Содержание

ВВЕДЕНИЕ

В современном мире технологии играют важную роль в различных сферах деятельности человека. Одной из таких технологий является LiDAR (Light Detection and Ranging), которая активно применяется в разных областях, например, в метеорологии, геодезии, картографии, автомобильной промышленности, робототехнике, в системах машинного зрения и других областях.

Цель курсовой работы — разработка программного обеспечения для симуляции LiDAR и построение 3D-карты сцены. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) описать объекты и выбрать модель их представления;
2) изучить и провести анализ существующих алгоритмов компьютерной графики;
3) выбрать наиболее подходящие алгоритмы для реализации программы;
4) спроектировать программное обеспечение, предоставляющее пользователю необходимые функции;
5) реализовать спроектированное программное обеспечение;
6) провести сравнительный анализ времени построения кадра в зависимости от удаления объекта сцены от камеры.

1 Аналитический раздел

1.1 Описание объектов сцены

Сцена состоит из следующих объектов:

1) камера — объект сцены, из которого происходит наблюдение сцены, определяется местоположением и направлением просмотра;
2) источник света — объект сцены, который представляет из себя материальную точку, излучающую свет;
3) модель — трехмерный объект, расположенный на сцене, представленный в виде полигональной сетки;
4) облако точек — набор векторов вида (x, y, z).

1.2 LiDAR

LiDAR (Light Detection and Ranging) — это технология, которая использует лазерное излучение для обнаружения и определения дальности до объектов. Она применяется в разных областях, например, в метеорологии, геодезии, картографии и в системах машинного зрения.

Технология основана на принципе измерения времени прохождения лазерного импульса (ToF — Time of Flight). Лазерный излучатель генерирует короткие световые импульсы в инфракрасном или видимом диапазоне. Отражённые импульсы фиксируются датчиком, который измеряет время их возврата. Вычислительный модуль обрабатывает собранные данные и преобразует их в цифровую информацию.

Лидар работает с высокой частотой, испуская тысячи импульсов в секунду. Результаты объединяются в облака точек, представляющие собой трёхмерную модель.

В системах машинного зрения LiDAR используется для создания двумерных или трёхмерных изображений окружающего пространства.

1.2.1 Шаблоны сканирования

Чтобы собрать как можно больше данных, в LiDAR устанавливаются системы вращающихся зеркал, чтобы собрать как можно больше данных, не изменяя положения. Различные алгоритмы, по которым вращаеются зеркала, об-

разуют различные шаблоны (паттерны) сканирования — то, как движется луч при сканировании. Распространены следующие шаблоны сканирования:

— вертикальные полосы — сканирование происходит вертикально, от основания объекта до его вершины;

— горизонтальные полосы — сканирование происходит горизонтально, слева направо или справа налево;

— сетка — совмещение двух предыдущих шаблонов образует сканирование в виде сетки;

— с интервалом — сканирование происходит с определёнными интервалами.

1.3 Способы описания моделей

Модель является отображением формы и размеров объектов. Основное назначение модели - правильно отображать форму и модели объекта.

В основном используются 3 вида модели [1, 3]:

1) каркасная (проволочная);
2) поверхностная;
3) объемная.

1.3.1 Каркасная модель

В этой модели задается информация о вершинах и рёбрах объектов. Этим видам модели присущ один, но весьма существенный недостаток: не всегда модель правильно передает представление об объекте.

1.3.2 Поверхностная модель

Поверхность может описываться аналитически, либо может задаваться другим способом. Сложные криволинейные поверхности можно представлять в упрощенном виде, например, с помощью полигональной аппроксимации. В этом случае такая поверхность будет задаваться в виде поверхности многогранника.

Недостаток: отсутствует информация о том, с какой стороны поверхности находится собственный материал, а с какой стороны - пустота.

1.3.3 Объемная модель

Отличие от поверхностной модели состоит только в том, что в объемных модели к информации о поверхности добавляется информация о том, где расположен материал. Проще всего это можно сделать путём указания направления внутренней нормали.

1.3.4 Выбор модели

Т.к. LiDAR сканирует поверхности объектов, то каркасная модель не подходит из-за отсутствия поверхностей. Объёмная модель отличается от поверхностной наличием информации о нахождении материала, что не влияет на сканирование, поэтому не имеет значения для данной задачи. Поэтому в результате выбора модели была выбрана поверхностная модель.

