

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

КАФЕДРА КОМПЬЮТЕРНЫХ СИСТЕМ И ПРОГРАММНЫХ ТЕХНОЛОГИЙ

Отчёт по лабораторной работе №3-4
на тему: «Визуализация каркасной модели»
Курс: «Разработка графических приложений»

Выполнил студент:

Волкова М.Д.
Группа: 13541/2

Проверил:

Абрамов Н.А.

Санкт-Петербург
2018 г.

Содержание

1	Лабораторная работа №2	2
1.1	Цель работы	2
1.2	Описание программы	2
1.3	Ход работы	3
1.3.1	Алгоритм Брезенхема	3
1.3.2	Настройка фиксированной камеры	4
1.4	Результаты	7
1.4.1	CvLineDrawer	7
1.4.2	LineDrawer	8
1.4.3	TriangleDrawer	9
1.5	Вывод	13
1.6	Листинг	14

Лабораторная работа №2

1.1 Цель работы

Разработать программу на языке C для растеризации загруженной модели на экран

1.2 Описание программы

1. Возможности программы:

- (a) Загрузка трехмерной модели из OBJ-файла
- (b) Растеризация каркаса трехмерной модели
- (c) Обеспечение вращения камеры вокруг трехмерной модели
- (d) Растеризация линий своим алгоритмом
- (e) Растеризация треугольников своим алгоритмом
- (f) Вычисление координат и получение значения глубины для конкретного пикселя
- (g) Использование буфера глубины для отсека невидимых пикселей

2. Входные параметры программы:

- (a) Ширина и высота окна
- (b) Вертикальный угол обзора камеры для выполнения перспективной проекции
- (c) Ближняя и дальняя плоскости отсечения камеры
- (d) Дистанция от камеры до загруженной модели
- (e) Скорость вращения камеры вокруг модели (градус/сек)

3. Выходные параметры программы:

- (a) Последовательность кадров, выводимая на экран

4. Порядок работы программы:

- (a) Загрузка трехмерной модели в вершинные и индексные буфера
- (b) Определение центра модели (можно считать, что матрица мира для модели – единичная)
- (c) Формирование матрицы проекции
- (d) Далее – для очередного кадра:
 - i. Формирование матрицы вида исходя из координат центра модели, дистанции до модели и скорости вращения камеры
 - ii. Преобразование вершин модели в экранные координаты

1.3 Ход работы

В дополнение к уже установленной ранее библиотеке OpenCV дополнительно была установлена библиотека GLM, предназначенная для работы с векторами и матрицами размерности до 4-х. Для работы с форматом OBJ использована библиотека TinyObj.

Программа предоставлена в листинге.

1.3.1 Алгоритм Брезенхема

Алгоритм Брезенхема - это алгоритм, определяющий, какие точки двумерного раstra нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками.

Для проволочного рендеринга, сначала нам нужна функция, которая будет отрисовывать линии:

```
1  template<typename F>
2  inline void drawline(int x0, int y0, int x1, int y1, F plot) {
3
4      int dx = std::abs(x1 - x0);
5      int dy = std::abs(y1 - y0);
6
7      int directionX = x0 < x1 ? 1 : -1;
8      int directionY = y0 < y1 ? 1 : -1;
9      int err = (dx > dy ? dx : -dy) / 2;
10
11     for (;;) {
12         plot(x0, y0);
13         if (x0 == x1 && y0 == y1) break;
14         int e2 = err;
15         if (e2 > -dx) {
16             err -= dy;
17             x0 += directionX;
18         }
19         if (e2 < dy) {
20             err += dx;
21             y0 += directionY;
22         }
23     }
24 }
```

Для хранения модели мы используем формат wavefront obj. Формат файлов OBJ - это простой формат данных, который содержит только 3D геометрию, а именно, позицию каждой вершины, связь координат текстуры с вершиной, нормаль для каждой вершины, а также параметры, которые создают полигоны. Всё, что нам нужно для рендера, это прочитать из файла массив вершин вида:

```
1 v 0.608654 -0.568839 -0.416318
```

это координаты x,y,z, одна вершина на строку файла и граней:

```
1 f 7 6 1
```

еще в файле содержаться нормали (нормали могут быть не нормированными):

```
1 vn -0.966742 -0.255752 9.97231e-09
```

Нас интересуют первое число после каждого пробела, это номер вершины в массиве, который мы прочитали ранее. Таким образом эта строчка говорит, что вершины 7, 6 и 1 образуют треугольник.

Далее пишем функцию, которая принимает объект класса Drawer, вершины и координаты модели и рисует линии:

```
1 void render(Drawer &drawer, const std::vector<glm::vec3> &vertices, const std::vector<
2     unsigned int> &indices) {
3     for (auto i = 0; i < indices.size(); i += 3) {
4         Triangle triangle{vertices[indices[i]], vertices[indices[i + 1]], vertices[
5         indices[i + 2]]};
6         drawer.draw(triangle);
7     }
8 }
```

1.3.2 Настройка фиксированной камеры

В OpenGL при использовании фиксированного конвейера есть ровно две матрицы, относящихся к трансформациям точек и объектов:

- GL PROJECTION моделирует ортографическое или перспективное преобразование от трёхмерной усечённой пирамиды (т.е. от области видимости камеры) к трёхмерному кубу с длиной ребра, равной 2 (т.е. к нормализованному пространству).
- GL MODELVIEW сочетает в себе два преобразования: от локальных координат объекта к мировым координатам, а также от мировых координат к координатам камеры.

За рамками фиксированного конвейера можно использовать столько матриц, сколько захочется.

