Московский авиационный институт

(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Компьютерная графика»

Лабораторная работа № 2

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий.

Студент: Мукин Юрий
Группа: 80-304
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Задание: Написать и отладить программу, строящую каркасное изображение многогранника.

В моем варианте необходимо построить усеченную восьмигранную пирамиду.

Для отсечения невидимых линий был использован один из простейших алгоритмов **отсечения нелицевых граней.** Этот алгоритм предполагает определение видимости грани по сколярному произведению нормали грани и вектора наблюдателя.

Для вращения/масштабирования/сдвига использовались соответствующие матрицы преобразования для 3-х мерного пространства.

2. Описание программы

Задание выполнено на языка **C++** с использованием библиотек **openGL**, **glfw** и **boost.**

openGL - мультиплатформенная библиотека для работы с графикой, предоставляющая обширный инструментарий, который, однако, применен не был.

glfw - фреймворк для работы с openGL в среде языка C++. Позволяет отрисовывать окна и работать с ивентами.

boost - библиотека предоставляющая обширный функционал для математических расчетов. Мной был использован модуль **ublas/matrix.hpp** для облегчения матричных вычислений.

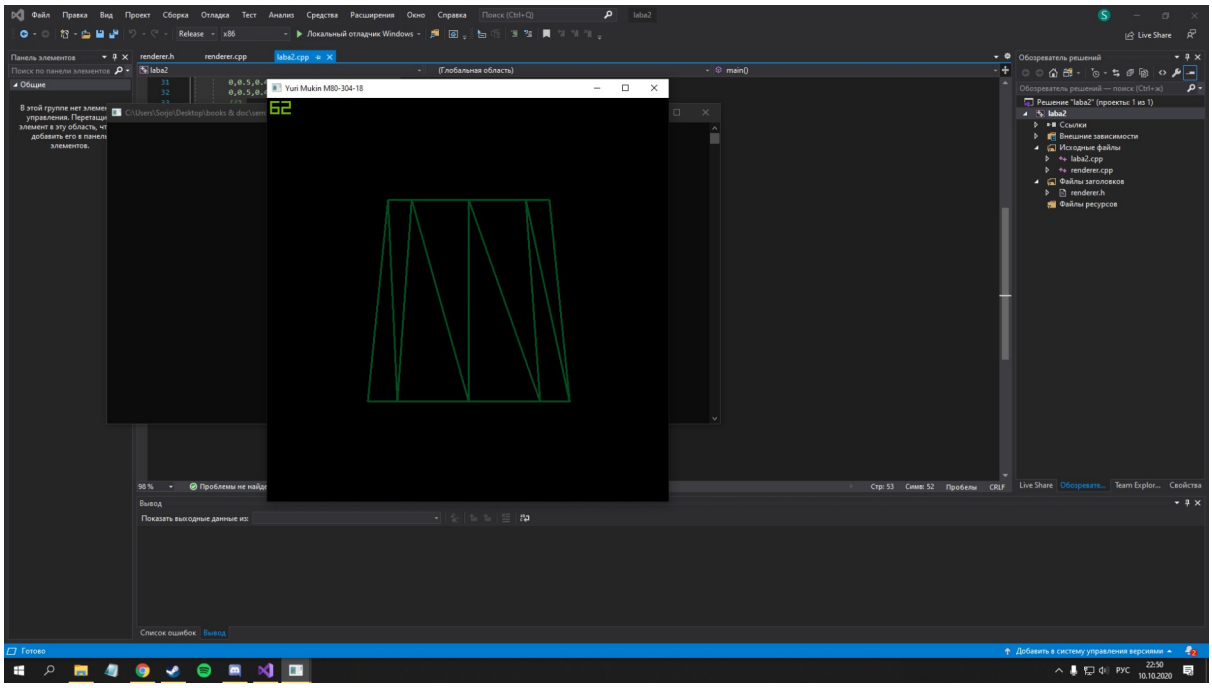
3. Набор тестов

набором тестов можно считать фигуру заданную в виде набора точек

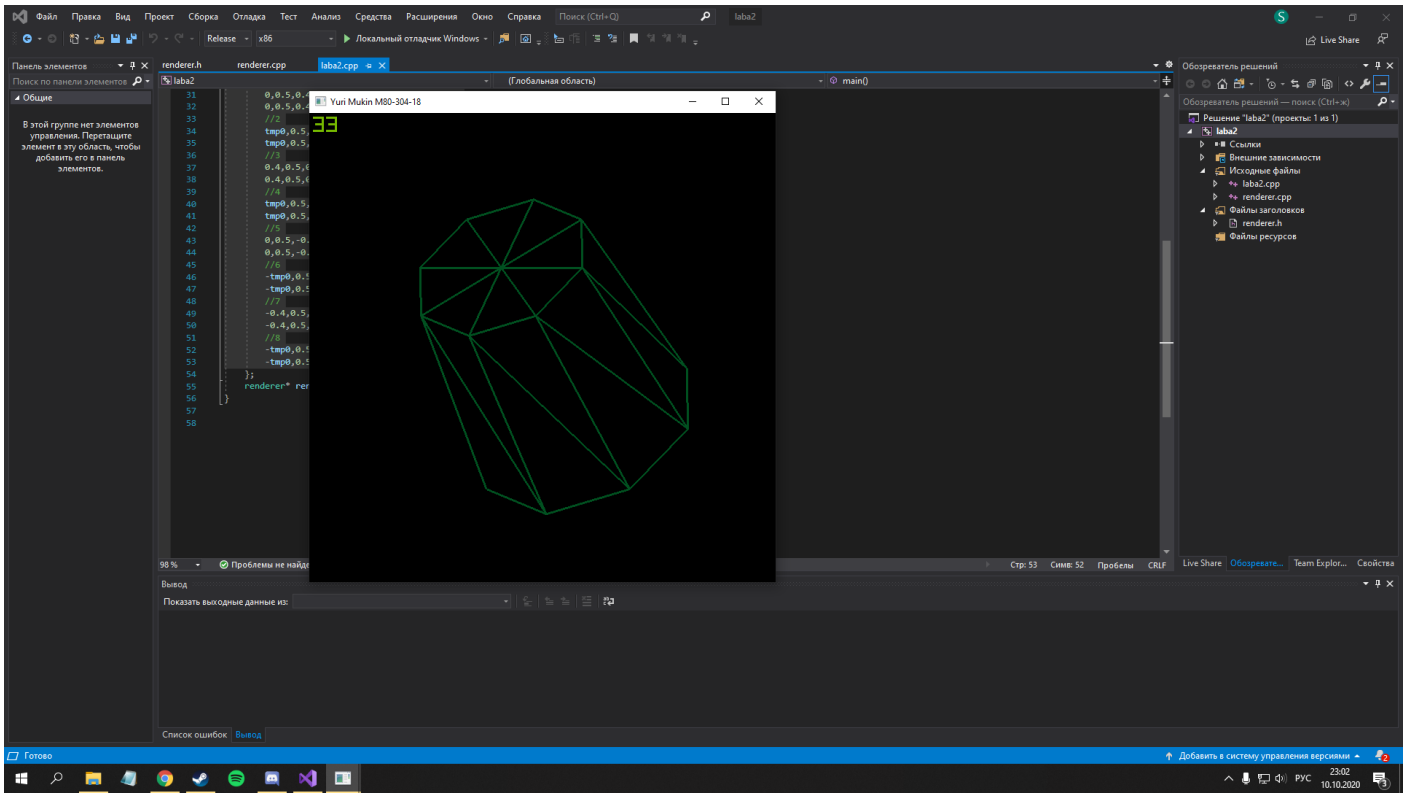
```
//bottom
0,-0.5,0, 0,-0.5,0.5,     tmp,-0.5,tmp,
0,-0.5,0, tmp,-0.5,tmp,   0.5,-0.5,0,
0,-0.5,0, 0.5,-0.5,0,     tmp,-0.5,-tmp,
0,-0.5,0, tmp,-0.5,-tmp,  0,-0.5,-0.5,
0,-0.5,0, 0,-0.5,-0.5,    -tmp,-0.5,-tmp,
0,-0.5,0, -tmp,-0.5,-tmp, -0.5,-0.5,0,
0,-0.5,0, -0.5,-0.5,0,    -tmp,-0.5,tmp,
0,-0.5,0, -tmp,-0.5,tmp,  0,-0.5,0.5,
//top
0,0.5,0.4,       0,0.5,0,  tmp0,0.5,tmp0,
tmp0,0.5,tmp0,   0,0.5,0,  0.4,0.5,0,
0.4,0.5,0,       0,0.5,0,  tmp0,0.5,-tmp0,
