

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:

Ерниязов Т.Е.

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
1.3.3	Z-буферизация	6
1.4	Результаты	8
1.4.1	CvLineDrawer	8
1.4.2	LineDrawer	9
1.4.3	TriangleDrawer	10
1.4.4	Z-буфер	14
1.5	Вывод	15
1.6	Листинги	16
1.6.1	Листинг 1. main.cpp	16
1.6.2	Листинг 2. Drawer.h	19
1.6.3	Листинг 3. render.h	21
1.6.4	Листинг 4. transformers.h	29

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.

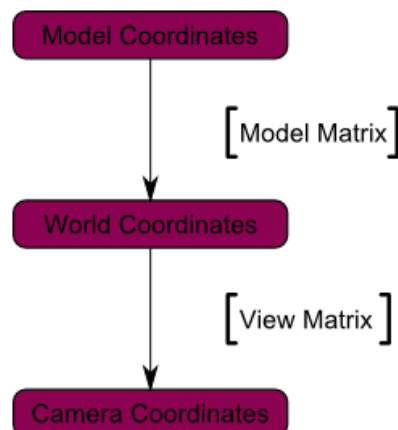
Начнём настройку камеры с GL MODELVIEW: зададим матрицу так, как будто бы камера смотрит с позиции camera position на точку model center, при этом направление “вверх” камеры задаёт вектор glm::vec3(0, 1, 0):

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

, где:

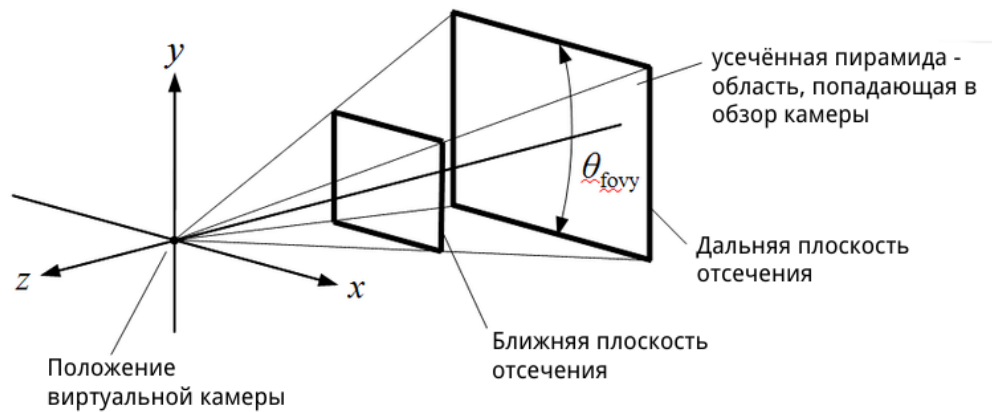
- camera position - Позиция камеры в мировом пространстве
- model center - Указывает куда вы смотрите в мировом пространстве
- glm::vec3(0, 1, 0) - Вектор, указывающий направление вверх

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.

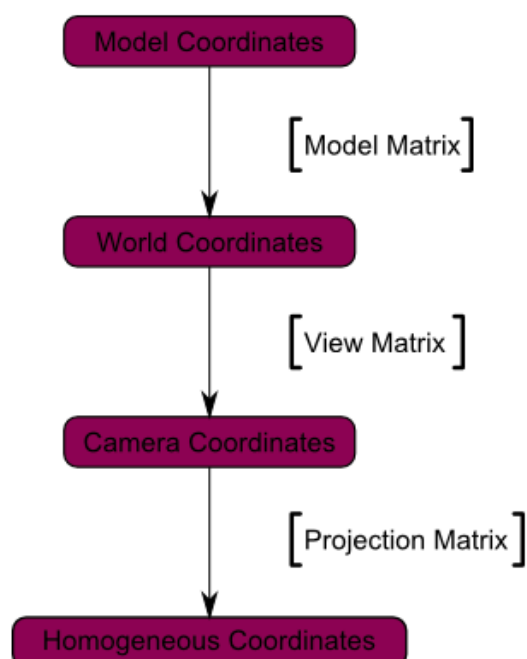
Это называется перспективной проекцией, к счастью, в glm имеем:

```
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

Z-буферизация - это способ учёта удалённости элемента изображения. Представляет собой один из вариантов решения «проблемы видимости». Очень эффективен и практически не имеет недостатков, если реализуется аппаратно.

Формальное описание алгоритма z-буфера таково:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета.
2. Заполнить z-буфер минимальным значением z.
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке.
4. Для каждого Пиксел(х,у) в многоугольнике вычислить его глубину z(х,у).
5. Сравнить глубину z(х,у) со значением Zбуфер(х,у), хранящимся в z-буфере в этой же позиции.
6. Если $z(х,у) > Zбуфер(х,у)$, то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить Zбуфер(х,у) на z(х,у). В противном случае никаких действий не производить.

Поскольку у нас экран двумерный, то z-буфер тоже должен быть двумерным:

```
1 cv::Mat1d zBufferDefaultValue(int height, int width) {  
2     return cv::Mat::ones(height, width, CV_64F) * 100000;  
3 }  
4
```

Затем в коде мы проходим по всем треугольникам и делаем вызов растеризатора, передавая ему и картинку, и z-буфер.

```
1 void triangleBarycentric(const ExtendedTriangle &t, cv::Mat1d &zBuffer, const  
MipMap<8> &mipMap) {  
2     auto &bbox = t.bbox();  
3     Coord from = bound(bbox.first);  
4     Coord to = bound(bbox.second);  
5     for (auto x = from.x; x <= to.x; ++x) {  
6         for (auto y = from.y; y <= to.y; ++y) {  
7             glm::dvec3 barycentric;  
8             bool pointInTriangle = get_barycentric(t, glm::vec4(x, y, 1, 1),  
barycentric);  
9             if (!pointInTriangle) continue;  
10            glm::dvec3 zInterpolated = glm::dvec3(t.a.z, t.b.z, t.c.z) * barycentric;  
11            double z = zInterpolated[0] + zInterpolated[1] + zInterpolated[2];  
12            if (z < zBuffer.at<double>(y, x)) {  
13                zBuffer.at<double>(y, x) = z;  
14            }  
15            glm::dvec3 barycentricX, barycentricY;  
16            get_barycentric(t, glm::vec4(x + 1, y, 1, 1), barycentricX);  
17            get_barycentric(t, glm::vec4(x, y + 1, 1, 1), barycentricY);  
18            glm::vec2 uv = interpolateTexture(t.texture, barycentric);  
19            glm::vec2 uvX = interpolateTexture(t.texture, barycentricX);  
20            glm::vec2 uvY = interpolateTexture(t.texture, barycentricY);  
21            glm::vec2 dx = uvX - uv;  
22            glm::vec2 dy = uvY - uv;  
23            glm::vec2 dx = uvX - uv;  
24            glm::vec2 dy = uvY - uv;  
25
```

```

26         float l = std::max(glm::dot(dx, dx), glm::dot(dy, dy));
27         float d = std::max(0.0f, 0.5f * std::log2(l));
28
29         cv::Vec3b texel = trilinearFilter(uv, d, mipMap);
30         plot(x, y, texel);
31     }
32 }
33 }
34 }
35 }
36

```

В этой функции мы переводим координаты в барицентрические, с помощью следующей функции:

```

1  bool get_barycentric(const ExtendedTriangle &t, const glm::vec4 &p, glm::dvec3 &
   barycentric) {
2      auto area = edgeFunction(t.a, t.b, t.c);
3      auto w0 = edgeFunction(t.b, t.c, p) / area;
4      auto w1 = edgeFunction(t.c, t.a, p) / area;
5      auto w2 = edgeFunction(t.a, t.b, p) / area;
6
7      barycentric = glm::dvec3(w0, w1, w2);
8      barycentric /= barycentric[0] + barycentric[1] + barycentric[2];
9      barycentric = glm::abs(barycentric);
10
11     return !(w0 < 0 || w1 < 0 || w2 < 0);
12 }
13

```

Все, что с отрицательным значением - отбрасывается.

