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 НА ПЛОСКОСТИ

Отчет по лабораторной работе №6, 7 дисциплины
«Компьютерная графика»

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

2016 г.

1 Постановка задачи

Написать программу, реализующую алгоритм анимации с применением аффинных преобразований на плоскости.

2 Краткие теоретические сведения

Система координат - совокупность правил, ставящих в соответствие каждой точке набор чисел (координат), число которых называется размерностью пространства. Допустим, на плоскости введена прямолинейная координатная система, тогда каждой точке ставится в соответствие упорядоченная пара чисел (x, y) - ее координат.

В компьютерной графике чаще всего используют аффинную и декартовую систему координат. Это прямоугольная координатная система, в которой выбирается точка O - начало координат и два приложенных к ней неколлинеарных единичных вектора e_x и e_y, которые задают оси координат. Если единичные отрезки на осях не равны, система называется аффинной, если равны и угол между осями координат прямой - прямоугольной декартовой.

Однородным представлением n-мерного объекта является его представление в (n+1)-мерном пространстве, полученное добавлением еще одной координаты - скалярного множителя. При решении задач компьютерной графики однородные координаты обычно вводятся так: произвольной точке M(x,y) на плоскости ставится в соответствие точка M(x,y,1) в пространстве.

Если нам необходимо преобразовать точку на плоскости с координатами (x,y) в другую точку то задача сводится к поиску новых координат для этой точки - (x’,y’). В случае аффинных преобразований такой поиск сведется к решению уравнений

$$x' = ax + by + m$$
$$y' = cx + dy + n,$$

где a, b, c, d, m, n - произвольные числа, причем: $\begin{bmatrix} ab \\ cd \end{bmatrix} \approx 0$.

В случае, когда m и n не равны нулю, для представления преобразования в матричной форме нужно исходные и преобразованные координаты точки записать в однородных координатах (x,y,1) и (x’,y’,1). Тогда в матричной форме общий вид преобразования будет следующим:

$$[x'\ y'\ 1] = [x\ y\ 1] * \begin{bmatrix} a & c & 0 \\ b & d & 0 \\ m & n & 1 \end{bmatrix} = [a*x + b*y + m \quad c*x + d*y + n \quad 1].$$

3 Разработка алгоритма

Анимация выполнена в виде двоичного трехразрядного счетчика на кривых Безье. Пользователь может перемещать, вращать и масштабировать счетчик, используя клавиатуру.

Схемы алгоритмов приведены в приложении А, листинг программ приведен в приложении Б, скриншоты программы приведены в приложении В.

4 Вывод

В ходе данной лабораторной работы были получены знания об аффинных преобразованиях на плоскости в общем виде и в частных случаях. В результате работы была написана программа, демонстрирующая анимацию с использованием аффинных преобразований.

