МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

АФФИННЫЕ ПРЕОБРАЗОВАНИЯ В ПРОСТРАНСТВЕ

Отчет по лабораторной работе № 12, 13 дисциплины
«Компьютерная графика»

Выполнил студент группы ИВТ-21 _____/Рзаев А.Э./
Проверил старший преподаватель _____/Вожегов Д.В./

2016 г.

1 Постановка задачи

Написать на языке высокого уровня программу:

1. Описывающую многогранник (куб) в приборной системе координат.
2. Смещающую его на n пикселов вправо, m - вниз, p - вглубь.
3. Зеркально отражающую относительно плоскостей координат.
4. Растягивающую (сжимающую) его вдоль координатных осей относительно некоторой заданной точки.
5. Вращающую его относительно линии, проходящей через начало координат (относительно координатных осей, диагонали многогранника).
6. Реализующую трехмерную анимацию.

2 Краткие теоретические сведения

Частные случаи аффинных преобразований:

1. Сдвиг - перенос точки (x, y, z) на m единиц по координате x, на n - по y, на l единиц- по z:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ m & n & l & 1 \end{bmatrix} = [x+m \quad y+n \quad z+l \quad 1]$$

2. Поворот точки (x, y, z) вокруг оси абсцисс на угол δ:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\delta & \sin\delta & 0 \\ 0 & -\sin\delta & \cos\delta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \quad y\cos\delta - z\sin\delta \quad -y\sin\delta + z\cos\delta$$

вокруг оси ординат на угол λ:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] * \begin{bmatrix} \cos\lambda & 0 & -\sin\lambda & 0 \\ 0 & 1 & 0 & 0 \\ \sin\lambda & 0 & \cos\lambda & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x\cos\lambda + z\sin\lambda \quad y \quad -x\sin\lambda + z\cos\lambda$$

вокруг оси аппликат на угол μ:

$$[z' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] * \begin{bmatrix} \cos\mu & \sin\mu & 0 & 0 \\ -\sin\mu & \cos\mu & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x\cos\mu - y\sin\mu \quad x\sin\mu + y\cos\mu \quad z$$

Вращение вокруг произвольной оси осуществляется переносом ее вместе с изображением в начало координат, вращением вокруг перенесенной

оси и обратным переносом изображения в исходное положение. Итоговая матрица будет определена умножением соответствующих простых матриц.

3. Симметрия относительно координатных плоскостей осуществляет зеркальное отображение 3D-изображения.

Относительно плоскости XOY

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & -z & 1 \end{bmatrix}$$

Плоскости XOZ

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & -y & z & 1 \end{bmatrix}$$

Плоскости YOZ

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -x & y & z & 1 \end{bmatrix}$$

Симметрия относительно произвольной плоскости - это сложное преобразование, осуществляемое из простых поэтапно:

- совмещение плоскости симметрии с одной из координатных
- отражение относительно этой координатной плоскости
- обратное преобразование, возвращающее плоскость симметрии в исходное состояние.

4. Масштабирование осуществляется диагональными элементами матрицы преобразования.

Относительно начала координат:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} * \begin{bmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x * k_x & y * k_y & z * k_z & 1 \end{bmatrix},$$

где kx, ky, kz - коэффициенты искажения вдоль осей ox, oy, oz, соответственно.

Любое другое преобразование может быть представлено суперпозицией вращений, переносов, отражений, растяжений (сжатий).

3 Вывод

В ходе данной лабораторной работы были закреплены знания по аффинным преобразованиям, как на плоскости, так и в пространстве. Также была написана программа, реализующая анимацию с помощью переноса, масштабирования и вращения координат фигуры. Листинг кода программы и экранные формы приведены в приложениях А и Б.

