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

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) Использования буффера глубины для отсечения невидимых пикселей
2. Входные параметры программы:
   (a) Ширина и высота окна
   (b) Вертикальный угол обзора камеры для выполненя 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 растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками.

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

```
    template<typename F>
    inline void drawline(int x0, int y0, int x1, int y1, F plot) {

        int dx = std::abs(x1 - x0);
        int dy = std::abs(y1 - y0);

        int directionX = x0 < x1 ? 1 : -1;
        int directionY = y0 < y1 ? 1 : -1;
        int err = (dx > dy ? dx : -dy) / 2;

        for (;;) {
            plot(x0, y0);
            if (x0 == x1 && y0 == y1) break;
            int e2 = err;
            if (e2 > -dx) {
                err -= dy;
                x0 += directionX;
            }
            if (e2 < dy) {
                err += dx;
                y0 += directionY;
            }
        }
    }
```

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:

$$v0.608654 - 0.568839 - 0.416318$$

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

$$f761$$

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

$$vn - 0.966742 - 0.2557529.97231e - 09$$

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 строит

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

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.

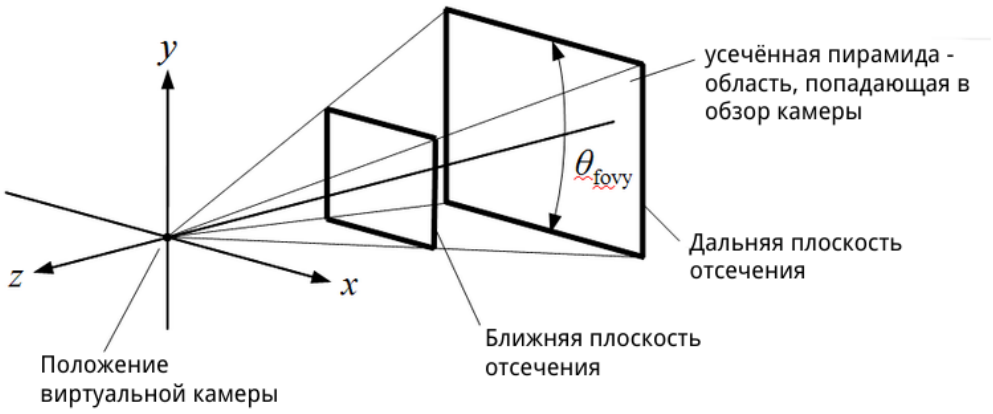
Настроим матрицу GL PROJECTION один раз для перспективного преобразования, а матрицу GL MODELVIEW будем постоянно модифицировать, когда локальная система координат очередного объекта не совпадает с мировой системой координат.

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):

```
camera = glm::lookAt(
                camera_position,
                model_center,
                glm::vec3(0, 1, 0)
        );
```

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:



```
projection = glm::perspective(
             glm::radians(fovy),
             screen_ratio,
             front,
             back)
```