```

1  double min, max;
2  cv::minMaxLoc(drawer.zBuffer, &min, &max);
3  cv::Mat zNorm;
4  cv::normalize(drawer.zBuffer, zNorm, 0.0, 255.0, cv::NORM_MINMAX, CV_8UC1);
5

```

Функция minMaxLoc находит минимальное и максимальное значения элементов и их положения.

normalize нормализует src таким образом, что минимальное значение выходного массива равно 0.0, а максимальное значение выходного массива равно 255.0.

1.4 Результаты

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

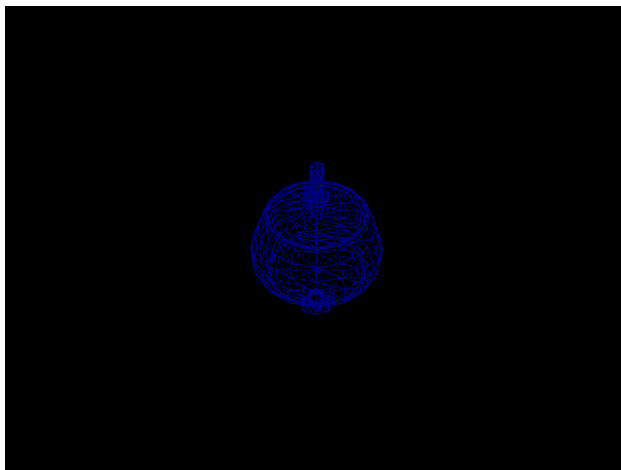
1.4.1 CvLineDrawer

Параметры:

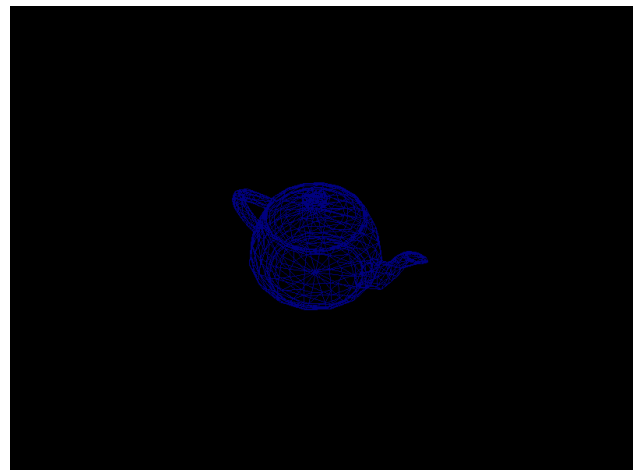
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<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

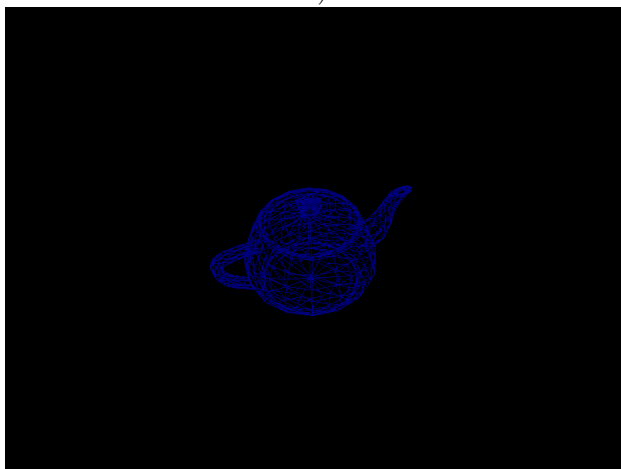
Результат работы:



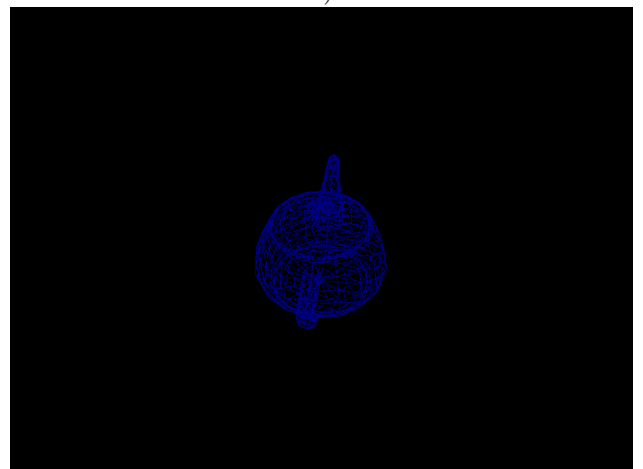
a)



b)



c)



d)

Рис. 1.2: Последовательно создаваемые изображения

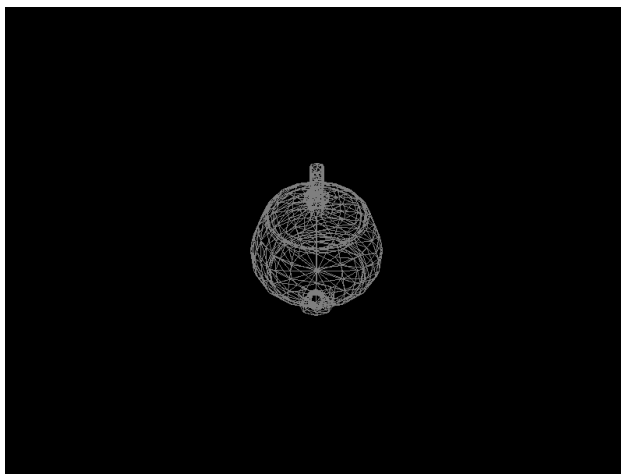
1.4.2 LineDrawer

Параметры:

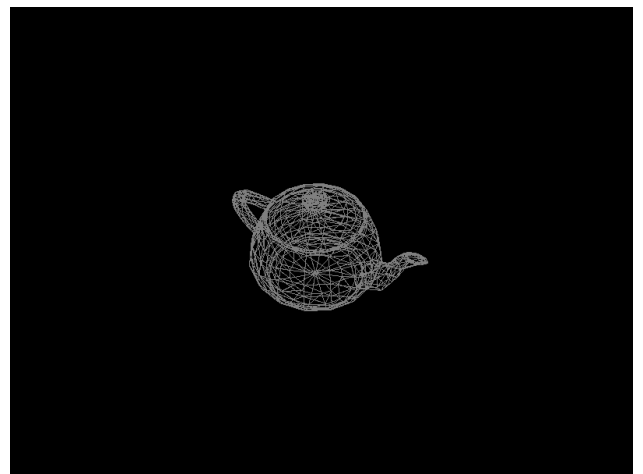
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<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

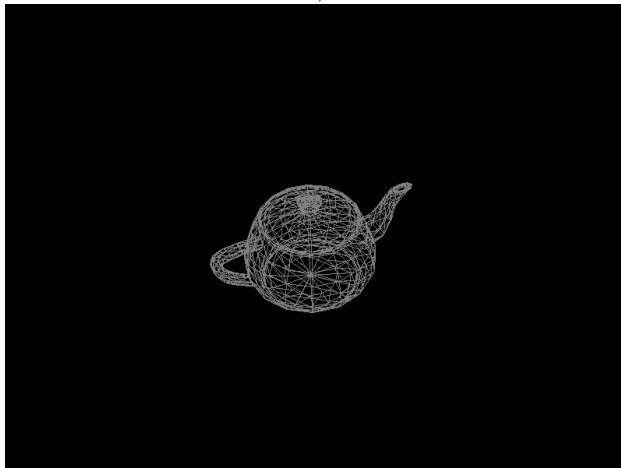
Результат работы:



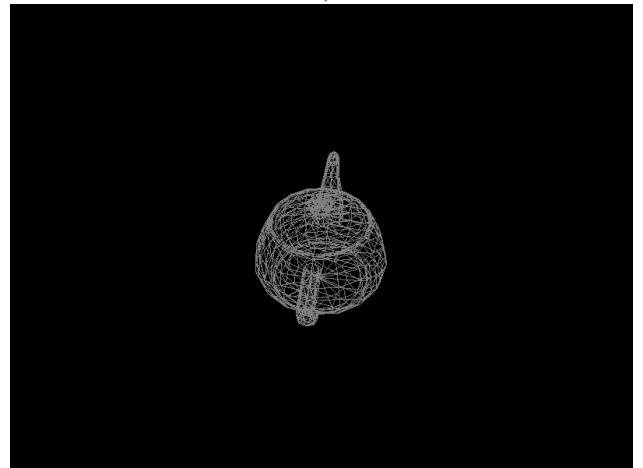
a)



b)



c)



d)

Рис. 1.4: Последовательно создаваемые изображения

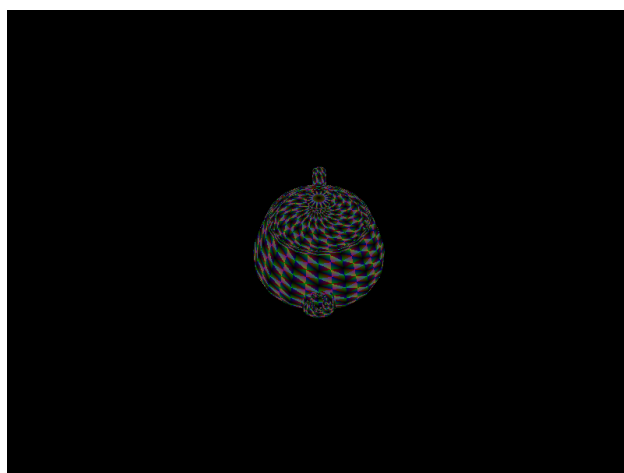
1.4.3 TriangleDrawer

Параметры:

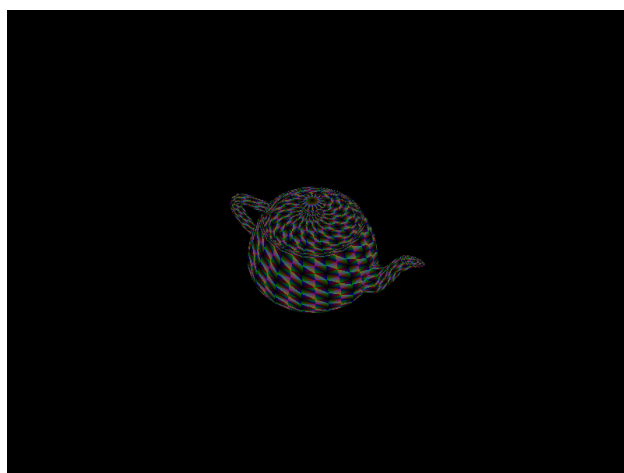
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<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