1.4 Способы задания объектов

Способы задания поверхностных моделей [1, 2]:

— аналитический — описание поверхности с помощью математических уравнений;

— полигональный — описания с использованием следующих элементов: вершины, отрезки прямых (векторы), полилинии, полигоны, полигональные поверхности;

Аналитичиский способ задания объекта ограничен невозможностью описывать произвольные объекты, что делает его неподходящим для решения задачи. Поэтому в результате был выбран полигональный способ задания объектов.

1.5 Анализ алгоритмов удаления невидимых линий и поверхностей

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства [1].

1.5.1 Алгоритм Робертса

Алгоритм Робертса — алгоритм, разработанный для удаления рёбер и граней, которые скрываются другими объектами. Алгоритм работает в пространстве объекта и сравнивает взаимное расположение поверхностей каждого тела.

1.5.2 Алгоритм художника

Алгоритм художника — алгоритм решения проблемы «видимости» в компьютерной графике. Он позволяет избежать дополнительных затрат памяти, изначально сортируя по расстоянию от точки обзора. Затем объекты проверяются в порядке глубины, начиная с самого дальнего. В этом случае при рассмотрении объекта нет необходимости проверять его z-координату, так как цвет записывается в буфер кадра. Значения, хранящиеся в буфере ранее, перезаписываются.

1.5.3 Алгоритм Z-буффера

Алгоритм z-буфера — это метод удаления невидимых частей объектов в трёхмерной сцене. В нём используется два буфера: буфер кадра и z-буфер. В буфере кадра хранится информация о цвете объекта для каждого пикселя, а в z-буфере — z-координата видимого объекта. Алгоритм с импользованием Z-буффера по шагам:

1) заполнить буфер кадра фоновым цветом.
2) заполнить z-буфер минимальным значением z (определяет самую удалённую границу сцены).
3) обработать каждый многоугольник в сцене: преобразовать его в растровую форму и вычислить глубину z для каждого пикселя.
4) сравнить глубину каждого пикселя с z-буфером: если z больше, чем в z-буфере, записать атрибуты цвета многоугольника в буфер кадра и обновить z-буфер. Если сравнение даёт противоположный результат, ничего не делать.

1.5.4 Алгоритм трассировки лучей

Алгоритм трассировки лучей предполагает отслеживание лучей света в обратном направлении, от наблюдателя к объекту. Каждый луч проходит через пиксель растра до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, если таковые существуют, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Пересечение с ближайшим объектом представляет видимую поверхность для данного пикселя.

1.5.5 Выбор алгоритма

LiDAR работает на основе отражения лазерных импульсов от объектов, поэтому для решения задачи симуляции LiDAR был выбран алгоритм трассировки лучей, т.к. он моделирует распространение лучей в пространстве и их взаимодействие с поверхностями объектов.

2 Конструкторский раздел

2.1 Требования к программному обеспечению

Разрабатываемое программное обеспечение должно предоставлять пользователю следующие возможности:

— добавление объекта на сцену (куб, четырехугольная пирамида, тетраэдр, шестиугольная призма);

— удаление объектов со сцены;

— изменение свойств выбранного объекта (положение в пространстве, поворот);

— выбор камеры;

— задание шаблона и параметра сканирования;

— задание сценария сканирования;

— добавление и удаление облаков точек;

— перемещение и поворот камеры с помощью клавиатуры.

При этом разрабатываемая программа должна удовлетворять следующему требованию:

— наличие визуализации облака точек.

2.2 Алгоритм трассировки лучей

На рисунке 2.1 представлена схема алгоритма трассировки лучей.

