**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 3**

Тема:

Студент: Лошманов Юрий Андреевич

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света.

Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант многогранника: Слой параболоида.

1. **Описание программы**

Программа состоит из окна для взаимодействия с многогранником, использует взаимодействие через клавиатуру для вращения многогранника и изменения некоторых его параметров.

1. **Набор тестов**

1. Изначальный вид.

2. Вращение многогранника.

3. Увеличение точности аппроксимации.

4. Увеличение коэффициента диффузного отражения рассеянного света.

5. Увеличение коэффициента диффузного отражения падающего света.

1. **Результаты выполнения тестов**

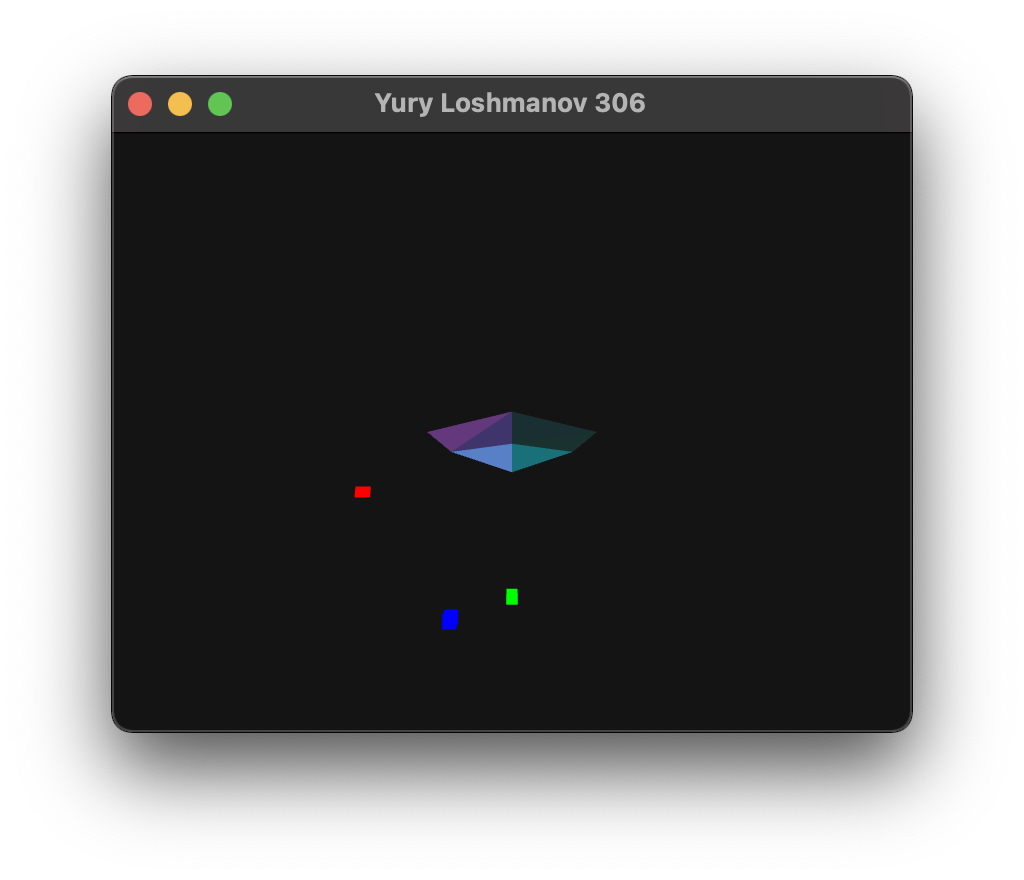
1. Изначальный вид.  


Рис. 4.1 Результат открытия окна

2. Вращение многогранника.



Рис. 4.2 Результат вращения многогранника

3. Увеличение точности аппроксимации.

