САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет
По лабораторной работе №3,4
Дисциплина: Разработка графических приложений

Выполнил студент группы: 13541/3: Покатило П.А.

Преподаватель: Беляевский К.О.

Санкт-Петербург
2018

Оглавление

Задание

Разработать программу на языке C++ для растеризации загруженной модели на экран (в cv::Mat).

Возможности программы:

- Загрузка трехмерной модели из OBJ-файла
- Растеризация каркаса трехмерной модели (в виде линий), используя средства opencv (cv::line)
- Обеспечение вращения камеры вокруг трехмерной модели
- Растеризация линий своим алгоритмом (вместо cv::line)
- Растеризация треугольников своим алгоритмом
- Вычисление барицентрических координат и получение значения глубины для конкретного пикселя
- Использование буфера глубины для отсечения невидимых пикселей

Ход работы

В работе использовалась библиотека GLM для работы с камерой и матричными преобразованиями, а также OBJ-loader для загрузки файлов типа .OBJ.

В результате работы программы были получены следующие результаты:

Макрос LINEDRAW обеспечивает отрисовку только линий объекта

Макрос RECORCD отвечает за запись видео в зависимости от LINEDRAW

Отрисовка линий производится следующим алгоритмом

```
for (int j = 0; j < mesh.Indices.size(); j += 3)
                        cv::Scalar color = cv::Scalar(255, 255, 255);
                        line(
                                image,
                                cv::Point(
                                        resultPoints.at(mesh.Indices[j]).x,
                                        resultPoints.at(mesh.Indices[j]).y),
                                cv::Point(
                                        resultPoints.at(mesh.Indices[j + 1]).x,
                                        resultPoints.at(mesh.Indices[j + 1]).y),
                                color);
                        line(
                                image,
                                cv::Point(
                                        resultPoints.at(mesh.Indices[j + 1]).x,
                                        resultPoints.at(mesh.Indices[j + 1]).y),
                                cv::Point(
                                        resultPoints.at(mesh.Indices[j + 2]).x,
                                        resultPoints.at(mesh.Indices[j + 2]).y),
                                color);
                        line(
                                image,
                                cv::Point(
                                        resultPoints.at(mesh.Indices[j]).x,
                                        resultPoints.at(mesh.Indices[j]).y),
                                cv::Point(
                                        resultPoints.at(mesh.Indices[j + 2]).x,
                                        resultPoints.at(mesh.Indices[j + 2]).y),
                                color);}
```