tmp0,0.5,-tmp0,  0,0.5,0,  0,0.5,-0.4,
0,0.5,-0.4,      0,0.5,0,  -tmp0,0.5,-tmp0,
-tmp0,0.5,-tmp0, 0,0.5,0,  -0.4,0.5,0,
-0.4,0.5,0,      0,0.5,0,  -tmp0,0.5,tmp0,
-tmp0,0.5,tmp0,  0,0.5,0,  0,0.5,0.4,
//further clockwise
//1
0,0.5,0.4,  tmp,-0.5,tmp,  0,-0.5,0.5,
0,0.5,0.4,  tmp0,0.5,tmp0, tmp,-0.5,tmp,
//2
tmp0,0.5,tmp0,  0.4,0.5,0,  0.5,-0.5,0,
tmp0,0.5,tmp0,  0.5,-0.5,0,  tmp,-0.5,tmp,
//3
0.4,0.5,0,  tmp0,0.5,-tmp0,  tmp,-0.5,-tmp,
```

```
0.4,0.5,0,  tmp,-0.5,-tmp,  0.5,-0.5,0,
//4
tmp0,0.5,-tmp0,  0,0.5,-0.4,   0,-0.5,-0.5,
tmp0,0.5,-tmp0,  0,-0.5,-0.5,  tmp,-0.5,-tmp,
//5
0,0.5,-0.4,  -tmp0,0.5,-tmp0,  -tmp,-0.5,-tmp,
0,0.5,-0.4,  -tmp,-0.5,-tmp,   0,-0.5,-0.5,
//6
-tmp0,0.5,-tmp0,  -0.4,0.5,0,   -0.5,-0.5,0,
-tmp0,0.5,-tmp0,  -0.5,-0.5,0,  -tmp,-0.5,-tmp,
//7
-0.4,0.5,0,  -tmp0,0.5,tmp0,  -tmp,-0.5,tmp,
-0.4,0.5,0,  -tmp,-0.5,tmp,   -0.5,-0.5,0,
//8
-tmp0,0.5,tmp0,  0,0.5,0.4,   0,-0.5,0.5,
-tmp0,0.5,tmp0,  0,-0.5,0.5,  -tmp,-0.5,tmp
```