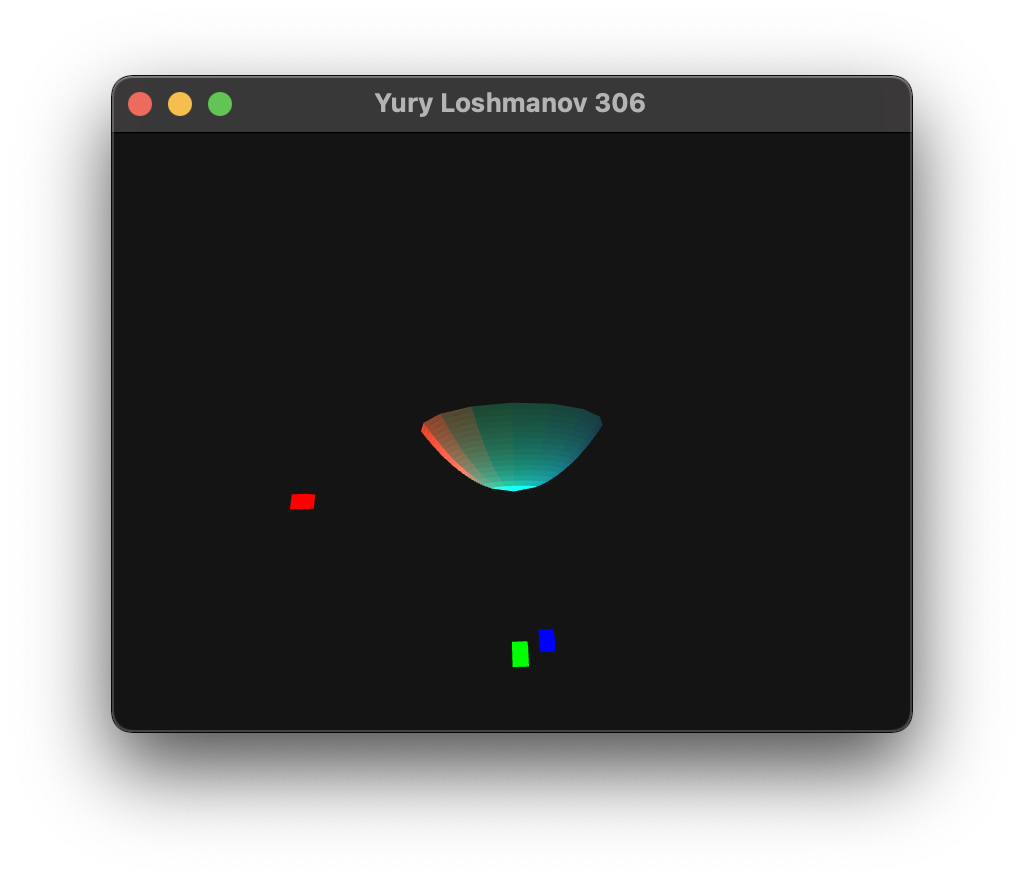


Рис. 4.3 Результат увеличения точности аппроксимации

4. Уменьшение коэффициента диффузного отражения рассеянного света.