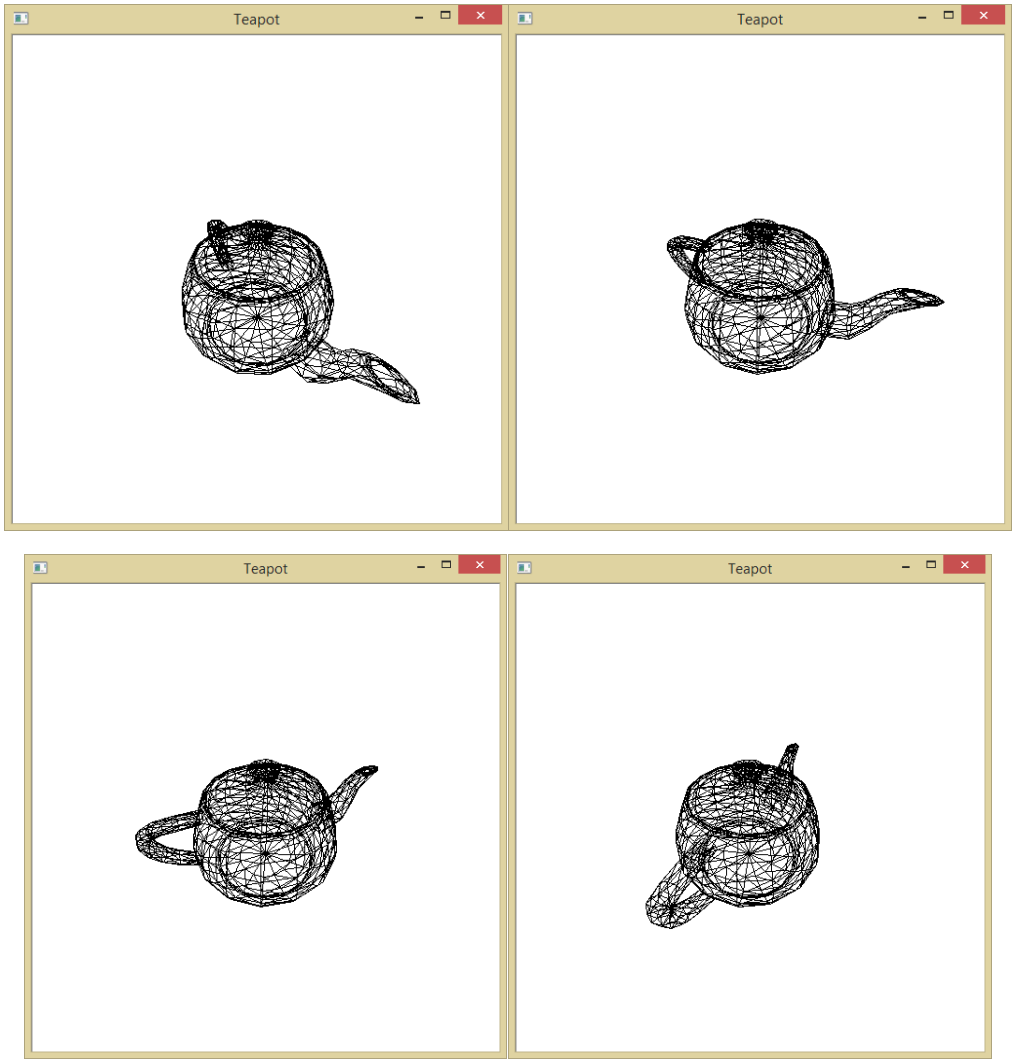

Рисунок 1 Отрисовка граней объекта

Отрисовка с учетом буфера глубины производится через заполнение треугольников с помощью барицентрических координат

```
for (int j = 0; j < mesh.Indices.size(); j += 3)
		{
			std::vector<cv::Scalar> colors;
			colors.push_back(cv::Scalar(80, 230, 170));
			colors.push_back(cv::Scalar(20, 170, 90));
			colors.push_back(cv::Scalar(170, 50, 230));
			cv::Scalar color = colors.at((j / 3) % 3);

			fillTriangleBarycentric(
				image,
				cv::Point3f(
					resultPoints.at(mesh.Indices[j]).x,
					resultPoints.at(mesh.Indices[j]).y,
					resultPoints.at(mesh.Indices[j]).z),
				cv::Point3f(
					resultPoints.at(mesh.Indices[j + 1]).x,
					resultPoints.at(mesh.Indices[j + 1]).y,
					resultPoints.at(mesh.Indices[j + 1]).z),
				cv::Point3f(
					resultPoints.at(mesh.Indices[j + 2]).x,
					resultPoints.at(mesh.Indices[j + 2]).y,
					resultPoints.at(mesh.Indices[j + 2]).z),
				buffer,
				color);
		}
#endif
```

