

# Оглавление

<b>Введение</b>	<b>4</b>
<b>1 Аналитическая часть</b>	<b>5</b>
1.1 Текстура, как характеристика поверхности трехмерного тела . . . . .	5
1.2 Описание метода внесения возмущений в нормаль . . . . .	5
1.3 Текстурные карты . . . . .	6
1.4 Представление объектов . . . . .	6
1.5 Анализ и выбор алгоритма удаления невидимых ребер и поверхностей . . . . .	6
1.5.1 Алгоритм буфера глубины . . . . .	7
1.5.2 Алгоритм Робертса . . . . .	8
1.5.3 Алгоритм обратной трассировки лучей . . . . .	8
1.6 Выбор модели освещения . . . . .	10
<b>2 Конструкторская часть</b>	<b>12</b>
2.1 Требования к программному обеспечению . . . . .	12
2.2 Общий алгоритм решения задачи . . . . .	12
2.3 Трассировка лучей . . . . .	13
2.3.1 Свет . . . . .	13
2.3.2 Наблюдатель и сцена . . . . .	14
2.3.3 Лучи . . . . .	14
2.3.4 Объекты . . . . .	14
2.3.5 Пересечение луча и грани . . . . .	15
2.3.6 Диффузное отражение . . . . .	16
2.3.7 Зеркальное отражение . . . . .	18
2.3.8 Тени . . . . .	21
2.3.9 Отражения . . . . .	21
2.4 Связь различных текстурных карт с данными о нормали . . . . .	22
2.4.1 Карты высот (англ. height maps) . . . . .	22
2.4.2 Карты нормалей (англ. normal maps) . . . . .	23
2.4.3 Карта параллактического отображения (англ. parallax map) . . . . .	24
2.5 Наложение текстуры на поверхность . . . . .	25

<b>3 Технологическая часть</b>	<b>27</b>
3.1 Средства реализации . . . . .	27
3.2 Структура программы . . . . .	27
3.3 Интерфейс программного обеспечения . . . . .	28
Вывод . . . . .	30
<b>4 Исследовательская часть</b>	<b>31</b>
4.1 Технические характеристики . . . . .	31
4.2 Результаты работы ПО . . . . .	31
4.3 Анализ производительности . . . . .	33
Вывод . . . . .	34
<b>Заключение</b>	<b>35</b>
<b>Список использованных источников</b>	<b>36</b>
<b>Приложение А</b>	<b>36</b>

# Введение

Современное компьютерное моделирование и визуализация требуют детального и реалистичного представления трехмерных объектов. Особенное внимание уделяется методам, позволяющим достичь высокой степени реалистичности без заметного увеличения вычислительной сложности или объема данных. Одним из таких методов является учет текстуры на поверхности тел путем внесения возмущений в нормаль.

Текстурирование позволяет не только изменять внешний вид объекта, но и модифицировать его геометрические и световые характеристики. С помощью текстур можно создавать реалистичные детали поверхности, такие как мельчайшие неровности, шероховатости и даже более сложные элементы, такие как: камни, узоры и рельефы, не добавляя при этом дополнительных геометрических объектов в сцену.

Цель работы – разработать программное обеспечение для учета текстуры на поверхности трехмерных тел с использованием метода внесения возмущения в нормаль.

Для достижения поставленной цели требуется решить следующие задачи:

- 1) Определить, какие объекты будут располагаться на сцене;
- 2) Выбрать алгоритмы для построения и обработки трехмерных объектов. Это включает в себя методы удаления невидимых поверхностей, создания теней и отражений;
- 3) Определить модель освещения, которая будет использоваться для создания реалистичных световых эффектов;
- 4) Спроектировать структуру программного обеспечения и выбрать подходящий способ представления данных;
- 5) Выбрать средства реализации алгоритмов;
- 6) Создать программное обеспечение, реализовав в нем выбранные алгоритмы;
- 7) Исследовать временные характеристики выбранных алгоритмов на основе созданного программного обеспечения.

# **1 Аналитическая часть**

В данном разделе производится формализация объектов сцены, анализ алгоритмов их визуализации, и выбирается наиболее подходящие для решения поставленной задачи.

## **1.1 Текстура, как характеристика поверхности трехмерного тела**

В компьютерной графике текстурой называется детализация структуры поверхности. Текстура – это одномерное или двумерное изображение, которое имеет множество ассоциированных с ним параметров, определяющих, каким образом производится наложение изображения на поверхность.

Обычно рассматривается два вида детализации. Первый состоит в том, чтобы на гладкую поверхность нанести заранее заданный узор – так называемая детализация светом. После этого поверхность все равно остается гладкой. Второй тип детализации заключается в создании неровностей на поверхности, реализуется путем внесения возмущений в параметры поверхности – детализация фактурой. [1]

## **1.2 Описание метода внесения возмущений в нормаль**

В этом методе для создания неровностей отдельно рассматривается каждый пиксел текстуры – текстел. Реалистичность изображения достигается за счет создания рельефа, а каждый текстел должен хранить информацию о возмущении в нормаль, которое требуется внести.

## 1.3 Текстурные карты

Для хранения информации о текстелях используют так называемые текстурные карты. Для метода внесения возмущений в нормаль чаще всего используют следующие:

- Карта высот (англ. height map). Каждый пиксел карты хранит одно значение. Поэтому изображения являются черно-белым, где белый – самые выпуклые точки на поверхности;
- Карта нормалей (англ. normal map). Представлены в виде RGB-изображений, где каналы R и G описывают наклон каждой нормали;
- Карта параллактического отображения (англ. parallax map). Эта карта использует особый визуальный эффект, чтобы создать иллюзию глубины и объемности. Она изменяет положение текстурных координат в зависимости от уровня смещения, создавая ощущение сложной геометрии на плоской поверхности.

## 1.4 Представление объектов

Для решения данной задачи важно учесть, как именно текстура будет располагаться на поверхности модели, поэтому требуется наиболее полное представление об объекте. Такую модель называют объемной.

## 1.5 Анализ и выбор алгоритма удаления невидимых ребер и поверхностей

Выделим критерии, которыми должен обладать выбранный алгоритм:

- Получение высокореалистичных изображений. Для успешного наложения текстур важно учитывать даже небольшие изменения в освещении, тенях и других световых эффектах.

- Отсутствие ограничений на входные модели. Если мы стремимся к глубокому исследованию текстурирования, мы должны убедиться, что выбранный алгоритм способен обрабатывать разнообразные модели, включая не только выпуклые, но и неоднородные, с отверстиями и так далее.

### 1.5.1 Алгоритм буфера глубины

Одним из самых распространенных алгоритмов удаления невидимых поверхностей является алгоритм буфера глубины (англ. Z-буфера) [2, 3]. Здесь буфер глубины представляет собой дополнительное пространство в памяти, где для каждого пикселя хранится значение глубины объектов перед ним (расстояние от наблюдателя до поверхности объекта). Задача алгоритма заключается в нахождении минимального значения глубины  $z$  для каждого пикселя экрана.

Принцип работы заключается в следующем: сначала весь буфер глубины заполняется значениями, соответствующими максимальной глубине. В процессе растеризации граней для каждого пикселя рассчитывается его глубина и сравнивается с текущим значением в буфере глубины. Если рассматриваемый пикセル находится ближе, он рисуется, и его глубина заменяет значение в буфере. Если пиксел дальше, он не рисуется, и буфер остается неизменным – это позволяет отбросить невидимые поверхности.

Преимущества алгоритма:

- Простота реализации;
- Высокая скорость работы;
- Применим к различным типам объектов.

Недостатки:

- Производительность может снижаться при обработке сложных сцен;
- Могут возникнуть проблемы с отображением прозрачных объектов;
- Требуется значительный объем памяти.

