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
по дисциплине «Компьютерная графика»

Тема Визуализация трехмерной динамической сцены с ветром и управляемым осве

Студент Вавилова В. Л.

Группа ИУ7-54Б

Преподаватели Куров А. В., Русакова З. Н.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Динамические сцены с природными объектами, такими как поля подсолнухов, находят широкое применение в компьютерных играх и виртуальных средах. Они позволяют создавать атмосферу и усиливать погружение пользователя в виртуальный мир. Однако реализация таких сцен требует решения ряда задач, включая моделирование движения объектов под воздействием внешних факторов (например, ветра) и учёт освещения в зависимости от времени суток.

Целью данной работы является разработка программного обеспечения для построения трёхмерной динамической сцены, состоящей из поля подсолнухов, которые качаются под воздействием ветра, и источника света, положение которого зависит от заданного пользователем времени суток.

Для достижения поставленной цели необходимо решить следующие задачи:

1) проанализировать предметую область;
2) проанализировать существующие подходы и алгоритмы для моделирования и визуализации природных объектов;
3) выбрать средства реализации программного обеспечения;
4) создать программное обеспечение;
5) создать систему освещения, которая будет изменяться в зависимости от времени суток, заданного пользователем;
6) разработать пользовательский интерфейс для настройки параметров сцены;
7) провести тестирование и исследование разработанного программного обеспечения для оценки его производительности.

1 Аналитическая часть

1.1 Формализация задачи

Объектами сцены являются:

1) **подсолнухи**;
   - — высота подсолнухов;
   - — количество подсолнухов на поле;
   - — амплитуда колебания подсолнухов под действием ветра;
   - — частота колебаний подсолнухов под действием ветра (количество колебаний в секунду).
2) **поле**;
   - — ширина поля;
   - — длина поля;
   - — характеристики поверхности:
     - – рассеянное отражение;
     - – диффузное отражение.
3) **источник света (солнце)**;
   - — расположение: определяется по вертикальной траектории, имеющей форму круга.
4) **наблюдатель (камера)**.
   - — расположение;
   - — угол обзора.

Формализация задачи в виде IDEF0 представлена на рисунке **??**

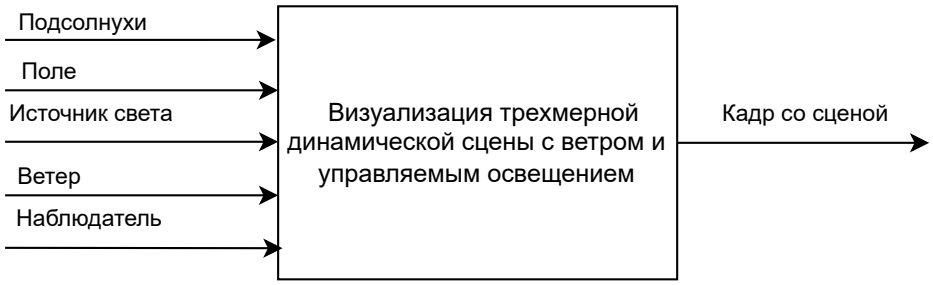

Рисунок 1.1 — Формализациязадачи в виде IDEF0.

1.2 Алгоритмы для моделирования и визуализации природных объектов

Для моделирования и визуализации природных объектов, таких как подсолнухи, существуют различные подходы и алгоритмы. Некоторые из них представлены ниже.

1.2.1 Полигональное 3D-моделирование

Полигональное моделирование — это метод создания трехмерных моделей с помощью полигонов, которые представляют собой плоские многоугольники, обычно треугольники или четырехугольники. Эти полигоны образуют сетку, которая формирует контур и поверхность объекта. citelit1

Преимущества:

— высокая детализация: чем больше полигонов, тем более детализированной становится модель.

Недостатки:

— сложность: требует значительного количества времени и усилий для создания сложных моделей;

— ограничения: может быть неэффективным для объектов с сложной внутренней структурой.

1.2.2 Методы рендеринга

Scanline Метод рендеринга, который обрабатывает изображение построчно.

Raytrace Этот метод симулирует путь света в реальном мире, отслеживая лучи, которые падают на объекты сцены и отражаются от них.

Raycasting Упрощенная версия Raytrace, которая рассчитывает только первое столкновение луча с поверхностью.

1.2.3 Выбор метода моделирования

Для визуализации поля с подсолнухами было выбрано полигональное 3D-моделирование. Оно позволяет создавать детальные модели подсолнухов. Для рендеринга сцены был выбран метод Raycasting для баланса между скоростью и

качеством изображения.

1.3 Модели освещения

В компьютерной графике модели освещения используются для имитации световых эффектов, когда свет аппроксимируется на основе физических законов.

Основными параметрами, определяющими освещение, являются:

— **свойства источников света**, включая их интенсивность и цвет,

— **свойства материала объекта**, такие как коэффициенты отражения, поглощения и пропускания света,

— **взаимодействие света с другими объектами** в сцене, например, затенение и переотражение,

— **цвет самой поверхности**, который определяется текстурами или параметрами материала.

1.3.1 Модель Ламберта

Модель Ламберта используется для расчета диффузного освещения. Формула освещения имеет вид:

$$I = I_a + I_d \cdot \max(0, \vec{L} \cdot \vec{N}),$$

где I_a — фоновая компонента, I_d — интенсивность источника света, $\vec{L}$ — вектор направления света, $\vec{N}$ — нормаль к поверхности.

Достоинства:

— простота реализации,

— высокая скорость вычислений.

Недостатки:

— не учитывает зеркальные отражения, что делает поверхность "матовой".

1.3.2 Модель Фонга

Модель Фонга включает три компоненты: фоновую, диффузную и зеркальную. Формула имеет вид:

$$I = I_a + I_d \cdot \max(0, \vec{L} \cdot \vec{N}) + I_s \cdot \max(0, \vec{R} \cdot \vec{V})^n,$$

где I_s — интенсивность зеркального света, $\vec{R}$ — отражённый вектор, $\vec{V}$ — направление к наблюдателю, n — параметр блеска.

Достоинства:

— реализует блеск и блики,

— подходит для гладких поверхностей.

Недостатки:

— зеркальная компонента может давать нереалистичные результаты при больших значениях параметра блеска.

1.3.3 Модель Блинна-Фонга

Модель Блинна-Фонга модифицирует зеркальную компоненту модели Фонга. Вместо отражённого вектора $\vec{R}$ используется вектор полупути $\vec{H}$, рассчитываемый как среднее между векторами света и наблюдателя:

$$\vec{H} = \frac{\vec{L} + \vec{V}}{\|\vec{L} + \vec{V}\|}.$$

Формула освещения:

$$I = I_a + I_d \cdot \max(0, \vec{L} \cdot \vec{N}) + I_s \cdot \max(0, \vec{N} \cdot \vec{H})^n.$$

Достоинства:

— более естественные блики,

— ускорение расчётов.

Недостатки:

— не подходит для анизотропных поверхностей.

1.3.4 Модификация Wrap-around

Модель Wrap-around улучшает модель Ламберта, добавляя дополнительный параметр k, который позволяет источнику света освещать поверхность даже в затенённых областях:

$$I_d = \max(k + (1 - k) \cdot (\vec{L} \cdot \vec{N}), 0).$$

Достоинства:

— улучшает реалистичность теней.

Недостатки:

— увеличивает сложность расчётов.

1.3.5 Выбор модели

Для данной задачи подходящей является **модель Блинна-Фонга**. Она обеспечивает баланс между реализмом и производительностью.

Эта модель:

1) учитывает как диффузное, так и зеркальное освещение, что важно для создания реалистичного вида подсолнухов,
2) даёт плавное распределение бликов,
3) эффективнее модели Фонга за счёт упрощения расчёта зеркальной компоненты.

Модель Ламберта слишком проста и не учитывает блеск поверхности, что делает её неподходящей для сцены с естественным освещением. Модификация Wrap-around не нужна, так как не требуется учитывать освещение затенённых областей.

Таким образом, **модель Блинна-Фонга** оптимально соответствует требованиям проекта, обеспечивая реалистичное освещение подсолнухов и высокую производительность.

1.4 Алгоритмы удаления невидимых рёбер и поверхностей

Ниже представлены некоторые из наиболее распространённых алгоритмов, их преимущества и недостатки.

1.4.1 Алгоритм Робертса

Описание: этот алгоритм работает в объектном пространстве и удаляет рёбра или грани, которые экранируются самим телом или другими объектами сцены. Требует, чтобы объекты были выпуклыми.

Преимущества:

— **точность:** позволяет точно определить видимые рёбра;

— **простота реализации:** использует простые математические методы.

Недостатки:

— **высокая вычислительная сложность:** объём вычислений растёт как квадрат числа объектов;

— **ограничение на невыпуклые объекты:** требует разбиения невыпуклых объектов на выпуклые части [**?**, 1, 2].