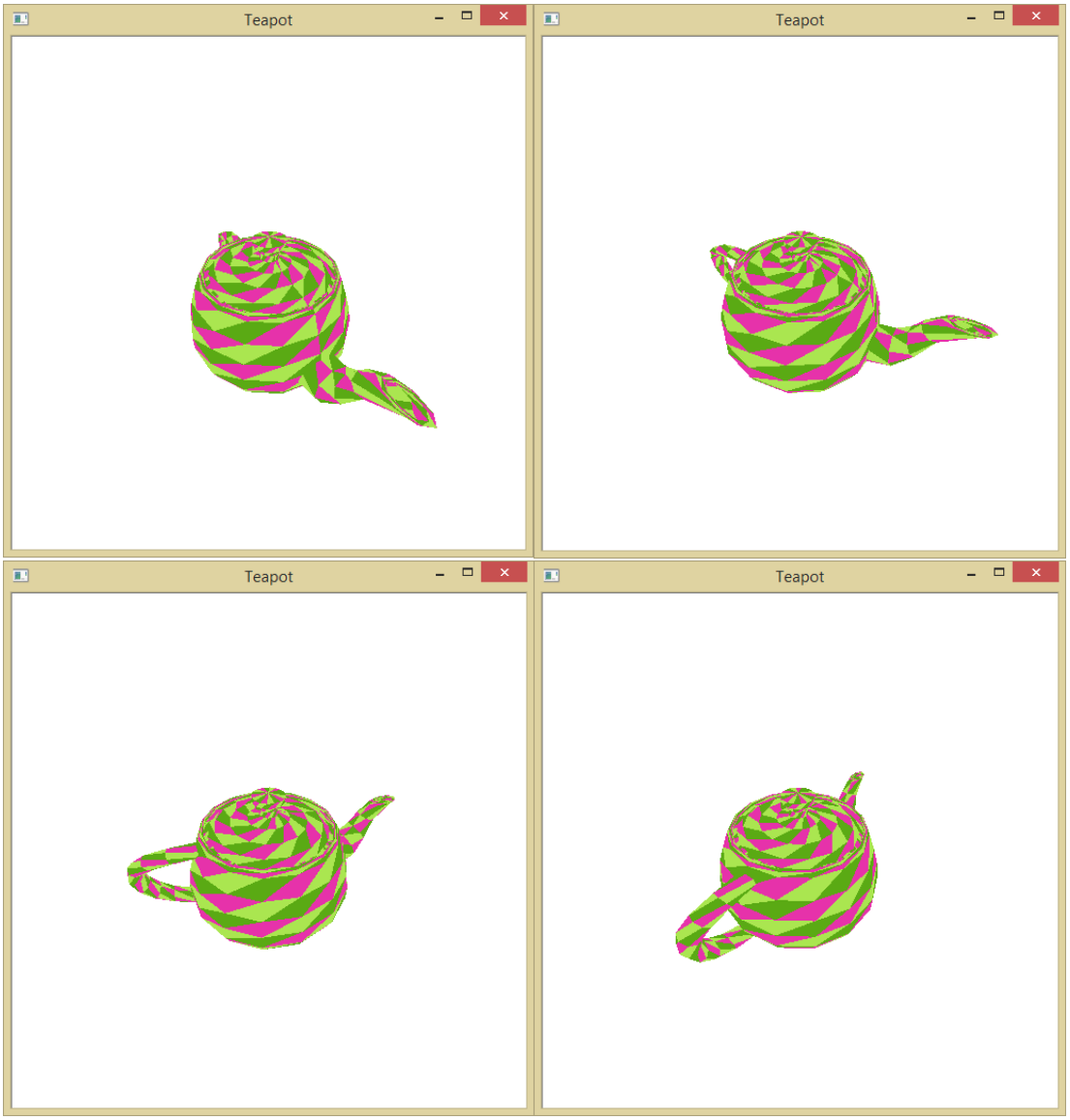

Рисунок 2 Поворот камеры вокруг объекта с закраской с помощью буфера глубины

Полный код проекта приведен в Приложении.

Выводы

В ходе данной лабораторной работы были закреплены основы работы с библиотекой OpenCV.

Также была разработана программа, выполняющая растеризацию 3D объектов, с возможностью их загрузки, отрисовки граней или отрисовки в зависимости от глубины положения вершин и скрытых пикселей.

Данная работа демонстрирует сам алгоритм отрисовки, дает понимание того, как работает растеризация трехмерных объектов. Однако скорость работы оставляет желать лучшего. Отсутствует поддержка текстурирования и затенения. Однако данный процесс выходит за рамки лабораторной работы, так как для корректного текстурирования необходимо осуществлять маппинг моделей и рисовать под них текстуры, либо использовать .mtl для .obj, который не поддерживается OBJ-loader'ом. Для корректного и эффективного затенения необходимо использовать шейдеры, карты нормалей и bump-карты.