Приложение А
(обязательное)
Листинг кода программы

```
#include <iostream>
#include <fstream>
#include <functional>
#include <memory>
#include <vector>
#include <cmath>
#include <SDL.h>
#include "Algo.h"


struct Point3D {
	int x, y, z;
	Point3D() = default;
	Point3D(int x, int y, int z) : x(x), y(y), z(z) {}
};

class Transformation {
public:
	double a11, a12, a13;
	double a21, a22, a23;
	double a31, a32, a33;
	Transformation() :
		a11(1), a12(0), a13(0),
		a21(0), a22(1), a23(0),
		a31(0), a32(0), a33(1) { }
	Point3D operator() (Point3D point) const {
		int x = point.x,
			y = point.y,
			z = point.z;
		return Point3D((int)(a11 * x + a21 * y + a31 * z),
					(int)(a12 * x + a22 * y + a32 * z),
					(int)(a13 * x + a23 * y + a33 * z));
	}
};

Transformation MakeRotationZTransform(double angle) {
	Transformation tr;
	tr.a11 = std::cos(angle);
	tr.a12 = std::sin(angle);
	tr.a21 = -std::sin(angle);
	tr.a22 = std::cos(angle);
	tr.a33 = 1;
	return tr;
}

Transformation MakeScaleTransform(double scale) {
	Transformation tr;
	tr.a11 = tr.a22 = tr.a33 = scale;
	return tr;
}

Point3D MovePoint3D(Point3D src, Point3D sh) {
	return Point3D(src.x + sh.x, src.y + sh.y, src.z + sh.z);
}

void ClearRender(const ren_ptr& renderer) {
	SDL_SetRenderDrawColor(renderer.get(), 0xff, 0xff, 0xff, 0);
	SDL_RenderClear(renderer.get());
}
```

```
void SetRenderColor(const ren_ptr& renderer, int r, int g, int b, int a) {
	SDL_SetRenderDrawColor(renderer.get(), r, g, b, a);
}

void DrawCube(ren_ptr ren, const std::vector<pii>& proj) {
	pii a = proj[0], b = proj[1],
		c = proj[2], d = proj[3],

		a1 = proj[4], b1 = proj[5],
		c1 = proj[6], d1 = proj[7];

	SetRenderColor(ren, 255, 0, 0, 0);
	SDL_RenderDrawLine(ren.get(), a.first, a.second, b.first, b.second);
	SDL_RenderDrawLine(ren.get(), b.first, b.second, c.first, c.second);
	SDL_RenderDrawLine(ren.get(), c.first, c.second, d.first, d.second);
	SDL_RenderDrawLine(ren.get(), d.first, d.second, a.first, a.second);

	SDL_RenderDrawLine(ren.get(), a1.first, a1.second, b1.first, b1.second);
	SDL_RenderDrawLine(ren.get(), b1.first, b1.second, c1.first, c1.second);
	SDL_RenderDrawLine(ren.get(), c1.first, c1.second, d1.first, d1.second);
	SDL_RenderDrawLine(ren.get(), d1.first, d1.second, a1.first, a1.second);

	SDL_RenderDrawLine(ren.get(), a.first, a.second, a1.first, a1.second);
	SDL_RenderDrawLine(ren.get(), b.first, b.second, b1.first, b1.second);
	SDL_RenderDrawLine(ren.get(), c.first, c.second, c1.first, c1.second);
	SDL_RenderDrawLine(ren.get(), d.first, d.second, d1.first, d1.second);
}

void MainLoop() {
	win_ptr window( SDL_CreateWindow("Test", 100, 100, 1024, 768,
SDL_WINDOW_SHOWN), &SDL_DestroyWindow );
	ren_ptr renderer( SDL_CreateRenderer(window.get(), -1,
SDL_RENDERER_ACCELERATED), &SDL_DestroyRenderer );

	std::vector<Point3D> cube = {
		Point3D(-50, -50, 0), Point3D(-50, -50, 100), Point3D(50, -50, 100),
Point3D(50, -50, 0),
		Point3D(-50, 50, 0), Point3D(-50, 50, 100), Point3D(50, 50, 100),
Point3D(50, 50, 0)
	};

	int rotation = 0, radius = 100;
	int xr, yr;
	while (true) {
		SDL_Delay(60);

		std::vector<Point3D> cube_copy = cube;
		xr = radius * cos(3.14 * rotation / 180 * 5);
		yr = radius * sin(3.14 * rotation / 180 * 5);
		for (int i = 0; i < 8; ++i) {
			cube_copy[i] = MakeRotationZTransform(3.14 * rotation /
180)(cube_copy[i]);
			cube_copy[i] = MakeScaleTransform(sin(3.14 * rotation / 180 * 2) +
1.5)(cube_copy[i]);
			cube_copy[i] = MovePoint3D(cube_copy[i], Point3D(400 + xr, 400 +
yr, 0));
		}
		rotation = (rotation + 1) % 360;

		std::vector<pii> proj(8);
		for (int i = 0; i < 8; ++i) {
			double d = 400;
			int x = cube_copy[i].x,
				y = cube_copy[i].y,
				z = cube_copy[i].z;
			proj[i] = {
```

```
                        x / (z / d + 1),
                        y / (z / d + 1) };
            }

            ClearRender(renderer);
            SetRenderColor(renderer, 255, 0, 0, 0);

            DrawCube(renderer, proj);
            SDL_RenderPresent(renderer.get());

            SDL_Event e;
            SDL_WaitEventTimeout(&e, 2);
            if (e.type = SDL_KEYDOWN && e.key.state == SDL_PRESSED &&
e.key.keysym.sym == SDLK_ESCAPE) {
                  break;
            }
      }
}

int main(int argc, char** args){
      if (SDL_Init(SDL_INIT_VIDEO) != 0){
            std::cout << "SDL_Init Error: " << SDL_GetError() << std::endl;
            return 1;
      }

      MainLoop();

      SDL_Quit();
      return 0;
}
```