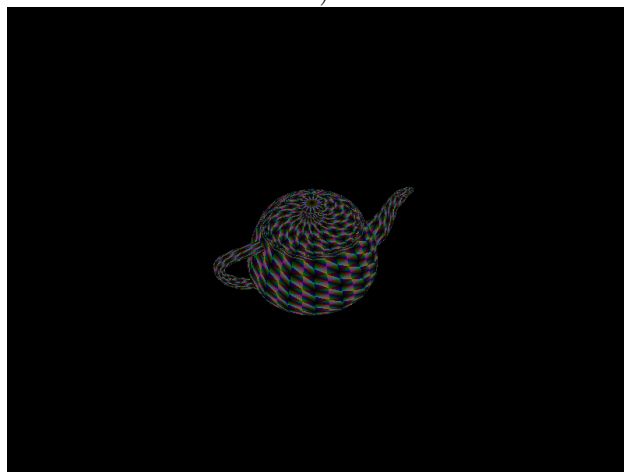
Результат работы:



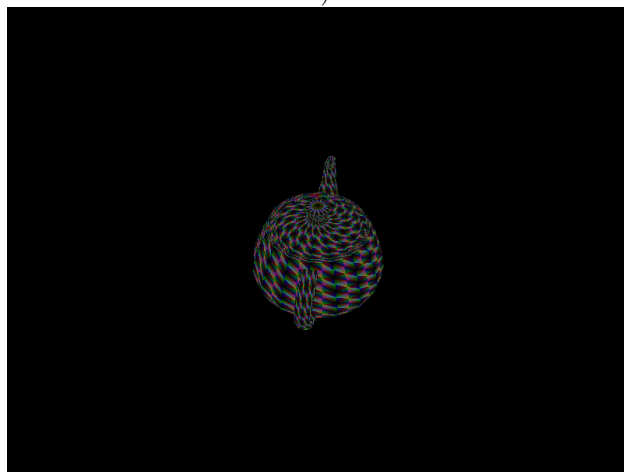
a)



b)



c)



d)

Рис. 1.6: Последовательно создаваемые изображения

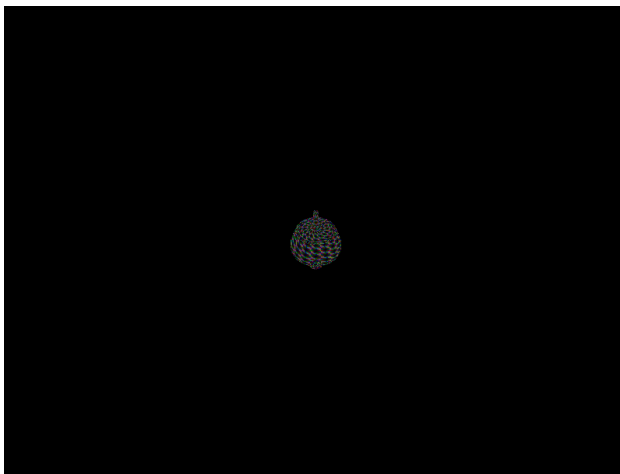
Приведем еще несколько результатов, изменяя параметры камеры:

Параметры:

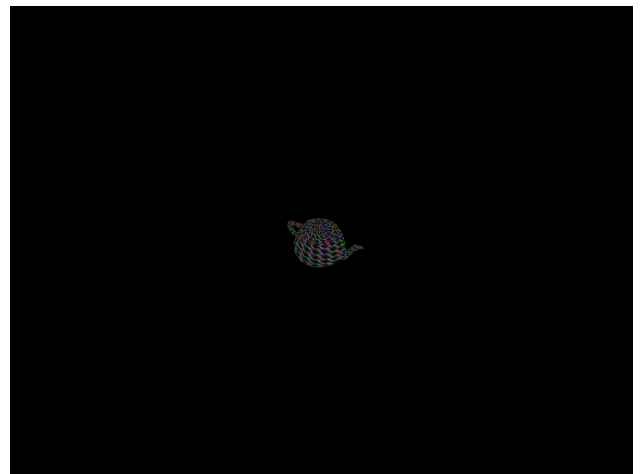
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<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

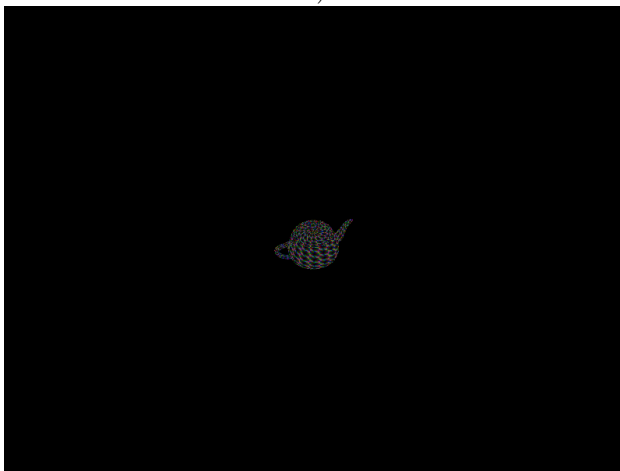
Результат работы:



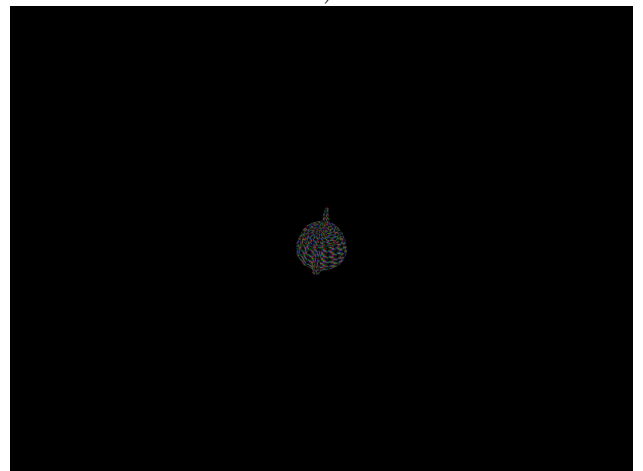
a)



b)



c)



d)

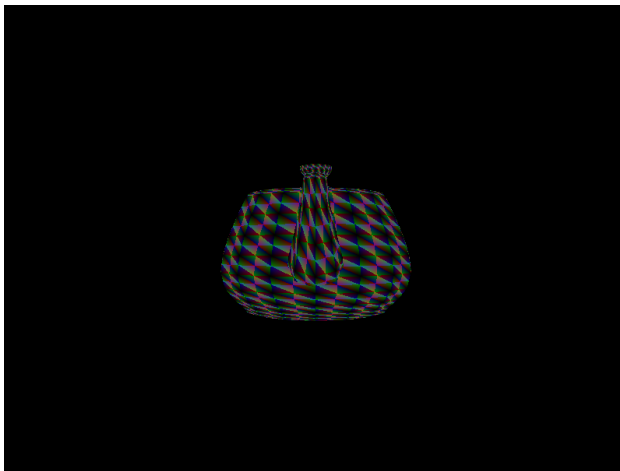
Рис. 1.8: Последовательно создаваемые изображения

Параметры:

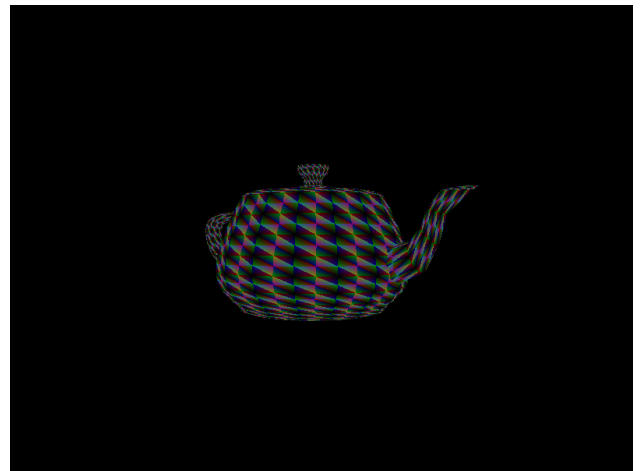
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<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

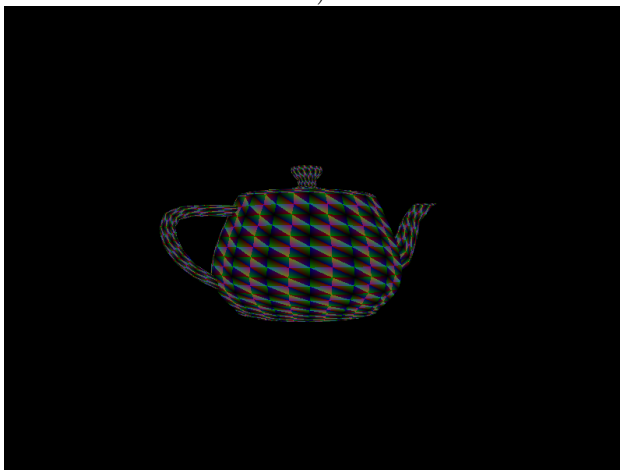
Результат работы:



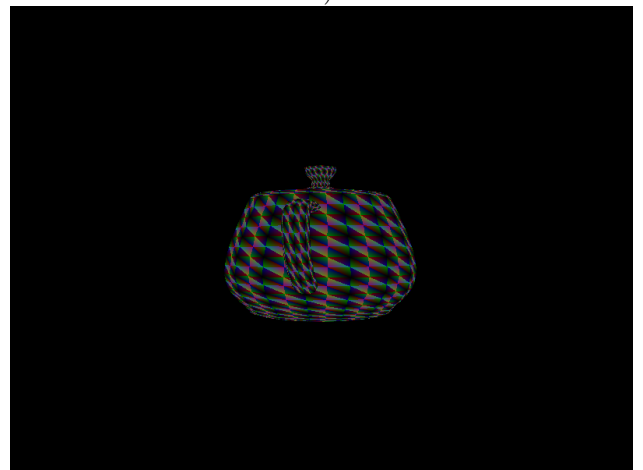
a)



b)



c)



d)

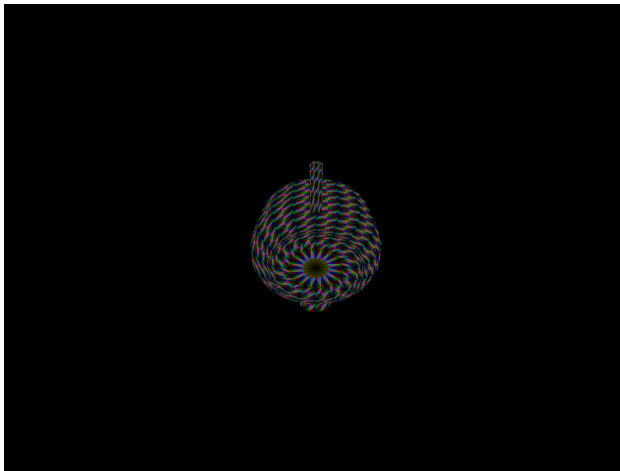
Рис. 1.10: Последовательно создаваемые изображения

Параметры:

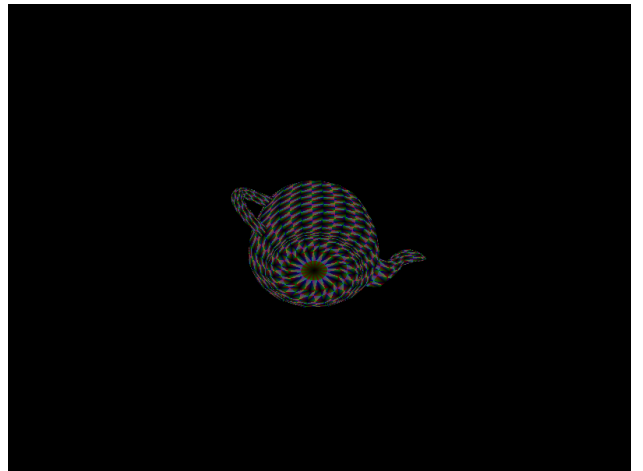
```
options.add_options()  
    ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))  
    ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))  
    ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("10.0"))  
    ("v,fovy", "fovy", cxxopts::value<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