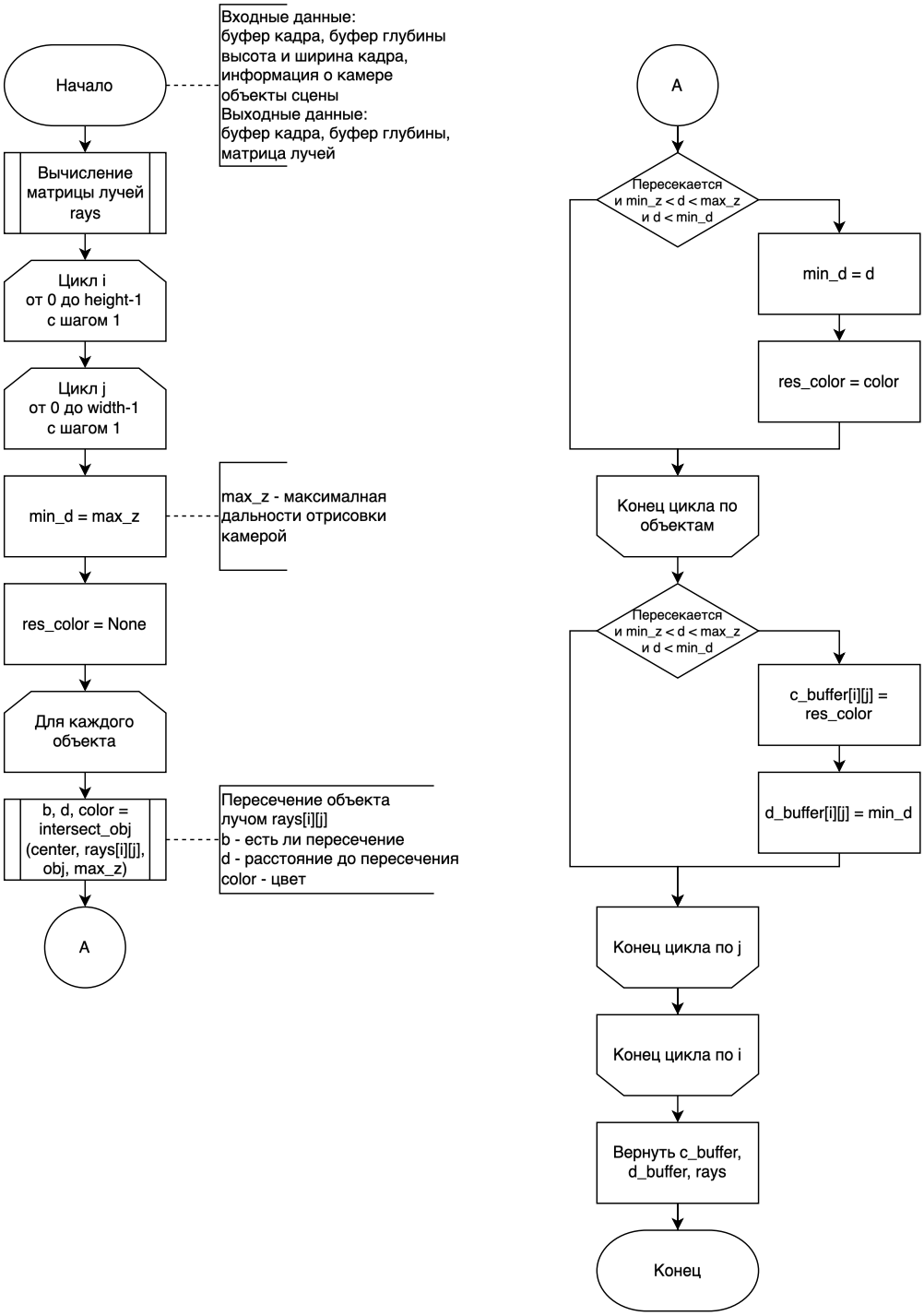

Рисунок 2.1 — Схема алгоритма трассировки лучей

2.3 Типы данных

В таблице 2.1 представлены структуры данных и их поля.

Таблица 2.1 — Выбор структур данных для представления объектов

Объект	**Представление**
Сцена	Объект Scene: — objs — массив объектов сцены; — point_clouds — массив облаков точек.
Объект	Объект MeshModel: — points — массив точек; — triangles — массив треугольников; — name — наименование объекта (в таблице объектов).
Облако точек	Объект PointCloud: — points — массив точек.
Камера	Объект Camera: — center — центр камеры; — h_dir — горизонтальный вектор задания направления камеры; — v_dir — вертикальный вектор задания направления камеры; — h_fov — угол обзора по горизонтали; — v_fov — угол обзора по векртикали; — min_z — минимальная дальность отрисовки; — max_z — максимальная дальность отрисовки.
Полотно	Объект Canvas — width — ширина полотна; — height — высота полотна.

3 Технологический раздел

3.1 Выбор средств реализации

В качестве используемого языка программирования был выбран $Python$ [6] по слудующим причинам:

— средствами языка можно реализовать все неободимые алгоритмы;

— данный язык является объектно-ориентированным, что позволит структурировать программу;

— язык обладает необходимой для реализации задачи библитекой для создания графического интерфеса пользователя $PyQt6$ [7];

— язык обладает библитекой для визуализации данных $Matplotlib$ [5].