Приложение А
(обязательное)
Блок-схемы алгоритмов

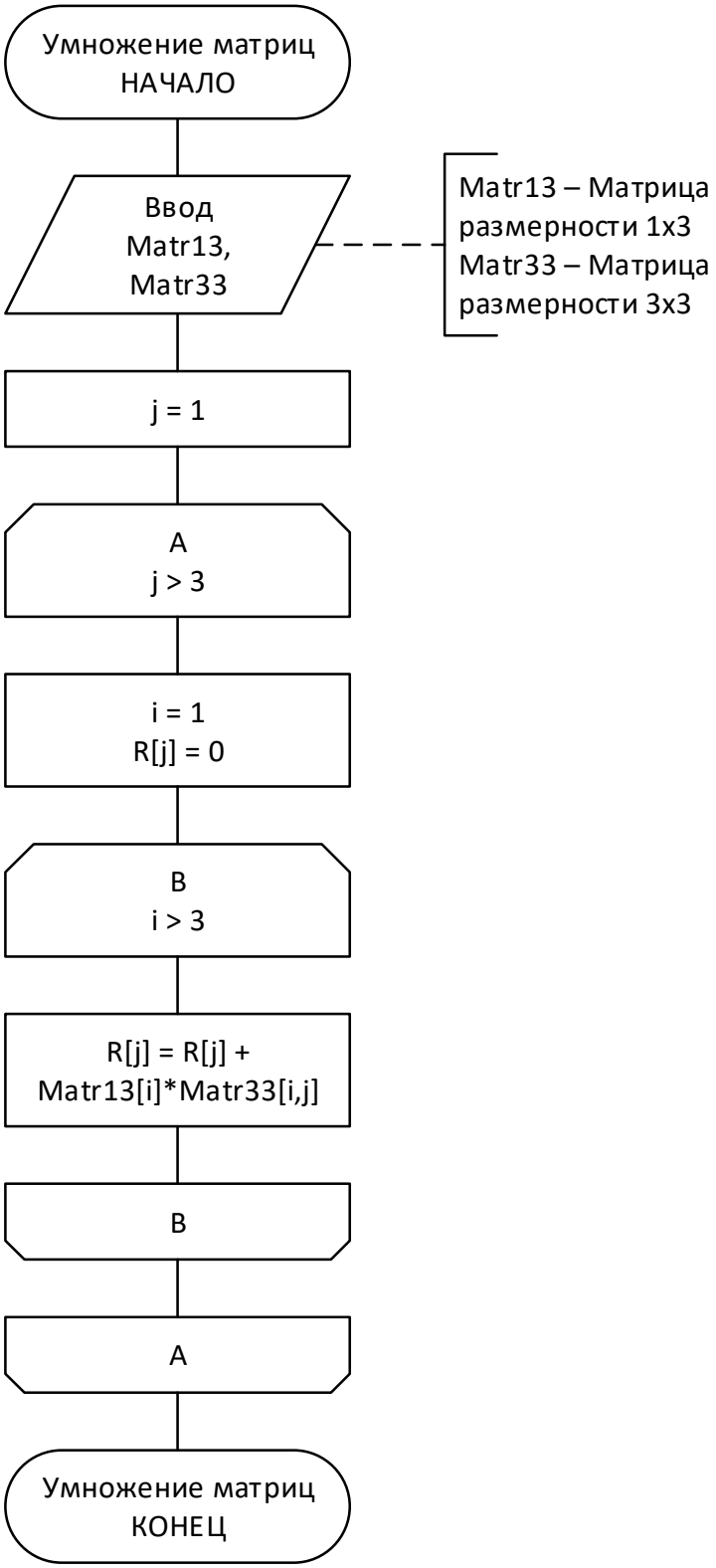

Рисунок А.1 – Схема алгоритма перемножения матриц

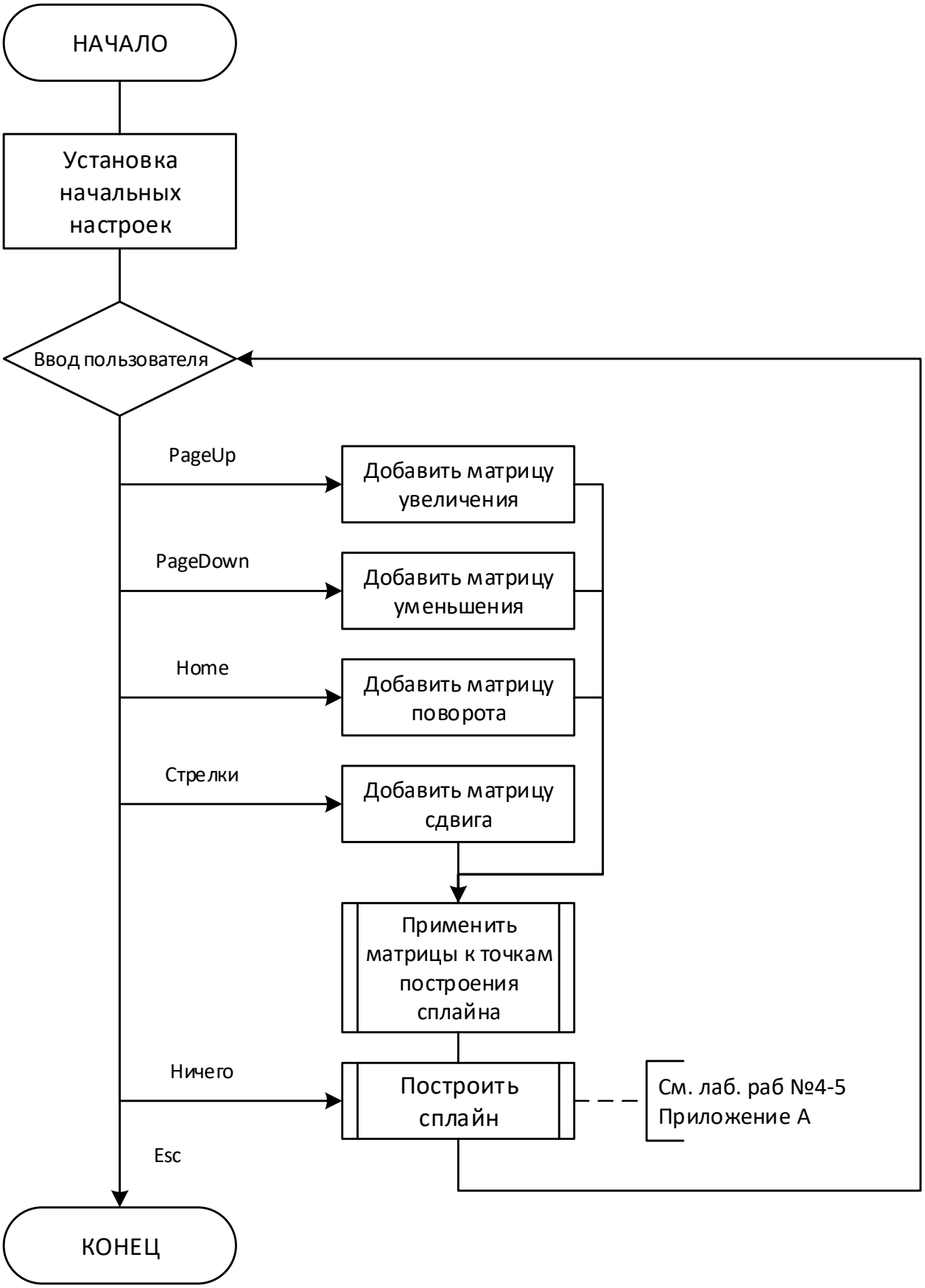

Рисунок А.2 – Схема алгоритма основного цикла программы

Приложение Б
(обязательное)
Листинг программы

CounterWidget.h

```
#pragma once
#include <SDL.h>
#include <fstream>
#include <functional>
#include <memory>
#include <vector>
#include <cmath>
#include <array>

class window_deleter {
public:
	void operator ()(SDL_Window *win) {
		SDL_DestroyWindow(win);
	}
};

class renderer_deleter {
public:
	void operator ()(SDL_Renderer *ren) {
		SDL_DestroyRenderer(ren);
	}
};

typedef std::unique_ptr < SDL_Window, window_deleter > win_ptr;
typedef std::unique_ptr < SDL_Renderer, renderer_deleter > ren_ptr;
typedef std::pair < int, int > pii;
typedef std::vector<std::vector<pii> > Digit;

class Counter {
public:
	std::array<pii, 3> Current() const {
		std::array<pii, 3> ret;
		ret[0] = pii(_prev % 2,     _prev % 2     == _current % 2     ? 0 :
_frame);
		ret[1] = pii(_prev / 2 % 2, _prev / 2 % 2 == _current / 2 % 2 ? 0 :
_frame);
		ret[2] = pii(_prev / 4 % 2, _prev / 4 % 2 == _current / 4 % 2 ? 0 :
_frame);
		return ret;
	}
	void Next() {
		_frame = (_frame + 1) % 11;
		if (_frame == 0) {
			_prev = _current;
			++_current;
		}
	}
	void Initialize() {
		_current = 1;
		_prev = 0;
		_frame = 0;
	}
private:
	unsigned int _current;
	unsigned int _prev;
	unsigned int _frame;
};

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) { }
	pii operator() (pii point) const {
		int x = point.first, y = point.second;
		return pii((int)(a11 * x + a21 * y + a31), (int)(a12 * x + a22 * y +
a32));
	}
};

SDL_Point MakePoint(int x, int y) {
	SDL_Point p; p.x = x; p.y = y;
	return p;
}

SDL_Point MakePoint(pii point) {
	SDL_Point p; p.x = point.first; p.y = point.second;
	return p;
}

Transformation MakeMoveTranform(pii point) {