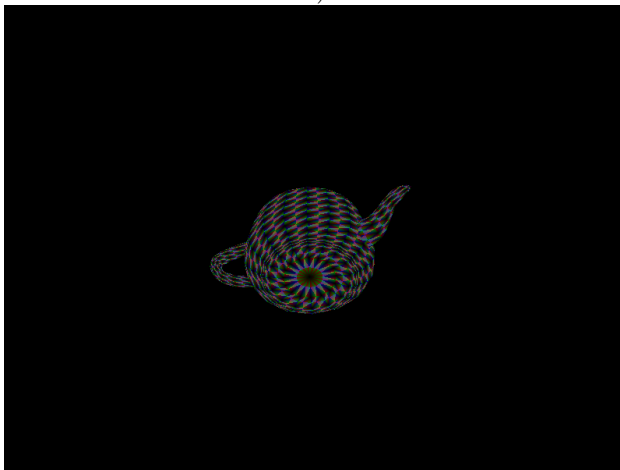
Результат работы:



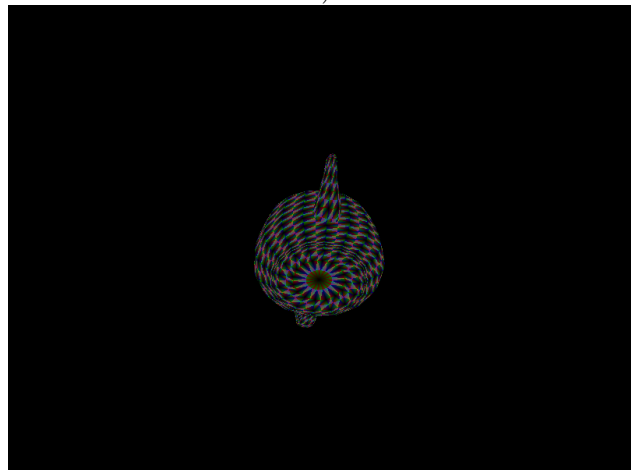
a)



b)



c)



d)