- поведение камеры описывается как ортографическим или перспективным преобразованием, так и положением камеры в мировом пространстве, то есть для моделирования камеры нужны GL PROJECTION и GL MODELVIEW одновременно
- с другой стороны, для трансформаций над телами — вращение предмета с помощью умножения координат на матрицу — нужна матрица GL MODELVIEW.

Настроим матрицу GL PROJECTION один раз для перспективного преобразования, а матрицу GL MODELVIEW будем постоянно модифицировать, когда локальная система координат очередного объекта не совпадает с мировой системой координат.

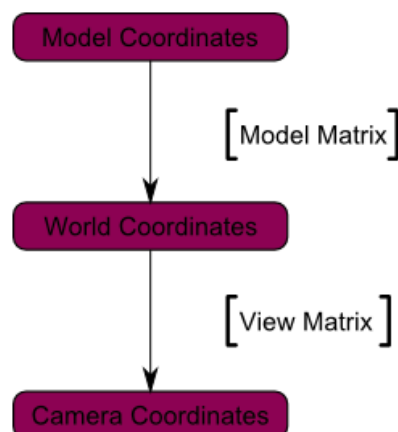
Начнём настройку камеры с GL MODELVIEW: зададим матрицу так, как будто бы камера смотрит с позиции camera position на точку model center, при этом направление “вверх” камеры задаёт вектор glm::vec3(0, 1, 0):

```
1 camera = glm::lookAt(  
2     camera_position ,  
3     model_center ,  
4     glm::vec3(0, 1, 0)  
5 );
```

, где:

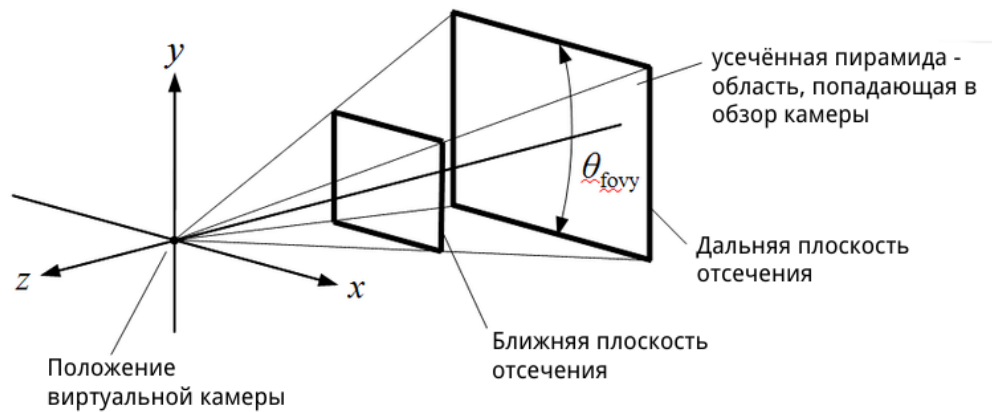
- camera position - Позиция камеры в мировом пространстве
- model center - Указывает куда вы смотрите в мировом пространстве
- glm::vec3(0, 1, 0) - Вектор, указывающий направление вверх

А вот диаграмма, которая показывает то, что мы делаем:



Для перспективного преобразования достаточно создать матрицу с помощью функции glm::perspective. Она принимает на вход несколько параметров преобразования: горизонтальный угол обзора камеры, соотношение ширины и высоты, а также две граничных координаты для отсечения слишком близких к камере и слишком далёких от камеры объектов.

Эти параметры легко увидеть на следующей иллюстрации:



Проекционная матрица

Теперь мы находимся в пространстве камеры, вершина, которая получит координаты $x == 0$ и $y == 0$ будет отображаться по центру экрана. Однако, при отображении объекта огромную роль играет также дистанция до камеры. Для двух вершин, с одинаковыми x и y , вершина имеющая большее значение по z будет отображаться ближе, чем другая.

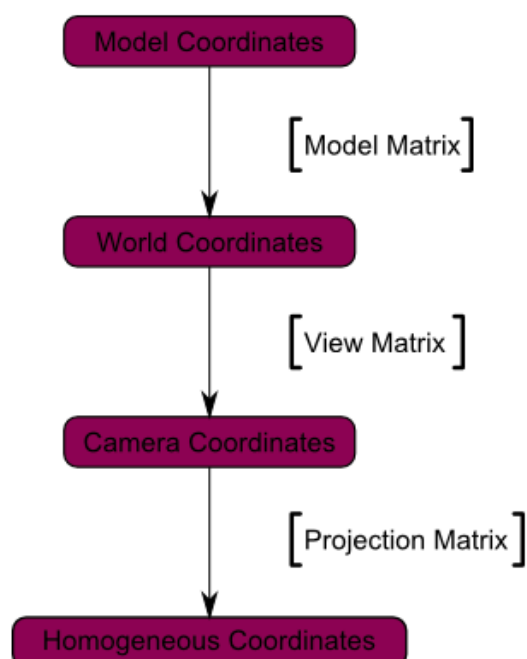
Это называется перспективной проекцией, к счастью, в glm имеем:

```
1 projection = glm::perspective(
2     glm::radians(fovy),
3     screen_ratio,
4     front,
5     back)
```

, где:

- `glm::radians(fovy)` - Вертикальное поле зрения в радианах.
- `screen_ratio` - Отношение сторон.
- `front` - Ближняя плоскость отсечения.
- `back` - Дальняя плоскость отсечения.