3.2 Структура программы

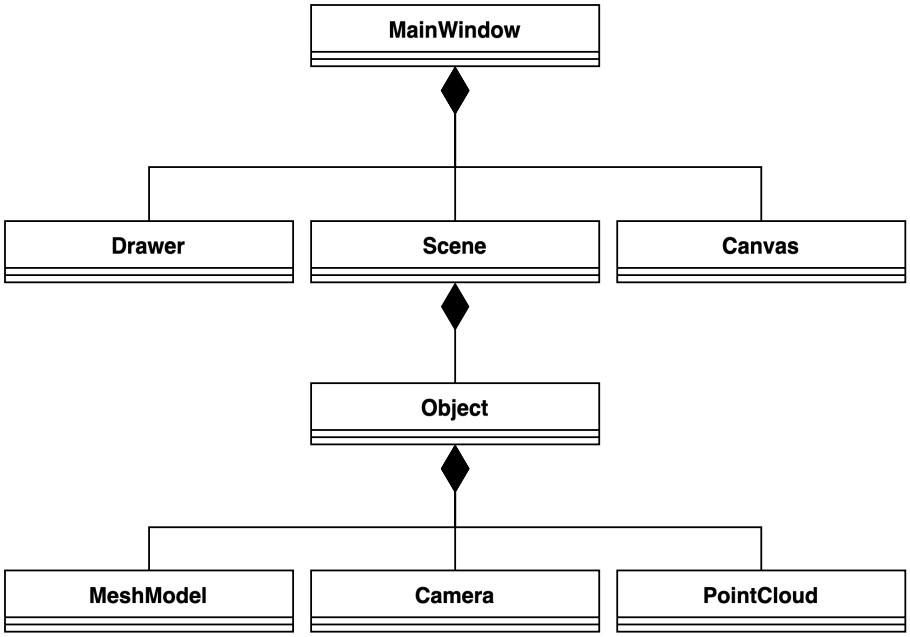

Рисунок 3.1 — Диаграмма классов разработанного ПО

3.3 Детали реализации

В листингах 3.1-3.4 представлена реализация алгоритма трассировки лучей.

Листинг 3.1 — Реализация алгоритма трассировки лучей

```
def ray_tracing(c_buffer, d_buffer, height, width, camera, objs):
  center, h_dir, v_dir, h_fov, v_fov, min_z, max_z = camera
  rays = comp_rays(height, width, h_dir, v_dir, h_fov, v_fov)
  for i in range(height):
    for j in range(width):
      min_d = max_z
      res_color = None
      for k in range(len(objs)):
        points, triangles = objs[k]
        b, d, color = intersect_obj((center, rays[i][j]), points,
            triangles, max_z)
        if b and d < min_d and min_z < d < max_z:
          min_d = d
          res_color = color
      if res_color is not None:
        c_buffer[i][j] = res_color
        d_buffer[i][j] = min_d
  return c_buffer, d_buffer, rays
```

Листинг 3.2 — Реализация алгоритма вычисления лучей

```
def comp_rays(height, width, h_dir, v_dir, h_fov, v_fov):
  z = np.cross(h_dir, v_dir)
  z /= np.linalg.norm(z, ord=2)
  h = 2 * tan(pi * h_fov / 360) / width
  v = 2 * tan(pi * v_fov / 360) / height
  rays = np.zeros((height, width, 3))
  for i in range(height):
    for j in range(width):
      res = (j + 0.5 - width / 2) * h * h_dir + (i + 0.5 - height
          / 2) * v * v_dir + z
      res /= np.linalg.norm(res, ord=2)
      rays[i][j] = res
  return rays
```

Листинг 3.3 — Реализация алгоритма определения пересечения луча с треугольником

```
def intersect_triangle(ray, v0, v1, v2):
  v0, v1, v2 = v0[:3], v1[:3], v2[:3]
  origin, direction = ray
  origin = origin[:3]
  edge1, edge2 = v1 - v0, v2 - v0

  pvec = np.cross(direction, edge2)
  det = np.dot(edge1, pvec)
  if -EPSILON < det < EPSILON:
    return -1, None

  inv_det = 1.0 / det

  tvec = origin - v0
  u = np.dot(tvec, pvec) * inv_det
  if u < -EPSILON or u > 1.0 + EPSILON:
    return -1, None

  qvec = np.cross(tvec, edge1)
  v = np.dot(direction, qvec) * inv_det

  if v < -EPSILON or u + v > 1.0 + EPSILON:
    return -1, None

  t = np.dot(edge2, qvec) * inv_det
  if t < 0.0:
    return -1, None
  if u < EPSILON or v < EPSILON or u + v > 1.0 - EPSILON:
    return t, np.array([0, 0, 255], dtype=np.uint8)
  return t, np.array([255, 255, 255], dtype=np.uint8)
```