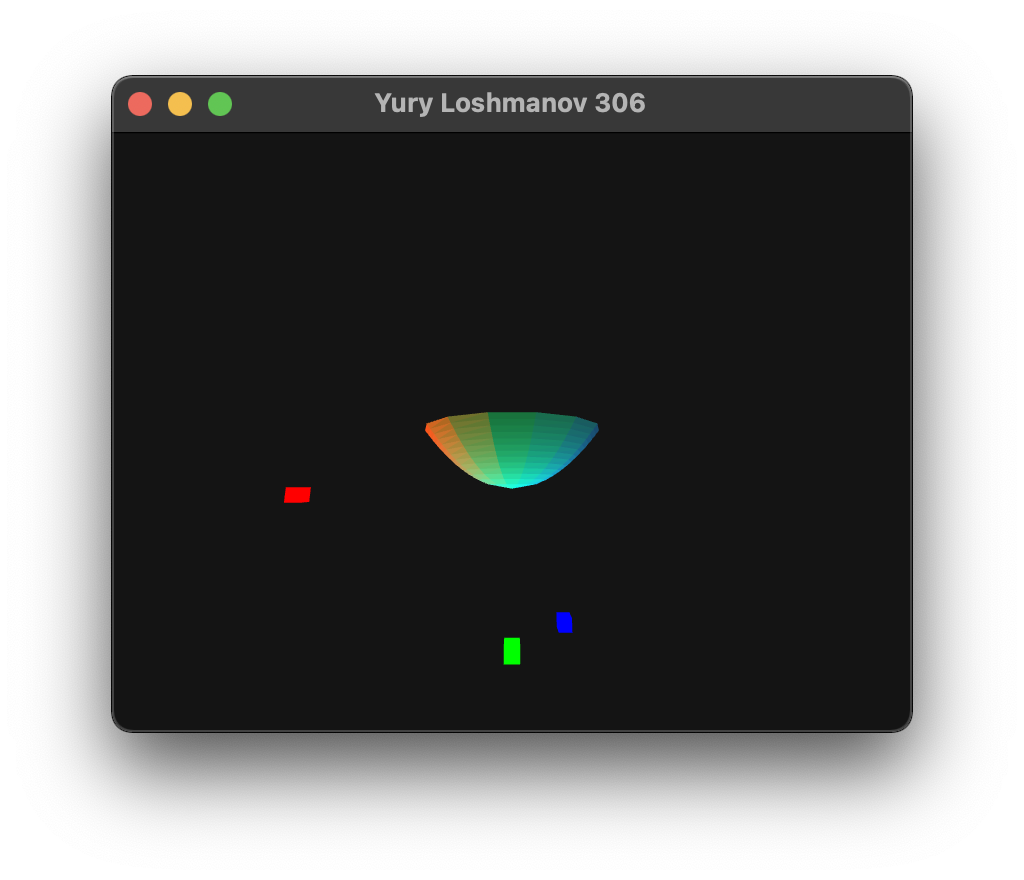


Рис. 4.4 Результат уменьшения коэффициента диффузного отражения рассеянного света

5. Увеличение коэффициента диффузного отражения падающего света.