### **1.5.2 Алгоритм Робертса**

Алгоритм Робертса [2] представляет собой первое известное решение задачи об удалении невидимых линий. Это математический метод, работающий в объектном пространстве.

В алгоритме Робертса требуется, чтобы все изображаемые тела или объекты были выпуклыми. Невыпуклые тела должны быть разбиты на выпуклые части. Выпуклое многогранное тело с плоскими гранями должно представляться набором пересекающихся плоскостей.

Алгоритм Робертса включает в себя три этапа:

- На первом этапе каждое тело анализируется индивидуально с целью удаления нелицевых плоскостей;
- На втором этапе проверяется экранирование оставшихся в каждом теле ребер всеми другими телами с целью обнаружения невидимых отрезков;
- На третьем этапе вычисляются отрезки, которые образуют новые ребра при протыкании телами друг друга.

Преимущества алгоритма:

- Высокая точность изображения на выходе.

Недостатки:

- Сложность реализации;
- Высокий рост сложности алгоритма при увеличении числа объектов;
- Необходимость разбивать невыпуклые объекты на выпуклые, что может значительно замедлить выполнение программы.

### **1.5.3 Алгоритм обратной трассировки лучей**

Методы трассировки лучей [2, 3] считаются наиболее мощными и универсальными методами создания реалистичных изображений. Существует множество успешных реализаций алгоритмов трассировки для отображения даже самых сложных трехмерных сцен.

Суть алгоритма заключается в том, что из источника наблюдения проводится луч в каждую точку картинной плоскости. Анализируя траекторию луча, мы можем определить, какие объекты он пересекает и от каких отражается. Это подобно тому, как человеческий глаз воспринимает световые лучи, но в обратном порядке.

При практической реализации метода обратной трассировки вводят ограничения. Вот некоторые из них:

- 1) Среди типов объектов выделяются источники света, они могут только излучать свет;
- 2) Свойства отражающих поверхностей описываются суммой двух компонент — диффузной и зеркальной;
- 3) Зеркальность разбивается на две составляющие: отражение от других объектов (кроме источников света) и световые блики от источников;
- 4) При диффузном отражении учитываются только лучи от источников света. Точки, находящиеся в тени, определяются тем, если луч на источник света блокируется другим объектом;
- 5) Для прозрачных объектов часто упрощается моделирование преломления: без учета зависимости от длины волны. Прозрачность может также моделироваться без преломления, когда направление преломленного луча совпадает с направлением падающего луча.

Преимущества алгоритма:

- Высокая степень реалистичности полученных изображений;
- Вычислительная сложность не сильно зависит от количества объектов на сцене.

Недостатки:

- Ограничения в производительности из-за интенсивного вычислительного процесса.

Алгоритм обратной трассировки лучей является предпочтительным для использования в задачах внесения возмущений в нормали и текстурной модификации по следующим причинам:

- Алгоритм обратной трассировки лучей уже включает в себя вычисление нормалей для каждой точки поверхности. Это означает, что мы можем легко внести дополнительные изменения в нормали, соответствующие текстуре;
- Алгоритм обратной трассировки лучей позволяет точно моделировать освещение, что имеет ключевое значение при работе с текстурными изменениями. Этот метод обеспечивает более реалистичные эффекты теней, подсветки и отражений, учитывая изменения, внесенные в нормали;
- В данной задаче акцент делается на визуальное воздействие, а не на обработку больших объемов данных в кратчайшие сроки. Хотя алгоритм обратной трассировки лучей может потреблять больше ресурсов, он предоставляет значительно более качественное визуальное представление сцены.

## 1.6 Выбор модели освещения

При текстурировании в трассировке лучей, предпочтительной моделью освещения является модель Фонга. Эта модель выделяется своей способностью учесть разнообразные характеристики поверхностей в зависимости от направления света и точки наблюдения. Такой выбор модели обеспечивает создание более реалистичных текстур на объектах.

Модель освещения Фонга включает как диффузную, так и зеркальную компоненты освещения. Диффузная составляющая позволяет текстуре равномерно рассеивать свет в разные направления, что особенно важно для отображения матовых и неровных поверхностей. Зеркальная составляющая создает блеск и отражения на более гладких участках поверхности, что позволяет подчеркнуть детали текстуры.

## Вывод

При процессе текстурирования с внесением изменений в нормали, главной целью является достижение высокой степени реалистичности.

Применяя текстурные карты, необходимо использовать объемные модели, чтобы обеспечить правильное расположение текстуры на поверхности.

Выбор алгоритма обратной трассировки лучей обусловлен его способностью создавать более точное и реалистичное изображение поверхности, сохраняя детали и освещение. Отличным дополнением стала модель освещения Фонга, которая добавит еще больше реалистичности и позволит наиболее точно проанализировать наложение текстур.

Важно отметить, что благодаря такому подбору методов, при трассировке лучей мы сразу же учитываем отражающие и преломляющие способности поверхности, а также сразу можем накладывать текстурные карты. Это предоставляет возможность в несколько раз ускорить работу программы: нет необходимости производить вычисления поэтапно, анализируя одну и ту же поверхность несколько раз.

## **2 Конструкторская часть**

В данном разделе будут более подробно рассмотрены выбранные в предыдущем разделе методы и алгоритмы, а также предоставлены требования к программному обеспечению.

### **2.1 Требования к программному обеспечению**

- Возможность отображать заданные объекты с учетом текстуры;
- Поддержка различных типов источников света: точечные, направленные и рассеянные;
- Учет теней, света и отражений;
- Поддержка различных текстурных карт: карт высот, нормалей, параллактического отображения.

### **2.2 Общий алгоритм решения задачи**

- Разместить источники света: определить их позиции и характеристики (типы и интенсивность);
- Добавить объект на сцену;
- Загрузить для объекта текстурные карты;
- Трассировка лучей и визуализация:
  - Для каждой точки картинной плоскости сгенерировать луч, исходящий из источника наблюдения;
  - Проанализировать путь луча через сцену, учет взаимодействия с гранями объекта;
  - Внесение изменений в нормали объектов с учетом текстурных карт;
  - Расчет освещенности, теней и отражений на каждой точке поверхности объекта;

- Формирование окончательного изображения на экране с учетом всех эффектов и модификаций.

## 2.3 Трассировка лучей

В реальном мире свет исходит от источников освещения, отражается от нескольких объектов и достигает наших глаз [4]. Процесс моделирования пути света называется трассировкой лучей. В компьютерной графике же используются алгоритмы обратной трассировки. Основная идея заключается в анализе путей лучей, исходящих не от источников света, а от “глаз”, смотрящих на предметы. Такой подход соответствует всем привычным нам законам физики, при этом является реальным для реализации, в отличие от симуляции фотонов.

### 2.3.1 Свет

Для наилучшего отображения реального освещения определим три типа источников света:

- Точечные источники света: испускают его равномерно во всех направлениях из определенной точки в пространстве;
- Направленные источники света: имеют фиксированное направление света, их можно рассматривать как бесконечно удаленные точечные источники, расположенные в заданном направлении;
- Рассеянные источники света: привносят часть освещения в каждую точку сцены, независимо от ее положения. Упрощает визуализацию реальной модели, когда свет, достигнув объекта, рассеивает часть обратно в сцену.

Таким образом, на сцене мы определим один источник рассеянного света и несколько точечных и направленных источников.