Теперь наша диаграмма будет выглядеть так:



Матрица поворота

Функция `rotate` в `glm` поворачивает 3D вектор на заданный угол вокруг заданной оси (представленной орт-вектором):

```
1 glm::rotate(glm::mat4(1), angle, glm::vec3(0, 1, 0));
```

Следующий шагом объединяем трансформации, что реализуется по следующей формуле:

```
1 camera_position = (rotation_matrix * start_camera_position)
```

1.3.3 Z-буферизация

1.4 Результаты

В качестве тестовой модели для проверки работоспособности программы использовалась модель чайничка, экспортированная стандартными средствами в формат OBJ.

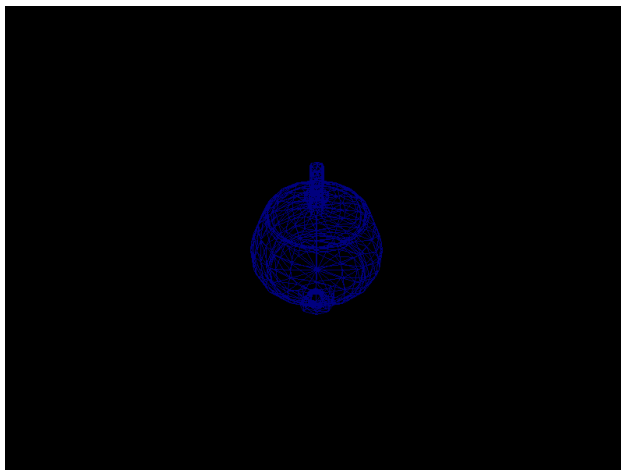
1.4.1 CvLineDrawer

Параметры:

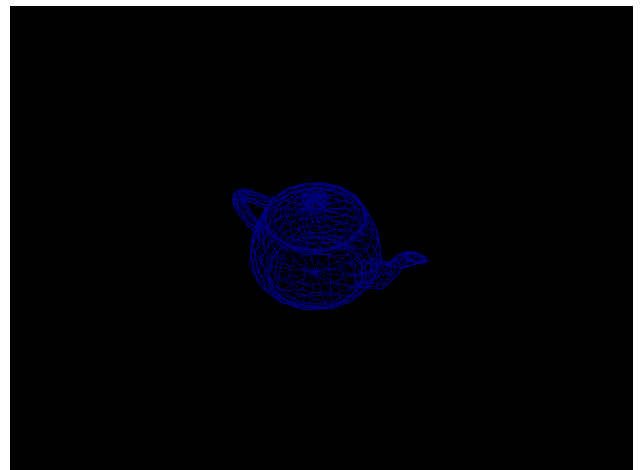
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-50.0"))  
    ("dx", "Distance to model", cxxopts::value<int>()->default_value("300"))  
    ("dy", "Distance to model", cxxopts::value<int>()->default_value("300"))  
    ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("1.0"))  
    ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("100.0"))  
    ("i,in_file", "Input filename ", cxxopts::value<std::string>()->default_value(default_file_path))  
    ("o,out_file", "Output filename ", cxxopts::value<std::string>()->default_value(default_save_path));
```

Рис. 1.1: Параметры

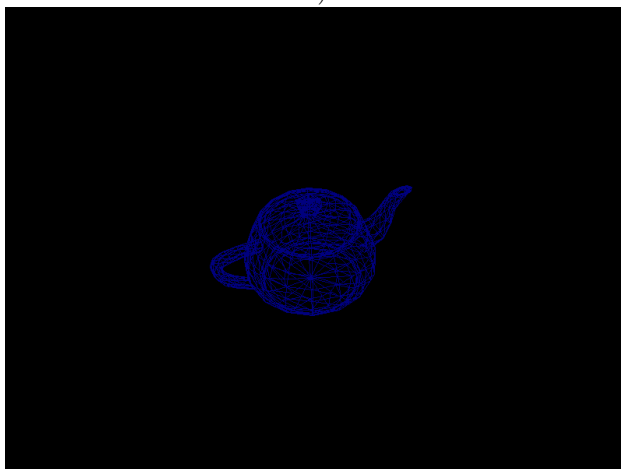
Результат работы:



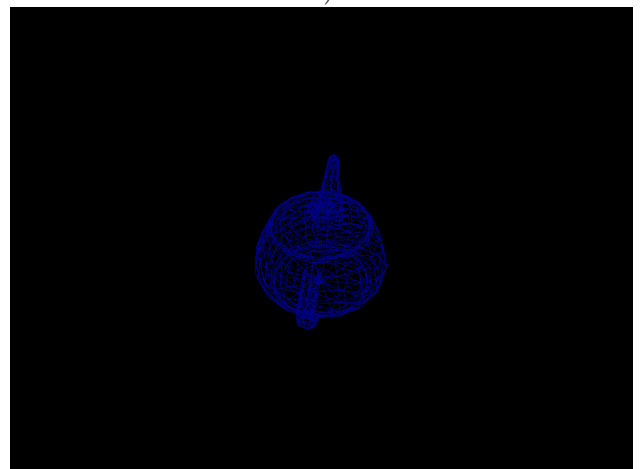
a)



b)



c)



d)

Рис. 1.2: Последовательно создаваемые изображения

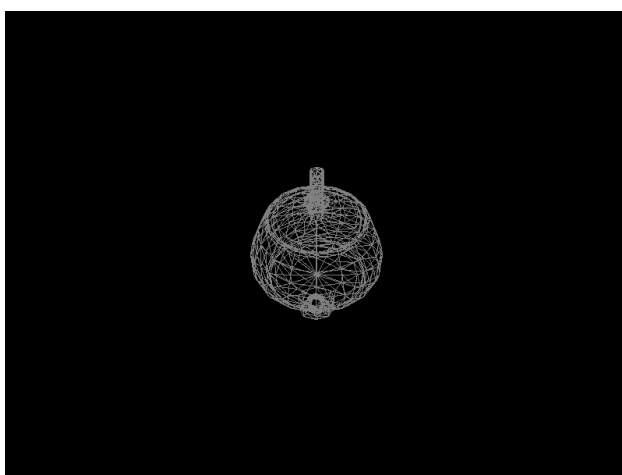
1.4.2 LineDrawer

Параметры:

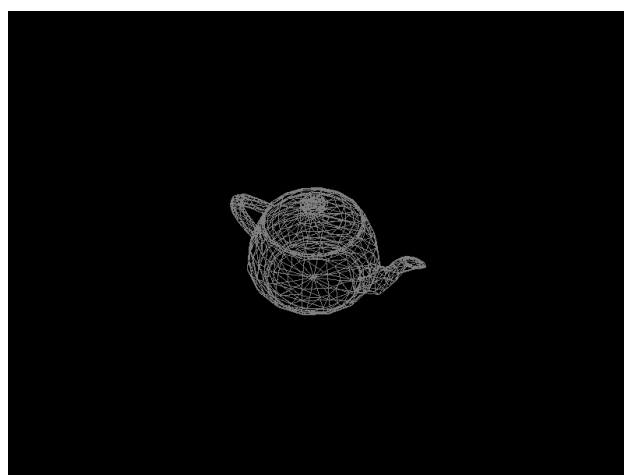
```
options.add_options()
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))
    ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-50.0"))
    ("dx", "Distance to model", cxxopts::value<int>()->default_value("300"))
    ("dy", "Distance to model", cxxopts::value<int>()->default_value("300"))
    ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("1.0"))
    ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("100.0"))
    ("i,in_file", "Input filename ", cxxopts::value<std::string>()->default_value(default_file_path))
    ("o,out_file", "Output filename ", cxxopts::value<std::string>()->default_value(default_save_path));
```

Рис. 1.3: Параметры

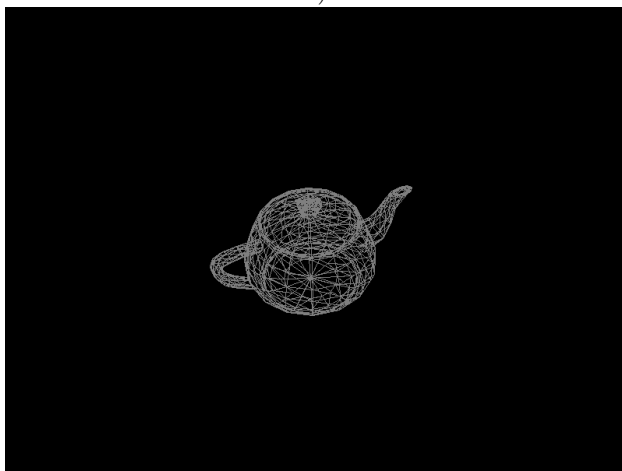
Результат работы:



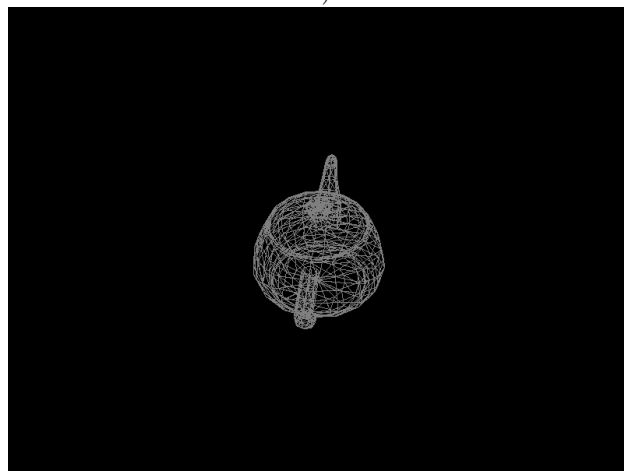
a)



b)



c)



d)

Рис. 1.4: Последовательно создаваемые изображения

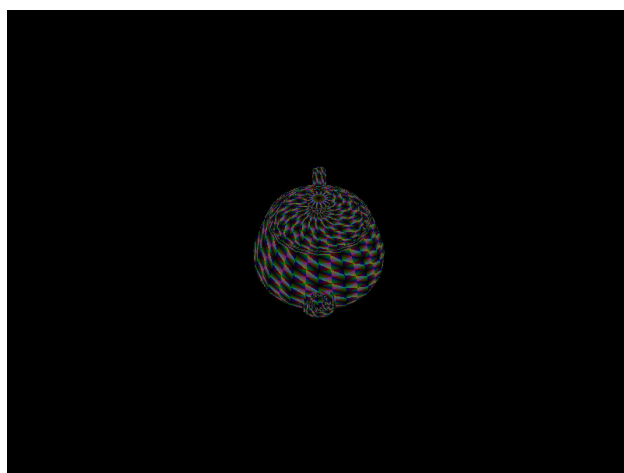
1.4.3 TriangleDrawer

Параметры:

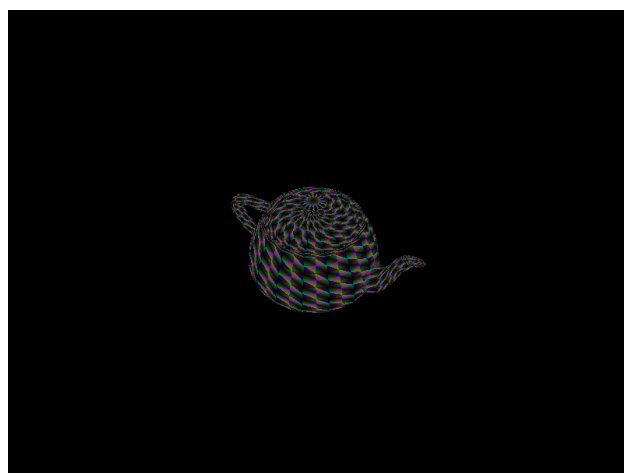
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-50.0"))  
    ("dx", "Distance to model", cxxopts::value<int>()->default_value("800"))  
    ("dy", "Distance to model", cxxopts::value<int>()->default_value("-300"))  
    ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("1.0"))  
    ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("100.0"))  
    ("i,in_file", "Input filename ", cxxopts::value<std::string>()->default_value(default_file_path))  
    ("o,out_file", "Output filename ", cxxopts::value<std::string>()->default_value(default_save_path));
```