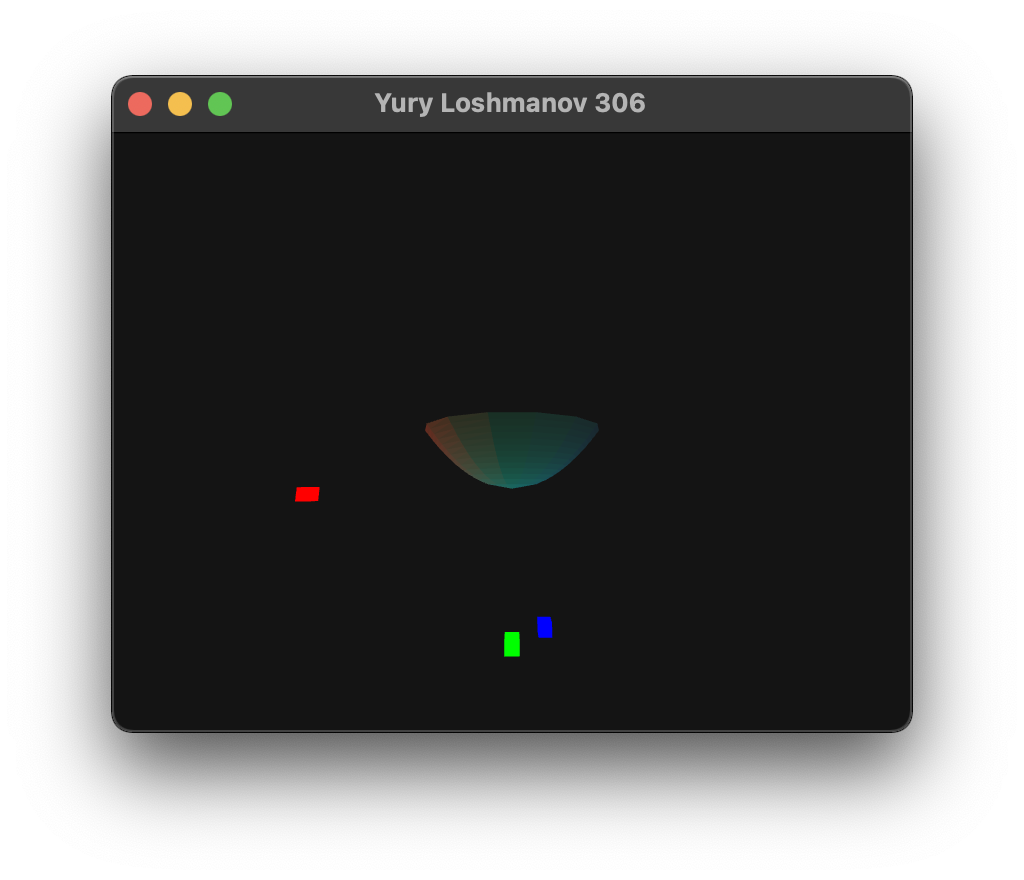


Рис. 4.5 Результат увеличения коэффициента диффузного падающего света

1. **Листинг программы**

**main.cpp**

//

// Yury Loshmanov

// 306

//

#include <iostream>

#include <SFML/Graphics.hpp>

#include "matrices.hpp"

#include "figure.hpp"

#include "math\_help.hpp"

constexpr int width = 800;

constexpr int height = 600;

Figure customFigure(size\_t precision) {

if (precision < 2) {

throw std::runtime\_error("minimum precision is 2");

}

std::vector<float> radiuses = math::linspace(2.0f, 0.0f, precision);

std::vector<std::vector<std::pair<float, float>>> circles(precision);

for (size\_t i = 0; i < precision; i++) {

circles[i] = math::regularPolygon(precision+1, std::sqrt(radiuses[i]), 0.0, 0.0);

circles[i].push\_back(circles[i][0]);

}

std::vector<Triangle> triangles;

double z = 0.0;

double zStep = 1.0 / circles.size();

for (size\_t i = 0; i < circles.size()-1; i++) {

for (size\_t j = 0; j < circles[i].size()-1; j++) {

triangles.emplace\_back(Triangle(

{circles[i][j].first, circles[i][j].second, z},

{circles[i][j+1].first, circles[i][j+1].second, z},

{circles[i+1][j].first, circles[i+1][j].second, z + zStep}

));

if (i+1 == circles.size()-1) {

continue;

}

triangles.emplace\_back(Triangle(

{circles[i][j+1].first, circles[i][j+1].second, z},

{circles[i+1][j+1].first, circles[i+1][j+1].second, z + zStep},

{circles[i+1][j].first, circles[i+1][j].second, z + zStep}

));

}

z += zStep;

}

return Figure(triangles);

}

Figure cubeFigure() {

Figure f;

std::vector<Point> p({

Point(1.0, 1.0, -1.0), //0

Point(-1.0, 1.0, -1.0),

Point(-1.0, -1.0, -1.0),

Point(1.0, -1.0, -1.0),

Point(1.0, 1.0, 1.0), //4

Point(-1.0, 1.0, 1.0),

Point(-1.0, -1.0, 1.0),

Point(1.0, -1.0, 1.0),

});

f.addTriangle(Triangle(p[0], p[1], p[4]));

f.addTriangle(Triangle(p[1], p[5], p[4]));

f.addTriangle(Triangle(p[1], p[2], p[5]));

f.addTriangle(Triangle(p[2], p[6], p[5]));

f.addTriangle(Triangle(p[2], p[3], p[6]));

f.addTriangle(Triangle(p[3], p[7], p[6]));

f.addTriangle(Triangle(p[0], p[7], p[3]));

f.addTriangle(Triangle(p[0], p[4], p[7]));

f.addTriangle(Triangle(p[4], p[5], p[6]));

f.addTriangle(Triangle(p[6], p[7], p[4]));

f.addTriangle(Triangle(p[0], p[2], p[1]));

f.addTriangle(Triangle(p[0], p[3], p[2]));

return f;