### 2.3.2 Наблюдатель и сцена

Первое, что определяется на сцене – положение наблюдателя и окна просмотра. Введем обозначения. Пусть точка  $O = (O_x, O_y, O_z)$  – позиция камеры. Ширина и высота окна просмотра -  $V_w, V_h$  соответственно. Так же определим расстояние от точки  $O$  до плоскости окна –  $d$ , количество пикселей в окне приложения –  $C_w$  (по ширине),  $C_h$  (по высоте) и координаты пикселя на холсте -  $C_x, C_y$ . Для перехода от координат холста  $(C_x, C_y)$  к пространственным координатам  $(V_x, V_y, V_z)$  необходимо будет выполнить следующие преобразования:

$$V_x = C_x * \frac{V_w}{C_w}; \quad V_y = C_y * \frac{V_h}{C_h}; \quad V_z = d. \quad (2.1)$$

Итак, для каждой точки холста мы можем определить соответствующую ей точку в окне просмотра.

### 2.3.3 Лучи

Когда мы определили источник лучей (т.  $O$ ), и их направления ( $V$ ), можно задать луч ( $P$ ). Удобнее всего это сделать с помощью параметрического уравнения:

$$P = O + t(V - O), \quad (2.2)$$

где  $t$  – любое неотрицательное вещественное число. Обозначим направление луча как:  $\vec{D} = (V - O)$ . Тогда:

$$P = O + t\vec{D}. \quad (2.3)$$

### 2.3.4 Объекты

Поскольку для представления объектов мы выбрали объемную модель, для универсальности программы можно использовать популярный формат

– .obj (Wavefront OBJ) файлы. Данный формат – текстовый и используется для хранения трехмерных моделей. Он содержит следующую информацию:

- Вершины (англ. Vertices). Координаты точек в трехмерном пространстве, которые определяют форму объекта;
- Нормали (англ. Normals). Направления поверхности в каждой вершине, важные для расчета освещения.
- Текстурные координаты (англ. Texture Coordinates). Координаты, используемые для нанесения текстуры на поверхность модели.
- Границы (англ. Faces). Определяют полигоны объекта, указывая индексы вершин, текстурных координат и нормалей, составляющих грани.

Таким образом, каждая грань объекта будет задана набором точек ( $T_i$ ).

### 2.3.5 Пересечение луча и грани

Для того чтобы определить пересечения луча и грани можно проанализировать как проходит луч относительно каждого ребра при проходе их в заданном порядке. Если луч проходит с внешней стороны от какой-либо грани – пересечения не будет.

Для каждого ребра грани, образованного смежными вершинами  $V_i$  и  $V_{i+1}$ , где  $i = 0, 1, \dots, n - 1$  (где  $n$  – количество вершин), мы должны выполнить следующие шаги:

- 1) Пусть  $P_{intersection}$  – точка пересечения луча с гранью. Чтобы ее определить с помощью параметрического уравнения необходимо вычислить параметр  $t_L$ . Для этого воспользуемся следующим уравнением:

$$t_L = \frac{N * P_L}{N * \vec{D}}, \quad (2.4)$$

где  $P_L$  – вектор от начальной точки луча до первой точки вершины грани. Итак:

$$P_{intersection} = O + t_L \vec{D}; \quad (2.5)$$

- 2) Найдем вектора, соединяющие текущую вершину с точкой пересечения луча с плоскостью грани. Пусть  $V_i$  и  $V_{i+1}$  - текущее ребро грани.
- 3) Найдем вектора ребер  $E_1$  и  $E_2$ :

$$E_1 = V_i - P_{intersection}; \quad E_2 = V_{i+1} - P_{intersection}; \quad (2.6)$$

- 4) Вычислим векторное произведение ребер:

$$N_{edge} = E_1 \times E_2; \quad (2.7)$$

- 5) Найдем скалярное произведение вектора ребра  $N_{edge}$  и нормали грани  $N$ :

$$D_P = E_1 \times E_2; \quad (2.8)$$

- 6) Если скалярное произведение меньше нуля, значит, луч проходит с внешней стороны от текущего ребра, и пересечения с гранью нет.

### 2.3.6 Диффузное отражение

Диффузное отражение это процесс, при котором луч света, сталкиваясь с объектом, рассеивается обратно в сцену равномерно во всех направлениях. Именно оно придает матовым объектам матовость. Рассмотрим луч света с направлением  $\vec{L}$  и интенсивностью  $I$ , который достигает поверхности с нормалью  $N$ . Представим интенсивность света как «ширину» луча. Его энергия распределяется по поверхности размером  $A$ . Отобразжу ситуацию на рисунке, обозначив вспомогательные углы и точки:

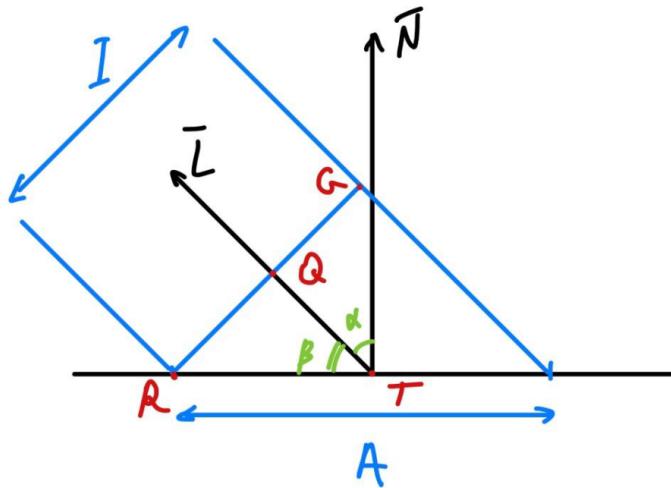


Рисунок 2.1 – Диффузное отражение света

Луч света шириной  $I$  достигает поверхности в точке  $T$  под углом  $\beta$ . Нормаль в  $T$  это  $\vec{N}$ , а переносимая лучом энергия распределяется по поверхности  $A$ . В случае, когда луч света имеет одинаковое направление с нормалью,  $I = A$ . С другой стороны, по мере того, как угол между  $\vec{L}$  и  $\vec{I}$  приближается к  $90^\circ$ ,  $A \rightarrow \infty$ . Значит,  $\lim_{x \rightarrow \infty} \frac{I}{A} = 0$ . Выходит, для определения диффузного отражения, нам необходимо узнать что находится на промежутке, выяснив значение  $\frac{I}{A}$ . Рассмотрим  $RG$  – ширину луча. Так как эта прямая перпендикулярна лучу света,  $QRT = \alpha$ . В треугольнике  $QRT$  сторона  $QR = \frac{1}{2}$ , а  $RT = \frac{A}{2}$ . По определению  $\cos \alpha = \frac{QR}{RT} = \frac{\frac{1}{2}}{\frac{A}{2}} = \frac{I}{A}$ . Так как  $\alpha$  – угол между  $(\vec{N})$  и  $\vec{L}$ , мы можем использовать скалярное произведение этих векторов, чтобы выразить угол как:

$$\frac{I}{A} = \cos \alpha = \frac{\langle \vec{N}, \vec{L} \rangle}{|\vec{N}| |\vec{L}|}. \quad (2.9)$$

Мы получили простое уравнение, дающее нам отражаемую долю света в виде функции угла между нормалью поверхности и направлением света. Если у нас получилось, что значение  $\alpha > 90$ , это означает, что свет освещает тыльную сторону поверхности, и учитывать это не нужно. В итоге, мы можем записать уравнение диффузного отражения для вычисления общего количества света, получаемого точкой  $T$  с нормалью  $\vec{N}$  в сцене с рассеянным светом с интенсивностью  $I_A$  и  $n$  точечных или направленных источников света с ин-

тенсивностью  $I_n$ , а так же световыми векторами  $\vec{L}_n$ :

$$I_T = I_A + \sum_{i=1}^n I_i * \frac{\langle \vec{N}, \vec{L} \rangle}{|\vec{N}| |\vec{L}|}. \quad (2.10)$$