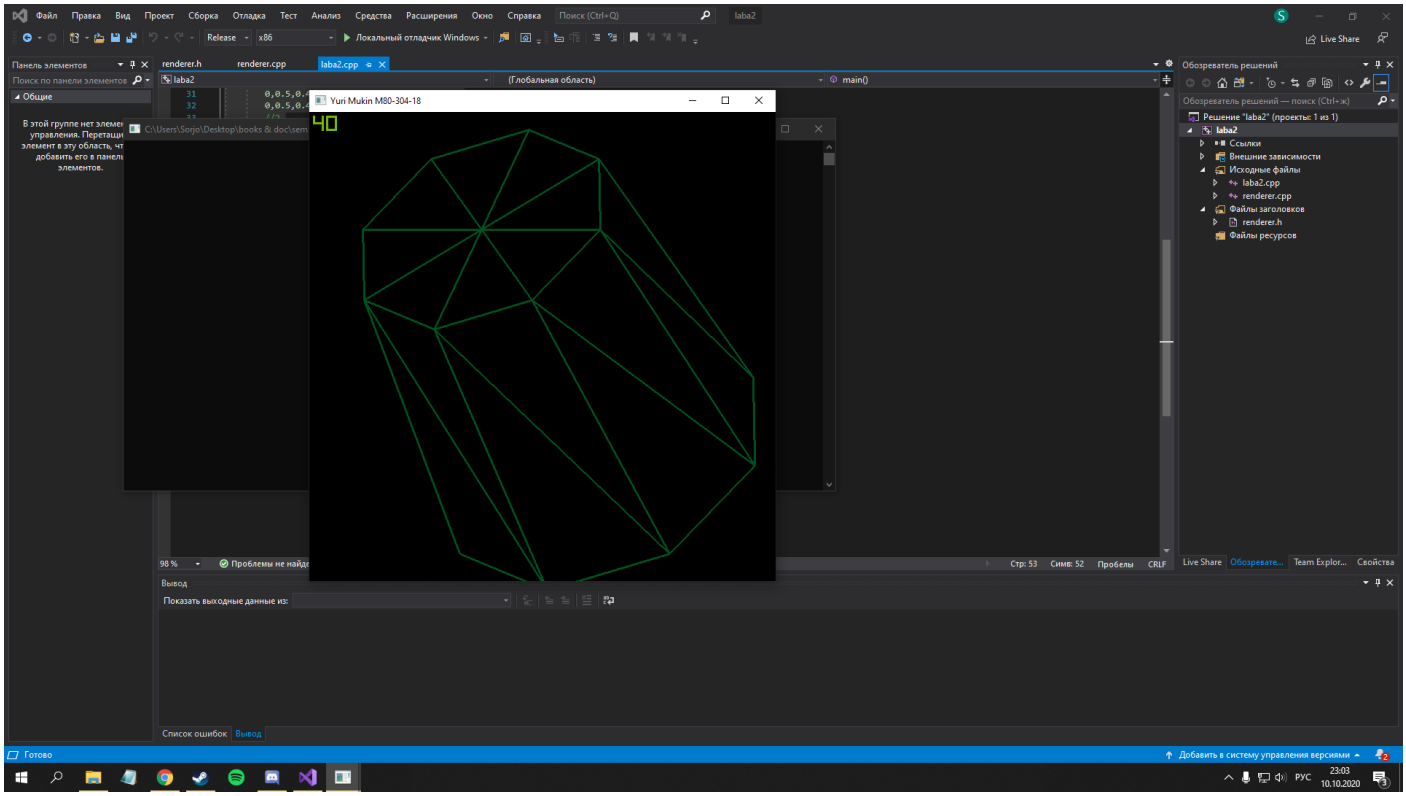
4. Результаты выполнения тестов



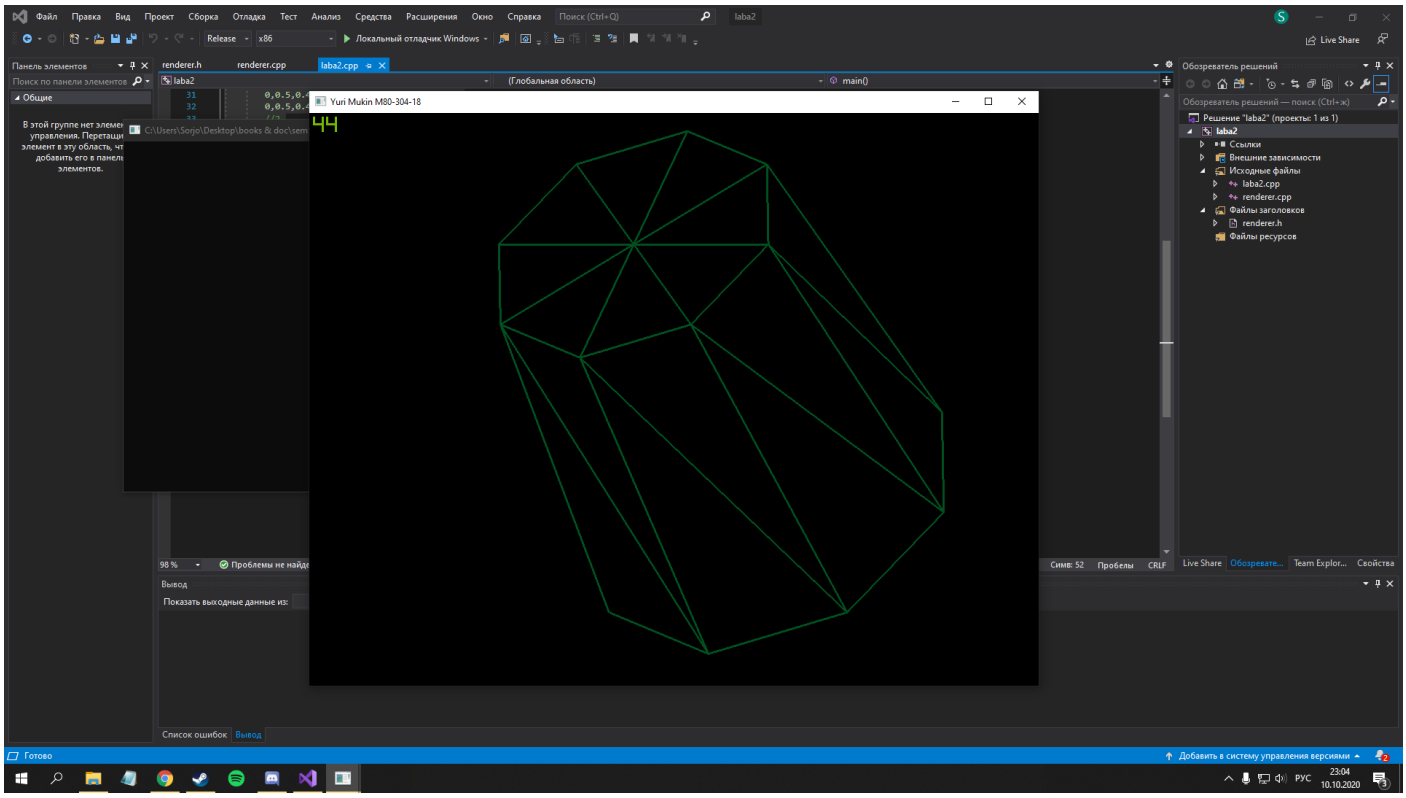
изображение при запуске



поворот



масштаб



изменение размеров окна

5. Листинг программы

```
void renderer::draw()
{
	GLFWwindow* window;
	if (!glfwInit())
		return;
	window = glfwCreateWindow(640, 640, "Yuri Mukin M80-304-18", NULL, NULL);// создаем окно
	if (!window)
	{
		glfwTerminate();
		return;
	}
	glfwMakeContextCurrent(window);
	glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);// устанавливаем режим отрисовки полигонов линиями
	glLineWidth(3);
	glfwSetScrollCallback(window, scroll_callback); //подписываемся на ивенты
	glfwSetWindowSizeCallback(window, window_size_callback);
	glfwSetMouseButtonCallback(window, mouse_button_callback);
	glfwSetCursorPosCallback(window, cursor_position_callback);
	while (!glfwWindowShouldClose(window)) //бесконечный цикл до закрытия окна
	{
		glfwWaitEvents(); // проверяем обновление ивентов
		renderer::newVertSet(); //персчитываем набор точек
		glClear(GL_COLOR_BUFFER_BIT); //сбрасываем цвет (ну мало ли что)
		glEnableClientState(GL_VERTEX_ARRAY);// включаем возможность строить примитивы по массиву
		glVertexPointer(3, GL_DOUBLE, 0, renderer::nPointSet); // подгружаем массив точчек
		glColor3ub(1, 80, 32); // задаем цвет
		for (int i = 0; i < 96; i+=3)
		{
			if (renderer::isRight(i*3))//проверяем видимость полигона
				glDrawArrays(GL_POLYGON, i, 3); //рисуем его
		}
		glfwSwapBuffers(window); //свапаем буфер окна