Рис. 1.12: Последовательно создаваемые изображения

1.4.4 Z-буффер

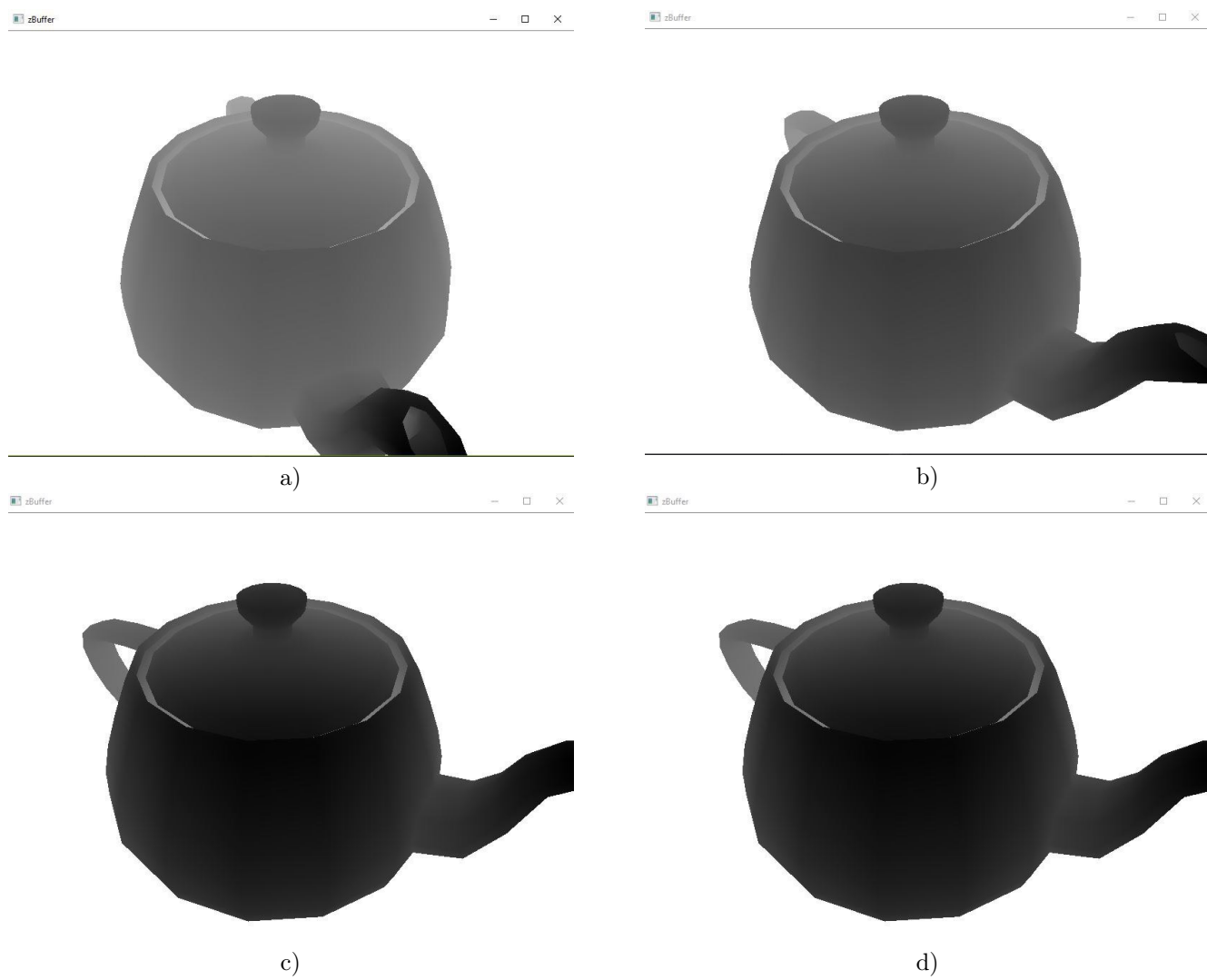


Рис. 1.13: Последовательно создаваемые изображения

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 Листинги

1.6.1 Листинг 1. main.cpp

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

```

60 }
61
62 void render(Drawer &drawer,
63             const std::vector<glm::vec3> &vertices,
64             const std::vector<glm::vec3> &normals,
65             const std::vector<glm::vec2> &texture,
66             const std::vector<unsigned int> &indices
67 ) {
68     for (auto i = 0; i < indices.size(); i += 3) {
69         Triangle triangle(vertices[indices[i]], vertices[indices[i + 1]], vertices[
indices[i + 2]]);
70         TriangleTexture tex{texture[indices[i]], texture[indices[i + 1]], texture[indices
[i + 2]]};
71         TriangleNormal norm{normals[indices[i]], normals[indices[i + 1]], normals[indices
[i + 2]]};
72         triangle.normal = norm;
73         triangle.texture = tex;
74         drawer.draw(triangle);
75     }
76 }
77
78 int main(int argc, char **argv) {
79     cxxopts::Options options("Lba3", "Render teapot and maybe something else");
80     std::string default_file_path = "../Corvette-F3.obj";
81     std::string default_texture_path = "../SF_Corvette-F3_diffuse.jpg";
82     std::string default_save_path = "../Corvette-F3.avi";
83
84     options.add_options()
85         ("w,width", "Width of image", cxxopts::value<int>()->default_value("800"))
86         ("h,height", "Height of image", cxxopts::value<int>()->default_value("600"))
87         ("s,speed", "Camera speed", cxxopts::value<float>()->default_value("2.0"))
88         ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-50.0"))
89         ("dx", "Distance to model", cxxopts::value<int>()->default_value("2000"))
90         ("dy", "Distance to model", cxxopts::value<int>()->default_value("2000"))
91         ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("0.1"))
92         ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("30000.0"
))
93         ("i,in_file", "Input filename ", cxxopts::value<std::string>()->default_value
(default_file_path))
94         ("t,texture", "Texture filename ", cxxopts::value<std::string>()->
default_value(default_texture_path))
95         ("o,out_file", "Output filename ", cxxopts::value<std::string>()->
default_value(default_save_path));
96
97
98     auto &&arguments = options.parse(argc, argv);
99
100     auto &&width = arguments["width"].as<int>();
101     auto &&height = arguments["height"].as<int>();
102     auto &&speed = arguments["speed"].as<float>();
103     auto &&fovy = arguments["fovy"].as<float>();
104     auto &&distanceX = arguments["dx"].as<int>();
105     auto &&distanceY = arguments["dy"].as<int>();
106     auto &&front = arguments["front"].as<float>();
107     auto &&back = arguments["back"].as<float>();
108     auto &&file_name = arguments["in_file"].as<std::string>();
109     auto &&texture_file_name = arguments["texture"].as<std::string>();
110     auto &&res_file_name = arguments["out_file"].as<std::string>();
111
112     objl::Loader loader;
113     loader.LoadFile(file_name);
114     auto &&mesh = loader.LoadedMeshes[0];
115
116     auto &&model_vertices = ToGLMVertices().applyList<objl::Vertex, glm::vec3>(mesh.
Vertices);

```

```

117     auto &&model_normals = ToGLMNormals().applyList<objl::Vertex, glm::vec3>(mesh.
Vertices);
118     auto &&model_texture_coordinates = ToGLMTexture().applyList<objl::Vertex, glm::vec2>(
mesh.Vertices);
119     cv::Mat3b texture = cv::imread(texture_file_name);
120
121     MipMap mipMap = MipMap<8>::compute(texture);
122
123     auto &&model_center = getModelCenter(model_vertices);
124
125     auto &&screen_ratio = static_cast<float>(width) / static_cast<float>(height);
126     auto &&projection = glm::perspective(
127         glm::radians(fovy),
128         screen_ratio,
129         front,
130         back
131     );
132
133     float angle = 0;
134     float angle_per_frame = speed / FRAME_PER_SECOND;
135
136     auto &&start_camera_position = glm::vec4(distanceX, distanceY, 0, 1);
137
138     TriangleDrawer drawer(width, height);
139     drawer.mipMap = mipMap;
140
141     for (auto i = 0; i < FRAME_COUNT; i++) {
142         drawer.resetImage();
143         glm::mat4 rotation_matrix = glm::rotate(glm::mat4(1), angle, glm::vec3(0, 1, 0));
144         glm::vec3 camera_position = (rotation_matrix * start_camera_position);
145         auto &&camera = glm::lookAt(
146             camera_position,
147             model_center,
148             glm::vec3(0, 1, 0)
149         );
150
151         double min, max;
152         cv::minMaxLoc(drawer.zBuffer, &min, &max);
153         cv::Mat zNorm;
154         cv::normalize(drawer.zBuffer, zNorm, 0.0, 1.0, cv::NORM_MINMAX, CV_64F);
155         cv::imshow("ZZZZ", zNorm);
156         cv::waitKey(2000);
157
158         angle += angle_per_frame;
159     }
160     return 0;
161 }