Листинг 3.4 — Реализация алгоритма трассировки лучей для выбранных пикселей

```
def ray_tracing_seq(c_buffer, d_buffer, height, width, camera,
   objs, ray_ids):
  center, h_dir, v_dir, h_fov, v_fov, min_z, max_z = camera
  rays = comp_rays_seq(height, width, h_dir, v_dir, h_fov, v_fov,
      ray_ids)
  for i in range(len(ray_ids)):
    min_d = max_z
    res_color = None
    for k in range(len(objs)):
      points, triangles = objs[k]
      b, d, color = intersect_obj((center, rays[i]), points,
         triangles, max_z)
      if b and d < min_d and min_z < d < max_z:
        min_d = d
        res_color = color
    if res_color is not None:
      c_buffer[ray_ids[i]] = res_color
      d_buffer[ray_ids[i]] = min_d
  return c_buffer, d_buffer, rays
```

3.4 Интерфейс программы

Интерфейс программы представлен на рисунках 3.2-3.5. На рисунке 3.2 представлено окно программы. Слева находится окно для визуализации сцены камерой. Справа находятся разделы меню программы: «Объекты», «Сканирование», «Облака точек». Движение камеры происходит при помощи клавиш клавиатуры: «W», «A», «S», «D». Вращение камеры происходит при помощи стрелок клавиатуры: вверх, влево, вниз, вправо.

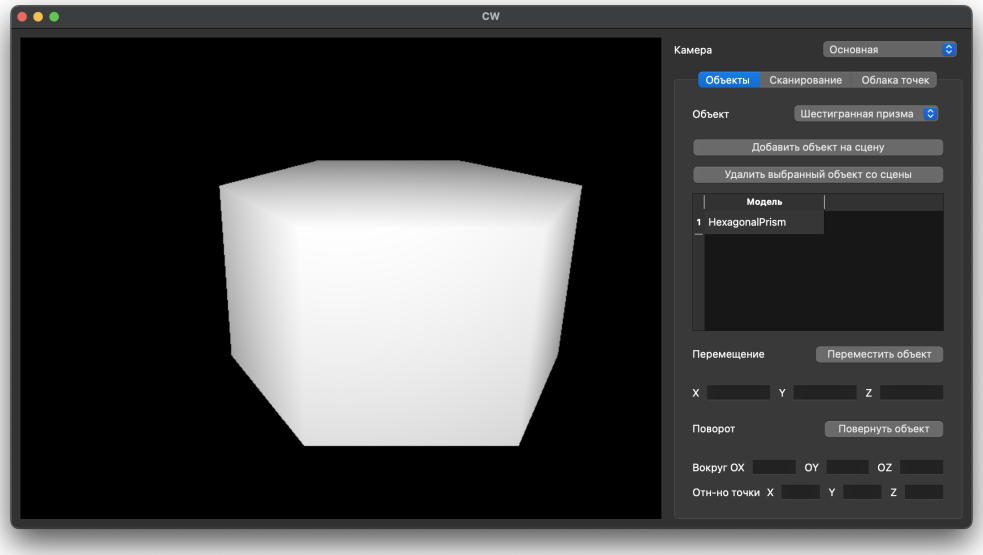

Рисунок 3.2 — Интерфейс программы

На рисунке 3.3 представлен раздел меню объектов. В нем задаются объекты сцены. Кнопка «Добавить объект на сцену» добавляет объект из выпадающего списка выше на сцену. Кнопка «Удалить объект со сцены» удаляет модели выбранные в таблице. Кнопка «Переместить объект» сдвигает объект на расстояния указанные в полях ниже. Кнопка «Повернуть объект» поворачивает объект вокруг осей на углы указанные в полях ниже и относительно точки, координаты которой указаны в полях ниже.