1.4.2 Алгоритм Аппеля

Описание: вводит понятие количественной невидимости, когда рёбра классифицируются по количеству граней, их закрывающих. [3].

Преимущества:

— упрощение анализа: позволяет упростить анализ видимости рёбер.

Недостатки:

— ограниченное применение: в основном используется для контурных линий и не подходит для всех типов объектов

1.4.3 Z-буфер

Описание: работает в пространстве изображения и использует буфер глубины для определения видимости пикселей. Каждый пиксель проверяется на глубину, чтобы определить, какой объект ближе к наблюдателю. [3, 4]

Преимущества:

— эффективность: быстрый и широко поддерживается современными видеокартами.

Недостатки:

— ограниченная точность: может давать ошибки при определении видимости в сложных сценах.

1.4.4 BSP-деревья

Описание: используются для оптимизации рендеринга статических сцен путём разбиения пространства на более мелкие части и определения видимости объектов. [3]

Преимущества:

— эффективность: позволяет быстро исключать невидимые объекты;

— применение в сложных сценах: подходит для архитектурных и других статических сцен.

Недостатки:

— сложность реализации: требует значительных усилий для построения и обновления дерева;

— ограниченное применение: в основном используется для статических сцен.

1.4.5 Иерархический Z-буфер

Описание: разделяет z-буфер на более мелкие части для более эффективного удаления невидимых поверхностей. [3]

Преимущества:

— улучшенная эффективность: позволяет быстрее исключать невидимые треугольники;

— применение в сложных сценах: подходит для больших сцен с множеством объектов.

Недостатки:

— Сложность реализации: Требует ручного обновления пирамиды при добавлении новых объектов.

— Высокие требования к памяти: Для хранения иерархии z-буфера.

Алгоритмы	Поддержка сложных сцен	Поддержка дин. объектов	Требования к памяти
Алг. Робертса	Нет	Нет	Низкие
Алг. Аппеля	Нет	Нет	Низкие
Z-буфер	Да	Да	Средние
BSP-деревья	Да (для стат.)	Нет	Высокие
Иерарх. Z-буфер	Да	Да	Высокие

Таблица 1.1 — Сравнение алгоритмов удаления невидимых рёбер и поверхностей

Для реализации поставленной задачи наиболее подходящим методом является Z-буфер. Он поддерживает сложные сцены с динамиескими объектами.

1.5 Скелетная Анимация

Скелетная анимация — это метод создания движений в трехмерных моделях, который использует внутренний «скелет» для управления деформацией и движением объекта. Этот скелет состоит из костей и суставов, которые функционируют аналогично человеческому скелету, позволяя аниматорам создавать реалистичные и плавные движения citelit2.

1.5.1 Механизм работы костей

— **Создание скелета:** аниматоры создают набор костей, которые соединены между собой суставами. Каждая кость может двигаться относительно других костей, что позволяет создавать сложные движения. Например, для моделирования руки человека создаются кости для плеча, предплечья и кисти, соединенные суставами. Это позволяет анимировать руку, сгибая её в локте или вращая кисть.

— **Привязка модели (скиннинг):** модель привязывается к скелету, что позволяет каждому полигону модели быть привязанным к определённой кости. Этот процесс называется скиннингом. Каждая вершина модели может быть привязана к одной или нескольким костям с определёнными весами. Веса определяют, насколько сильно каждая кость влияет на вершину. Формула для вычисления новой позиции вершины:

$$\mathbf{v}' = \sum_{i=1}^{n} w_i \cdot \mathbf{T}_i \cdot \mathbf{v},$$

где:

- $\mathbf{v}$ — исходная позиция вершины,
- $\mathbf{v}'$ — новая позиция вершины,
- w_i — вес вершины для i-й кости,
- $\mathbf{T}_i$ — матрица преобразования i-й кости,
- n — количество костей, влияющих на вершину.

Этот процесс позволяет модели деформироваться в соответствии с движением костей.

— **Анимация:** движения костей определяются с помощью ключевых кадров, которые задают положение и вращение костей в определенные моменты времени. Программа интерполирует промежуточные кадры, создавая плавные движения. Для интерполяции часто используется линейная или сплайновая интерполяция. Формула линейной интерполяции:

$$\mathbf{q}(t) = (1 - t) \cdot \mathbf{q}_1 + t \cdot \mathbf{q}_2,$$

где:

- $\mathbf{q}_1$ и $\mathbf{q}_2$ — ключевые кадры,
- t — параметр интерполяции ($0 \leq t \leq 1$).

Этот подход позволяет аниматору задавать только ключевые позы, а программа автоматически вычисляет промежуточные состояния.

1.5.2 Визуализация ветра

Для симуляции воздействия ветра на объекты, такие как растения, можно использовать математическую модель, которая описывает силу ветра как функцию времени и пространства. Рассмотрим функцию ветра:

$$F(t) = A \cdot \sin(2\pi f t + \phi),$$

где:

— A — амплитуда силы ветра,

— f — частота колебаний,

— ϕ — фаза колебаний,

— t — время.

Эта функция описывает периодическое воздействие ветра. В таком случае, для симуляции колыхания стебля растения под воздействием ветра, угол поворота кости может быть вычислен по формуле:

$$\theta(t) = k \cdot F(t),$$

где:

— k — коэффициент, зависящий от гибкости кости,

— $F(t)$ — сила ветра в момент времени t.

Для визуализации ветра скелетная анимация может быть полезна по нескольким причинам:

— **Реализм движений:** скелетная анимация позволяет создавать реалистичные и плавные движения.

— **Эффективность:** после создания скелета и привязки модели анимацию можно быстро создавать, используя инструменты интерполяции между ключевыми кадрами. Это ускоряет процесс анимации.

— **Гибкость:** скелетная анимация позволяет легко изменять позы и движения объектов, что дает воможность легко управлять положением отдельных частей модели для создания более сложной анимации.

1.5.3 Применение в визуализации ветра

Хотя скелетная анимация обычно используется для персонажей, она также может быть применена к растениям или другим объектам, на которые воздействуют внешние силы (например, ветер). Для этого создается скелет, который имитирует стебли растения, и привязывается к ним модель листьев и других элементов. Движения костей позволяют симулировать колыхание растений под воздействием ветра, создавая реалистичную и динамичную сцену.

1.6 Описание модели подсолнуха

Для выполнения поставленной задачи визуализации трехмерной сцены поля подсолнухов была выбрана модель подсолнуха в формате FBX. Данная модель была выбрана благодаря своей высокой детализации и наличию встроенного скелета, состоящего из костей.

На рисунке 1.2 представлена модель подсолнуха, загруженная в программу Blender. На изображении отображены основные компоненты модели, включая геометрию объекта и структуру скелета, состоящего из костей, которые используются для анимации.

На рисунке 1.3 показана модель подсолнуха в процессе выполнения программы.

Рисунок 1.2 — Модель подсолнуха, загруженная в программу Blender

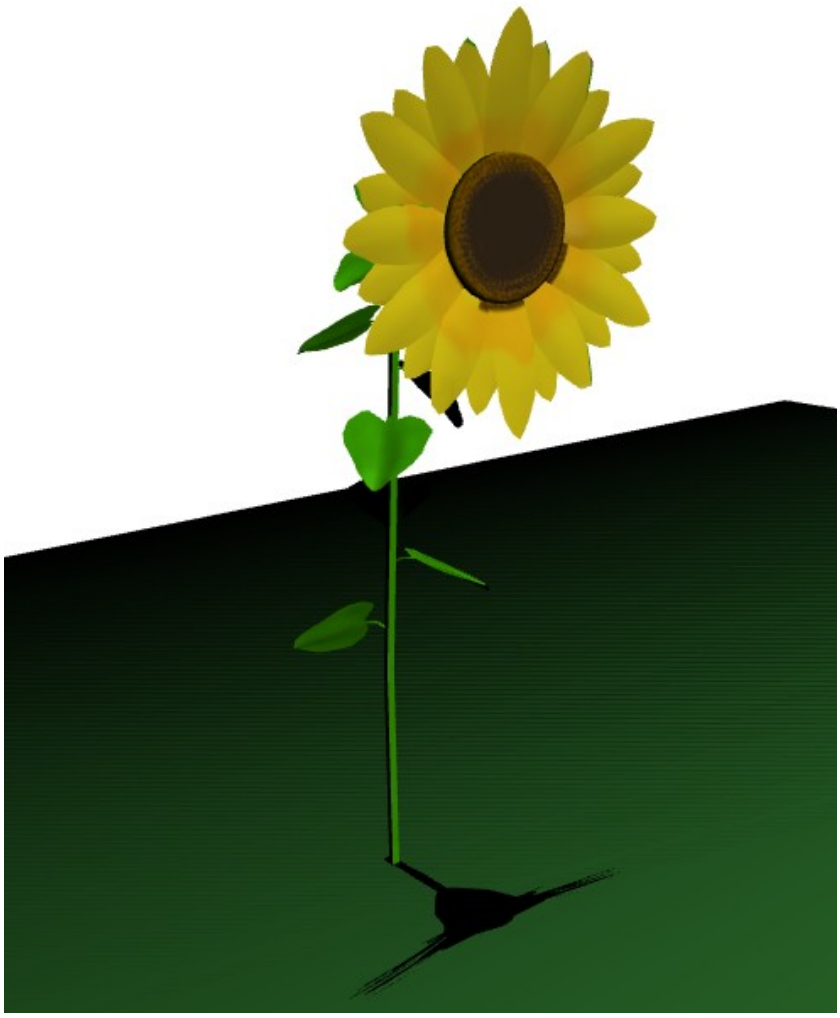

Рисунок 1.3 — Модель подсолнуха в процессе выполнения программы.