```

1.6.2 Листинг 2. Drawer.h

```
1 #pragma once
2
3 #include "render.h"
4 #include "transformers.h"
5
6
7 class Drawer {
8 public:
9     explicit Drawer(int width, int height) : renderer(width, height) {};
10
11     virtual void draw(const Triangle &t) = 0;
12
13     cv::Mat getImage() const {
14         return renderer.getImage();
15     }
16
17     virtual void resetImage() {
18         renderer.resetImage();
19     }
20
21 protected:
22     Renderer renderer;
23 };
24
25
26 class LineDrawer : public Drawer {
27 public:
28     LineDrawer(int width, int height) : Drawer(width, height), pipeline() {}
29
30     void draw(const Triangle &triangle) override {
31         auto &&t = pipeline->apply(triangle);
32         renderer.line(t.aCoord(), t.bCoord());
33         renderer.line(t.bCoord(), t.cCoord());
34         renderer.line(t.cCoord(), t.aCoord());
35     }
36
37     void updatePipeline(std::unique_ptr<TransformationPipeline<Triangle, Triangle>>
38 newPipeline) {
39         pipeline = std::move(newPipeline);
40     }
41
42 protected:
43     std::unique_ptr<TransformationPipeline<Triangle, Triangle>> pipeline;
44 };
45
46 class CvLineDrawer : public LineDrawer {
47     using LineDrawer::LineDrawer;
48
49     void draw(const Triangle &triangle) override {
50         auto &&t = pipeline->apply(triangle);
51         renderer.cv_line(t.aCoord(), t.bCoord());
52         renderer.cv_line(t.bCoord(), t.cCoord());
53         renderer.cv_line(t.cCoord(), t.aCoord());
54     }
55 };
56
57 class TriangleDrawer : public Drawer {
58 public:
59     TriangleDrawer(int width, int height) : Drawer(width, height), zBuffer(
60 zBufferDefaultValue(height, width)),
61                                     pipeline() {}
62
63     void draw(const Triangle &triangle) override {
```

```

63         auto &&t = pipeline->apply(triangle);
64         renderer.triangleBarycentric(t, zBuffer, mipMap);
65     }
66
67     cv::Mat1d zBufferDefaultValue(int height, int width) {
68         return cv::Mat::ones(height, width, CV_64F) * 100000;
69     }
70
71     void resetImage() override {
72         renderer.resetImage();
73         zBuffer = zBufferDefaultValue(renderer.height, renderer.width);
74     }
75
76
77     void updatePipeline(std::unique_ptr<TransformationPipeline<Triangle, ExtendedTriangle>> newPipeline) {
78         pipeline = std::move(newPipeline);
79     }
80
81     //protected:
82     std::unique_ptr<TransformationPipeline<Triangle, ExtendedTriangle>> pipeline;
83     cv::Mat1d zBuffer;
84     MipMap<8> mipMap;
85 };

```

1.6.3 Листинг 3. render.h

```
1 #pragma once
2
3 #include <iostream>
4
5 #include <opencv2/core.hpp>
6 #include <opencv2/highgui.hpp>
7 #include <opencv2/imgcodecs.hpp>
8 #include <opencv2/imgproc.hpp>
9
10 #include <glm/vec3.hpp>
11 #include <glm/vec4.hpp>
12 #include <glm/mat3x3.hpp>
13 #include <glm/geometric.hpp>
14
15 struct Coord {
16     int x, y;
17
18     cv::Point toCV() const {
19         return cv::Point(x, y);
20     }
21 };
22
23 struct TriangleNormal {
24     glm::vec3 a, b, c;
25 };
26
27 struct TriangleTexture {
28     glm::vec2 a, b, c;
29 };
30
31
32 template<typename T>
33 struct TriangleBase {
34     TriangleBase(const T &aa, const T &bb, const T &cc) : a(aa), b(bb), c(cc) {}
35
36     virtual Coord toCoord(const T &v) const = 0;
37
38     Coord aCoord() const {
39         return toCoord(a);
40     }
41
42     Coord bCoord() const {
43         return toCoord(b);
44     }
45
46     Coord cCoord() const {
47         return toCoord(c);
48     }
49
50     virtual glm::vec3 toVec(const T &v) const = 0;
51
52     glm::vec3 aVec() const {
53         return toVec(a);
54     }
55
56     glm::vec3 bVec() const {
57         return toVec(b);
58     }
59
60     glm::vec3 cVec() const {
61         return toVec(c);
62     }
63
64 }
```

```

65     T a, b, c;
66     TriangleNormal normal;
67     TriangleTexture texture;
68
69 };
70
71 inline cv::Vec3b getFrom(const cv::Mat3b &image, const glm::vec2 &coords) {
72     return image.at<cv::Vec3b>(
73         static_cast<int>(coords.y),
74         static_cast<int>(coords.x)
75     );
76 }
77
78 struct Triangle : public TriangleBase<glm::vec3> {
79     using TriangleBase::TriangleBase;
80
81     Coord toCoord(const glm::vec3 &v) const override {
82         int x = static_cast<int>(v.x);
83         int y = static_cast<int>(v.y);
84         return {x, y};
85     }
86
87     glm::vec3 toVec(const glm::vec3 &v) const override {
88         return v;
89     }
90 };
91
92 struct ExtendedTriangle : public TriangleBase<glm::vec4> {
93     using TriangleBase::TriangleBase;
94
95     Coord toCoord(const glm::vec4 &v) const override {
96         int x = static_cast<int>(v.x);
97         int y = static_cast<int>(v.y);
98         return {x, y};
99     }
100
101     glm::vec3 toVec(const glm::vec4 &v) const override {
102         return glm::vec3(v.x, v.y, v.z);
103     }
104
105
106     std::pair<Coord, Coord> box() const {
107         int minX = static_cast<int>(std::floor(std::min({a.x, b.x, c.x})));
108         int minY = static_cast<int>(std::floor(std::min({a.y, b.y, c.y})));
109         int maxX = static_cast<int>(std::ceil(std::max({a.x, b.x, c.x})));
110         int maxY = static_cast<int>(std::ceil(std::max({a.y, b.y, c.y})));
111         return std::make_pair(Coord{minX, minY}, Coord{maxX, maxY});
112     }
113
114 };
115
116
117 std::ostream &operator<<(std::ostream &res, const glm::vec4 &v) {
118     res << "(" << v.x << ", " << v.y << ", " << v.z << ")";
119     return res;
120 }
121
122 std::ostream &operator<<(std::ostream &res, const glm::vec3 &v) {
123     res << "(" << v.x << ", " << v.y << ", " << v.z << ")";
124     return res;
125 }
126
127
128 std::ostream &operator<<(std::ostream &res, const glm::vec2 &v) {
129     res << "(" << v.x << ", " << v.y << ")";
130     return res;

```

```

131 }
132
133 template<int level = 8>
134 struct MipMap {
135     static MipMap<level> compute(const cv::Mat &original) {
136         MipMap<level> mipmap;
137         mipmap.atLevel[0] = original;
138         for (auto i = 1; i < level; i++) {
139             mipmap.atLevel[i] = imageAverage(mipmap.atLevel[i - 1]);
140         }
141         return mipmap;
142     }
143
144     static cv::Mat imageAverage(const cv::Mat &image) {
145         auto height = image.size().height, width = image.size().width;
146         cv::Mat result(height / 2, width / 2, image.type());
147         for (auto x = 0; x < width; x += 2) {
148             for (auto y = 0; y < height; y += 2) {
149                 cv::Mat block = image(cv::Rect(x, y, 2, 2));
150                 cv::Scalar averaged = cv::mean(block);
151                 result.at<cv::Vec3b>(y / 2, x / 2) = cv::Vec3b(
152                     static_cast<uchar>(averaged[0]),
153                     static_cast<uchar>(averaged[1]),
154                     static_cast<uchar>(averaged[2])
155                 );
156             }
157         }
158         return result;
159     }
160
161     cv::Mat3b get(int d) const {
162         int realD = std::max(0, std::min(d, level - 1));
163         return atLevel[realD];
164     }
165
166     glm::vec2 scale(int d, const glm::vec2 &uv) const {
167         int realD = std::max(0, std::min(d, level - 1));
168         auto &&size = atLevel[realD].size();
169         return uv * glm::vec2(size.width, size.height);
170     }
171
172 private:
173     cv::Mat3b atLevel[level];
174 };
175
176
177 class Renderer {
178 public:
179     explicit Renderer(int w, int h) : width(w), height(h), image(cv::Mat::zeros(h, w,
180         CV_8U)) {}
181
182     void line(const Coord &from, const Coord &to) {
183         drawline(from.x, from.y, to.x, to.y);
184     }
185
186     void cv_line(const Coord &from, const Coord &to) {
187         cv::line(image, from.toCV(), to.toCV(), cv::Scalar(255, 0, 0));
188     }
189
190     template<int index>
191     inline float interpolateTextureCoordinate(const TriangleTexture &texture, const glm::
192         dvec3 &barycentric) const {
193         return static_cast<float>(texture.a[index] * barycentric[0]
194             + texture.b[index] * barycentric[1]
195             + texture.c[index] * barycentric[2]);
196     }
197
198     }