Рис. 1.5: Параметры

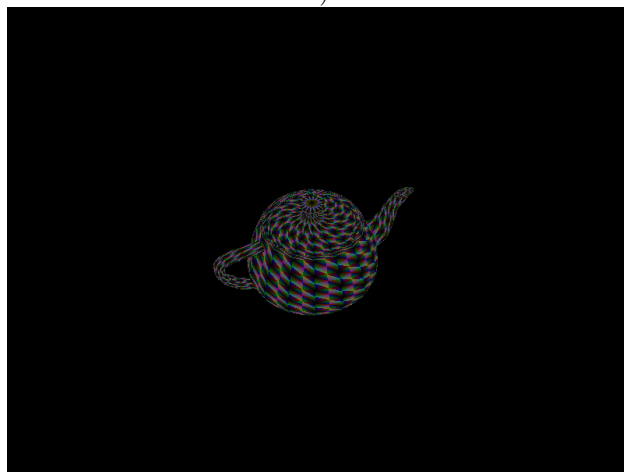
Результат работы:



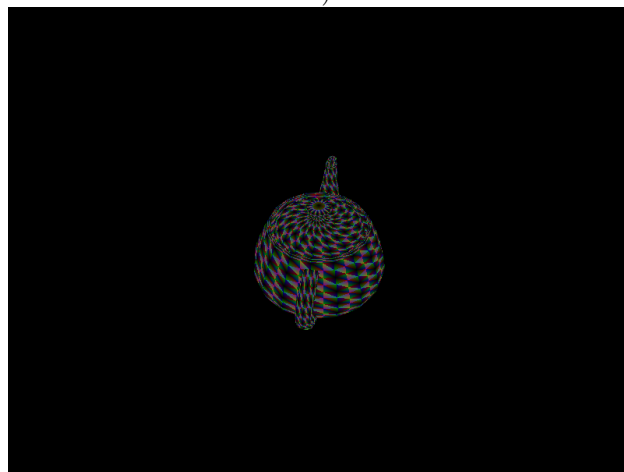
a)



b)



c)



d)

Рис. 1.6: Последовательно создаваемые изображения

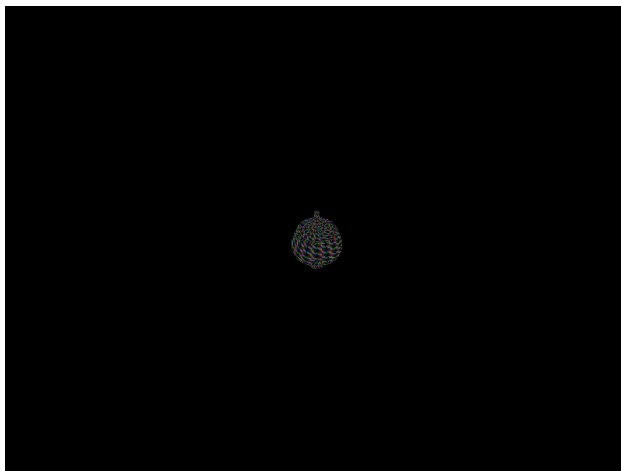
Приведем еще несколько результатов, изменяя параметры камеры:

Параметры:

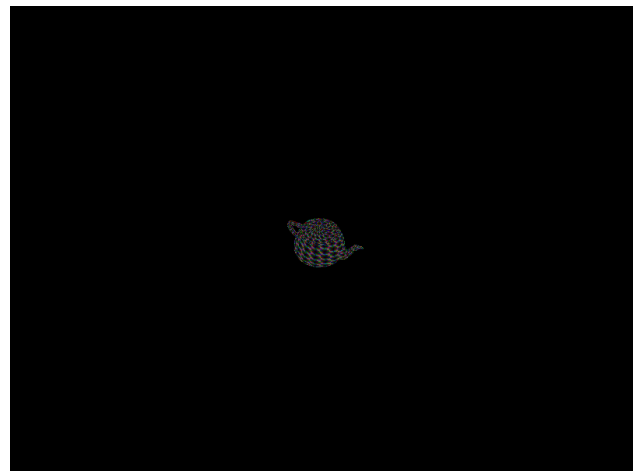
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-100.0"))  
    ("dx", "Distance to model", cxxopts::value<int>()->default_value("300"))  
    ("dy", "Distance to model", cxxopts::value<int>()->default_value("300"))  
    ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("1.0"))  
    ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("100.0"))  
    ("i,in_file", "Input filename ", cxxopts::value<std::string>()->default_value(default_file_path))  
    ("o,out_file", "Output filename ", cxxopts::value<std::string>()->default_value(default_save_path));
```

Рис. 1.7: Параметры

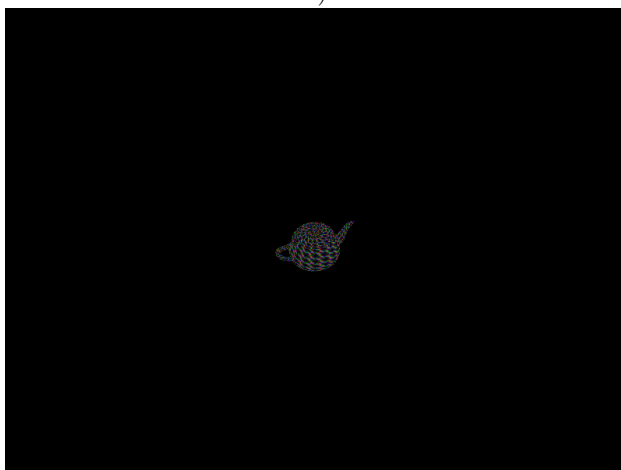
Результат работы:



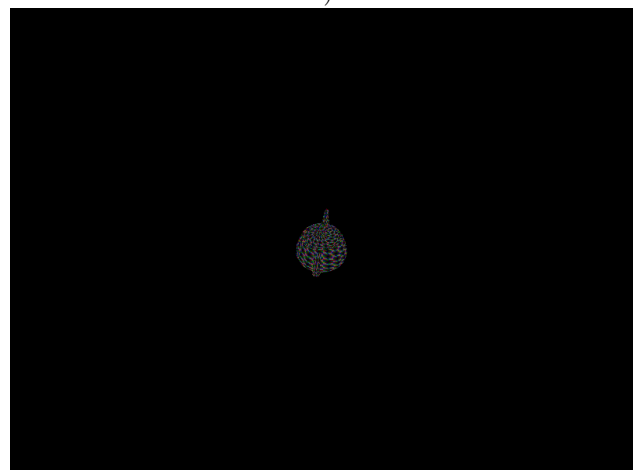
a)



b)



c)



d)

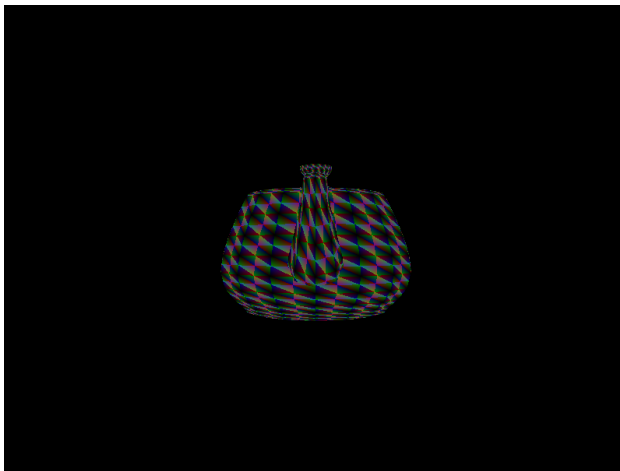
Рис. 1.8: Последовательно создаваемые изображения

Параметры:

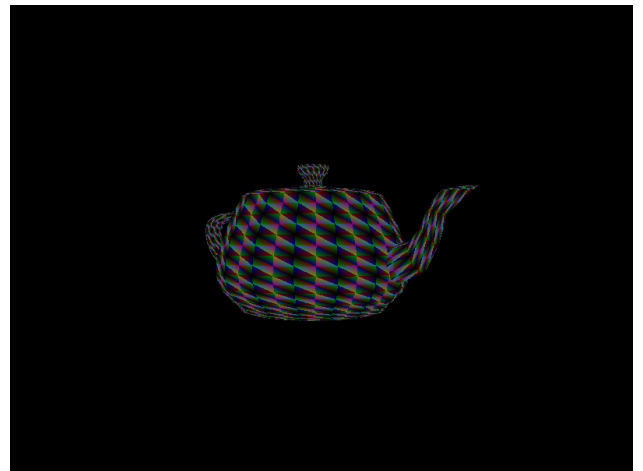
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-50.0"))  
    ("dx", "Distance to model", cxxopts::value<int>()->default_value("300"))  
    ("dy", "Distance to model", cxxopts::value<int>()->default_value("0"))  
    ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("1.0"))  
    ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("100.0"))  
    ("i,in_file", "Input filename ", cxxopts::value<std::string>()->default_value(default_file_path))  
    ("o,out_file", "Output filename ", cxxopts::value<std::string>()->default_value(default_save_path));
```

Рис. 1.9: Параметры

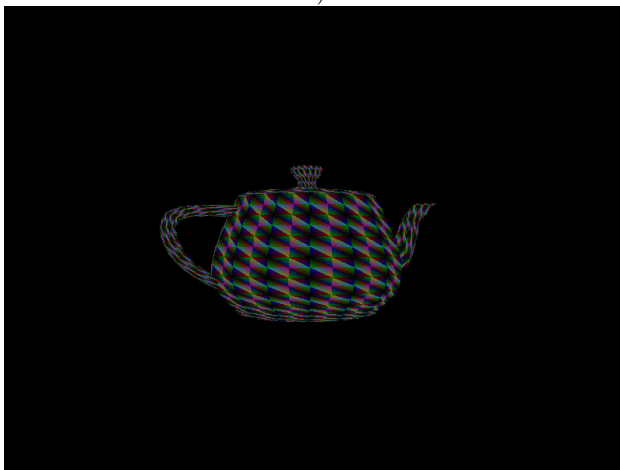
Результат работы:



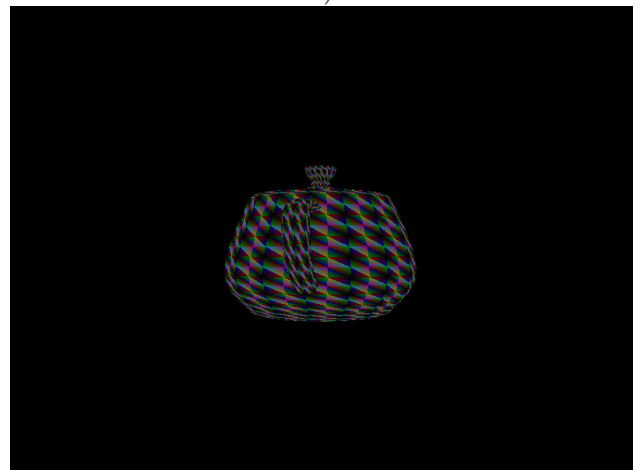
a)



b)



c)



d)

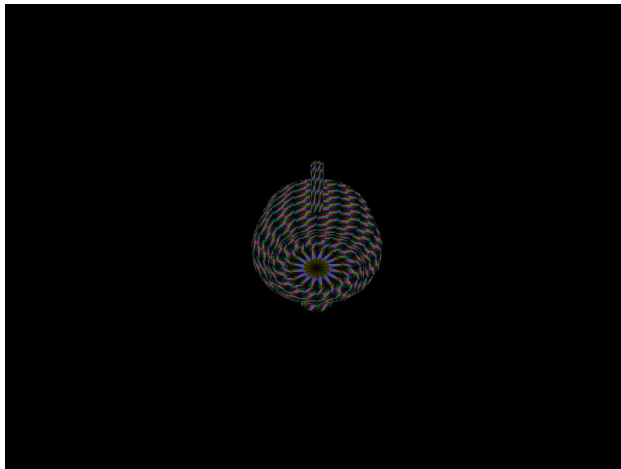
Рис. 1.10: Последовательно создаваемые изображения

Параметры:

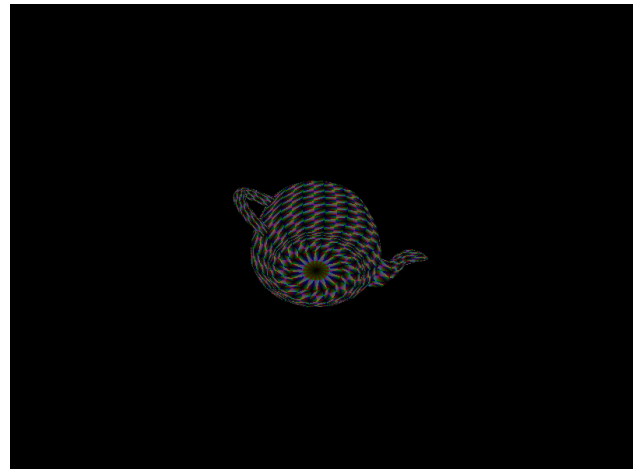
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-50.0"))  
    ("dx", "Distance to model", cxxopts::value<int>()->default_value("300"))  
    ("dy", "Distance to model", cxxopts::value<int>()->default_value("300"))  
    ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("150.0"))  
    ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("100.0"))  
    ("i,in_file", "Input filename ", cxxopts::value<std::string>()->default_value(default_file_path))  
    ("o,out_file", "Output filename ", cxxopts::value<std::string>()->default_value(default_save_path));
```

Рис. 1.11: Параметры

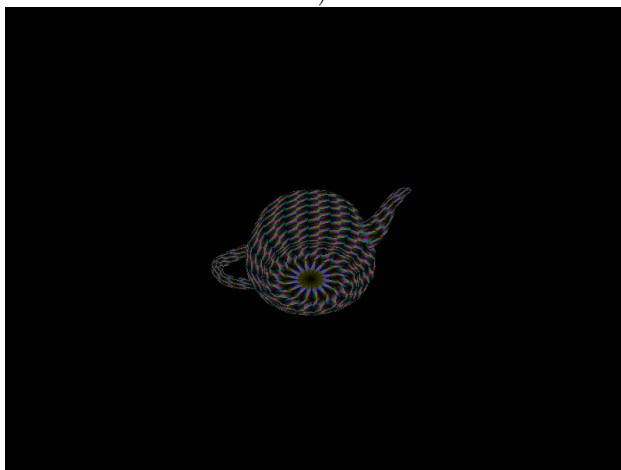
Результат работы:



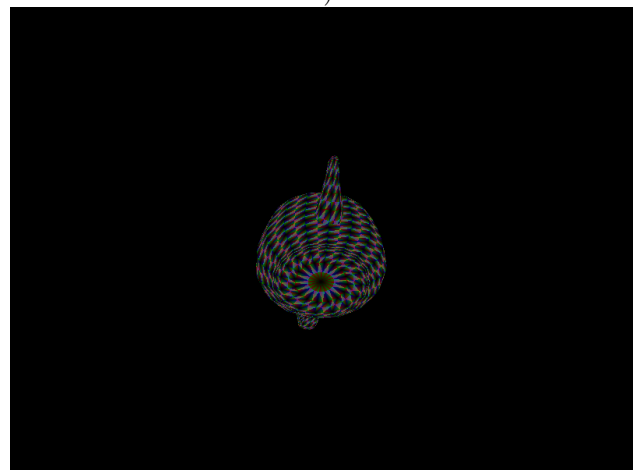
a)



b)



c)



d)

Рис. 1.12: Последовательно создаваемые изображения

Результатом работы стало двумерное анимированное изображение вращающегося каркаса выбранной ранее модели, которая в любой дискретный момент времени была повернута на некоторый угол вокруг мировых осей OX, OY, OZ.

1.5 Вывод

В данной работе была изучена библиотека GLM и составлена программа для визуализации трехмерной модели в виде проволочного каркаса с использованием средств библиотеки OpenGL.

Результаты визуализации отвечают ожиданиям при заданном смещении, повороте и масштабе модели. Для создания более полного представления наблюдателя о внешнем виде исходной модели, необходимо в дальнейшем реализовать отображение поверхностей модели, посредством треугольников, учитывая, что используемая библиотека TinyObj позволяет проводить разбиение произвольного полигона на треугольники автоматически при чтении файла модели.

1.6 Листинг