	Transformation tr;
	tr.a31 = point.first; tr.a32 = point.second;
	return tr;
}

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

Transformation MakeScaleTranform(double s, pii center) {
	Transformation tr;
	int x = center.first, y = center.second;
	tr.a11 = s;		tr.a22 = s;
	//tr.a31 = -x * s + x; tr.a32 = -y * s + y;
	return tr;
}

class Widget {
public:
	void ScaleUp(double s) {
		_scale *= s;
		_scale_tr = MakeScaleTranform(_scale, _center);
	}
	void ScaleDown(double s) {
		_scale /= s;
		_scale_tr = MakeScaleTranform(_scale, _center);
	}
	void Move(pii point) {
		_center = point;
		_move_tr = MakeMoveTranform(_center);
	}
	void MoveRel(pii point) {
		_center.first += point.first;
		_center.second += point.second;
		_move_tr = MakeMoveTranform(_center);
	}
```

```
	void Rotate(unsigned int delta) {
		_angle = (_angle + delta + 360) % 360;
		_angle_tr = MakeRotateTranform(_angle * 2 * 3.14 / 360);
	}
	void Draw(const ren_ptr& renderer) {
		std::array<pii, 3> c = _counter.Current();
		for (int i = 0; i < 3; ++i) {
			_DrawSpline(renderer, _digit_frames[c[3 - i - 1].first][c[3 - i -
1].second][0], 30 + i * 120, 30);
			_DrawSpline(renderer, _digit_frames[c[3 - i - 1].first][c[3 - i -
1].second][1], 30 + i * 120, 30);
		}
		SDL_RenderPresent(renderer.get());
		_counter.Next();
	}
	void Initialize(const std::array<std::vector<Digit>, 2>& digit_frames) {
		_scale = 1;
		_angle = 0;
		_center = pii(0, 0);
		_digit_frames = digit_frames;
		_counter.Initialize();
	}
private:
	double			_scale;
	unsigned int _angle;
	pii				_center;
	std::array<std::vector<Digit>, 2> _digit_frames;
	Transformation _move_tr, _scale_tr, _angle_tr;
	Counter _counter;


	void _DrawCircle(const ren_ptr& ren, int x, int y, double r) {
		std::vector<SDL_Point> points;

		for (int i = 0; i <= 360; i += 36) {
			int x1 = x + r * std::cos(i * 6.28 / 360);
			int y1 = y + r * std::sin(i * 6.28 / 360);