### 2.3.7 Зеркальное отражение

Зеркальное отражение отображает глянцевость объектов. Их вид меняется при смещении точки обзора. Рассмотрим луч света  $\vec{L}$ . Нам нужно определить, сколько света от него отражается обратно в направлении точки обзора. Пусть  $\vec{V}$  – вектор обзора, указывающий из точки  $T$  в сторону камеры, а  $\alpha$  – угол между  $\vec{R}$  и  $\vec{V}$ , тогда получим следующую картину:

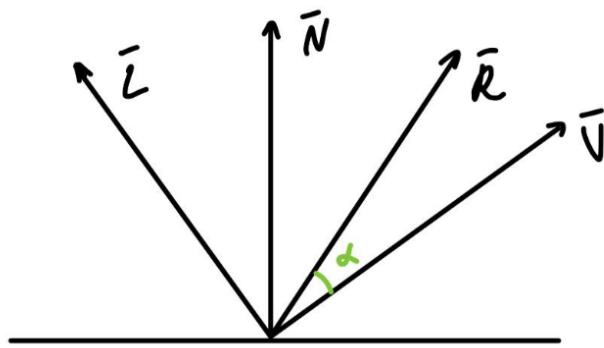


Рисунок 2.2 – Зеркальное отражение света

При  $\alpha = 0$  весь свет отражается в направлении  $\vec{V}$ . При  $\alpha = 90$  свет вообще не отражается. Как и в случае с диффузным отражением, нам нужно математическое выражение, позволяющее определить, что происходит при промежуточных значениях  $\alpha$ . Данная модель не имеет физического прототипа, но отлично отражает необходимые свойства и проста в вычислении. Рассмотрим свойство  $\cos \alpha : \cos 0 = 1 \cos \pm 90 = 0$ . Это мы и будем использовать. Теперь необходимо учесть глянцевость поверхности. Глянцевость – это мера того, как быстро уменьшается функция отражения при возрастании  $\alpha$ . Простой способ это сделать – возвести  $\cos \alpha$  в положительную степень  $s$ :

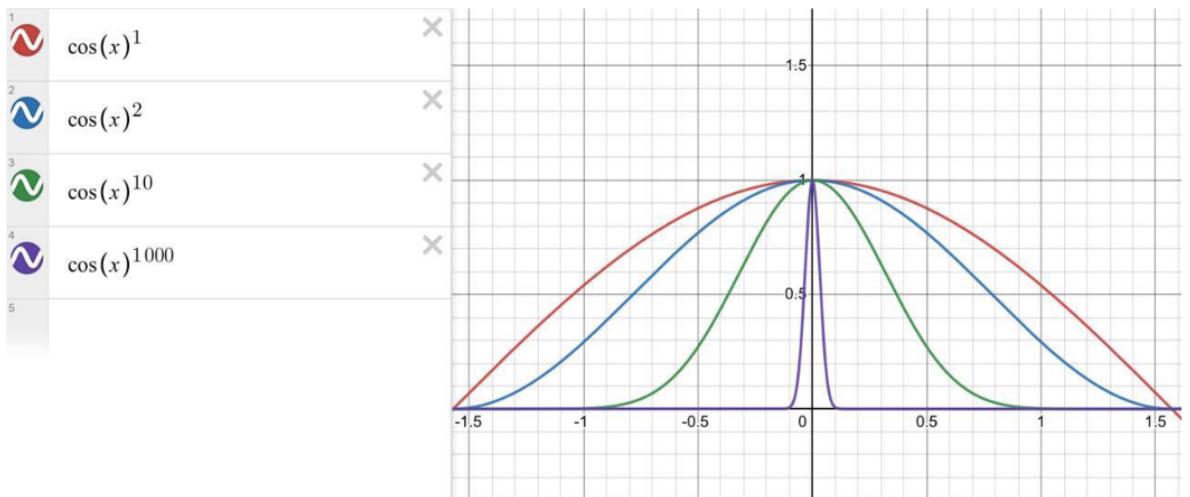


Рисунок 2.3 – График для  $\cos x^s$

$s$  – характеристика блеска поверхности. Для начала вычислим  $\vec{R}$  из  $\vec{N}$  и  $\vec{L}$ . Для этого разделим  $\vec{L}$  на  $\vec{L}_P$  и  $\vec{L}_N$  так, чтобы  $\vec{L} = \vec{L}_P + \vec{L}_N$ :

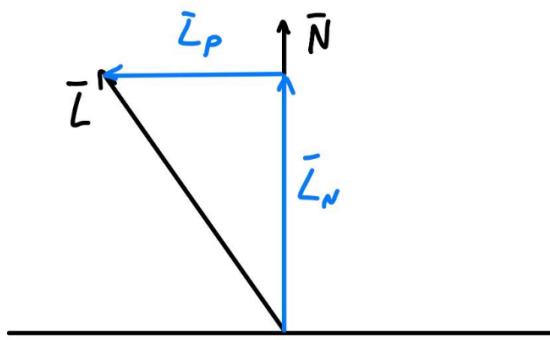


Рисунок 2.4 – Разделение  $\vec{L}$  на  $\vec{L}_P$  и  $\vec{L}_N$

$\vec{L}_N$  – проекция  $\vec{L}$  на  $\vec{N}$ . Исходя из свойств скалярного произведения и того, что длина нормали равняется единице, длина проекции равна  $\langle \vec{N}, \vec{L} \rangle$ . Так как  $|\vec{L}_N| |\vec{N}| = \vec{N} * \langle \vec{N}, \vec{L} \rangle$ . Получим  $\vec{L}_P = \vec{L} - \vec{N} * \langle \vec{N}, \vec{L} \rangle$ . Теперь рассмотрим  $\vec{R}$ :

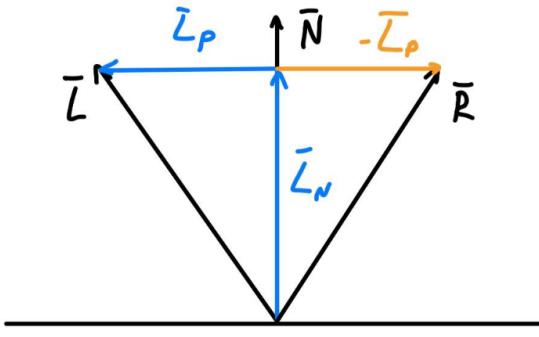


Рисунок 2.5 – Расположение векторов  $\vec{R}$  и  $\vec{L}$

Исходя из рисунка:  $\vec{R} = \vec{L}_N - \vec{L}_P$ . Подставив полученное выше:

$$\vec{R} = \vec{N} * \langle \vec{N}, \vec{L} \rangle - \vec{L} + \vec{N} * \langle \vec{N}, \vec{L} \rangle = 2 * \vec{N} * \langle \vec{N}, \vec{L} \rangle - \vec{L} \quad (2.11)$$

Теперь, по аналогии с диффузным отражением, можно составить уравнение для зеркального:

$$\vec{R} = 2 * \vec{N} * \langle \vec{N}, \vec{L} \rangle - \vec{L}; \quad (2.12)$$

$$I_S = I_L * \left( \frac{\langle \vec{R}, \vec{V} \rangle}{|\vec{R}| |\vec{V}|} \right)^S. \quad (2.13)$$

Как и в случае с диффузным отражением, если  $\cos \alpha$  оказался отрицательным, его нужно игнорировать. Кроме того, для матовых объектов выражение зеркальности не должно вычисляться. Это можно предусмотреть заранее, пометив соответствующую поверхность, чтобы сократить количество вычислений. Вычислив зеркальное и диффузное отражение, составим уравнение полного освещения в точке  $T$ :

$$I_T = I_A + \sum_{i=1}^n I_i * \left[ \frac{\langle \vec{N}, \vec{L} \rangle}{|\vec{N}| |\vec{L}|} + \left( \frac{\langle \vec{R}, \vec{V} \rangle}{|\vec{R}| |\vec{V}|} \right)^S \right], \quad (2.14)$$

где  $I_A$  – интенсивность рассеянного света,  $\vec{N}$  – нормаль к поверхности в точке

$T$ ,  $V$  – вектор от  $T$  к камере,  $s$  – зеркальная характеристика поверхности,  $I_i$  – интенсивность потока света  $i$ ,  $L_i$  – вектор из  $T$  к свету  $i$ , а  $R_i$  – вектор отражения в  $T$  для потока света  $i$ .