На рисунке 3.4 представлен раздел меню сканирования. В нем задаются паттерн сканирования, его параметр и сценарий сканирования. Выпадающее меню «Паттерн» позволяет выбрать шаблон сканирования, ниже находится поле параметра, в котором задаются, к примеру, число линий или интервал сканирования. Кнопка «Применить» обновляет значение параметра в программе. Кнопка «Сканировать» создает облако точек на основе текущего положения и направле-

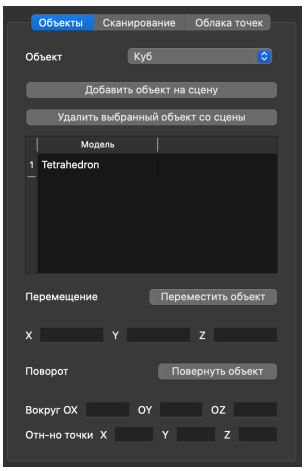

Рисунок 3.3 — Интерфейс программы, раздел объектов

ния основной камеры и выбранного паттерна сканирования. Кнопка «Добавить действие в сценарий» добавляет точку сценария на основе информации, указанной выше. Кнопка «Удалить выбранное действие из сценария» удаляет точки выбранные в таблице. Кнопка «Выполнить сценарий» производит сканирование сцены согласно сценарию.

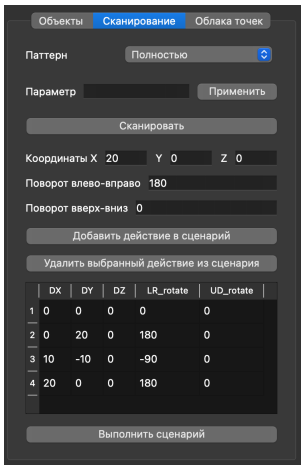

Рисунок 3.4 — Интерфейс программы, раздел сканирования

На рисунке 3.5 представлен раздел меню облаков точек. В нем находится таблица облаков точек. Кнопка «Удалить выбранное облако точек» удаляет облака точек выбранные в таблице.

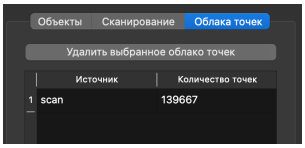

Рисунок 3.5 — Интерфейс программы, раздел облаков точек

3.5 Демонстрация работы

На рисунках 3.6-3.7 представлена демонстрация работы программы.