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

1.4 Результаты

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

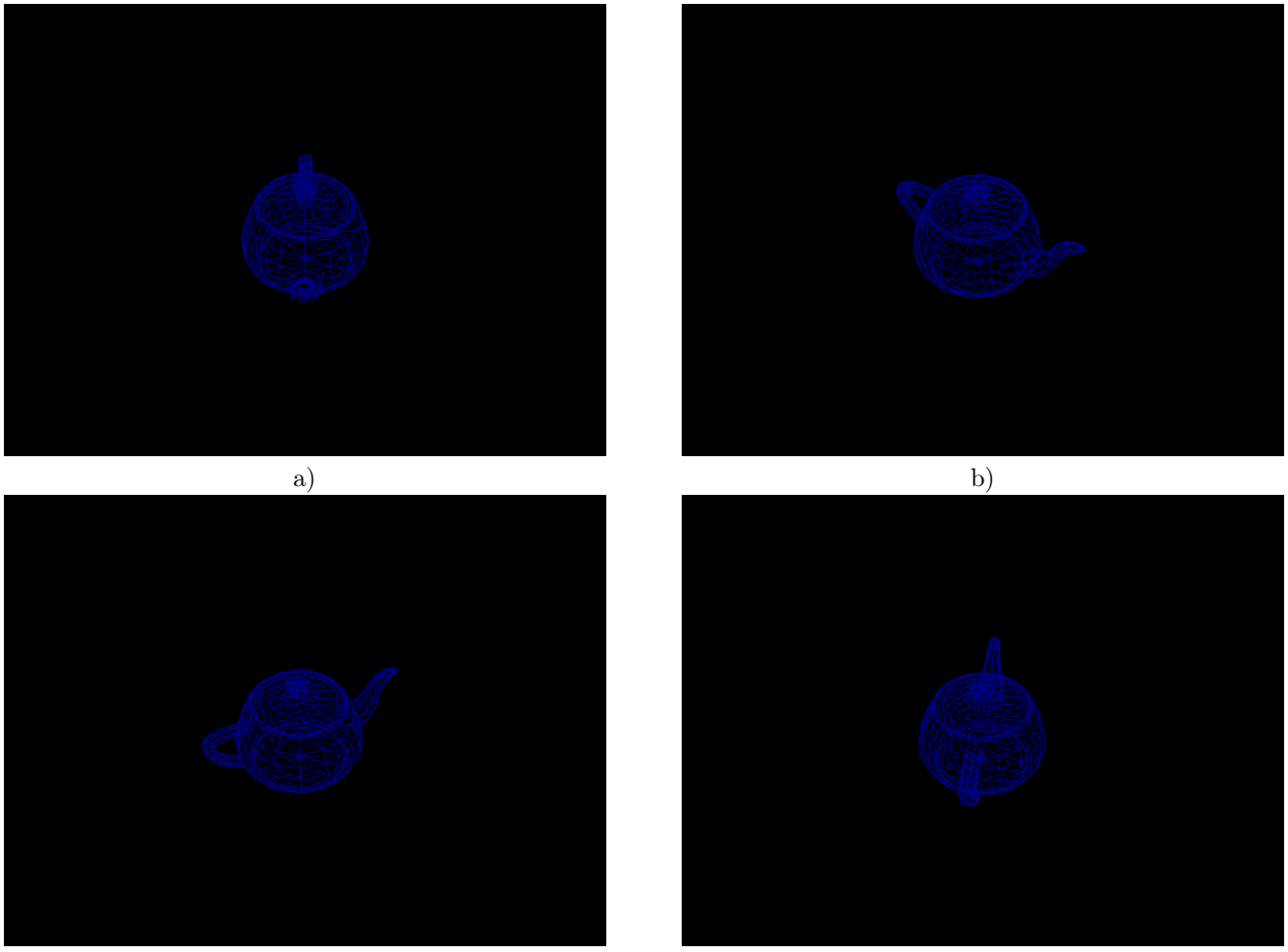