### 2.3.8 Тени

Тени возникают, когда лучи света не могут достичь объекта из-за встреченного на пути препятствия. Начнем с направленного света. Нам известна точка  $T$  на поверхности, а также луч света  $\vec{L}$ . Зная это, мы можем определить луч  $(O + t\vec{L})$ , проходящий из этой точки поверхности к бесконечно удаленному источнику света. Если этот луч пересекается с чем-либо, то точка будет находиться в тени и освещения от этого источника нужно игнорировать. В ином случае, мы добавляем освещение данного источника, как было показано ранее. Точечные источники можно рассматривать так же, но надо учитывать то, что мы не хотим, чтобы объекты дальше источника света могли отбрасывать тени на  $T$ . Установим  $t_{max} = 1$ , чтобы при достижении источника света луч останавливался. Так же не стоит забывать, что  $t_{min} = eps$ , где  $eps$  – очень близкое к нулю число справа, чтобы мы не нашли пересечение с поверхностью, из которой и пускаем луч.

### 2.3.9 Отражения

Когда мы смотрим в зеркало, мы видим отражаемые им лучи света. Они отражаются симметрично относительно нормали поверхности. Итак, нам необходимо выяснить, куда идет луч, отражаемый от зеркальной поверхности. Таким образом, мы получим рекурсивный алгоритм. При его создании нам нужно убедится, что мы не порождаем бесконечный цикл. У нас будет два условия выхода: когда луч сталкивается с неотражающим объектом, и когда он ни с чем не сталкивается. Также важно вспомнить об эффекте «бесконечного коридора», который образуется, если поставить два зеркала друг напротив друга. Самый простой способ предотвратить бесконечный подсчет отражений: ограничить рекурсию каким-либо числом. В нашей задачи нет необходимости отображать зеркальные поверхности, а тем более визуализи-

ровать эффект «бесконечного коридора», поэтому можно обойтись достаточно малым значением в 2-3 единицы.

## 2.4 Связь различных текстурных карт с данными о нормали

Как было сказано ранее, существует несколько применяемых способов вносить возмущения в нормали. Рассмотрим же соответствующие текстурные карты от наиболее примитивной, к наиболее сложной.

### 2.4.1 Карты высот (англ. height maps)

Карты высот имеют черно-белый цвет, что означает, что все каналы RGB (red, green, blue) равны между собой. Таким образом, они хранят единственное значение. Пример:

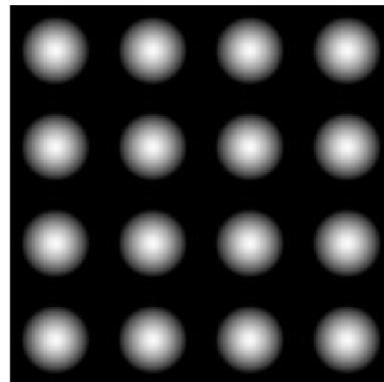


Рисунок 2.6 – Карта высот  $h(x, y)$

Чтобы получить возмущение, сначала нужно вычислить аппроксимацию производных по направлениям  $x$  и  $y$  с помощью разностного аналога [5]:

$$h_x(x, y) = \frac{h(x + 1, y) - h(x - 1, y)}{2}; \quad h_y(x, y) = \frac{h(x, y + 1) - h(x, y - 1)}{2}. \quad (2.15)$$

В изображении это будет выглядеть так:

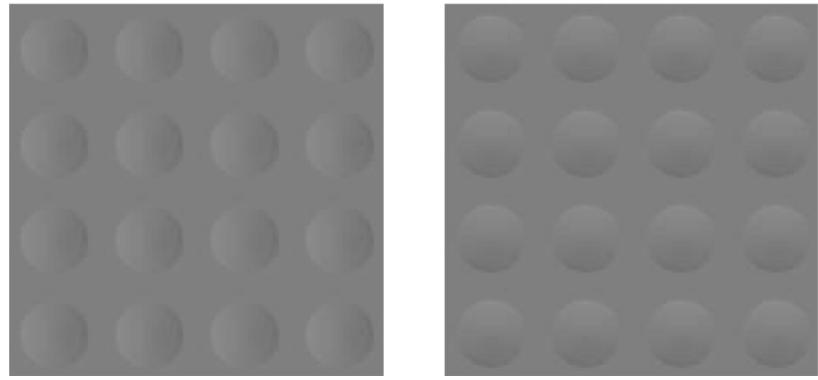


Рисунок 2.7 – Аппроксимация по  $x$  (слева) и  $y$  (справа)

Тогда ненормализованное значение нормали в текстеле  $(x, y)$ :

$$N'(x, y) = (-h_x(x, y) - h_y(x, y), 1). \quad (2.16)$$

#### 2.4.2 Карты нормалей (англ. normal maps)

Карты нормалей содержат значения в двух каналах. Принято хранить  $h_x$  в канале R, а  $h_y$  в G. Тогда значение канала B не используется, и все карты имеют голубоватый оттенок

В изображении это будет выглядеть так:

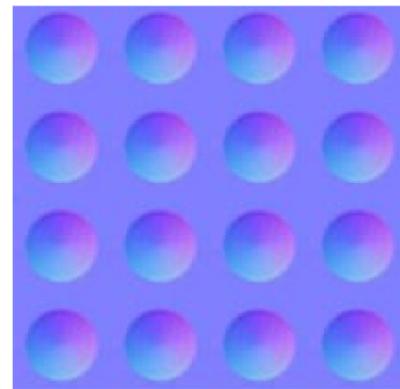


Рисунок 2.8 – Карта нормалей

### 2.4.3 Карта параллактического отображения (англ. parallel map)

С предыдущими типами карт есть проблема: при смещении угла обзора, это никак не влияет на рельеф. Если мы посмотрим вдоль реальной кирпичной стены под некоторым углом, мы не увидим зазор между кирпичами. Это происходит, потому что мы лишь изменили нормали. Идея параллакса заключается в том, что положение объектов относительно друг друга должно изменяться с движением наблюдателя. Неровности должны увеличиваться в высоту [5].

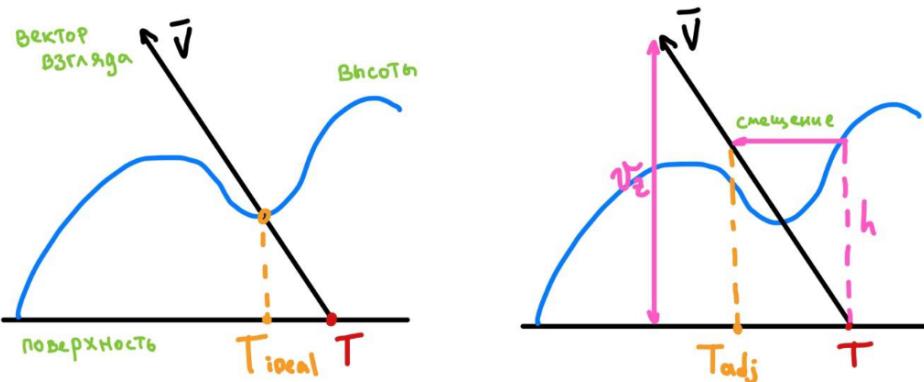


Рисунок 2.9 – Фактическое положение поверхности определяется лучом взгляда (слева). Параллактическое отображение выполняет аппроксимацию, используя высоту для нахождения положения новой точки (справа).

