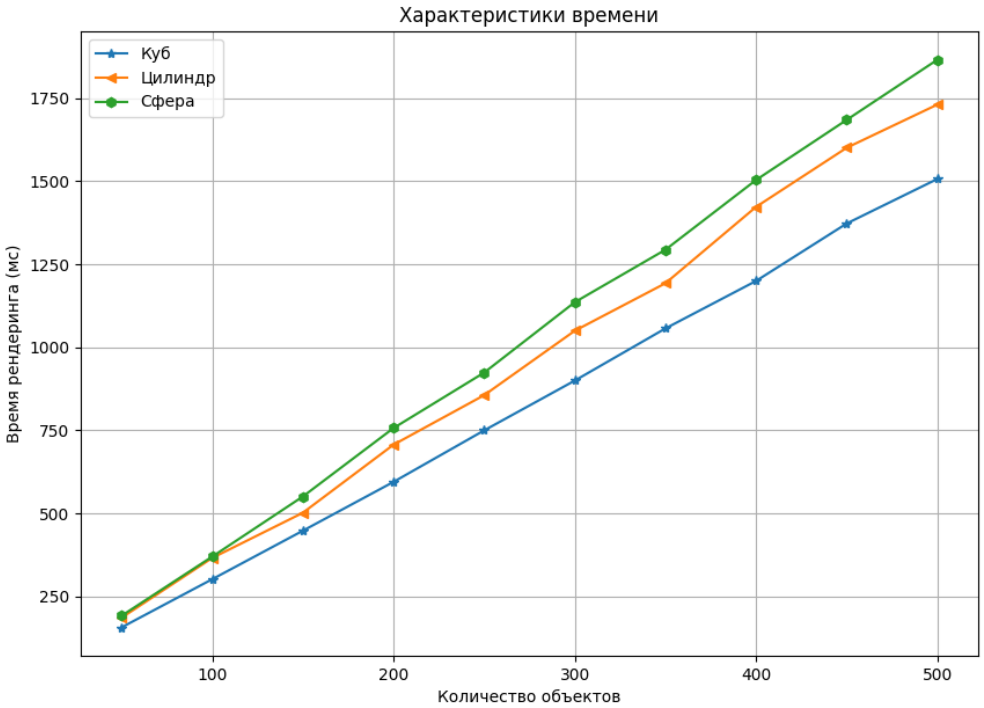

Рисунок 15 – График зависимости времени отрисовки от числа объектов

ства объектов линейно.

Следующим этапом исследования разработанной программы является исследование зависимости времени построения сцены от текстуры при фиксированном количестве объектов. В ходе исследования количество сфер менялось от 50 до 500 с шагом 50.

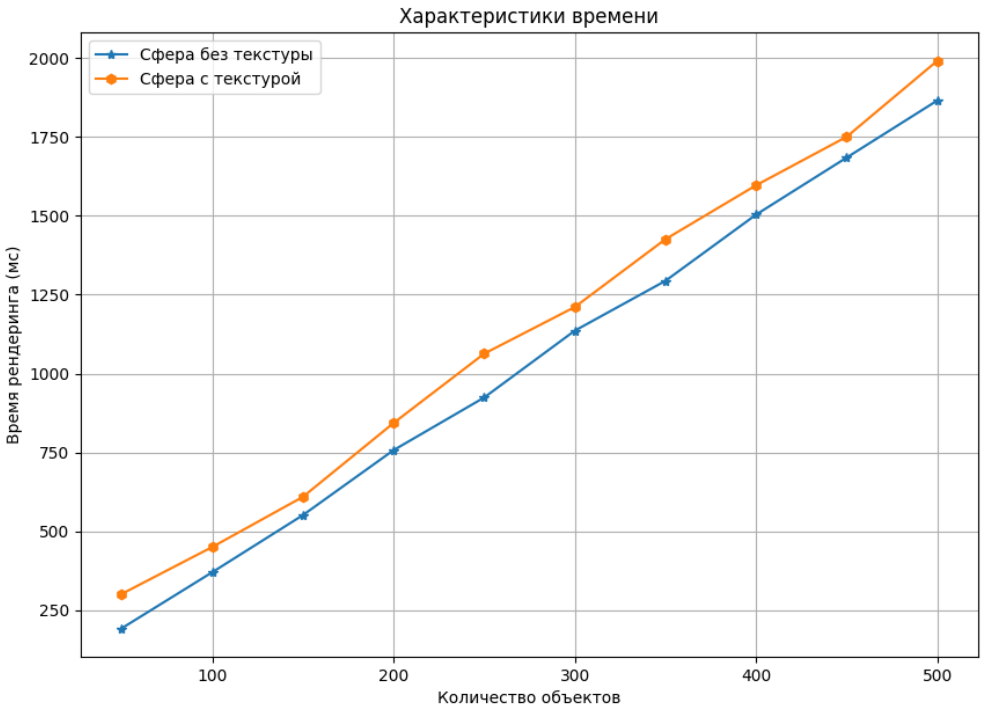

Рисунок 16 – График зависимости времени отрисовки от учёта текстуры

Из проведенного исследования можно сделать вывод, что время визуализации сцены линейно зависит от количества объектов на сцене. В случаях визуализации с учётом текстуры время растет объясняется сложностью при вычислении текстурных координат.

Вывод

В данном разделе приведены результаты работы программного обеспечения.

Результаты исследования совпали с ожидаемыми, так как в ходе исследования было установлено, что время работы увеличивается с увеличением количества объектов на сцене. Также время выполнения зависит от текстуры, которая накладывается на объект.

ЗАКЛЮЧЕНИЕ

Цель курсовой работы была достигнута, то есть был разработано программное обеспечение для наложения текстур на трёхмерные объекты.

Для достижение цели были решены все задачи:

— проведен анализ существующих алгоритмов компьютерной графики, используемых для создание трехмерной сцены;

— выбраны алгоритмы для решения поставленной задачи;

— выбраны язык программирования и среда разработки для реализации поставленной задачи;

— создано программное обеспечение, реализующее выбранные алгоритмы;

— проведены замеры временных характеристик разработанного программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Горюнова А.М. Омельченко Т.В. Омельченко П.Н. Муслимов Д.А. Применение компьютерной графики для решения экономических и инженерных задач : учебное пособие. — Оренбург: ОГУ, 2018. — С. 153.

2 Д. Роджерс. Алгоритмические основы машинной графики: Пер. с англ. — СПб: БХВ-Петербург, 1989. — С. 512.

3 Алгоритм, использующий Z-буфер [Электронный ресурс]. — Режим доступа: `https://web.posibnyky.vntu.edu.ua/fitki/8romanyuk_komp_grafika/zmg/zmg/45.htm` (дата обращения: 20.10.2023).

4 Алгоритм обратной трассировки лучей [Электронный ресурс]. — Режим доступа: `https://dzen.ru/a/YozcqGjmRidwVBUz` (дата обращения: 20.10.2023).

5 Документация по языку C++ [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160` (дата обращения: 30.10.2023).

6 Qt Creator Manual [Электронный ресурс]. — Режим доступа: `https://doc.qt.io/qtcreator` (дата обращения: 30.10.2023).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация содержит 14 слайдов.