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма Z-буфера.

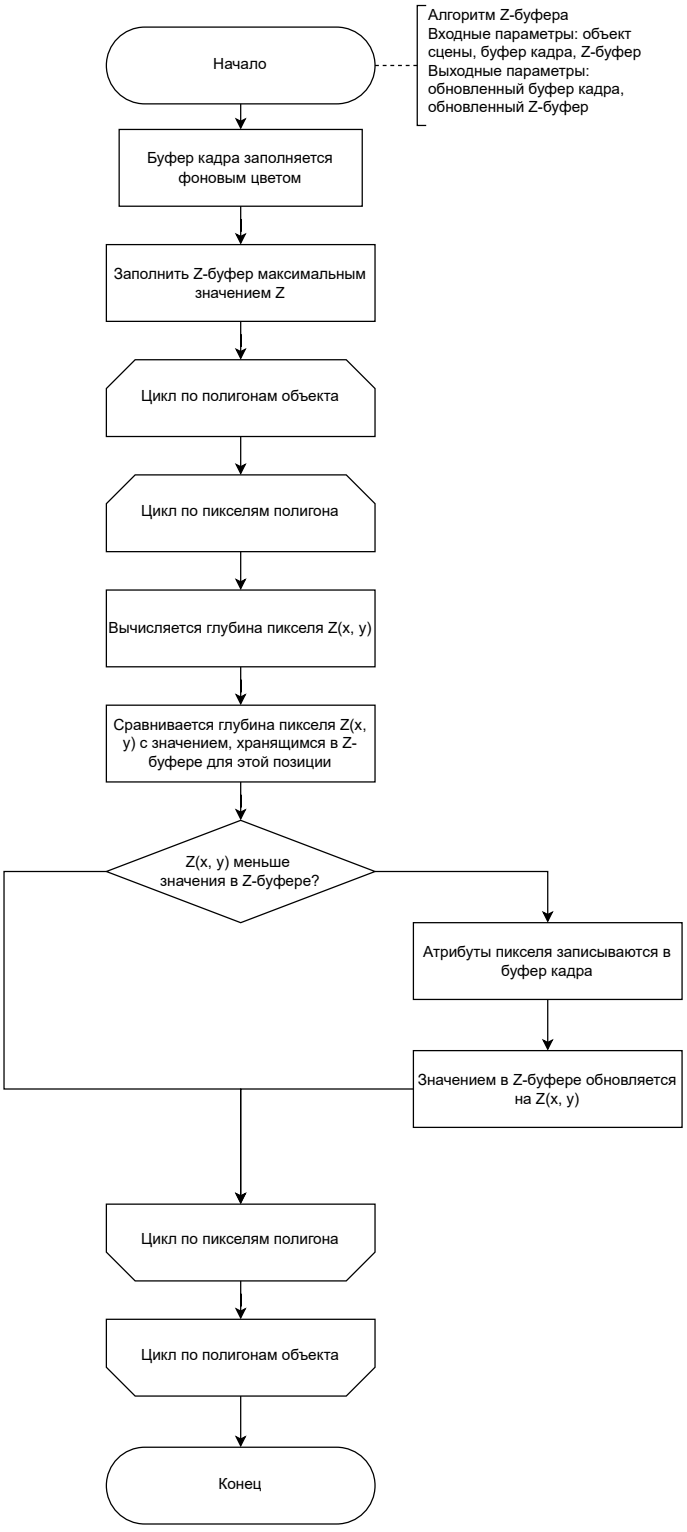

Рисунок 2.1 — Схема алгоритма Z-буфера.

На рисунках 2.2, 2.3 представлена схема алгоритма реализации ветра.

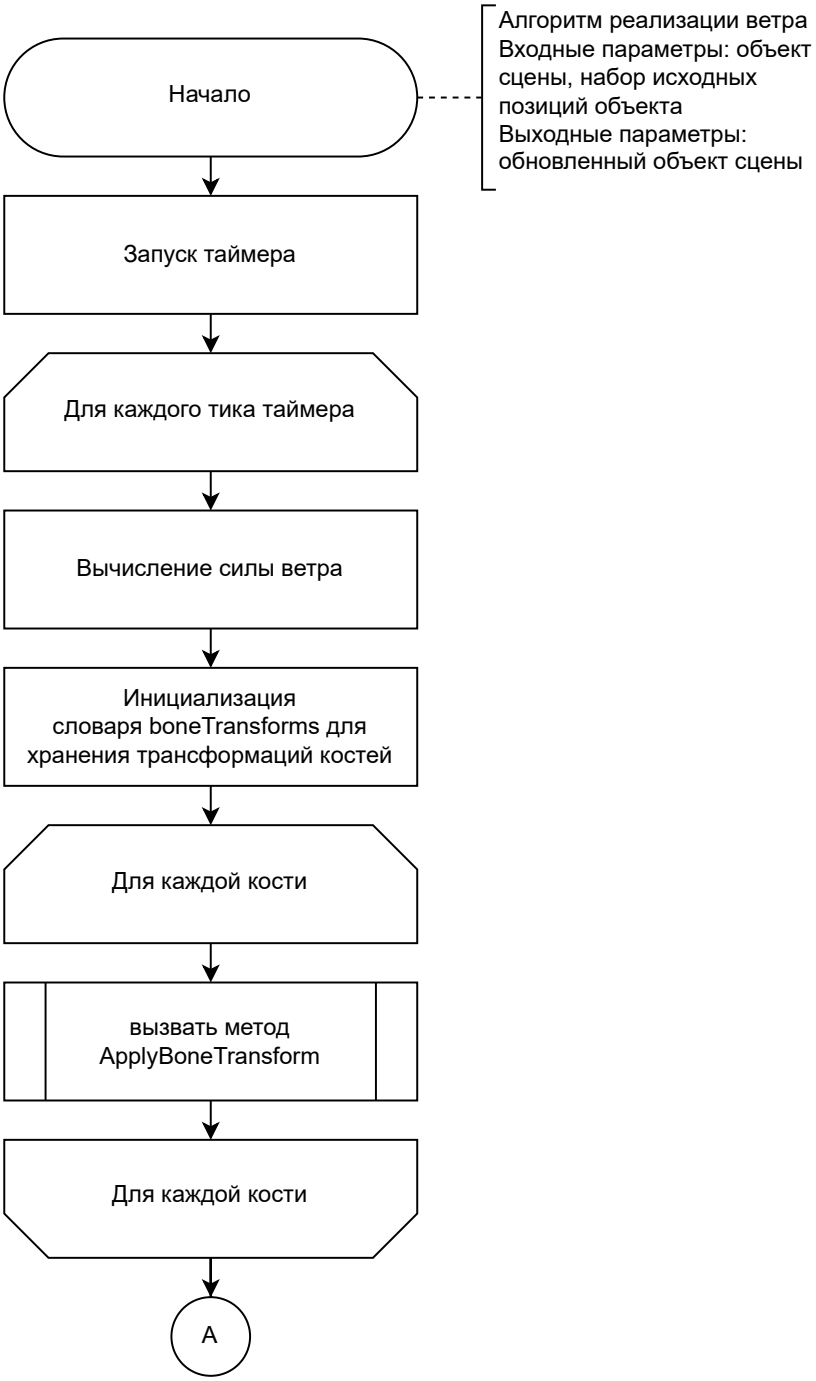

Рисунок 2.2 — Схема алгоритма реализации ветра. Часть 1.