Теперь, помимо направления нормали, нам необходимо хранить высоту  $h$ . Учитывая местоположение с координатами текстуры  $T$ , высоту  $h$ , нормализованный вектор вида  $V$  со значением высоты  $V_z$  и горизонтальной составляющей  $V_{xy}$ , новая скорректированная по параллаксу текстурная координата:

$$T_{adj} = T + \frac{h * V_{xy}}{V_z}. \quad (2.17)$$

Однако, когда вектор обзора находится вблизи горизонта поверхности, небольшое изменение высоты приводит к большому смещению координат текстуры. Аппроксимация не будет корректной, поскольку полученное новое местоположение практически не коррелирует по высоте с исходным местоположением на поверхности. Чтобы решить эту проблему, мы можем ограничить смещение

ние: оно не должно превышать высоту. Тогда уравнение будет иметь вид:

$$T_{adj} = T + h * V_{xy}. \quad (2.18)$$

При крутых углах, это уравнение почти совпадает с исходным, поскольку  $V_z \approx 1$ . При малых углах смещение становится ограниченным. Но даже с такими ограничениями параллактическое смещение должно обеспечивать более реалистичное представление.

## 2.5 Наложение текстуры на поверхность

Теперь, когда у нас есть все необходимые уравнения и преобразования, остается лишь связать текстурную карту с поверхностью. То есть нужно определить, какой пиксель  $h_{xy}$  текстурной карты соответствует точке на поверхности.

В .obj файле фигуры у нас заданы текстурные координаты каждой из вершин, а для того чтобы определить текстурные координаты  $(u, v)$  любой точки  $(P)$  грани выполним следующие шаги:

- 1) Для каждого треугольника, образованного смежными вершинами  $V_i, V_{i+1}, V_{i+2}$  вычислим барицентрические координаты  $(\alpha_i, \beta_i, \gamma_i)$ . Для этого сначала определим векторы сторон треугольника:

$$\vec{AB} = B - A; \quad (2.19)$$

$$\vec{AC} = C - A. \quad (2.20)$$

И вектор  $\vec{AP}$ :

$$\vec{AP} = P - A. \quad (2.21)$$

Затем определим  $\alpha_i, \beta_i, \gamma_i$ :

$$\alpha_i = \frac{\vec{AC} \times \vec{AP}}{\vec{AC} \times \vec{AB}}; \quad (2.22)$$

$$\beta_i = \frac{\vec{AB} \times \vec{AP}}{\vec{AB} \times \vec{AC}}; \quad (2.23)$$

$$\gamma_i = 1 - \alpha_i - \beta_i. \quad (2.24)$$

2) Вычислим теперь текстурные координаты:

$$u = \sum_{i=1}^n \alpha_i u_i; v = \sum_{i=1}^n \beta_i v_i, \quad (2.25)$$

где  $\alpha_i, \beta_i$  - барицентрические координаты для  $i$ -й вершины, а  $u_i, v_i$  - ее текстурные координаты.

## Вывод

В данном разделе были подробно рассмотрены алгоритмы, необходимые для реализации поставленной цели.

# 3 Технологическая часть

В данном разделе представлены средства реализации, структура программы и интерфейс ПО.

## 3.1 Средства реализации

Для разработки программы был выбран язык JavaScript — мульти paradigmенный язык программирования [?]. Выбор данного языка обусловлен наличием опыта разработки с его использованием, наличием большого количества библиотек с открытым исходным кодом, направленных на визуализацию в трехмерном пространстве, а также большим количеством литературы по компьютерной графике ( $\approx 334000$  результатов по запросу «js computer graphics algorithms» в поисковой системе Google Академия).

Для создания пользовательского интерфейса программного обеспечения будет использоваться модуль datgui [?]. Для мониторинга производительности будет использоваться модуль Stats [?]. Этот модуль позволяет отслеживать FPS (количество кадров в секунду). Данная информация позволит оценить производительность ПО. В качестве среды разработки выбран текстовый редактор Visual Studio Code [?], содержащий большое количеством плагинов и инструментов для различных языков программирования, в том числе для JavaScript. Такие инструменты облегчают и ускоряют процесс разработки программного обеспечения.

## 3.2 Структура программы

Разработанная программа состоит из следующих классов.

- 1) Сцена представляет собой объект класса *Scene* с полями *camera*, *lights*, *cloth*
- 2) Ограничитель представляет собой объекта класса *Constraint* с полями *p1*, *p2* и *distance*.

- 3) Частица представляет объекта класса *Particle* с полями *position*, *previous*, *original*.
- 4) Тканевая сетка представляет собой объект класса *Cloth* с полями *particles*, *constraints*.
- 5) Источник света представляет собой объекта класса *DirectionalLight* с полями *color*, *intensity*, *position*.
- 6) Камера представляет собой объекта класса *PerspectiveCamera* с полями *position*, *lookAt*.

### 3.3 Интерфейс программного обеспечения

На рисунках 3.1 – 3.3 представлен интерфейс программы. При запуске загружается сцена с предварительно заданными объектами. Управление камерой производится с помощью компьютерной мыши путем зажатия левой кнопки мыши и передвижения в сторону. Для отдаления используется колесо мыши. Начальный интерфейс программного обеспечения отображен на рисунке 3.1.

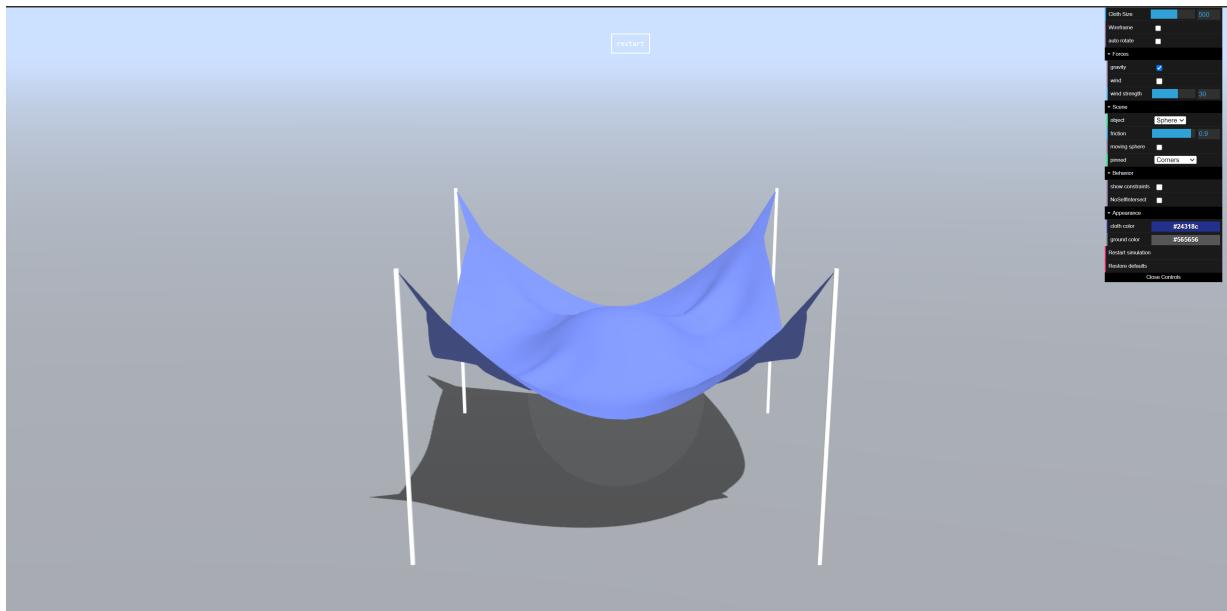


Рисунок 3.1 – Интерфейс ПО (вид по умолчанию)

В правом верхнем углу находится меню с настройками ткани и внешних сил. Для скрытия меню используется латинская буква «H». Выпадающее меню представлено на рисунке 3.2.