```
1 #include <utility>
2
3 #include <iostream>
4 #include <any>
5 #include <OBJ_Loader.h>
6 #include <cxopts.hpp>
7
8 #include <glm/vec3.hpp>
9 #include <glm/geometric.hpp>
10 #include <glm/gtc/matrix_transform.hpp>
11
12 #include <opencv2/core.hpp>
13 #include <opencv2/highgui.hpp>
14 #include <opencv2/imgcodecs.hpp>
15 #include <opencv2/imgproc.hpp>
16
17 #include "render.h"
18 #include "Drawer.h"
19 #include "transformers.h"
20
21 const int FRAME_PER_SECOND = 10;
22 const int FRAME_COUNT = 10000;
23
24
25 template<int index>
26 float min(const std::vector<glm::vec3> &vertices) {
27     float result = FLT_MAX;
28     for (auto &&vex : vertices) {
29         result = std::min(result, vex[index]);
30     }
31     return result;
32 }
33
34 template<int index>
35 float max(const std::vector<glm::vec3> &vertices) {
36     float result = FLT_MIN;
37     for (auto &&vex : vertices) {
38         result = std::max(result, vex[index]);
39     }
40     return result;
41 }
42
43 template<int index>
44 float getCenter(const std::vector<glm::vec3> &vertices) {
45     auto &&min_point = min<index>(vertices);
46     auto &&max_point = max<index>(vertices);
47     return (min_point + max_point) / 2;
48 }
49
50 glm::vec3 getModelCenter(const std::vector<glm::vec3> &vertices) {
51     auto &&center_x = getCenter<0>(vertices);
52     auto &&center_y = getCenter<1>(vertices);
53     auto &&center_z = getCenter<2>(vertices);
54     return glm::vec3(center_x, center_y, center_z);
55 }
56
57 void render(Drawer &drawer, const std::vector<glm::vec3> &vertices, const std::vector<
    unsigned int> &indices) {
58     for (auto i = 0; i < indices.size(); i += 3) {
59         Triangle triangle{vertices[indices[i]], vertices[indices[i + 1]], vertices[
            indices[i + 2]]};
60         drawer.draw(triangle);
61     }
```

```

62 }
63
64
65 int main(int argc, char **argv) {
66     cxxopts::Options options("Lba3", "Render teapot and maybe something else");
67     std::string default_file_path = "../teapot.obj";
68     std::string default_save_path = "../teapot.avi";
69
70     options.add_options()
71         ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))
72         ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))
73         ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("2.0"))
74         ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-50.0"))
75         ("dx", "Distance to model", cxxopts::value<int>()->default_value("120"))
76         ("dy", "Distance to model", cxxopts::value<int>()->default_value("100"))
77         ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("0.1"))
78         ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("10000.0"
79     ))
80     ("i,in_file", "Input filename ", cxxopts::value<std::string>()->default_value(
81     default_file_path))
82     ("o,out_file", "Output filename ", cxxopts::value<std::string>()->
83     default_value(default_save_path));
84
85     auto &&arguments = options.parse(argc, argv);
86
87     auto &&width = arguments["width"].as<int>();
88     auto &&height = arguments["height"].as<int>();
89     auto &&speed = arguments["speed"].as<float>();
90     auto &&fovy = arguments["fovy"].as<float>();
91     auto &&distanceX = arguments["dx"].as<int>();
92     auto &&distanceY = arguments["dy"].as<int>();
93     auto &&front = arguments["front"].as<float>();
94     auto &&back = arguments["back"].as<float>();
95     auto &&file_name = arguments["in_file"].as<std::string>();
96     auto &&res_file_name = arguments["out_file"].as<std::string>();
97
98     objl::Loader loader;
99     loader.LoadFile(file_name);
100     auto &&mesh = loader.LoadedMeshes[0];
101
102     auto &&model_vertices = ToGLMVertices().applyList<objl::Vertex, glm::vec3>(mesh.
103     Vertices);
104     auto &&model_center = getModelCenter(model_vertices);
105
106     auto &&screen_ratio = static_cast<float>(width) / static_cast<float>(height);
107     auto &&projection = glm::perspective(
108         glm::radians(fovy),
109         screen_ratio,
110         front,
111         back
112     );
113
114     float angle = 0;
115     float angle_per_frame = speed / FRAME_PER_SECOND;
116
117     auto &&start_camera_position = glm::vec4(distanceX, distanceY, 0, 1);
118
119     TriangleDrawer drawer(width, height);
120
121     for (auto i = 0; i < FRAME_COUNT; i++) {
122         drawer.resetImage();
123         glm::mat4 rotation_matrix = glm::rotate(glm::mat4(1), angle, glm::vec3(0, 1, 0));
124         glm::vec3 camera_position = (rotation_matrix * start_camera_position);
125         auto &&camera = glm::lookAt(
126             camera_position,

```



```

124         model_center ,
125         glm::vec3(0, 1, 0)
126     );
127
128     drawer.updatePipeline(std::make_unique<TriangleTransformationPipeline>(camera ,
129     projection , width , height));
130     render(drawer , model_vertices , mesh.Indices);
131
132     cv::imshow("res" , drawer.getImage());
133     cv::waitKey(2000);
134     angle += angle_per_frame;
135 }
136
137 return 0;

```