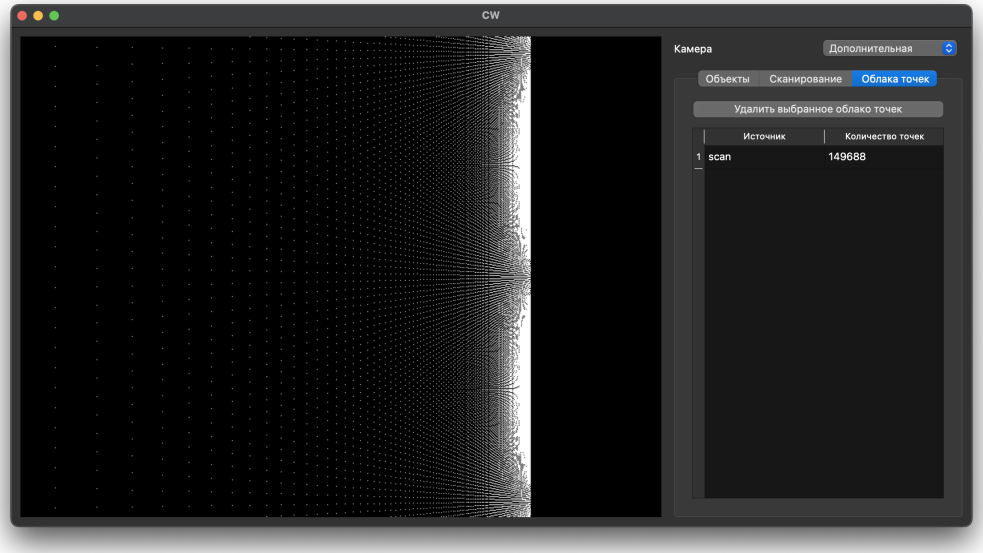

Рисунок 3.6 — Пример работы программы — облако точек грани куба вблизи

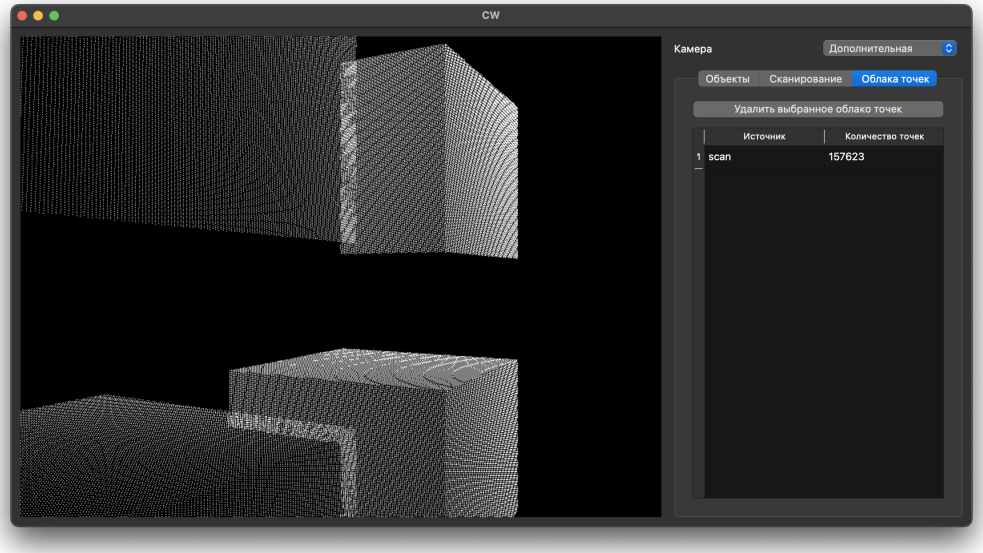

Рисунок 3.7 — Пример работы программы — облако точек четырех кубов

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устроства, на котором выполнялось тестирование:

— операционная система — macOS Sonoma 14.1 (23B2073);

— процессор — Apple M3 [4];

— оперативная память — 16 Гб.

Замеры проводилось на ноутбуке, включенном в сеть питания, нагруженным процессами операционной системы, приложениями окружения и замеряемой программой.

4.2 Описание исследования

В ходе исследования необходимо проанализировать зависимость времени генерации кадра размером 800×600 от расстояния от камеры до объекта. На сцене содержится только один, заданный исследованием объект. Расстояние от камеры до объекта означает расстояние между точкой центра камеры и средней точкой всех точек объекта (центром масс объекта).

Исследование проводилось на следующих объектах:

— куб (12 треугольников);

— квадратная правильная пирамида (8 треугольников);

— треугольная правильная пирамида (тетраэдр) (4 треугольника);

— шестиугольная правильная призма (20 треугольников).

Результаты измерений для различных объектов представлены в таблицах 4.1-4.4.

На рисунке 4.1 представлен графики зависимости времени построения кадра от расстояния камеры до объекта.

Вывод

Исследование показало:

— время построения кадра уменьшается при удалении объекта от камеры;

— время построения кадра увеличивается при увеличении числа треугольников, из которых состоит объект.

Таблица 4.1 — Результаты измерений времени построения одного кадра в зависимости от расстояния до куба

Расстояние до объекта	**Время построения одного кадра в миллисекундах**
4	792.751
5	593.561
6	385.714
7	269.575
8	201.955
9	160.791
10	154.974
11	139.017
12	124.865
13	113.250
14	92.610
15	83.913
16	93.297
17	81.428
18	82.049
19	71.318
20	68.965

Таблица 4.2 — Результаты измерений времени построения одного кадра в зависимости от расстояния до квадратной пирамиды

Расстояние до объекта	**Время построения одного кадра в миллисекундах**
4	327.318
5	240.370
6	161.174
7	140.256
8	118.707
9	102.096
10	93.984
11	85.103
12	82.478
13	73.347
14	71.728
15	69.722
16	63.068
17	51.587
18	62.824
19	64.616
20	62.851

Таблица 4.3 — Результаты измерений времени построения одного кадра в зависимости от расстояния до тетраэдра

Расстояние до объекта	**Время построения одного кадра в миллисекундах**
4	193.817
5	145.702
6	133.799
7	112.043
8	98.981
9	89.180
10	81.717
11	76.380
12	71.771
13	69.786
14	67.180
15	63.685
16	64.003
17	63.105
18	61.160
19	56.520
20	57.133

Таблица 4.4 — Результаты измерений времени построения одного кадра в зависимости от расстояния до шестиугольной призмы