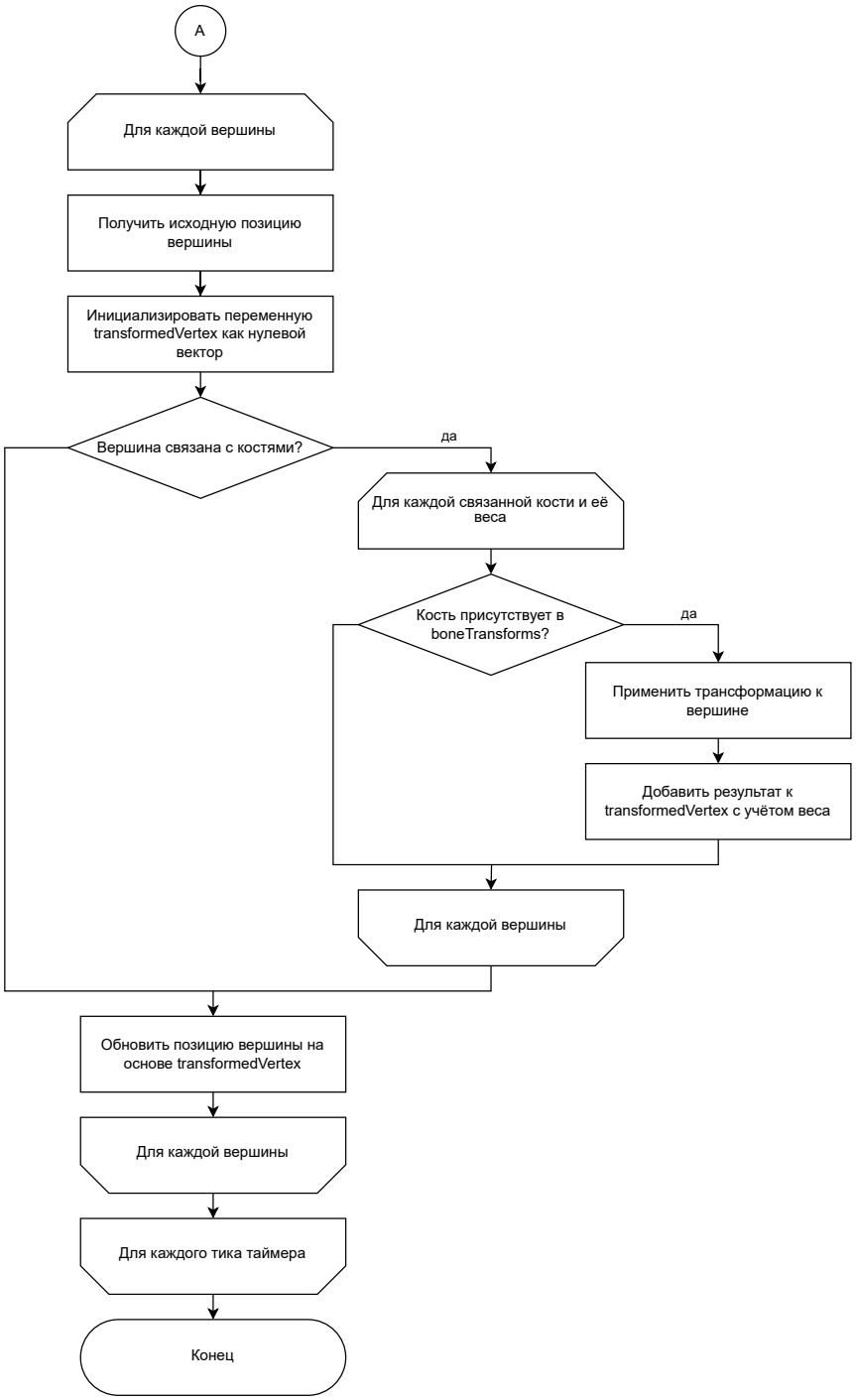

Рисунок 2.3 — Схема алгоритма реализации ветра. Часть 2.

На рисунке 2.4 представлен метод для применения вращения к вершинам кости.

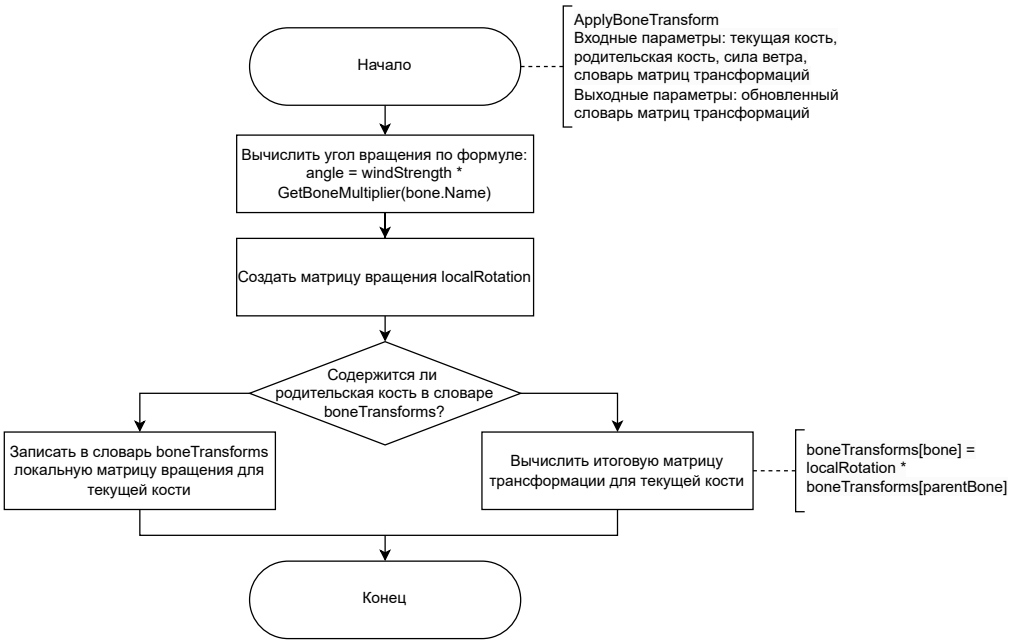

Рисунок 2.4 — Метод для применения вращения к вершинам кости.

На рисунке 2.5 представлена схема алгоритма рендеринга методом Raycasting.

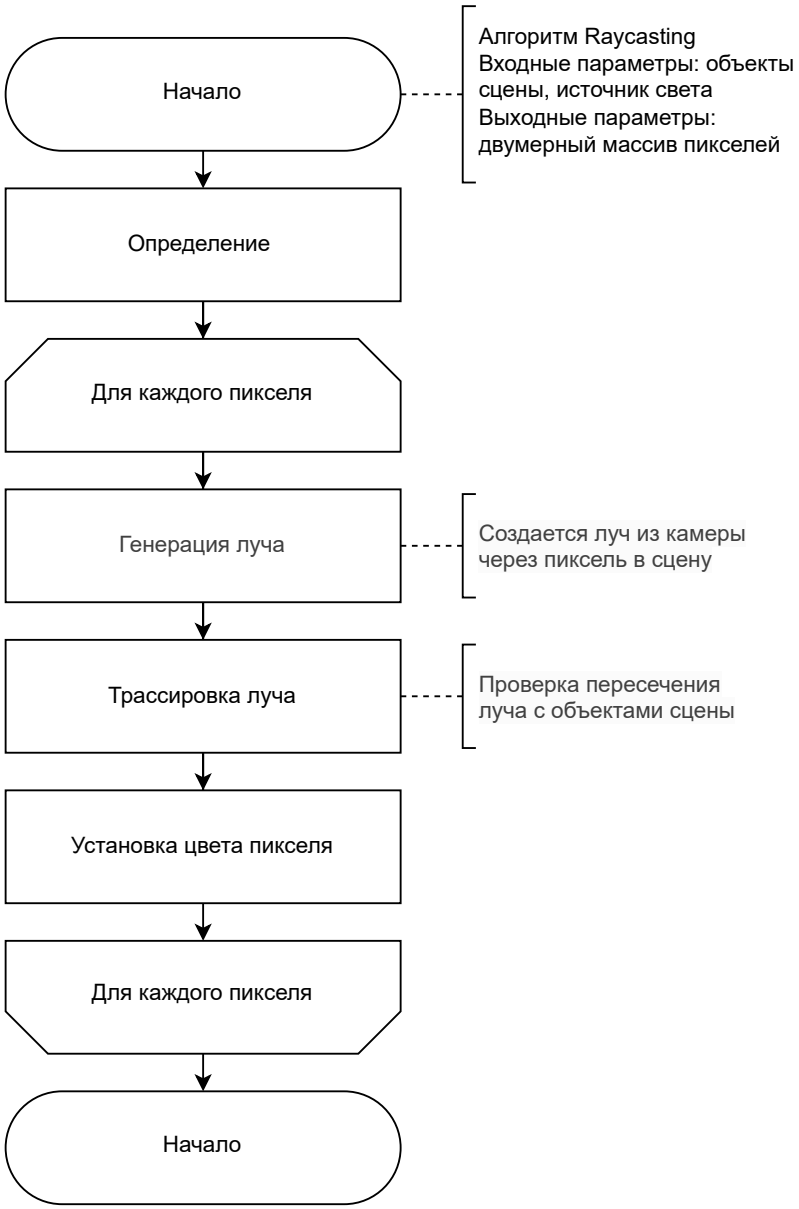

Рисунок 2.5 — Схема алгоритма рендеринга методом Raycasting.