1.4.1 CvLineDrawer

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

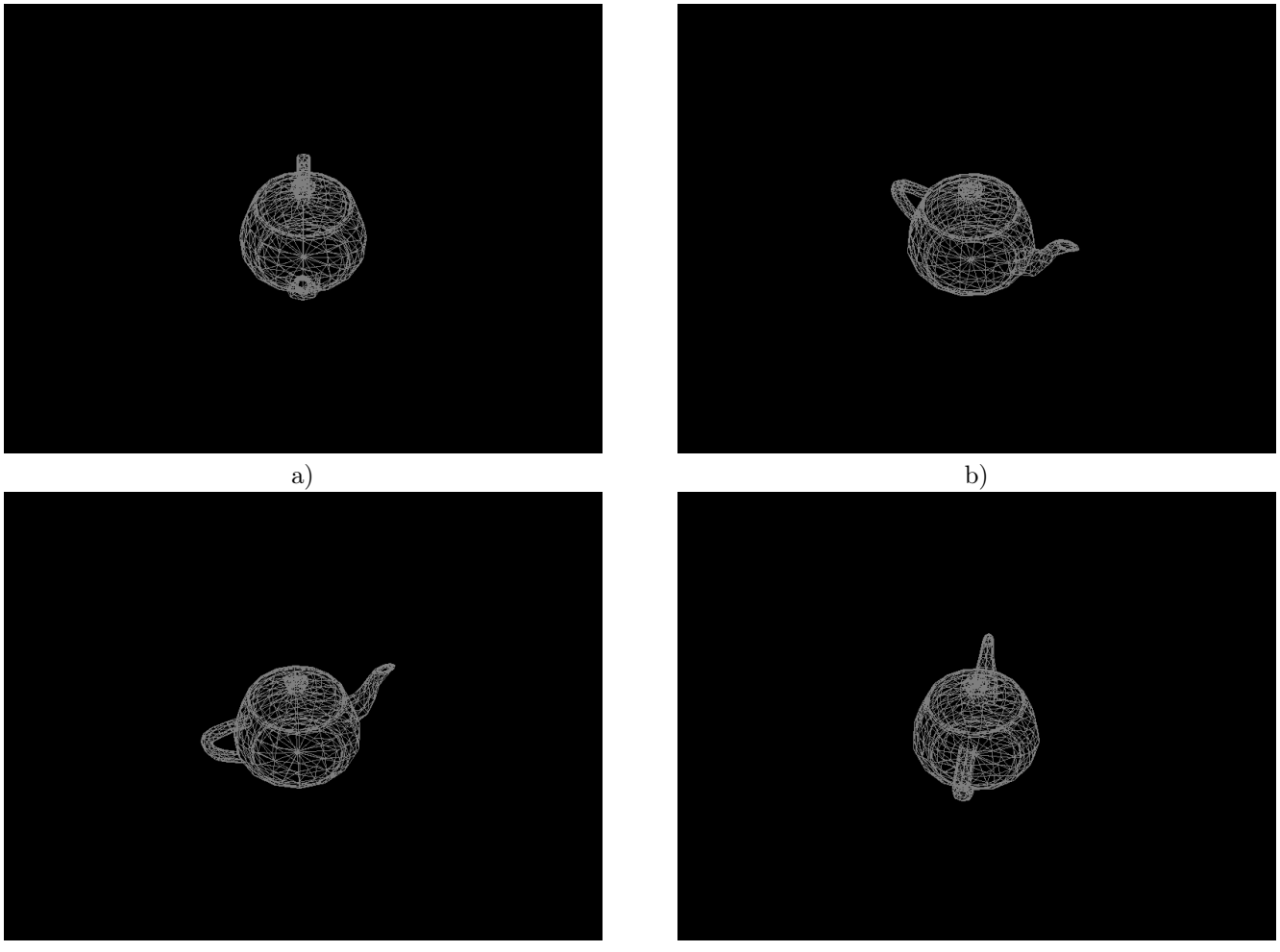
1.4.2 LineDrawer