			points.push_back(MakePoint(x1, y1));
		}

		SDL_RenderDrawLines(ren.get(), points.data(), points.size());
		points.clear();

	}

	void _DrawRect(const ren_ptr& renderer, int x1, int y1, int x2, int y2) {
		std::vector<SDL_Point> points;
		points.push_back(MakePoint(_move_tr(_angle_tr(_scale_tr(pii(x1, y1))))));
		points.push_back(MakePoint(_move_tr(_angle_tr(_scale_tr(pii(x2, y1))))));
		points.push_back(MakePoint(_move_tr(_angle_tr(_scale_tr(pii(x2, y2))))));
		points.push_back(MakePoint(_move_tr(_angle_tr(_scale_tr(pii(x1, y2))))));
		points.push_back(MakePoint(_move_tr(_angle_tr(_scale_tr(pii(x1, y1))))));
		SDL_RenderDrawLines(renderer.get(), points.data(), points.size());
	}

	void _DrawSpline(const ren_ptr& ren, const std::vector<pii>& points, int bx,
int by) {
		auto round = [](double val) -> int { return (int)std::ceil(val - 0.5); };
		std::vector<pii> R = points, P = points, spline_points;
		int m = points.size();
		double t = 0, step = 0.05;

		pii current_point(P[0]);
		pii point = _move_tr(_angle_tr(_scale_tr(
			pii(current_point.first + bx, current_point.second + by))));
		_DrawCircle(ren, point.first, point.second, 2 * _scale);
```

```
            while (t < 1) {
                  R = P;
                  for (int j = m; j > 1; --j) {
                        for (int i = 0; i < j - 1; ++i) {
                              R[i].first = R[i].first + round(t * (R[i + 1].first -
R[i].first));
                              R[i].second = R[i].second + round(t * (R[i + 1].second
- R[i].second));
                        }
                  }

                  t += step;
                  current_point = R[0];
                  pii point = _move_tr(_angle_tr(_scale_tr(
                        pii(current_point.first + bx, current_point.second + by))));
                  _DrawCircle(ren, point.first, point.second, 2 * _scale);
            }
      }
};
```