}

std::vector<mm::vec3> light1Trajectory() {

size\_t n = 500;

std::vector<mm::vec3> res(n);

std::vector<std::pair<float, float>> regPolygon = math::regularPolygon(n, 3.0, 0.0, 0.0);

for(size\_t i = 0; i < n; i++) {

res[i][0] = regPolygon[i].first;

res[i][1] = regPolygon[i].second;

res[i][2] = 1.0;

}

return res;

}

std::vector<mm::vec3> light2Trajectory() {

size\_t n = 500;

std::vector<mm::vec3> res(n);

std::vector<std::pair<float, float>> regPolygon = math::regularPolygon(n, 3.0, 0.0, 0.0);

for(size\_t i = 0; i < n; i++) {

res[i][0] = regPolygon[i].first;

res[i][1] = 0.0;

res[i][2] = regPolygon[i].second;

}

return res;

}

std::vector<mm::vec3> light3Trajectory() {

size\_t n = 500;

std::vector<mm::vec3> res(n);

std::vector<std::pair<float, float>> regPolygon = math::regularPolygon(n, 3.0, 0.0, 0.0);

for(size\_t i = 0; i < n; i++) {

res[i][0] = 0.0;

res[i][1] = regPolygon[i].first;

res[i][2] = regPolygon[i].second;

}

return res;

}

int main(){

std::vector<mm::vec3> light1PositionList = light1Trajectory();

std::vector<mm::vec3> light2PositionList = light2Trajectory();

std::vector<mm::vec3> light3PositionList = light3Trajectory();

size\_t rotateIdx = 0;

mm::vec3 cameraPos = {0.0, 0.0, -5.0};

double cameraSpeed = 0.25;

double rotateSpeed = 2.0;

double FOV = 90;

double rotateAngleX = 45.0;

double rotateAngleY = 0.0;

double rotateAngleZ = 90.0;

size\_t figurePrecision = 3;

float specularPow = 8;

float ambientStrength = 0.1;

float diffStrength = 0.7;

float specularStrength = 1.0;

sf::ContextSettings settings;

settings.antialiasingLevel = 8;

sf::RenderWindow window(sf::VideoMode(width, height), "Yury Loshmanov 306",

sf::Style::Default, settings);

sf::Color bgColor = sf::Color(20, 20, 20);

Figure cube1 = cubeFigure();

cube1.setColor(sf::Color::Red);

cube1.setOutlineThickness(0.0f);

Figure cube2 = cubeFigure();

cube2.setColor(sf::Color::Green);

cube2.setOutlineThickness(0.0f);

Figure cube3 = cubeFigure();

cube3.setColor(sf::Color::Blue);

cube3.setOutlineThickness(0.0f);

rotateAngleX = -80;

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

break;

} else if(event.type == sf::Event::KeyPressed) {

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Numpad8)) {

cameraPos[1] -= cameraSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Numpad2)) {

cameraPos[1] += cameraSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Numpad4)) {

cameraPos[0] -= cameraSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Numpad6)) {

cameraPos[0] += cameraSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Numpad0)) {

cameraPos[2] -= cameraSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Numpad1)) {

cameraPos[2] += cameraSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {

FOV += 1.0;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {

FOV -= 1.0;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Z)) {

rotateAngleX += rotateSpeed;

std::cout << rotateAngleX << std::endl;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::X)) {

rotateAngleX -= rotateSpeed;

std::cout << rotateAngleX << std::endl;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::C)) {

rotateAngleY += rotateSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::V)) {

rotateAngleY -= rotateSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::B)) {

rotateAngleZ += rotateSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::N)) {

rotateAngleZ -= rotateSpeed;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {

figurePrecision++;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {

figurePrecision--;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::PageUp)) {

specularPow++;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::PageDown)) {

specularPow--;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::F1)) {

ambientStrength -= 0.1;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::F2)) {

ambientStrength += 0.1;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::F3)) {

diffStrength -= 0.1;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::F4)) {

diffStrength += 0.1;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::F5)) {

specularStrength -= 0.1;