Приложение

Код программы на языке C

```
#pragma comment(lib,"I:\\Downloads\\opcv\\opencv\\build\\x64\\vc15\\lib\\opencv_world343.lib")

#include <iostream>
#include <opencv2/opencv.hpp>
#include <iostream>
#include <OBJ_Loader.h>
#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>

//#define LINEDRAW 1    //Отрисовка только граней
#define RECORD 1        //Запись видео
#define FRAMES_COUNT 720
// отрисовка линии
void printLine(
        cv::Mat& src, float x1in,
        float x2in, float y1in,
        float y2in, cv::Mat& buffZ,
        float z1in, float z2in);

float sign(cv::Point2f p1, cv::Point2f p2, cv::Point2f p3);

bool pointInTriangle(cv::Point2f pt, cv::Point2f v1, cv::Point2f v2, cv::Point2f v3);

// нахождении z координаты из уравнения плоскости
float equationPlaneZ(
        float x1, float y1, float z1,
        float x2, float y2, float z2,
        float x3, float y3, float z3,
        float x4, float y4);

// закраска полигона
void fillTriangleBarycentric(
        cv::Mat& src,
        cv::Point3f v1, cv::Point3f v2, cv::Point3f v3,
        cv::Mat& buffZ,
        cv::Scalar color);

int main()
{
        float avgX = 0, avgY = 0, avgZ = 0;

        objl::Loader Loader;
        bool loadout = Loader.LoadFile("teapot.obj");
        objl::Mesh mesh = Loader.LoadedMeshes[0];

        for (int j = 0; j < mesh.Vertices.size(); j++)
        {
                avgX += mesh.Vertices[j].Position.X;
                avgY += mesh.Vertices[j].Position.Y;
                avgZ += mesh.Vertices[j].Position.Z;
        }

        avgX /= mesh.Vertices.size();
        avgY /= mesh.Vertices.size();
        avgZ /= mesh.Vertices.size();

#ifdef RECORD
#ifdef LINEDRAW
        cv::VideoWriter video("EdgedTeapot.avi", CV_FOURCC('M', 'J', 'P', 'G'), 10, cv::Size(500, 500));
#else
        cv::VideoWriter video("ColoredTeapot.avi", CV_FOURCC('M', 'J', 'P', 'G'), 10, cv::Size(500, 500));
#endif
#endif
        for (int k = 0; k < FRAMES_COUNT; k++)
        {
                cv::Mat image(500, 500, CV_8UC3, cv::Scalar(0, 0, 0));
                cv::Mat buffer(500, 500, CV_64F, 1000.0f);

                std::vector<glm::vec4> resultPoints;

                for (int j = 0; j < mesh.Vertices.size(); j++) {
```

```
                // задаем глобальную систему координат
                glm::mat4 globalMatrix = glm::translate(glm::mat4(1.f), glm::vec3(0.f, 0.f, 0.f));

                // выбор оси вращения чайника
                glm::vec3 yAxis = { 0.0f, 1.0f, 0.0f };

                // вращение матрицы по оси Y
                globalMatrix = glm::rotate(globalMatrix, k * 3.14f / 180, yAxis);

                // вектор для текущего градуса
                glm::vec4 currVec = glm::vec4(
                        mesh.Vertices[j].Position.X,
                        mesh.Vertices[j].Position.Y,
                        mesh.Vertices[j].Position.Z,
                        1.f);

                // трансформируем вершину в однородных координатах
                glm::vec4 transformedVector = globalMatrix * currVec;

                // устанавливаем позицию в отдалени в 100 единиц от центра масс
                glm::vec3 cameraPos = glm::vec3(100.0f, 100.0f, 0.0f);

                // наводим камеру на центр масс
                glm::vec3 cameraTarget = glm::vec3(avgX, avgY, avgZ);

                // переносим в пространство вида
                glm::mat4 camera = glm::lookAt(cameraPos, cameraTarget, yAxis);

                // проекция с перспективой и углом обзора 45 градусов
                glm::mat4 projection = glm::perspective(glm::radians(45.0f), 1.0f, 1.0f, 300.0f);

                // проекция вектора с учетом перспективы
                transformedVector = projection * camera * transformedVector;
                transformedVector = transformedVector / transformedVector.w;

                // увеличение полученного результата
                glm::mat4 scaling = glm::scale(glm::mat4(1.f), glm::vec3(90, 90, 1));

                // переворот Y
                glm::mat4 revertYAxis = glm::mat4(
                        1, 0, 0, 0,
                        0, -1, 0, 0,
                        0, 0, 1, 0,
                        0, 0, 0, 1);

                transformedVector = revertYAxis * scaling * transformedVector;
                transformedVector = transformedVector + glm::vec4(250, 250, 0, 1.f);

                resultPoints.push_back(transformedVector);
        }

#ifdef LINEDRAW
        // отрисовка линий
        for (int j = 0; j < mesh.Indices.size(); j += 3)
        {
                cv::Scalar color = cv::Scalar(255, 255, 255);
                line(
                        image,
                        cv::Point(
                                resultPoints.at(mesh.Indices[j]).x,
                                resultPoints.at(mesh.Indices[j]).y),
                        cv::Point(
                                resultPoints.at(mesh.Indices[j + 1]).x,
                                resultPoints.at(mesh.Indices[j + 1]).y),
                        color);
                line(
                        image,
                        cv::Point(
                                resultPoints.at(mesh.Indices[j + 1]).x,
                                resultPoints.at(mesh.Indices[j + 1]).y),
                        cv::Point(
                                resultPoints.at(mesh.Indices[j + 2]).x,
                                resultPoints.at(mesh.Indices[j + 2]).y),
                        color);