2.2 Вывод

В данном разделе были представлены схемы используемых алгоритмов.

3 Технологическая часть

В данном разделе были рассмотрены средства, использованные для разработки, а также приведены коды реализованных алгоритмов.

3.1 Средства реализации

Выбор языка программирования C# для реализации курсовой работы обусловлен его техническими возможностями и поддержкой объектно-ориентированного программирования. Также обеспечивание высокой производительности благодаря компиляции в промежуточный язык и широкая документация упрощают процесс разработки, а автоматическое управление памятью снижает риск ошибок, связанных с утечками ресурсов.

3.2 Программный интерфейс

На рисунке 3.1 представлен графический интерфейс программы.

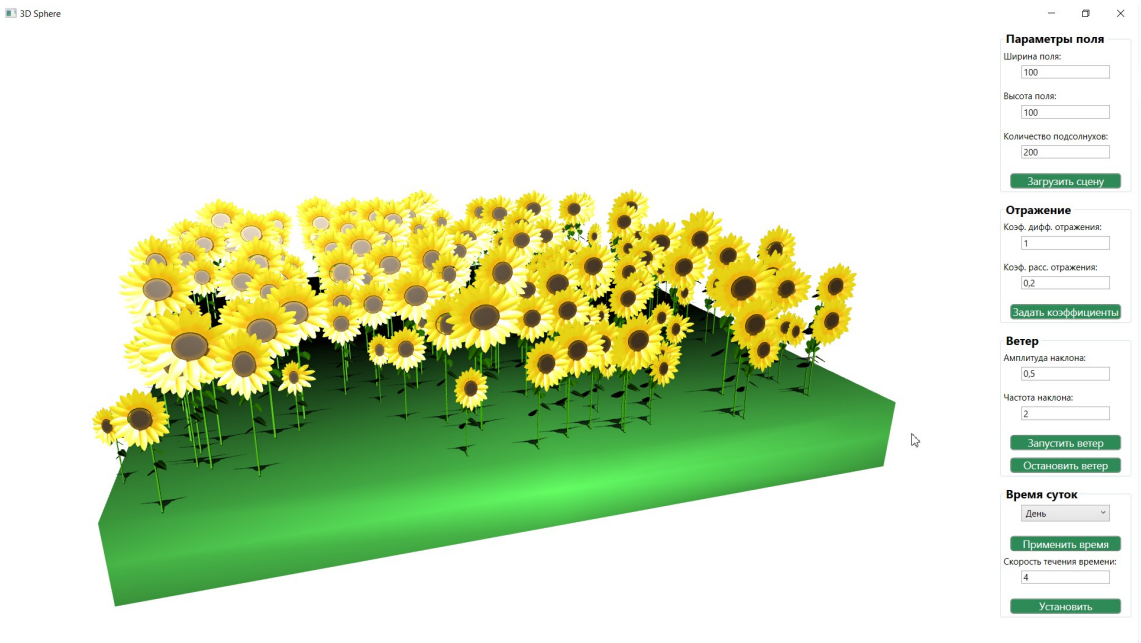

Рисунок 3.1 — Графический интерфейс программы.

На рисунках 3.2, 3.3, 3.4, 3.5 представлены результаты работы программы при различных заданных пользователем временах суток: утро, день, вечер и ночь соответственно.

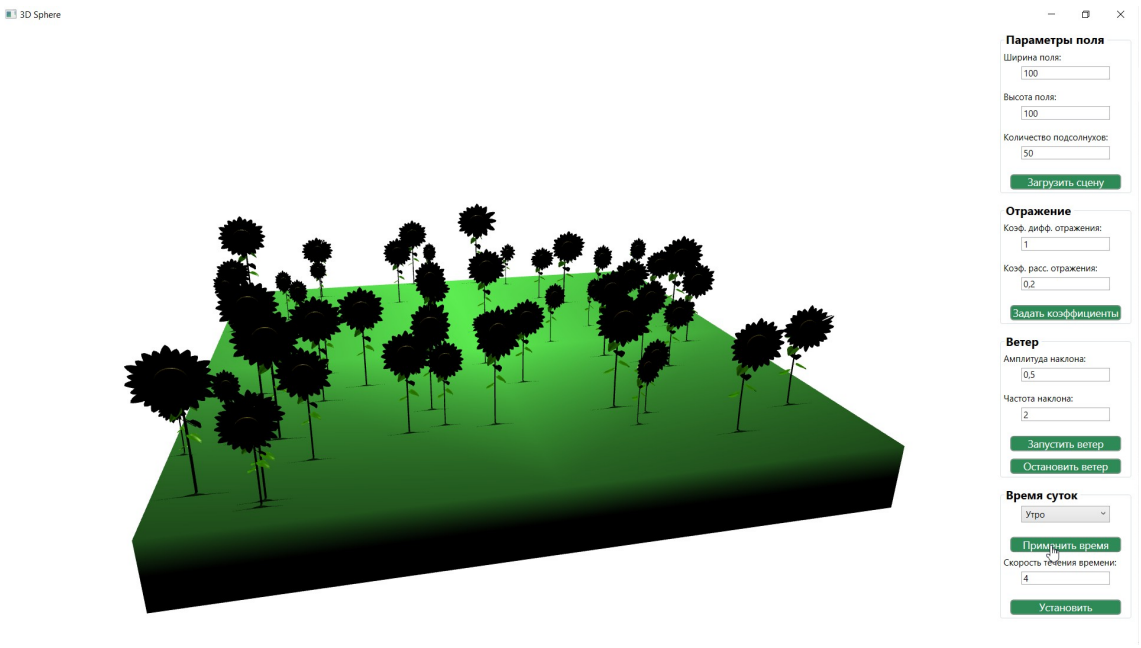

Рисунок 3.2 — Графический интерфейс программы.

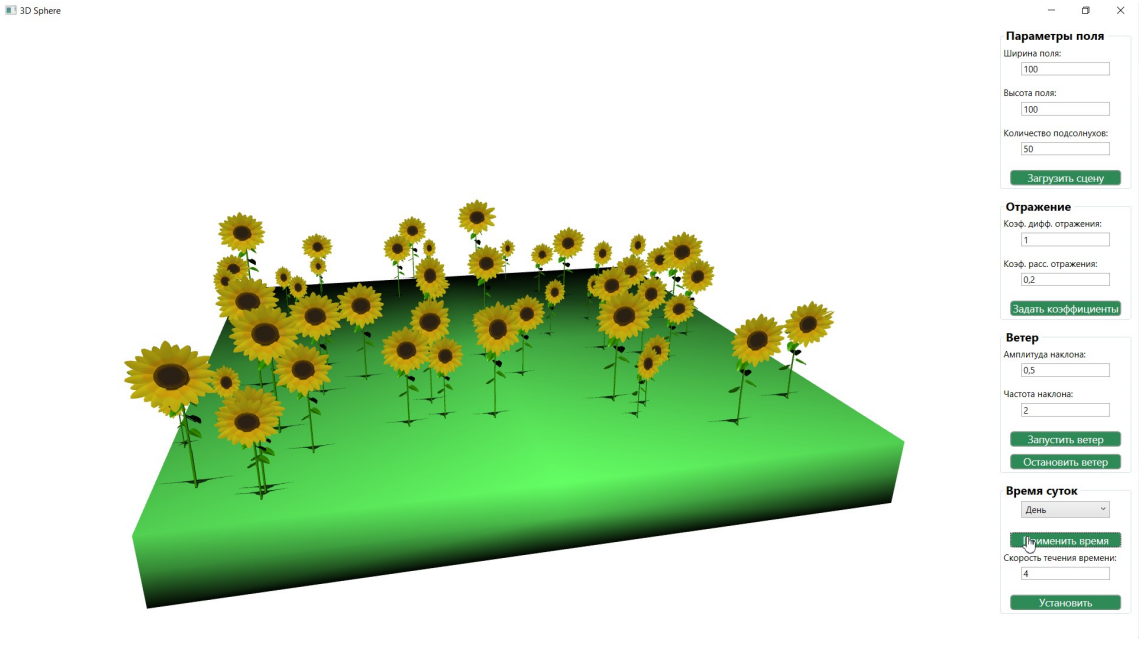

Рисунок 3.3 — Графический интерфейс программы.