Расстояние до объекта	**Время построения одного кадра в миллисекундах**
4	1353.018
5	1362.413
6	1341.379
7	1091.668
8	852.148
9	641.851
10	491.618
11	391.221
12	322.493
13	274.486
14	237.936
15	211.138
16	185.139
17	171.488
18	141.959
19	146.449
20	132.270

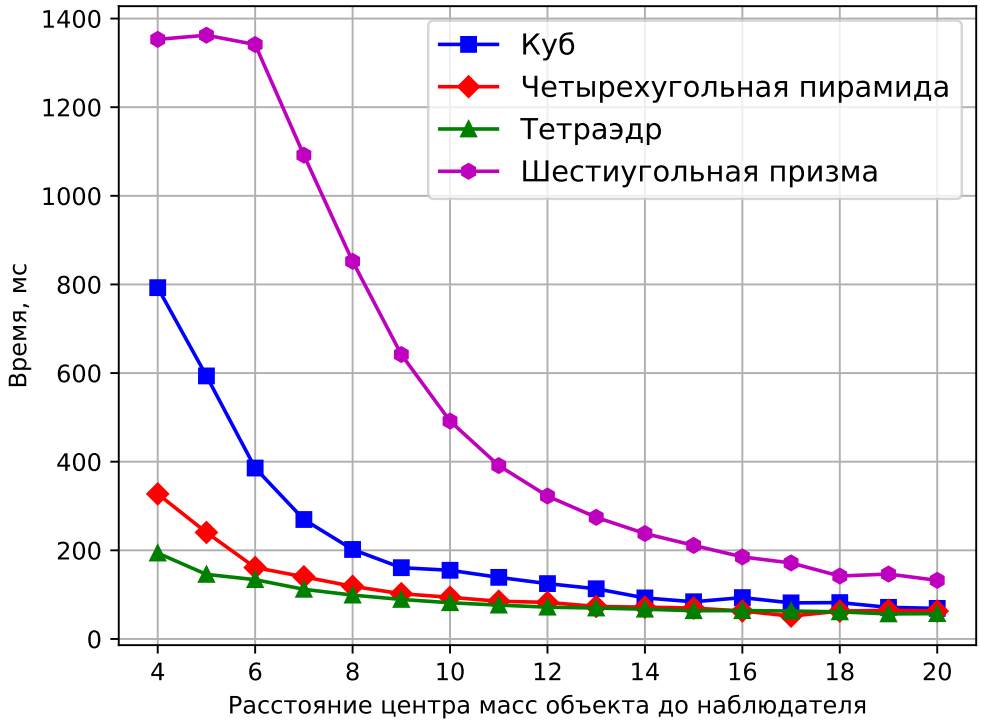

Рисунок 4.1 — Графики зависимости времени построения кадра от расстояния камеры до объекта

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы поставленная цель была достигнута: разработка программного обеспечения для симуляции LiDAR и построение 3D-карты сцены.

Были выполнены следующие задачи:

— описаны объекты и выбрать модель их предсатвления;

— изучены и провести анализ существующих алгоритмов компьютерной графики;

— выбраны наиболее подходящие алгоритмы для реализации программы;

— спроектировано программное обеспечение, предоставляющее пользователю необходимые функции;

— реализовано спроектированное программное обеспечение;

— проведен сравнительный анализ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роджерс Д. «Алгоритмические основы машинной графики». Москва – «Мир»: Редакция литературы по математическим наукам, 1989, С. 504.

2. Шикин Е., Боресков А. «КОМПЬЮТЕРНАЯ ГРАФИКА. Полигональные модели». – «ДИАЛОГ-МИФИ», 2001. – С. 464.

3. Соломенцева С. Б. «3D-моделирование и визуализация: учебно-методическое пособие». — Елец, 2019. — С. 80.

4. Технические характеристики MacBook Pro [Электронный ресурс]. Режим доступа: https://support.apple.com/en-by/117736 (дата обращения 11.12.24)

5. Документация библиотеки Matplotlib [Электронный ресурс]. Режим доступа: https://matplotlib.org/ (дата обращения 13.10.24)

6. Документация языка Python [Электронный ресурс]. Режим доступа: https://docs.python.org/3/ (дата обращения 20.11.24)

7. Документация библиотеки PyQt6 [Электронный ресурс]. Режим доступа: https://doc.qt.io/qtforpython-6/ (дата обращения 20.11.24)

Приложение А

Презентация состоит из 13 слайдов.