	}
```

```
}

void renderer::rotate(int of)
{
	boost::numeric::ublas::matrix<double>dot(4, 1);

	boost::numeric::ublas::matrix<double> x(4, 4);// матрица поворота относительно оси x
	x(0, 0) = 1.; x(0, 1) = 0.; x(0, 2) = 0.; x(0, 3) = 0.;
	x(1, 0) = 0.; x(1, 1) = cos(nx); x(1, 2) = -sin(nx); x(1, 3) = 0.;
	x(2, 0) = 0.; x(2, 1) = sin(nx); x(2, 2) = cos(nx); x(2, 3) = 0.;
	x(3, 0) = 0.; x(3, 1) = 0.; x(3, 2) = 0.; x(3, 3) = 1.;

	boost::numeric::ublas::matrix<double> y(4, 4);// матрица поворота относительно оси y
	y(0, 0) = cos(-ny); y(0, 1) = 0.; y(0, 2) = sin(-ny); y(0, 3) = 0.;
	y(1, 0) = 0.; y(1, 1) = 1.; y(1, 2) = 0.; y(1, 3) = 0.;
	y(2, 0) = -sin(-ny); y(2, 1) = 0.; y(2, 2) = cos(-ny); y(2, 3) = 0.;
	y(3, 0) = 0.; y(3, 1) = 0.; y(3, 2) = 0.; y(3, 3) = 1.;
	for (int i = of; i < of+9; i += 3)
	{
		dot(0, 0) = renderer::nPointSet[i]; dot(1, 0) = renderer::nPointSet[i + 1];
		dot(2, 0) = renderer::nPointSet[i + 2]; dot(3, 0) = 1;
		boost::numeric::ublas::matrix<double>dot0 = prod(x, dot);// поворачиваем относительно x, затем
относительно y
		boost::numeric::ublas::matrix<double>dot1 = prod(y, dot0);
		renderer::nPointSet[i] = dot1(0,0); renderer::nPointSet[i + 1] = dot1(1,0); renderer::nPointSet[i + 2]
= dot1(2,0);
	}
}

void renderer::offset(int of)
{
	boost::numeric::ublas::matrix<double>dot(4, 1);

	boost::numeric::ublas::matrix<double> shiftMatrix(4, 4);// матрица смещения
	shiftMatrix(0, 0) = 1.; shiftMatrix(0, 1) = 0.; shiftMatrix(0, 2) = 0.; shiftMatrix(0, 3) = 0.;