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

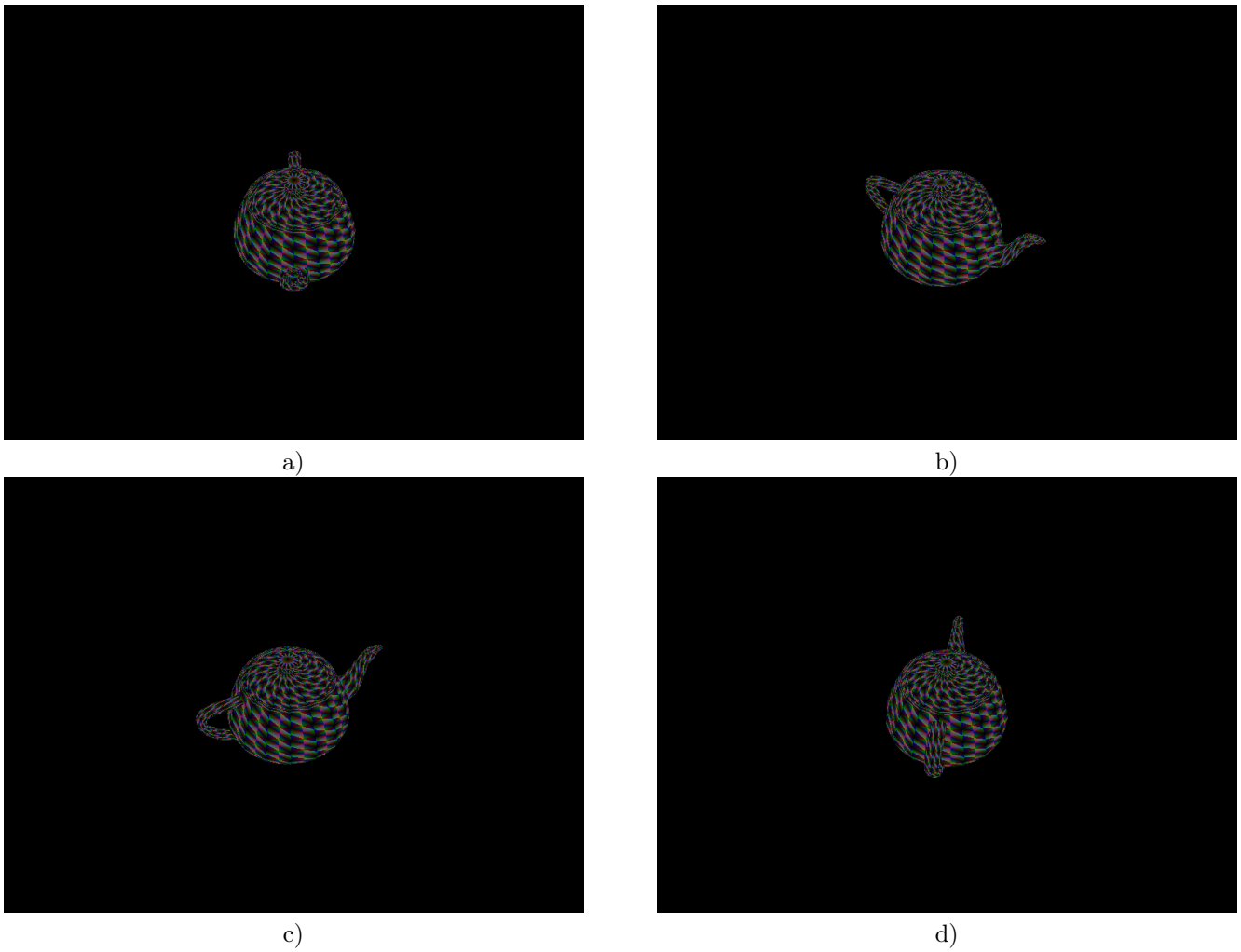
1.4.3 TriangleDrawer

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