```



```

195
196 glm::vec2
197 interpolateTexture(const TriangleTexture &texture, const glm::dvec3 &barycentric)
198 const {
199     auto u = interpolateTextureCoordinate<0>(texture, barycentric);
200     auto v = interpolateTextureCoordinate<1>(texture, barycentric);
201     return glm::vec2(u, v);
202 }
203
204 void triangleBarycentric(const ExtendedTriangle &t, cv::Mat1d &zBuffer, const MipMap
205 <8> &mipMap) {
206     auto &&bbox = t.bbox();
207     Coord from = bound(bbox.first);
208     Coord to = bound(bbox.second);
209     for (auto x = from.x; x <= to.x; ++x) {
210         for (auto y = from.y; y <= to.y; ++y) {
211             glm::dvec3 barycentric;
212             bool pointInTriangle = get_barycentric(t, glm::vec4(x, y, 1, 1),
213 barycentric);
214             if (!pointInTriangle) continue;
215             glm::dvec3 zInterpolated = glm::dvec3(t.a.z, t.b.z, t.c.z) * barycentric;
216             double z = zInterpolated[0] + zInterpolated[1] + zInterpolated[2];
217             if (z < zBuffer.at<double>(y, x)) {
218                 zBuffer.at<double>(y, x) = z;
219
220                 glm::dvec3 barycentricX, barycentricY;
221                 get_barycentric(t, glm::vec4(x + 1, y, 1, 1), barycentricX);
222                 get_barycentric(t, glm::vec4(x, y + 1, 1, 1), barycentricY);
223
224                 glm::vec2 uv = interpolateTexture(t.texture, barycentric);
225                 glm::vec2 uvX = interpolateTexture(t.texture, barycentricX);
226                 glm::vec2 uvY = interpolateTexture(t.texture, barycentricY);
227
228                 glm::vec2 dx = uvX - uv;
229                 glm::vec2 dy = uvY - uv;
230
231                 float l = std::max(glm::dot(dx, dx), glm::dot(dy, dy));
232                 float d = std::max(0.0f, 0.5f * std::log2(l));
233
234                 cv::Vec3b texel = trilinearFilter(uv, d, mipMap);
235                 plot(x, y, texel);
236             }
237         }
238     }
239 }
240
241 cv::Mat getImage() const {
242     return image;
243 }
244
245 void resetImage() {
246     image = cv::Mat::zeros(height, width, CV_8UC3);
247 }
248
249 int width, height;
250
251 private:
252
253 cv::Vec3b interpolate(const cv::Mat3b &level, const glm::vec2 &xy, const glm::vec2 &
254 dxy) const {
255     return (1 - dxy.x) * (1 - dxy.y) * getFrom(level, xy)
256         + dxy.x * (1 - dxy.y) * getFrom(level, xy + glm::vec2(1, 0))
257         + (1 - dxy.x) * dxy.y * getFrom(level, xy + glm::vec2(0, 1))

```

```

257         + (dxy.x * dxy.y) * getFrom(level, xy + glm::vec2(1, 1));
258
259     }
260
261     cv::Vec3b trilinearFilter(const glm::vec2 &uv, float df, const MipMap<8> &mipmap)
262     const {
263         int d = static_cast<int>(std::floor(df));
264         float dd = df - d;
265         glm::vec2 fxy0 = mipmap.scale(d, uv);
266         glm::vec2 fxy1 = mipmap.scale(d + 1, uv);
267
268         glm::vec2 xy0 = glm::floor(fxy0);
269         glm::vec2 xy1 = glm::floor(fxy1);
270
271         glm::vec2 dxy0 = fxy0 - xy0;
272         glm::vec2 dxy1 = fxy1 - xy1;
273
274         cv::Mat3b &&level0 = mipmap.get(d);
275         cv::Mat3b &&level1 = mipmap.get(d + 1);
276
277         cv::Vec3b v0 = interpolate(level0, xy0, dxy0);
278         cv::Vec3b v1 = interpolate(level1, xy1, dxy1);
279
280         return (1 - dd) * v0 + dd * v1;
281     }
282
283     inline int bound(int x, int bnd) {
284         return std::max(0, std::min(bnd - 1, x));
285     }
286
287     inline Coord bound(const Coord &coord) {
288         return Coord{bound(coord.x, width), bound(coord.y, height)};
289     }
290
291     inline double edgeFunction(const glm::vec4 &a, const glm::vec4 &b, const glm::vec4 &c) {
292         return (c.x - a.x) * (b.y - a.y) - (c.y - a.y) * (b.x - a.x);
293     }
294
295     bool get_barycentric(const ExtendedTriangle &t, const glm::vec4 &p, glm::dvec3 &
296     barycentric) {
297         auto area = edgeFunction(t.a, t.b, t.c);
298         auto w0 = edgeFunction(t.b, t.c, p) / area;
299         auto w1 = edgeFunction(t.c, t.a, p) / area;
300         auto w2 = edgeFunction(t.a, t.b, p) / area;
301
302         barycentric = glm::dvec3(w0, w1, w2);
303         barycentric /= barycentric[0] + barycentric[1] + barycentric[2];
304         barycentric = glm::abs(barycentric);
305
306         return !(w0 < 0 || w1 < 0 || w2 < 0);
307     }
308
309     inline bool check_coordinates(int x, int y) {
310         if (x < 0) return false;
311         if (x >= width) return false;
312         if (y < 0) return false;
313         if (y >= height) return false;
314         return true;
315     }
316
317     inline void plot(int x, int y, const glm::vec3 &color) {
318         if (!check_coordinates(x, y))
319             return;
320
321         image.at<cv::Vec3b>(y, x) = cv::Vec3b(

```

```

320         static_cast<uchar>(color[0] * 255),
321         static_cast<uchar>(color[1] * 255),
322         static_cast<uchar>(color[2] * 255)
323     );
324 }
325
326 inline void plot(int x, int y, const cv::Vec3b &color) {
327     if (!check_coordinates(x, y))
328         return;
329     image.at<cv::Vec3b>(y, x) = color;
330 }
331
332
333 void line(const Coord &from, const Coord &to, std::vector<Coord> &result) {
334     drawline(from.x, from.y, to.x, to.y, [&](int x, int y) mutable {
335         if (check_coordinates(x, y))
336             result.emplace_back(Coord{x, y});
337     });
338 }
339
340 void drawline(int x0, int y0, int x1, int y1) {
341     drawline(x0, y0, x1, y1, [&](int x, int y) { plot(x, y, glm::vec3(0.5f)); });
342 }
343
344 template<typename F>
345 inline void drawline(int x0, int y0, int x1, int y1, F plot) {
346
347     int dx = std::abs(x1 - x0);
348     int dy = std::abs(y1 - y0);
349
350     int directionX = x0 < x1 ? 1 : -1;
351     int directionY = y0 < y1 ? 1 : -1;
352     int err = (dx > dy ? dx : -dy) / 2;
353
354     for (;;) {
355         plot(x0, y0);
356         if (x0 == x1 && y0 == y1) break;
357         int e2 = err;
358         if (e2 > -dx) {
359             err -= dy;
360             x0 += directionX;
361         }
362         if (e2 < dy) {
363             err += dx;
364             y0 += directionY;
365         }
366     }
367 }
368
369 inline void orderVerticesByY(Coord &v1, Coord &v2, Coord &v3) {
370     if (v1.y > v2.y) {
371         std::swap(v1, v2);
372     }
373     if (v2.y > v3.y) {
374         std::swap(v2, v3);
375     }
376     if (v1.y > v2.y) {
377         std::swap(v1, v2);
378     }
379 }
380
381 inline int signum(int x) {
382     if (x == 0) return 0;
383     else if (x > 0) return 1;
384     else return -1;
385 }