Приложение Б
(обязательное)
Экранные формы программы

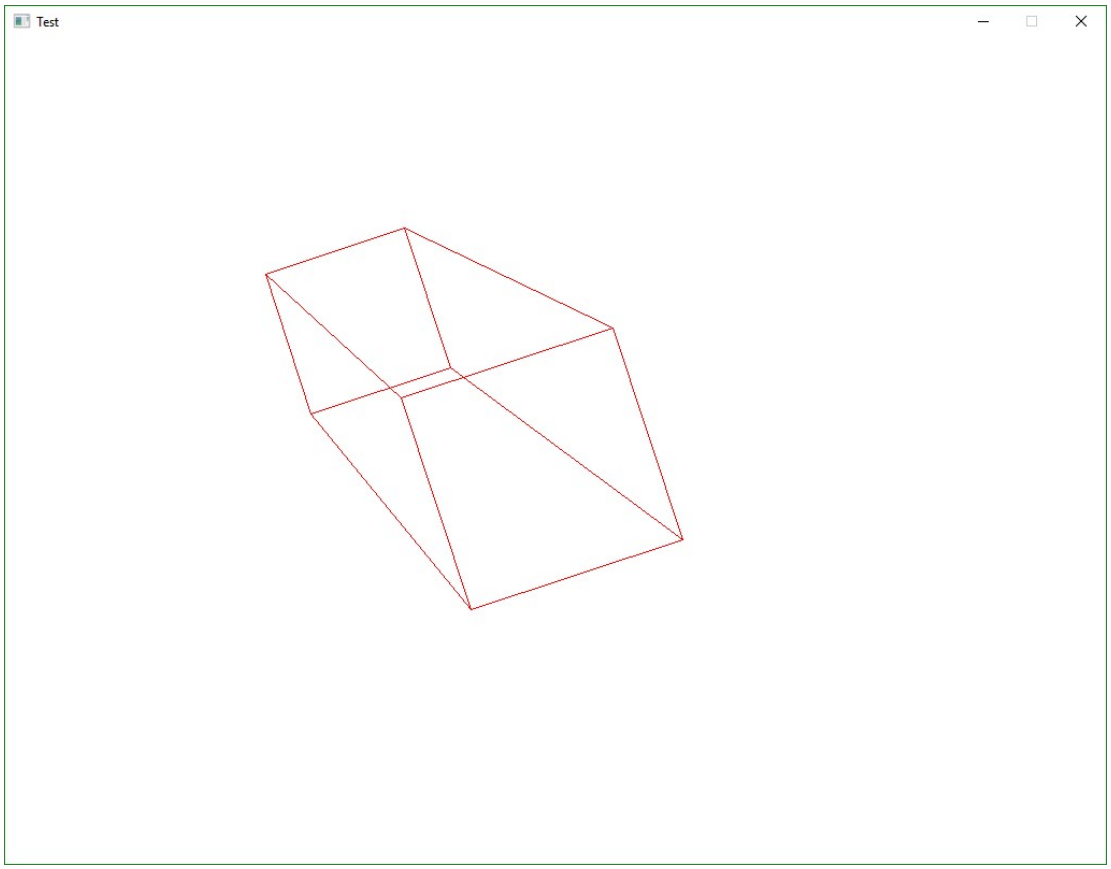

Рисунок Б.1 – Основной экран программы