                line(
                        image,
                        cv::Point(
                                resultPoints.at(mesh.Indices[j]).x,
                                resultPoints.at(mesh.Indices[j]).y),
                        cv::Point(
                                resultPoints.at(mesh.Indices[j + 2]).x,
                                resultPoints.at(mesh.Indices[j + 2]).y),
```

```
                                color);
                }
#else
                for (int j = 0; j < mesh.Indices.size(); j += 3)
                {
                        std::vector<cv::Scalar> colors;
                        colors.push_back(cv::Scalar(80, 230, 170));
                        colors.push_back(cv::Scalar(20, 170, 90));
                        colors.push_back(cv::Scalar(170, 50, 230));
                        cv::Scalar color = colors.at((j / 3) % 3);

                        fillTriangleBarycentric(
                                image,
                                cv::Point3f(
                                        resultPoints.at(mesh.Indices[j]).x,
                                        resultPoints.at(mesh.Indices[j]).y,
                                        resultPoints.at(mesh.Indices[j]).z),
                                cv::Point3f(
                                        resultPoints.at(mesh.Indices[j + 1]).x,
                                        resultPoints.at(mesh.Indices[j + 1]).y,
                                        resultPoints.at(mesh.Indices[j + 1]).z),
                                cv::Point3f(
                                        resultPoints.at(mesh.Indices[j + 2]).x,
                                        resultPoints.at(mesh.Indices[j + 2]).y,
                                        resultPoints.at(mesh.Indices[j + 2]).z),
                                buffer,
                                color);
                }
#endif

#ifdef RECORD
                video.write(image);
#else
                imshow("Teapot", image);
#endif
                resultPoints.clear();
        }
#ifdef RECORD
        video.release();
#else
        cv::waitKey(0);
#endif

}


void printLine(cv::Mat& src, float x1in, float x2in, float y1in, float y2in, cv::Mat& buffZ, float z1in, float z2in)
{
        float dy = y2in - y1in;
        float dx = x2in - x1in;

        if (dx < 1) return;

        float x1 = x1in;
        float x2 = x2in;
        float y1 = y1in;

        for (int x = (int)x1; x <= (int)x2; x++)
        {
                float y = y1 + dy * (x - x1) / dx;
                src.at<cv::Vec3b>((int)y, x).val[0] = 0;
                src.at<cv::Vec3b>((int)y, x).val[1] = 255;
                src.at<cv::Vec3b>((int)y, x).val[2] = 0;
        }
}

float sign(cv::Point2f p1, cv::Point2f p2, cv::Point2f p3) {
        return (p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y - p3.y);
}

bool pointInTriangle(cv::Point2f pt, cv::Point2f v1, cv::Point2f v2, cv::Point2f v3)
{
        float d1 = sign(pt, v1, v2);
        float d2 = sign(pt, v2, v3);
        float d3 = sign(pt, v3, v1);

        bool has_neg = (d1 < 0) || (d2 < 0) || (d3 < 0);
        bool has_pos = (d1 > 0) || (d2 > 0) || (d3 > 0);

        return !(has_neg && has_pos);
}
```

```
float equationPlaneZ(
        float x1, float y1, float z1,
        float x2, float y2, float z2,
        float x3, float y3, float z3,
        float x4, float y4)
{
        float a1 = x2 - x1;
        float b1 = y2 - y1;
        float c1 = z2 - z1;
        float a2 = x3 - x1;
        float b2 = y3 - y1;
        float c2 = z3 - z1;
        float a = b1 * c2 - b2 * c1;
        float b = a2 * c1 - a1 * c2;
        float c = a1 * b2 - b1 * a2;
        float d = (-a * x1 - b * y1 - c * z1);
        return -(a * x4 + b * y4 + d) / c;
}

void fillTriangleBarycentric(
        cv::Mat& src,
        cv::Point3f v1, cv::Point3f v2, cv::Point3f v3,
        cv::Mat& buffZ,
        cv::Scalar color)
{
        float min_x = std::min({ v1.x, v2.x, v3.x });
        float max_x = std::max({ v1.x, v2.x, v3.x });
        float min_y = std::min({ v1.y, v2.y, v3.y });
        float max_y = std::max({ v1.y, v2.y, v3.y });
        if (max_x - min_x < 1 || max_y - min_y < 1) return;
        for (int i = (int)min_x; i <= (int)max_x; i++)
        {
                for (int j = (int)min_y; j <= (int)max_y; j++)
                {
                        if (pointInTriangle(
                                cv::Point2f(i, j),
                                cv::Point2f(v1.x, v1.y),
                                cv::Point2f(v2.x, v2.y),
                                cv::Point2f(v3.x, v3.y)))
                        {
                                float zCoord = equationPlaneZ(
                                        v1.x, v1.y, v1.z,
                                        v2.x, v2.y, v2.z,
                                        v3.x, v3.y, v3.z,
                                        i, j);
                                if (zCoord < buffZ.at<double>(j, i))
                                {
                                        buffZ.at<double>(j, i) = zCoord;
                                        src.at<cv::Vec3b>(j, i).val[0] = color[0];
                                        src.at<cv::Vec3b>(j, i).val[1] = color[1];
                                        src.at<cv::Vec3b>(j, i).val[2] = color[2];
                                }
                        }
                }
        }
}
```