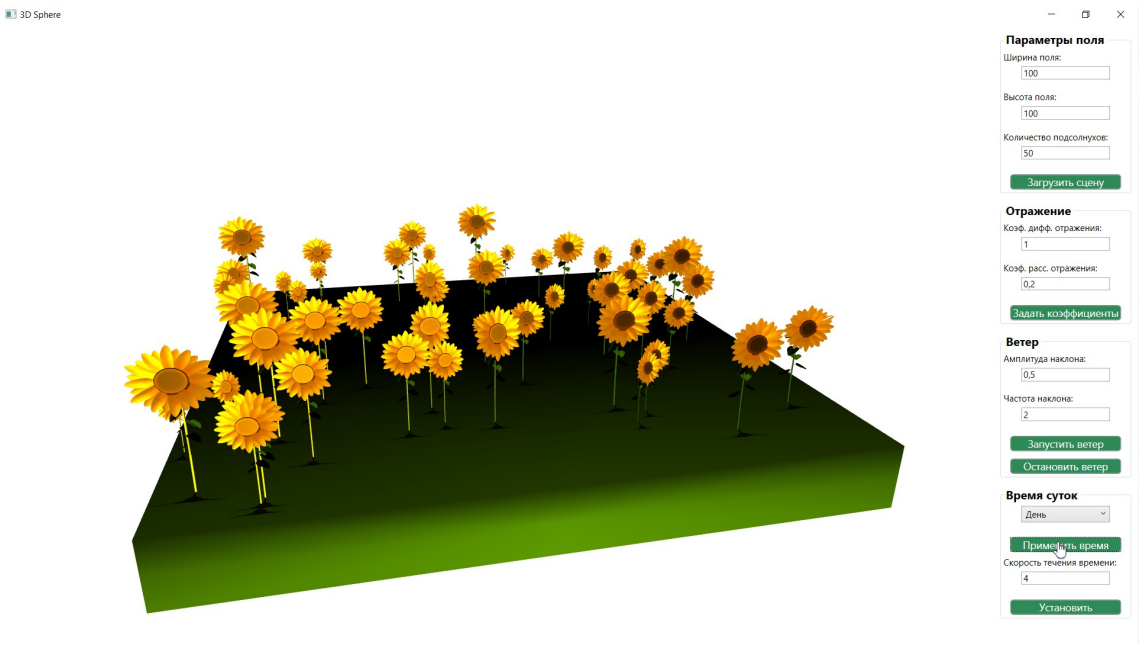

Рисунок 3.4 — Графический интерфейс программы.

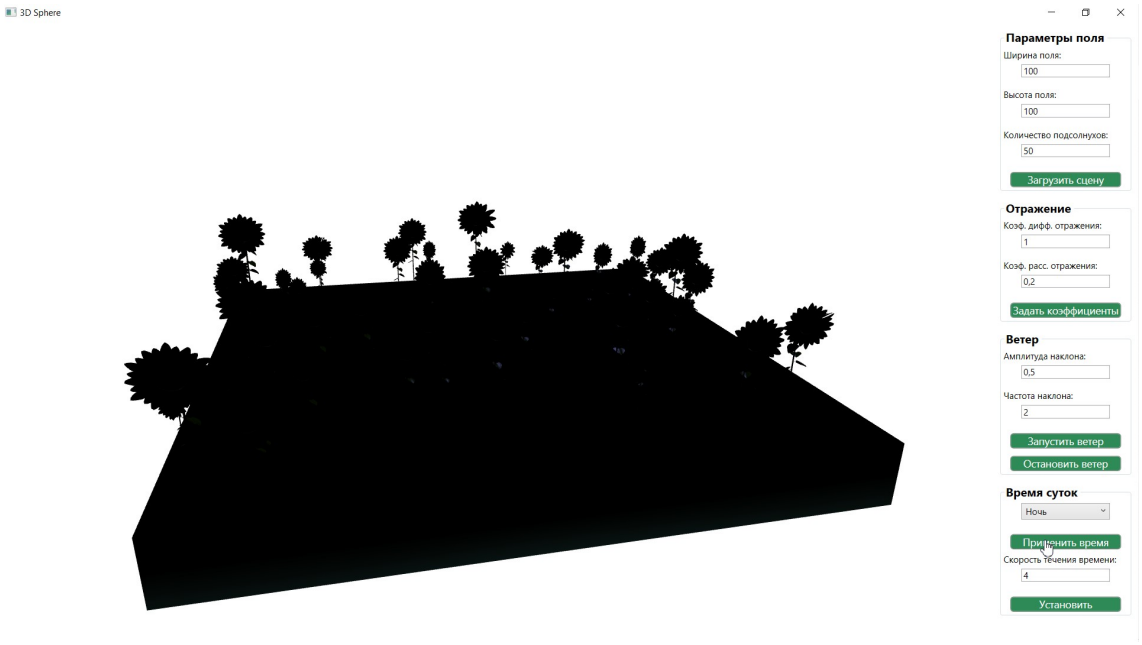

Рисунок 3.5 — Графический интерфейс программы.

3.2.1 Реализация алгоритмов

В листинге 3.3 представлен алгоритм применения ветра к модели.

```
1  private Transform3D CalculateTransform(GeometryModel3D geometryModel)
2          {
3              var meshGeometry = geometryModel.Geometry as MeshGeometry3D;
4              if (meshGeometry == null) return geometryModel.Transform;
5
```

```
6               var transform = new Transform3DGroup();
7               transform.Children.Add(_originalTransforms[geometryModel]);
8
9               double windStrength = _amplitude * Math.Sin(_timeElapsed *
                    _frequency * 2 * Math.PI);
10
11              var boneTransforms = new Dictionary<Bone, Matrix3D>();
12
13              var bone00 = _bones.FirstOrDefault(b => b.Name == "root");
14              var bone01 = _bones.FirstOrDefault(b => b.Name ==
                    "Sunflower01");
15              var bone02 = _bones.FirstOrDefault(b => b.Name ==
                    "Sunflower02");
16              var bone03 = _bones.FirstOrDefault(b => b.Name ==
                    "Sunflower03");
17              var bone04 = _bones.FirstOrDefault(b => b.Name ==
                    "Sunflower04");
18              var bone05 = _bones.FirstOrDefault(b => b.Name ==
                    "Sunflower05");
19              var bone5 = _bones.FirstOrDefault(b => b.Name ==
                    "Sunflower5");
20
21              ApplyBoneTransform(bone00, null, windStrength,
                    boneTransforms);
22              ApplyBoneTransform(bone01, bone00, windStrength,
                    boneTransforms);
23              ApplyBoneTransform(bone02, bone01, windStrength,
                    boneTransforms);
24              ApplyBoneTransform(bone03, bone02, windStrength,
                    boneTransforms);
25              ApplyBoneTransform(bone04, bone03, windStrength,
                    boneTransforms);
26              ApplyBoneTransform(bone05, bone04, windStrength,
                    boneTransforms);
27              ApplyBoneTransform(bone5, bone05, windStrength,
                    boneTransforms);
28
29              foreach (var bone in _bones)
30              {
31                  if (boneTransforms.ContainsKey(bone))
32                  {
33                      ApplyRotationToBone(meshGeometry, bone,
                            boneTransforms[bone]);
34                  }
35              }
36
```

```
37              var newMeshGeometry = new MeshGeometry3D
38              {
39                  Positions = meshGeometry.Positions,
40                  TriangleIndices = meshGeometry.TriangleIndices,
41                  Normals = meshGeometry.Normals,
42                  TextureCoordinates = meshGeometry.TextureCoordinates
43              };
44
45              geometryModel.Geometry = newMeshGeometry;
46              return transform;
47          }
```

Листинг 3.1 — Алгоритм применения ветра к модели

В листинге 3.3 представлен метод для применения вращения к вершинам кости.

```
1   private void ApplyBoneTransform(Bone bone, Bone parentBone, double
        windStrength, Dictionary<Bone, Matrix3D> boneTransforms)
2           {
3               if (bone == null) return;
4
5               double angle = windStrength * GetBoneMultiplier(bone.Name);
6               var localRotation = new RotateTransform3D(new
                    AxisAngleRotation3D(new Vector3D(1, 0, 0), angle)).Value;
7
8
9               if (parentBone != null &&
                    boneTransforms.ContainsKey(parentBone))
10              {
11                  boneTransforms[bone] = localRotation *
                        boneTransforms[parentBone];
12              }
13              else
14              {
15                  boneTransforms[bone] = localRotation;
16              }
17          }
```

Листинг 3.2 — Метод для применения вращения к вершинам кости

В листинге 3.3 представлен алгоритм преобразования координат вершин трехмерной модели, связанных с определенной "костью"(bone), с использованием заданной матрицы поворота.

```
1  private void ApplyRotationToBone(MeshGeometry3D meshGeometry, Bone bone,
       Matrix3D rotationMatrix)
2          {
3              foreach (var vertexIndex in bone.ConnectedVertices)
4              {
5                  if (vertexIndex < meshGeometry.Positions.Count)
6                  {
7                      var originalPosition =
                           meshGeometry.Positions[vertexIndex];
8                      var newPosition =
                           rotationMatrix.Transform(originalPosition);
9                      meshGeometry.Positions[vertexIndex] = newPosition;
10                 }
11             }
12         }
```