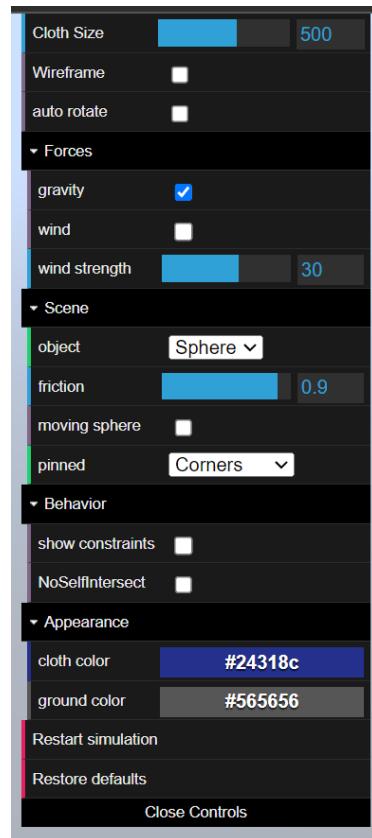


Рисунок 3.2 – Интерфейс ПО (выпадающее меню)

Пример измененных настроек представлен на рисунке 3.3.

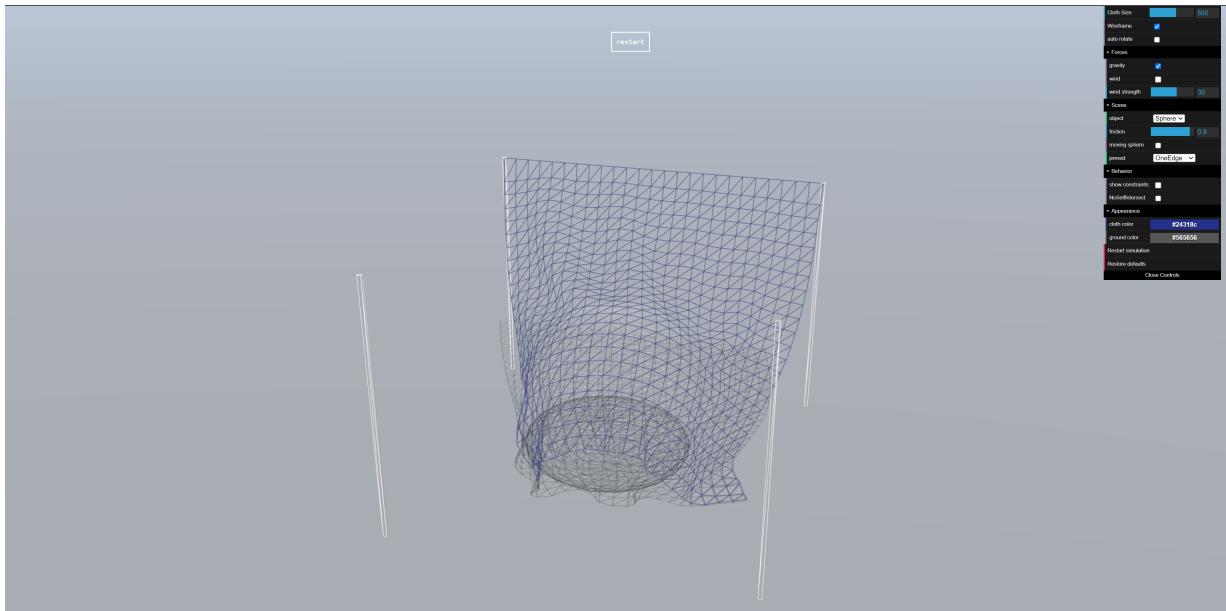


Рисунок 3.3 – Интерфейс ПО (пример работы программы)

## Вывод

В данном разделе были выбраны и обоснованы средства реализации, описана структура программы, а также рассмотрен интерфейс программного обеспечения.

## 4 Исследовательская часть

В данном разделе приведены технические характеристики устройства, на котором проводился анализ производительности, а также результаты работы программного обеспечения.

## 4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование, следующие.

- Операционная система Windows 11 [?].
  - Оперативная память: 16 ГБ.
  - Процессор: Intel(R) Core(TM) i7-10700F CPU @ 2.90 ГГц [?].
  - Количество ядер: 8 физических и 16 логических.

## 4.2 Результаты работы ПО

На рисунках 4.1 – 4.4 представлены изображения, полученные с помощью разработанного ПО.