	shiftMatrix(1, 0) = 0.; shiftMatrix(1, 1) = 1.; shiftMatrix(1, 2) = 0.; shiftMatrix(1, 3) = -shift;
	shiftMatrix(2, 0) = 0.; shiftMatrix(2, 1) = 0.; shiftMatrix(2, 2) = 1.; shiftMatrix(2, 3) = 0.;
	shiftMatrix(3, 0) = 0.; shiftMatrix(3, 1) = 0.; shiftMatrix(3, 2) = 0.; shiftMatrix(3, 3) = 1.;
	for (int i = of; i < of + 9; i += 3)
	{
		dot(0, 0) = renderer::nPointSet[i]; dot(1, 0) = renderer::nPointSet[i + 1];
		dot(2, 0) = renderer::nPointSet[i + 2]; dot(3, 0) = 1;
		boost::numeric::ublas::matrix<double>dot0 = prod(shiftMatrix, dot);
		renderer::nPointSet[i] = dot0(0, 0); renderer::nPointSet[i + 1] = dot0(1, 0); renderer::bPointSet[i +
2] = dot0(2, 0);
	}
}

void renderer::Scale(int of)
{
	boost::numeric::ublas::matrix<double>dot(4, 1);

	boost::numeric::ublas::matrix<double> scaleMatrix(4, 4);//матрица масштаба
	scaleMatrix(0, 0) = scale; scaleMatrix(0, 1) = 0.; scaleMatrix(0, 2) = 0.; scaleMatrix(0, 3) = 0.;
	scaleMatrix(1, 0) = 0.; scaleMatrix(1, 1) = scale; scaleMatrix(1, 2) = 0.; scaleMatrix(1, 3) = 0.;
	scaleMatrix(2, 0) = 0.; scaleMatrix(2, 1) = 0.; scaleMatrix(2, 2) = scale; scaleMatrix(2, 3) = 0.;
	scaleMatrix(3, 0) = 0.; scaleMatrix(3, 1) = 0.; scaleMatrix(3, 2) = 0.; scaleMatrix(3, 3) = 1.;
	for (int i = of; i < of + 9; i += 3)
	{
		dot(0, 0) = renderer::nPointSet[i]; dot(1, 0) = renderer::nPointSet[i + 1];
		dot(2, 0) = renderer::nPointSet[i + 2]; dot(3, 0) = 1;
		boost::numeric::ublas::matrix<double>dot0 = prod(scaleMatrix, dot);