Листинг 3.3 — Алгоритм преобразования координат вершин модели

В листинге 3.4 представлен тест для исследования времени загрузки сцены в зависимости от количества подсолнухов.

```
1  public void TestSunflowerFieldLoading(object sender, RoutedEventArgs e)
2          {
3              double width = 100.0;
4              double height = 100.0;
5
6              for (int countSunflowers = 1; countSunflowers <= 401;
                   countSunflowers += 20)
7              {
8                  ClearScene();
9
10                 Stopwatch stopwatch = Stopwatch.StartNew();
11
12                 _sunflowerField = new SunflowerField(width, height,
                       _fieldPath, countSunflowers, _sunflowerModelPath,
                       _sunflowerTexturePath);
13
14                 stopwatch.Stop();
15
16                 Console.WriteLine($"countSunflowers: {countSunflowers},
                       Time: {stopwatch.Elapsed.TotalMilliseconds} ms");
17             }
18         }
```

Листинг 3.4 — Тест для исследования времени загрузки сцены в зависимости от количества подсолнухов

В листинге 3.5 представлен тест для исследования времени загрузки сцены в зависимости от длины и ширины поля.

```
public void TestSunflowerFieldLoadingWithVariableSceneSize(object
    sender, RoutedEventArgs e)
        {
            int countSunflowers = 50;

            for (double width = 10; width <= 2010; width += 200)
            {
                for (double height = 10; height <= 2010; height += 200)
                {
                    ClearScene();

                    Stopwatch stopwatch = Stopwatch.StartNew();

                    _sunflowerField = new SunflowerField(width, height,
                        _fieldPath, countSunflowers, _sunflowerModelPath,
                        _sunflowerTexturePath);

                    stopwatch.Stop();

                    Console.WriteLine($"Width: {width}, Length:
                        {height}, Time downloading
                        {stopwatch.Elapsed.TotalMilliseconds} ms");
                }
            }
        }
```

Листинг 3.5 — Тест для исследования времени загрузки сцены в зависимости от длины и ширины поля

В листинге 3.6 представлен тест для исследования загруженности процессора при включенном ветре и различном количестве подсолнухов.

```
public void TestWindPerformance(object sender, RoutedEventArgs e)
        {
            double width = 100.0;
```

```
4               double height = 100.0;
5
6               for (int countSunflowers = 1; countSunflowers <= 401;
                    countSunflowers += 20)
7               {
8                   ClearScene();
9
10                  _sunflowerField = new SunflowerField(width, height,
                        _fieldPath, countSunflowers, _sunflowerModelPath,
                        _sunflowerTexturePath);
11
12                  _sunflowerField.StartWind(amplitude: 0.1, frequency:
                        0.5);
13                  double cpuUsage = MeasureCpuUsage(duration: 10000);
14
15                  _sunflowerField.StopWind();
16
17                  Console.WriteLine($"countSunflowers: {countSunflowers},
                        CPU usage: {cpuUsage:F2}%");
18              }
19          }
```

Листинг 3.6 — Тест для исследования загруженности процессора при включенном ветре и различном количестве подсолнухов.

Все тесты были пройдены успешно.

Вывод

В технологической части была рассмотрена реализация программного обеспечения для моделирования поля подсолнухов с учетом ветра. Тестирование показало эффективность программы, а также выявило возможности для дальнейшей оптимизации.

4 Исследовательская часть

Технические характеристики устройства:

— процессор: Intel(R) Core(TM) i5-10300H с тактовой частотой 2500000000 герц;

— ядра: 4;

— логические процессоры: 8;

— оперативная система: Windows 10;

— оперативная память: 8 ГБ.

4.1 Анализ времени загрузки сцены в зависимости от количества подсолнухов

Таблица 4.1 — Время загрузки сцены в зависимости от количества подсолнухов

Количество подсолнухов	**Время загрузки (мс)**
1	118.23
21	229.69
41	471.19
61	760.25
81	1201.38
101	1101.29
121	1269.22
141	1907.86
161	2139.81
181	2201.83
201	2376.41
221	2935.29
241	2769.19
261	2993.32
281	3676.13
301	3488.16
321	3478.32
341	3964.03
361	4107.77
381	4485.53
401	4510.49

На рисунке 4.1 представлен график времени загрузки сцены в зависимости

от количества подсолнухов.

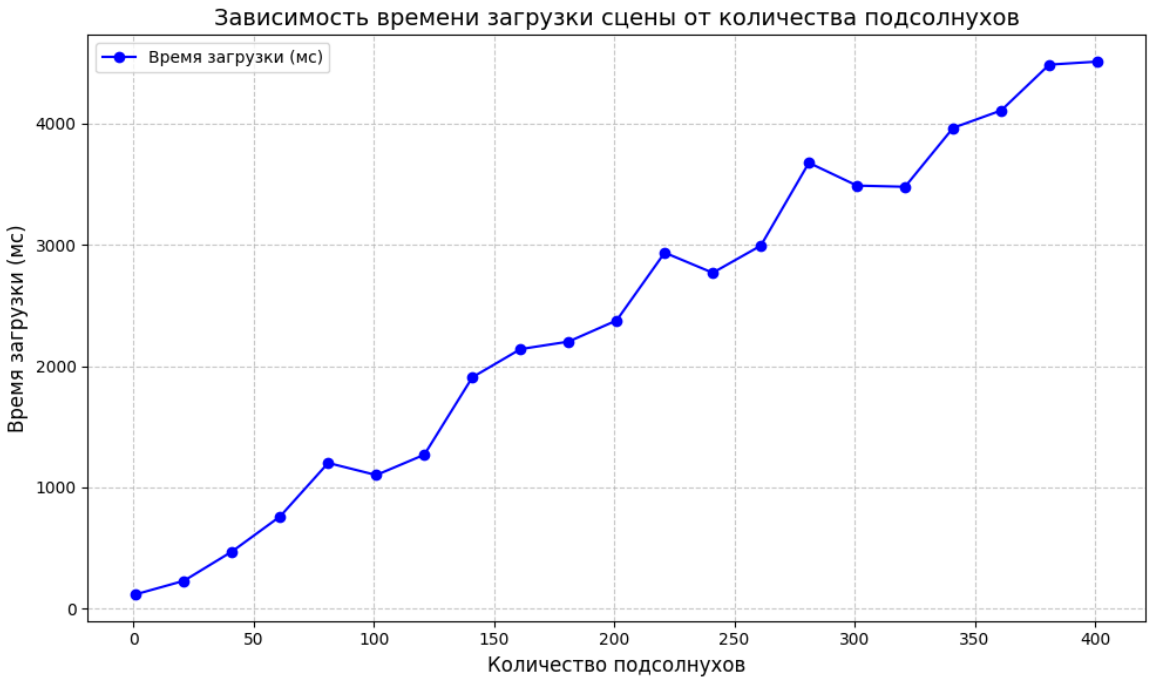

Рисунок 4.1 — График времени загрузки сцены в зависимости от количества подсолнухов.

4.1.1 Общая тенденция

— Время загрузки увеличивается с ростом количества подсолнухов. Это ожидаемо, так как больше объектов требуют больше ресурсов для создания и рендеринга.

4.1.2 Локальные колебания

— В некоторых случаях время загрузки может немного уменьшаться (например, для 101 подсолнуха время меньше, чем для 81). Это может быть связано с тем, что каждая модель подсолнуха имеет случайные характеристики.

4.1.3 Вывод

— Данные показывают, что время загрузки увеличивается с ростом количества подсолнухов, но могут наблюдаться локальные колебания из-за случайных факторов.

4.2 Анализ времени загрузки сцены в зависимости от размеров сцены

Таблица 4.2 — Время загрузки сцены (в мс) при различных размерах сцены

Ширина	10	210	410	610	810	1010	1210	1410	1610	1810	2010
10	732	600	616	588	520	606	567	515	577	685	579
210	545	541	596	601	556	577	833	551	556	708	523
410	666	536	625	511	549	533	556	592	780	505	682
610	532	679	569	548	586	513	518	542	515	542	569
810	508	556	538	535	509	549	534	548	561	520	566
1010	527	539	534	530	500	565	562	551	524	534	543
1210	590	600	520	516	514	527	531	543	543	507	499
1410	601	645	527	542	533	528	512	612	556	723	529
1610	566	548	570	525	496	877	788	782	536	530	536
1810	544	515	595	526	537	522	589	541	538	561	545
2010	490	539	669	690	560	565	527	531	502	572	552

На рисунке 4.2 представлена тепловая карта времени загрузки сцены в зависимости от её ширины и высоты.