main.cpp

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

std::vector<Digit> LoadDigits() {
      std::ifstream file("points.txt");
      std::vector<Digit> digits(2);

      for (int k = 0; k < 2; ++k) {
            int n; file >> n;
            digits[k].resize(n);
            for (int i = 0; i < n; ++i) {
                  for (int j = 0; j < 4; ++j) {
                        int x, y; file >> x >> y;
                        x -= 180, y -= 120;
                        digits[k][i].emplace_back(x, y);
                  }
            }
      }

      return digits;
}

std::vector<Digit> MakeFrames(const Digit& first, const Digit& second) {
      std::vector<Digit> frames(11);
      for (int i = 0; i < 11; ++i) {
            frames[i].resize(2);
            for (int j = 0; j < 2; ++j) {
                  frames[i][j].resize(4);
                  for (int p = 0; p < 4; ++p) {
                        frames[i][j][p] = pii(
                              first[j][p].first + (second[j][p].first -
first[j][p].first) * i / 10,
                              first[j][p].second + (second[j][p].second -
first[j][p].second) * i / 10
                              );
                  }
            }
      }
```

```
	return frames;
}

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);
}

bool WaitTimeout(int timeout, SDL_EventType type) {
	SDL_Event e;
	if (SDL_WaitEventTimeout(&e, timeout)) {
		if (e.type == type) {
			return true;
		}
	}
	return false;
}

void WaitUntill(SDL_EventType type) {
	SDL_Event e;
	SDL_WaitEvent(&e);
	while (e.type != type) {
		SDL_WaitEvent(&e);
	}
}

void MainLoop() {
	win_ptr window( SDL_CreateWindow("Test", 100, 100, 1024, 850, SDL_WINDOW_SHOWN)
);
	ren_ptr renderer(SDL_CreateRenderer(window.get(), -1,
SDL_RENDERER_ACCELERATED));

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

	std::vector<Digit> digits = LoadDigits();
	std::array<std::vector<Digit>, 2> digit_frames = { MakeFrames(digits[0],
digits[1]), MakeFrames(digits[1], digits[0]) };

	Widget w;
	w.Initialize(digit_frames);
	w.Move(pii(320, 240));
	bool stop = false;
	while (!stop) {
		SDL_Event e;
		SDL_WaitEventTimeout(&e, 2);
		if (e.type = SDL_KEYDOWN && e.key.state == SDL_PRESSED) {
			switch (e.key.keysym.sym) {
			case SDLK_LEFT:
				w.MoveRel(pii(-10, 0));
				break;
			case SDLK_RIGHT:
				w.MoveRel(pii(10, 0));
				break;
			case SDLK_UP:
				w.MoveRel(pii(0, -10));
				break;
			case SDLK_DOWN:
				w.MoveRel(pii(0, 10));
				break;
			case SDLK_PAGEDOWN:
```

```
                        w.Rotate(5);
                        break;
                  case SDLK_PAGEUP:
                        w.Rotate(-5);
                        break;
                  case SDLK_HOME:
                        w.ScaleUp(1.2);
                        break;
                  case SDLK_END:
                        w.ScaleDown(1.2);
                        break;
                  case SDLK_ESCAPE:
                        stop = true;
                        break;
                  }
            }

            ClearRender(renderer);
            SetRenderColor(renderer, 255, 0, 0, 0);
            w.Draw(renderer);
            SDL_Delay(80);
      }
}