		renderer::nPointSet[i] = dot0(0, 0); renderer::nPointSet[i + 1] = dot0(1, 0); renderer::nPointSet[i +
2] = dot0(2, 0);
	}
```

```
}

void renderer::newVertSet()
{
	for (int i = 0; i < renderer::num_pairs; i++)
		renderer::nPointSet[i] = renderer::bPointSet[i]; //сбрасываем все изменения и просчитываем заново
	for (int i = 0; i < renderer::num_pairs; i += 9)
	{
		renderer::offset(i);
		renderer::Scale(i);
		renderer::rotate(i);
	}
}

bool renderer::isRight(int of)
{
	double x1, x2, y1, y2, z1, z2;//определяем 2 вектора на грани считаем их векторное произведение, а затем скалярное с вектором наблюдения (смотрим из точки 0,0,1 в 0,0,0)
	x1 = renderer::nPointSet[of + 3] - renderer::nPointSet[of];	x2 = renderer::nPointSet[of + 6] - renderer::nPointSet[of];
	y1 = renderer::nPointSet[of + 4] - renderer::nPointSet[of + 1]; y2 = renderer::nPointSet[of + 7] - renderer::nPointSet[of + 1];
	z1 = renderer::nPointSet[of + 5] - renderer::nPointSet[of + 2]; z2 = renderer::nPointSet[of + 8] - renderer::nPointSet[of + 2];
	//xn = y1 * z2 - z1 * y2 - x1; yn = -x1 * z2 + x2 * z1; zn = x1 * y2 - x2 * y1;
	double crossProd = x1 * y2 - x2 * y1;
	return crossProd < 0;
}
```

Список литературы

1. Документация **glfw**. URL: https://www.glfw.org/documentation.html.
2. Справочник по языку **C++**. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=vs-2019
3. Документация **boost**. URL: https://www.boost.org/doc/libs/1_65_1/libs/numeric/ublas/doc/index.html
4. Руководство **openGL**. URL: https://www.opengl.org.ru/docs/pg/index.html