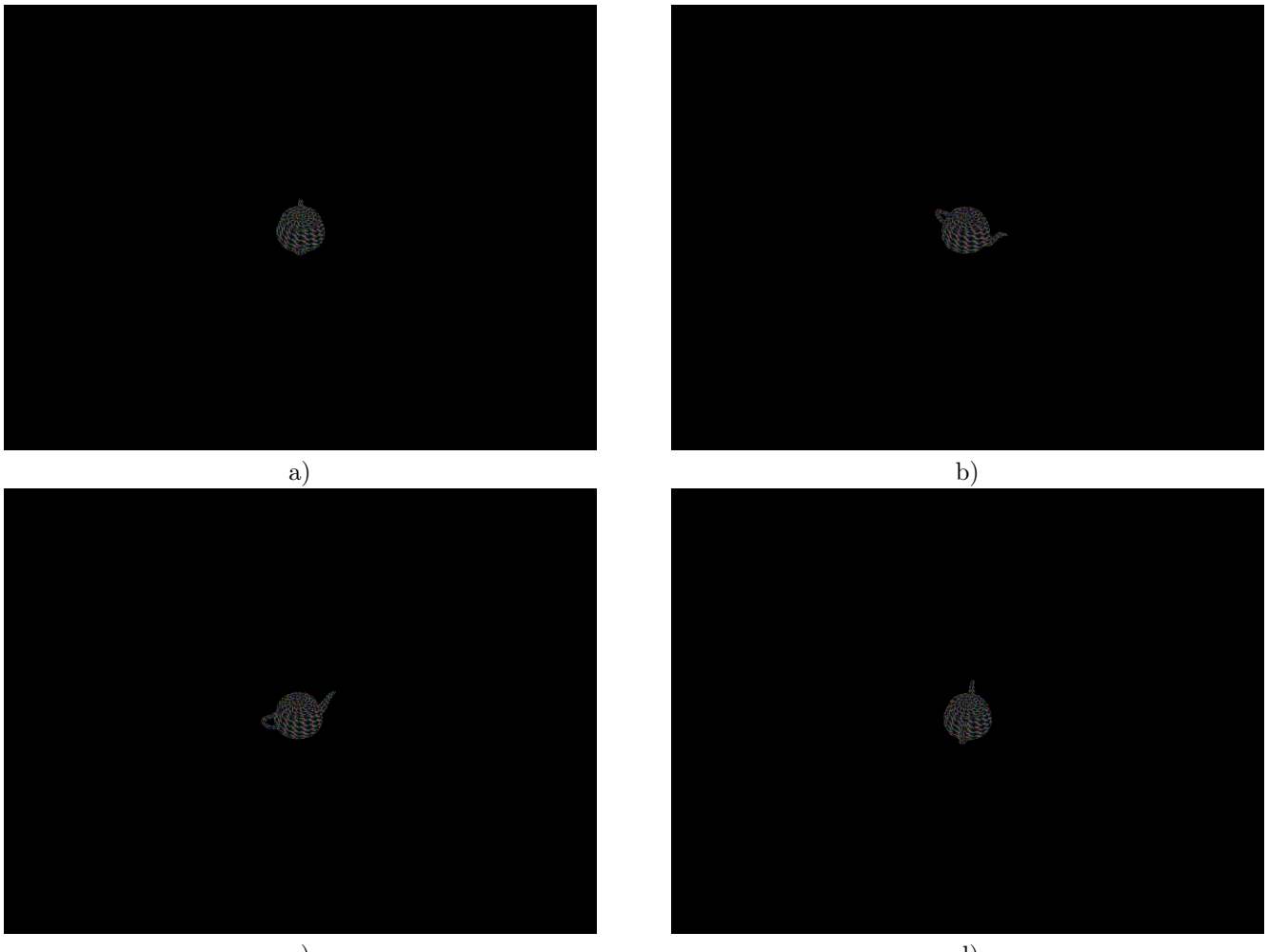
Приведем еще несколько результатов, изменяя параметры камеры:

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)