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

      MainLoop();

      SDL_Quit();
      return 0;
}
```

Приложение В
(обязательное)
Экранные формы программы

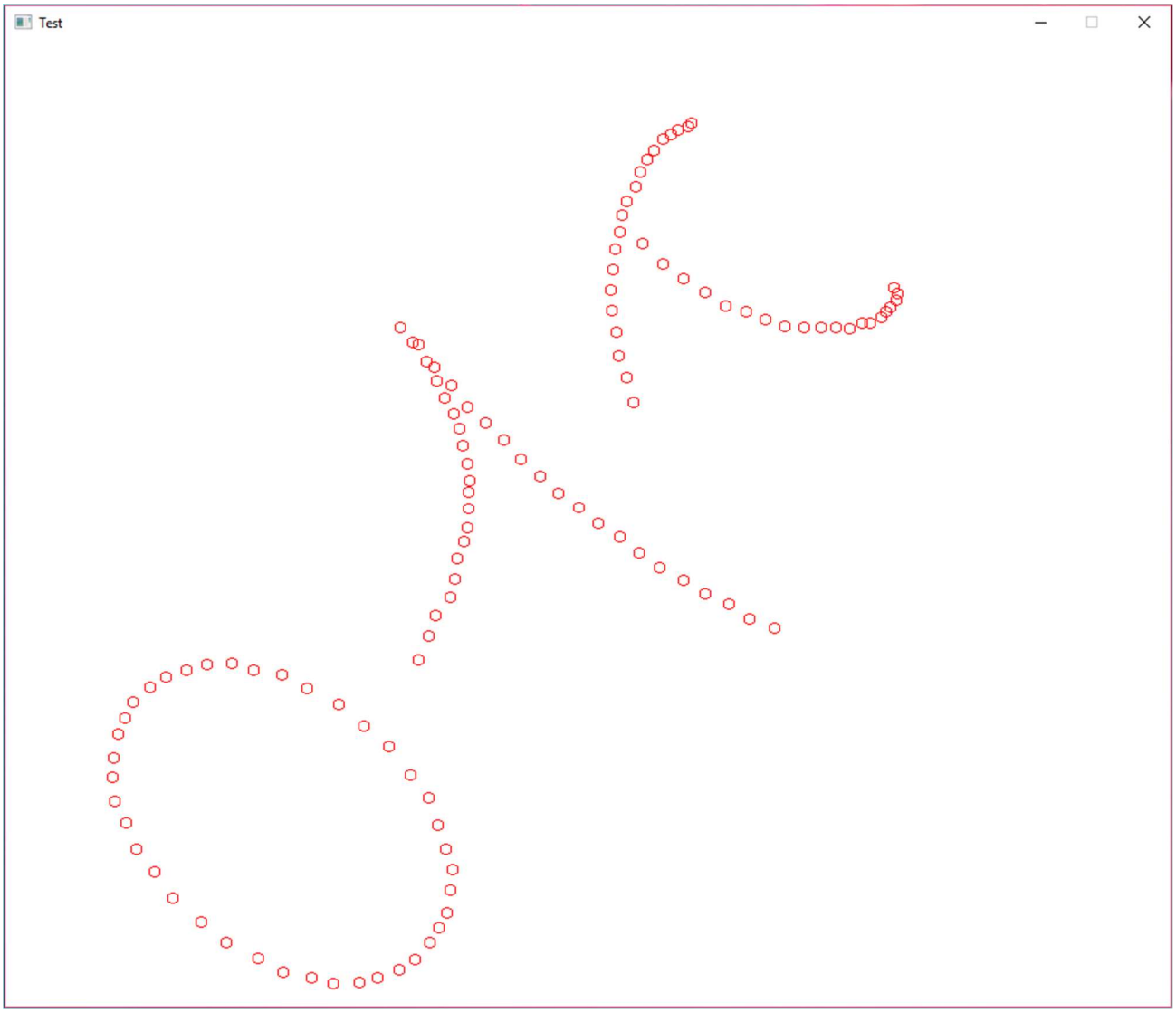

Рисунок В.1 – Основной экран программы