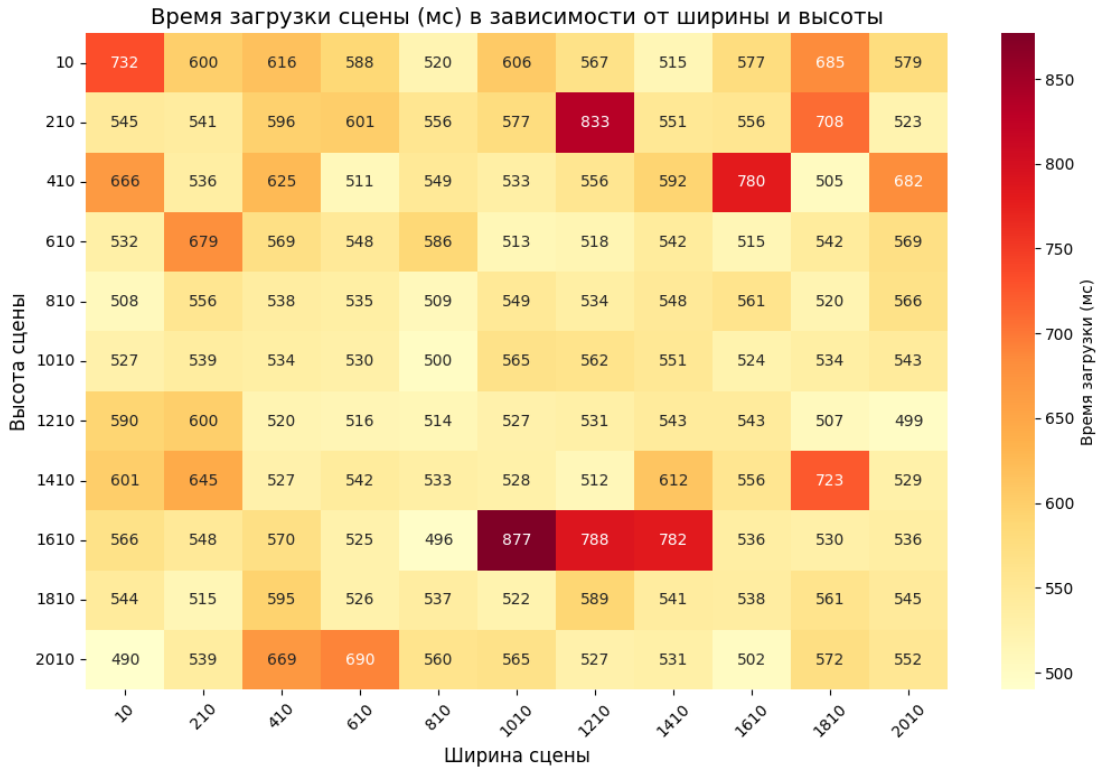

Рисунок 4.2 — Тепловая карта времени загрузки сцены в зависимости от её ширины и высоты.

4.2.1 Общая тенденция

— Время загрузки сцены варьируется в диапазоне от **489,74 мс** до **877,27 мс**.

— Наблюдается отсутствие явной линейной зависимости между размером сцены (шириной и длиной) и временем загрузки.

4.2.2 Зависимость от ширины и длины

— **Ширина сцены**:

  – для ширины 10 время загрузки варьируется от 515,48 мс до 731,87 мс;

  – для ширины 2010 время загрузки варьируется от 489,74 мс до 689,58 мс;

  – нет явной закономерности, что увеличение ширины приводит к увеличению времени загрузки.

— **Длина сцены**:

  – для длины 10 время загрузки варьируется от 489,74 мс до 731,87 мс;

  – для длины 2010 время загрузки варьируется от 498,59 мс до 684,53 мс;

  – аналогично, увеличение длины не всегда приводит к увеличению времени загрузки.

4.2.3 Аномальные значения

— Наблюдаются выбросы в данных:

  – **877,27 мс** (ширина: 1610, длина: 1010).

  – **833,34 мс** (ширина: 210, длина: 1210).

  – **781,50 мс** (ширина: 1610, длина: 1410).

— Эти значения могут быть связаны с различными причинами, такими как:

  – пиковая нагрузка на систему в момент загрузки;

  – особенности рендеринга для определённых размеров сцены;

  – фоновые процессы, которые могли повлиять на замеры.

4.2.4 Стабильность времени загрузки

— Для большинства сочетаний ширины и длины время загрузки находится в диапазоне 500–600 мс.

— Это указывает на стабильную производительность системы при изменении размеров сцены.

4.2.5 Вывод

— Данные показывают, что время загрузки сцены с 50 подсолнухами не зависит напрямую от размеров сцены.

— Наблюдаются локальные колебания, которые могут быть связаны с особенностями работы системы или случайными факторами.

4.3 Анализ загруженностьи процессора в зависимости от количества подсолнухов

На рисунке 4.3 представлен график анализа загруженности процессора в зависимости от количества подсолнухов.

Рисунок 4.3 — График анализа загруженности процессора в зависимости от количества подсолнухов.

Таблица 4.3 — Загруженность процессора в зависимости от количества подсолнухов

Количество подсолнухов	**Загруженность процессора (%)**
1	2,73
21	2,23
41	1,49
61	1,92
81	1,50
101	2,28
121	1,98
141	2,14
161	1,36
181	4,98
201	2,98
221	2,33
241	2,04
261	4,19
281	3,02
301	2,27
321	1,60
341	1,96
361	1,89
381	3,06
401	4,48

4.3.1 Общая тенденция

Загруженность процессора остаётся относительно низкой для большинства значений количества подсолнухов (в среднем от 1,36% до 4,98%). Наблюдается незначительный рост загруженности процессора при увеличении количества подсолнухов, но он не является линейным.

4.3.2 Локальные колебания

В некоторых случаях загруженность процессора может снижаться при увеличении количества подсолнухов (например, для 41 подсолнуха загруженность ниже, чем для 21). Это может быть связано с:

— **Распараллеливанием:** использование многопоточности позволяет эффективно распределять нагрузку на процессор;

— **Кэшированием вершин:** кэширование данных, связанных с костями, снижает нагрузку на процессор.

4.3.3 Стабильность загруженности процессора

Для большинства значений количества подсолнухов загруженность процессора остаётся в диапазоне 1,36% – 4,98%. Это указывает на эффективность использования ресурсов процессора благодаря распараллеливанию и кэшированию.

4.3.4 Вывод

Данные показывают, что загруженность процессора остаётся низкой для большинства значений количества подсолнухов благодаря использованию распараллеливания и кэширования вершин.

4.4 Вывод

В исследовательской части курсовой работы была проведена оценка производительности программного обеспечения для моделирования поля подсолнухов. Анализ времени загрузки сцены показал, что время увеличивается с ростом количества подсолнухов, хотя могут наблюдаться локальные колебания, вызванные случайными факторами. Также было установлено, что загруженность процессора остается на низком уровне благодаря эффективному использованию многопоточности и кэшированию.

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены следующие задачи:

1) проанализирована предметная область;
2) изучены существующие подходы и алгоритмы для моделирования и визуализации природных объектов;
3) выбраны средства реализации программного обеспечения;
4) создано программное обеспечение, которое моделирует трёхмерную динамическую сцену с подсолнухами, реагирующими на ветер;
5) реализована система освещения, изменяющаяся в зависимости от времени суток;
6) разработан пользовательский интерфейс, позволяющий настраивать параметры сцены;
7) проведено тестирование и исследование разработанного программного обеспечения, результаты которого подтвердили его высокую производительность и стабильность при различных настройках.

В результате проведённого исследования была разработана трёхмерная динамическая сцена, демонстрирующая реалистичное поведение подсолнухов под воздействием ветра и адаптивное освещение. Это программное обеспечение может быть использовано в различных приложениях, включая компьютерные игры и виртуальные симуляции. Полученные результаты подтвердили целесообразность выбранных подходов и методов, а также их практическую применимость в области компьютерной графики и анимации.

Литература

1. Головаченко С.А. Виды 3D моделирования: полигональное и сплайновое.

2. Степанов Д.И. Основы скелетной анимации: от простых объектов до сложных персонажей. [Электронный ресурс]. URL: https://apni.ru/article/10753-osnovy-skeletnoj-animacii-ot-prostyh-obuektov-do-slozhnyh-pe (дата обращения: 01.12.2024).

3. Гонахчян В.И. Алгоритм удаления невидимых поверхностей на основе программных проверок видимости.

4. Турлапов В.Е. Удаление невидимых поверхностей. Оптимизация. Тени. [Электронный ресурс]. URL: http://www.graph.unn.ru/rus/materials/CG/CG13_HSROptimization.pdf (дата обращения: 12.12.2024).