```

```

386
387 void fillFlatSideTriangle(const Coord &v1, const Coord &v2, const Coord &v3, std::
vector<Coord> &trianglePixels) {
388
389     int dx1 = std::abs(v2.x - v1.x);
390     int dy1 = std::abs(v2.y - v1.y);
391
392     int dx2 = std::abs(v3.x - v1.x);
393     int dy2 = std::abs(v3.y - v1.y);
394
395     int signx1 = signum(v2.x - v1.x);
396     int signx2 = signum(v3.x - v1.x);
397
398     int signy1 = signum(v2.y - v1.y);
399     int signy2 = signum(v3.y - v1.y);
400
401     bool edge1Steep = dy1 > dx1;
402     bool edge2Steep = dy2 > dx2;
403
404     if (edge1Steep) {
405         std::swap(dx1, dy1);
406     }
407     if (edge2Steep) {
408         std::swap(dx2, dy2);
409     }
410
411     Coord vTmp1{v1.x, v1.y};
412     Coord vTmp2{v1.x, v1.y};
413
414     int e1 = 2 * dy1 - dx1;
415     int e2 = 2 * dy2 - dx2;
416
417     for (int i = 0; i <= dx1; i++) {
418         line(vTmp1, vTmp2, trianglePixels);
419
420         while (e1 >= 0 && dx1 > 0) {
421             if (edge1Steep) {
422                 vTmp1.x += signx1;
423             } else {
424                 vTmp1.y += signy1;
425             }
426             e1 = e1 - 2 * dx1;
427         }
428
429         if (edge1Steep) {
430             vTmp1.y += signy1;
431         } else {
432             vTmp1.x += signx1;
433         }
434
435         e1 = e1 + 2 * dy1;
436
437         while (vTmp2.y != vTmp1.y) {
438             while (e2 >= 0 && dx2 > 0) {
439                 if (edge2Steep) {
440                     vTmp2.x += signx2;
441                 } else {
442                     vTmp2.y += signy2;
443                 }
444                 e2 = e2 - 2 * dx2;
445             }
446
447             if (edge2Steep) {
448                 vTmp2.y += signy2;
449             } else {
450                 vTmp2.x += signx2;

```

```

451         }
452
453         e2 = e2 + 2 * dy2;
454     }
455 }
456 }
457 }
458
459
460 std::vector<Coord> rasterizeTriangle(Coord v1, Coord v2, Coord v3) {
461     std::vector<Coord> trianglePixels;
462     orderVerticesByY(v1, v2, v3);
463     if (v2.y == v3.y) {
464         fillFlatSideTriangle(v1, v2, v3, trianglePixels);
465     } else if (v1.y == v2.y) {
466         fillFlatSideTriangle(v3, v1, v2, trianglePixels);
467     } else {
468         float v2mv1 = v2.y - v1.y;
469         float v3mv1 = v3.y - v1.y;
470         int v4x = static_cast<int>(v1.x + (v2mv1 / v3mv1) * (v3.x - v1.x));
471         Coord v4{v4x, v2.y};
472         fillFlatSideTriangle(v1, v2, v4, trianglePixels);
473         fillFlatSideTriangle(v3, v2, v4, trianglePixels);
474     }
475     return trianglePixels;
476 }
477
478
479 cv::Mat image;
480 };

```

1.6.4 Листинг 4. transformers.h

```
1 #pragma once
2
3 #include <any>
4 #include <utility>
5 #include <vector>
6 #include <memory>
7
8 #include <OBJ_Loader.h>
9
10 #include <glm/vec3.hpp>
11 #include <glm/geometric.hpp>
12 #include <glm/gtc/matrix_transform.hpp>
13
14 #include "render.h"
15
16 class VertexTransformer {
17 public:
18     virtual std::any apply(const std::any &vex) = 0;
19
20     template<typename T, typename R>
21     std::vector<R> applyList(const std::vector<T> &vertices) {
22         std::vector<R> result;
23         for (auto &&v: vertices) {
24             result.emplace_back(std::any_cast<R>(apply(v)));
25         }
26         return result;
27     }
28 };
29
30 glm::vec3 to_glm(const objl::Vector3 &vec) {
31     return glm::vec3{vec.X, vec.Y, vec.Z};
32 }
33
34 glm::vec2 to_glm(const objl::Vector2 &vec) {
35     return glm::vec2{vec.X, vec.Y};
36 }
37
38 class ToGLMVertices : public VertexTransformer {
39
40
41     std::any apply(const std::any &vertex) override {
42         return to_glm(std::any_cast<objl::Vertex>(vertex).Position);
43     }
44 };
45
46 class ToGLMNormals : public VertexTransformer {
47
48
49     std::any apply(const std::any &vertex) override {
50         return to_glm(std::any_cast<objl::Vertex>(vertex).Normal);
51     }
52 };
53
54 class ToGLMTexture : public VertexTransformer {
55     std::any apply(const std::any &vertex) override {
56         return to_glm(std::any_cast<objl::Vertex>(vertex).TextureCoordinate);
57     }
58 };
59
60 template<typename T, typename R>
61 class TransformationPipeline {
62 public:
63     virtual R apply(const T &triangle) = 0;
64 };
```

```

65
66
67 class GLMLineTransformationPipeline : public TransformationPipeline<Triangle, Triangle> {
68
69 public:
70     explicit GLMLineTransformationPipeline(const glm::mat4 &cam, const glm::mat4 &proj,
71         int width, int height)
72         : camera(cam), projection(proj), viewport(0, 0, width, height), w(width), h(
73             height) {}
74
75     glm::vec3 transformVertex(const glm::vec3 &vertex) {
76         auto &&scaled = glm::project(vertex, camera, projection, viewport);
77         return scaled;
78     }
79
80     Triangle apply(const Triangle &t) override {
81         return {transformVertex(t.a), transformVertex(t.b), transformVertex(t.c)};
82     }
83
84 private:
85     glm::mat4 camera, projection;
86     glm::vec4 viewport;
87     int w, h;
88 };
89
90 class LineTransformationPipeline : public TransformationPipeline<Triangle, Triangle> {
91 public:
92     explicit LineTransformationPipeline(const glm::mat4 &cam, const glm::mat4 &proj, int
93         width, int height)
94         : transformation(proj * cam), w(width), h(height), scale(width, height, 1) {}
95
96     glm::vec3 transformVertex(const glm::vec3 &vertex) {
97         auto homoCoord = glm::vec4(vertex, 1.0);
98         glm::vec4 v = transformation * homoCoord;
99         float w = v[3];
100         glm::vec3 cartesian = glm::vec3(v[0] / w, v[1] / w, v[2] / w);
101         glm::vec3 &&scaled = ((cartesian + 1.0f) / 2.0f) * scale;
102         return scaled;
103     }
104
105     Triangle apply(const Triangle &t) override {
106         return {transformVertex(t.a), transformVertex(t.b), transformVertex(t.c)};
107     }
108
109 private:
110     glm::mat4 transformation;
111     glm::vec3 scale;
112     int w, h;
113 };
114
115
116 class TriangleTransformationPipeline : public TransformationPipeline<Triangle,
117     ExtendedTriangle> {
118 public:
119     explicit TriangleTransformationPipeline(const glm::mat4 &cam, const glm::mat4 &proj,
120         int width, int height)
121         : transformation(proj * cam), w(width), h(height), scale(width, height, 1) {}
122
123     glm::vec4 transformVertex(const glm::vec3 &v) {
124         auto &&homoCoord = glm::vec4(v, 1.0);
125         glm::vec4 &&res = transformation * homoCoord;
126         float w = res[3];

```

```

126         glm::vec3 &&cartesian = glm::vec3(res.x / w, res.y / w, res.z / w);
127         glm::vec3 &&scaled = ((cartesian + glm::vec3(1.0f, 1.0f, 0.0f)) / glm::vec3(2.0f,
128         2.0f, 1.0f)) * scale;
129         return glm::vec4(scaled, w);
130     }
131     ExtendedTriangle apply(const Triangle &triangle) override {
132         ExtendedTriangle tri{transformVertex(triangle.a), transformVertex(triangle.b),
133         transformVertex(triangle.c)};
134         tri.texture = triangle.texture;
135         tri.normal = triangle.normal;
136         return tri;
137     }
138 private:
139     glm::mat4 transformation;
140     glm::vec3 scale;
141     int w, h;
142 };
143

```