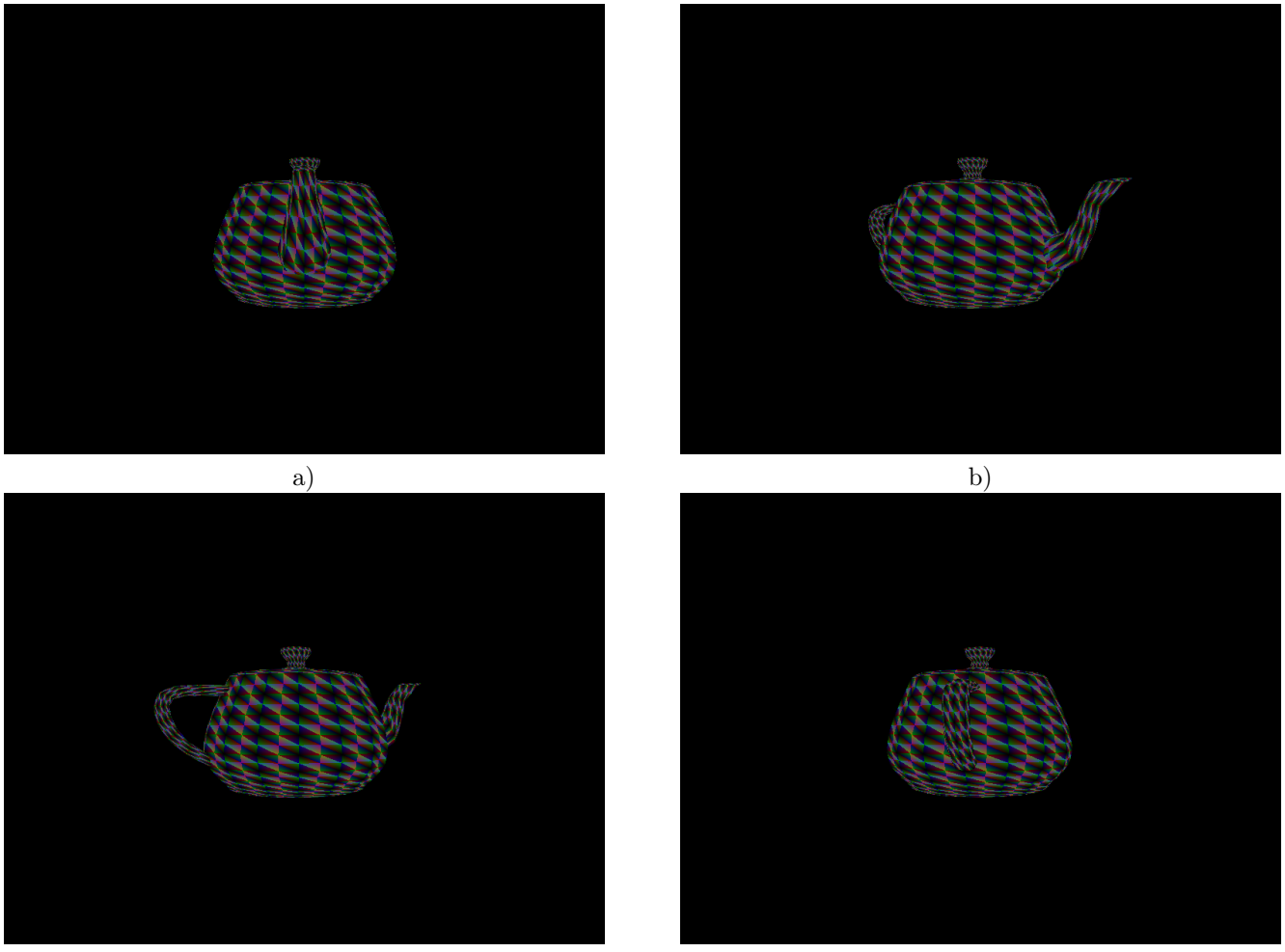
Рис. 1.8: Последовательно создаваемые изображения

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)