} else if (sf::Keyboard::isKeyPressed(sf::Keyboard::F6)) {

specularStrength += 0.1;

}

if (specularPow < 1) {

specularPow = 1;

}

if (specularPow > 32) {

specularPow = 32;

}

if (FOV > 180.0 || FOV < 0) {

FOV = 45.0;

}

if (rotateAngleX > 360.0 || rotateAngleX < -360.0) {

rotateAngleX = 0.0;

}

if (figurePrecision < 2) {

figurePrecision = 2;

}

} else if (event.type == sf::Event::MouseMoved) {

rotateIdx += 1;

rotateIdx %= light1PositionList.size();

}

window.clear(bgColor);

Figure myFigure = customFigure(figurePrecision);

myFigure.setColor(sf::Color(255, 255, 255));

myFigure.setOutlineColor(bgColor);

myFigure.setOutlineThickness(0.0f);

auto model = mm::mat4(1.0);

auto modelCube1 = mm::translate(model, light1PositionList[rotateIdx]);

modelCube1 = mm::scale(modelCube1, mm::vec3(0.1, 0.1, 0.1));

auto modelCube2 = mm::translate(model, light2PositionList[rotateIdx]);

modelCube2 = mm::scale(modelCube2, mm::vec3(0.1, 0.1, 0.1));

auto modelCube3 = mm::translate(model, light3PositionList[rotateIdx]);

modelCube3 = mm::scale(modelCube3, mm::vec3(0.1, 0.1, 0.1));

auto modelFigure = mm::mat4(1.0);

auto view = mm::mat4(1.0);

view = mm::translate(view, cameraPos);

view = mm::rotate(view, mm::radians(rotateAngleX), mm::vec3(1.0, 0.0, 0.0));

view = mm::rotate(view, mm::radians(rotateAngleY), mm::vec3(0.0, 1.0, 0.0));

view = mm::rotate(view, mm::radians(rotateAngleZ), mm::vec3(0.0, 0.0, 1.0));

mm::mat4 projection(1.0);

projection = mm::perspective(mm::radians(FOV), (double)width/height, 0.1, 5.0);

//projection = mm::ortho(-3.0, 3.0, -3.0, 3.0, 0.1, 5.0);

mm::mat4 resCube1 = projection\*view\*modelCube1;

mm::mat4 resCube2 = projection\*view\*modelCube2;

mm::mat4 resCube3 = projection\*view\*modelCube3;

mm::mat4 resFigure = projection\*view\*modelFigure;

mm::vec3 cameraVector = cameraPos;

cameraVector \*= -1.0;

cameraVector.normalize();

myFigure.addLightSrc({&cube1, modelCube1 \* cube1.triangles[0].points[0].asVector4()});

myFigure.addLightSrc({&cube2, modelCube2 \* cube2.triangles[0].points[0].asVector4()});

myFigure.addLightSrc({&cube3, modelCube3 \* cube3.triangles[0].points[0].asVector4()});

std::cout << "Specular power: " << specularPow << std::endl;

std::cout << "ambient: " << ambientStrength << ", diffusal: " << diffStrength <<

", specular: " << specularStrength << std::endl;

std::cout << std::string(10, '-') << std::endl;

myFigure.draw(&window, width, height, resFigure, cameraVector, Point(cameraPos[0], cameraPos[1], cameraPos[2]),

specularPow, ambientStrength, diffStrength, specularStrength);

cube1.draw(&window, width, height, resCube1, cameraVector);

cube2.draw(&window, width, height, resCube2, cameraVector);

cube3.draw(&window, width, height, resCube3, cameraVector);

window.display();