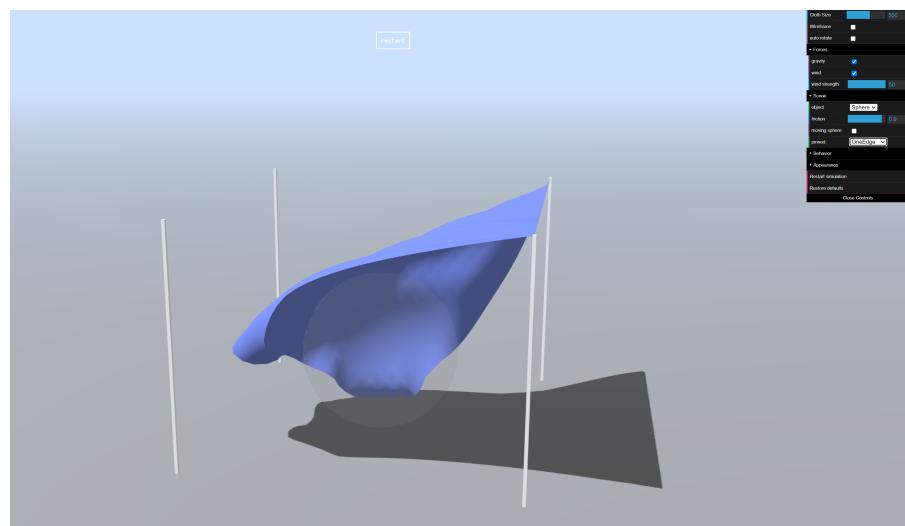


Рисунок 4.1 – Изображение №1, полученное с помощью разработанного ПО

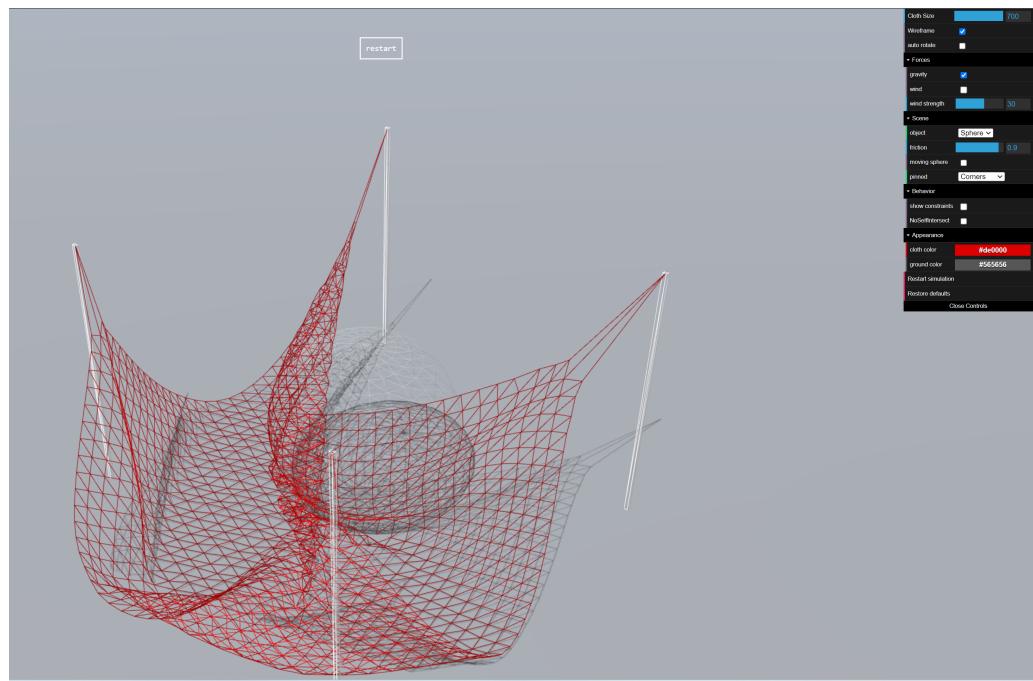


Рисунок 4.2 – Изображение №2, полученное с помощью разработанного ПО

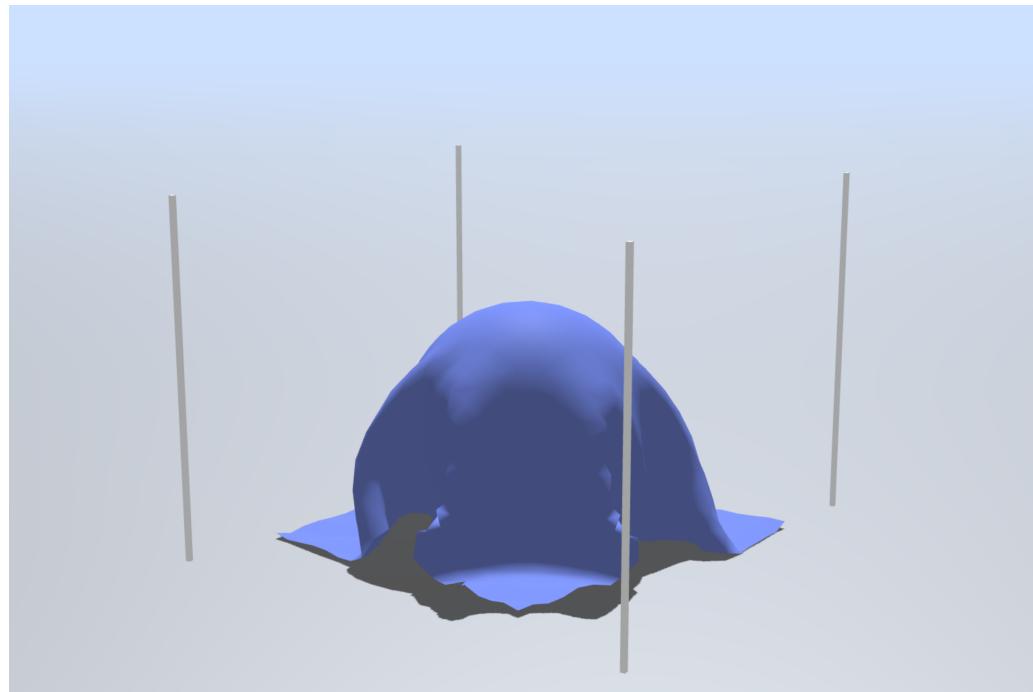


Рисунок 4.3 – Изображение №3, полученное с помощью разработанного ПО

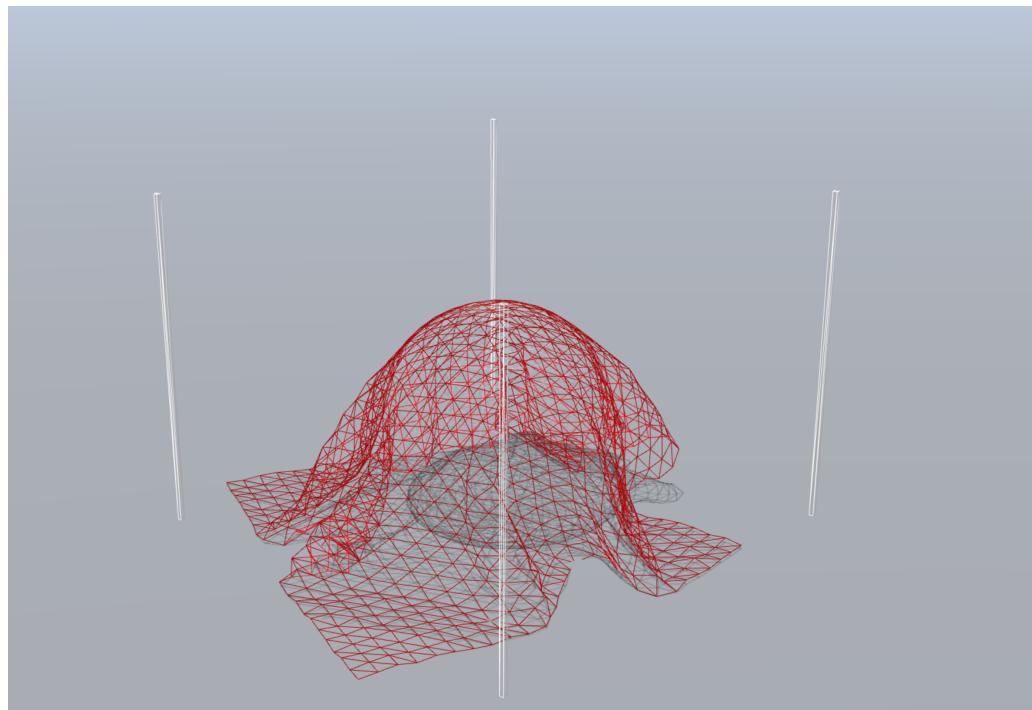


Рисунок 4.4 – Изображение №4, полученное с помощью разработанного ПО

### 4.3 Анализ производительности

Производительность будет оцениваться мерой количества кадров в секунду (FPS). Она будет меняться в зависимости от размера ткани. Количество кадров в секунду было замерено с помощью модуля *Stats*. Результаты приведены в таблице 4.1.

Таблица 4.1 – Производительность ПО при разном размере тканевой сетки.

Линейный размер ткани, частиц	FPS
100	60
500	60
1000	60
1500	60
2000	54
2500	49
5000	27

Исходя из результатов, можно заметить понижение количества кадров в секунду при большом размере тканевой сетки. Это происходит в силу боль-

шой вычислительной нагрузки, так как требуется расчет траектории для каждой частицы ткани.

## Вывод

В данном разделе приведен анализ производительности, а также результат работы программного обеспечения. В ходе анализа было установлено, что количество кадров в секунду уменьшается с увеличением размера ткани.

# Заключение

В ходе выполнения курсовой работы было разработано программное обеспечение, которое позволяет получить реалистичную модель физики ткани. Для достижения цели были выполнены следующие задачи:

- 1) проведён анализ методов представления ткани в трехмерном пространстве;
- 2) проведена формализация задачи построения кадра в виде IDEF0-диаграммы нулевого уровня;
- 3) реализованы необходимые алгоритмы;
- 4) спроектировано программное обеспечение для моделирования физики ткани;
- 5) был проведен анализ производительности разработанного программного обеспечения.

# Приложение А

Презентация к курсовой работе.