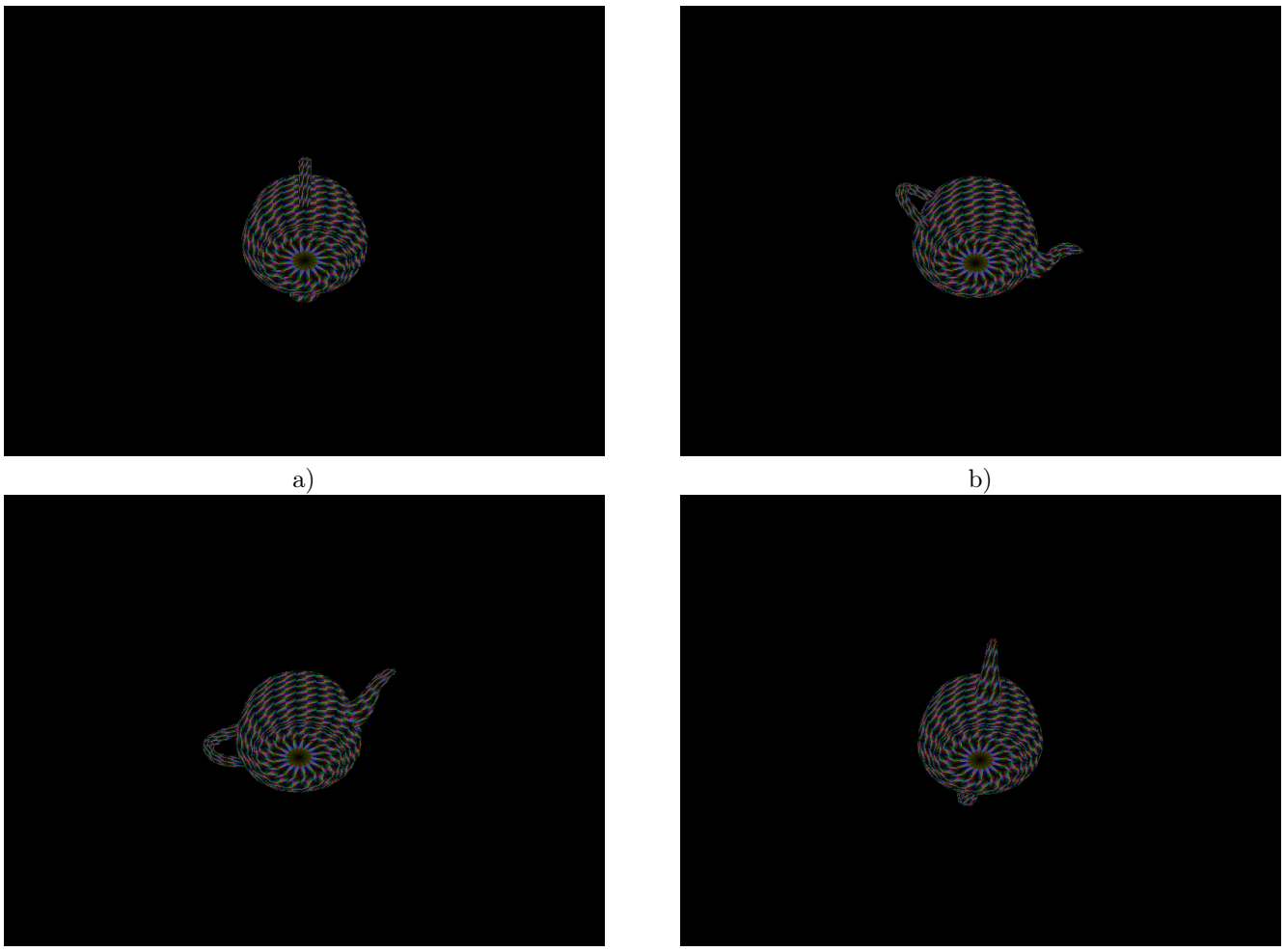
Рис. 1.10: Последовательно создаваемые изображения

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 OpenCV.