}

}

return 0;

}

**conversation.hpp**

//

// Yury Loshmanov

// 306

//

#pragma once

#include "matrices.hpp"

mm::vec2 normalizedToWindow(const mm::vec2 &normalizedCords, int width, int height);

#include "conversion.inl"

**figure.hpp**

//

// Yury Loshmanov

// 306

//

#pragma once

#include <array>

#include <vector>

#include <SFML/Graphics.hpp>

#include "matrices.hpp"

#include "conversion.hpp"

sf::Color operator\*(float f, sf::Color c);

sf::Color operator/(sf::Color c, float f);

class Point;

class Triangle;

class Figure;

#include "figure.inl"

**math\_help.hpp**

//

// Yury Loshmanov

// 306

//

#pragma once

#include <array>

#include <vector>

#include <SFML/Graphics.hpp>

#include "matrices.hpp"

#include "conversion.hpp"

sf::Color operator\*(float f, sf::Color c);

sf::Color operator/(sf::Color c, float f);

class Point;

class Triangle;

class Figure;

#include "figure.inl"

**matrices.hpp**

//

// Yury Loshmanov

// 306

//

#pragma once

#include <iostream>

#include <array>

#include <initializer\_list>

#include <cmath>

namespace mm {

const double pi = 3.14159265358979f;

const double comparePrecision = 1e-10f;

typedef size\_t length\_t;

typedef size\_t pos\_t;

template<typename T, length\_t L>

class vec;

template<typename T, length\_t L>

class mat;

double radians(double degrees);

template <typename T, length\_t L>

mat<T, L> translate(const mat<T, L> &identityMatrix, const vec<T, L-1> &translationVector);

template <typename T, length\_t L>

mat<T, L> scale(const mat<T, L> &identityMatrix, const vec<T, L-1> &scalingVector);

template <typename T>

mat<T, 4> rotate(const mat<T, 4> &identityMatrix, T angle, const vec<T, 3> &R);

mat<double, 4> makeOrthoMatrix(double l, double r, double b, double t, double n, double f);

mat<double, 4> makePerspectiveMatrix(double l, double r, double b, double t, double n, double f);

mat<double, 4> ortho(double left, double right, double bottom, double top, double front, double back);

mat<double, 4> perspective(double fovY, double aspectRatio, double front, double back);

template <typename T, length\_t L>

T cosBetween(const vec<T, L> &v1, const vec<T, L> &v2);

template <typename T, length\_t L>

T dotProduct(const vec<T, L> &v1, const vec<T, L> &v2);

template <typename T>

vec<T, 3> crossProduct(const vec<T, 3> &v1, const vec<T, 3> &v2);

template <typename T, length\_t L>

bool isPerpendicular(const vec<T, L> &v1, const vec<T, L> &v2);

bool floatEqual(double f1, double f2);

template <typename T, length\_t L>

vec<T, L> reflect(const vec<T, L> &v1, const vec<T, L> &v2);

typedef vec<double, 2> vec2;

typedef vec<double, 3> vec3;

typedef vec<double, 4> vec4;

typedef mat<double, 2> mat2;

typedef mat<double, 3> mat3;

typedef mat<double, 4> mat4;

} // namespace mm

#include "matrices.inl"

1. **Выводы**

Выполнив данную лабораторную работу, я закрепил знания по использованию SFML и запуску графического интерфейса, перерисовыванию его в зависимости от изменения окна и отрисовки многогранника.

**ЛИТЕРАТУРА**

1. Документация SFML [Электронный ресурс]URL: <https://www.sfml-dev.org> (Дата обращения: 29.10.2021).

2. Освещение и свет [Электронный ресурс]URL: <https://studizba.com/lectures/10-informatika-i-programmirovanie/305-kompyuternaya-grafika/4058-7-osveschenie-i-svet.html> (Дата обращения: 29.10.2021).