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

```
#include <utility>

#include <iostream>
#include <any>
#include <OBJ_Loader.h>
#include <cxxopts.hpp>

#include <glm/vec3.hpp>
#include <glm/geometric.hpp>
#include <glm/gtc/matrix_transform.hpp>

#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>

#include "render.h"
#include "Drawer.h"
#include "transformers.h"

const int FRAME_PER_SECOND = 10;
const int FRAME_COUNT = 10000;


template<int index>
float min(const std::vector<glm::vec3> &vertices) {
    float result = FLT_MAX;
    for (auto &&vex : vertices) {
        result = std::min(result, vex[index]);
    }
    return result;
}

template<int index>
float max(const std::vector<glm::vec3> &vertices) {
    float result = FLT_MIN;
    for (auto &&vex : vertices) {
        result = std::max(result, vex[index]);
    }
    return result;
}

template<int index>
float getCenter(const std::vector<glm::vec3> &vertices) {
    auto &&min_point = min<index>(vertices);
    auto &&max_point = max<index>(vertices);
    return (min_point + max_point) / 2;
}

glm::vec3 getModelCenter(const std::vector<glm::vec3> &vertices) {
    auto &&center_x = getCenter<0>(vertices);
    auto &&center_y = getCenter<1>(vertices);
    auto &&center_z = getCenter<2>(vertices);
    return glm::vec3(center_x, center_y, center_z);
}

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

```
}


int main(int argc, char **argv) {
    cxxopts::Options options("Lba3", "Render teapot and maybe something else");
    std::string default_file_path = "../teapot.obj";
    std::string default_save_path = "../teapot.avi";

    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("2.0"))
            ("v,fovy", "fovy", cxxopts::value<float>()->default_value("-50.0"))
            ("dx", "Distance to model", cxxopts::value<int>()->default_value("120"))
            ("dy", "Distance to model", cxxopts::value<int>()->default_value("100"))
            ("f,front", "Front cut plane", cxxopts::value<float>()->default_value("0.1"))
            ("b,back", "Back cut plane", cxxopts::value<float>()->default_value("10000.0"
    ))
            ("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));


    auto &&arguments = options.parse(argc, argv);

    auto &&width = arguments["width"].as<int>();
    auto &&height = arguments["height"].as<int>();
    auto &&speed = arguments["speed"].as<float>();
    auto &&fovy = arguments["fovy"].as<float>();
    auto &&distanceX = arguments["dx"].as<int>();
    auto &&distanceY = arguments["dy"].as<int>();
    auto &&front = arguments["front"].as<float>();
    auto &&back = arguments["back"].as<float>();
    auto &&file_name = arguments["in_file"].as<std::string>();
    auto &&res_file_name = arguments["out_file"].as<std::string>();

    objl::Loader loader;
    loader.LoadFile(file_name);
    auto &&mesh = loader.LoadedMeshes[0];

    auto &&model_vertices = ToGLMVertices().applyList<objl::Vertex, glm::vec3>(mesh.
    Vertices);
    auto &&model_center = getModelCenter(model_vertices);

    auto &&screen_ratio = static_cast<float>(width) / static_cast<float>(height);
    auto &&projection = glm::perspective(
            glm::radians(fovy),
            screen_ratio,
            front,
            back
    );

    float angle = 0;
    float angle_per_frame = speed / FRAME_PER_SECOND;

    auto &&start_camera_position = glm::vec4(distanceX, distanceY, 0, 1);

    TriangleDrawer drawer(width, height);

    for (auto i = 0; i < FRAME_COUNT; i++) {
        drawer.resetImage();
        glm::mat4 rotation_matrix = glm::rotate(glm::mat4(1), angle, glm::vec3(0, 1, 0));
        glm::vec3 camera_position = (rotation_matrix * start_camera_position);
        auto &&camera = glm::lookAt(
                camera_position,
```

```
                    model_center,
                    glm::vec3(0, 1, 0)
            );

            drawer.updatePipeline(std::make_unique<TriangleTransformationPipeline>(camera,
        projection, width, height));
            render(drawer, model_vertices, mesh.Indices);

            cv::imshow("res", drawer.getImage());
            cv::waitKey(2000);
            angle += angle_per_frame;
        }

        return 0;
}
```