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

РЕАЛИЗАЦИЯ ДВЕМЕРНОГО АЛГОРИТМА ОТСЕЧЕНИЯ ОТРЕЗКА САЗЕРЛЕНДА-КОЭНА

Отчет по лабораторной работе №10 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 PASCAL программу, реализующую алгоритм Сазерленда-Коэна, отсекающий отрезок по границам прямоугольного окна. Для показа результатов работы программы нарисовать на экране окно прямоугольной формы. Задав координаты окна и отрезка, продемонстрировать отсечение отрезка по границам окна. Рассмотреть все возможные случаи расположения отрезка относительно окна.

2 Краткие теоретические сведения

Необходимость отсечь выводимое изображение по границам некоторой области (окна) встречается довольно часто, особенно в многооконных диалоговых графических системах. В простейших случаях в качестве отсекающей области выступает прямоугольник. Существует много алгоритмов отсечения для 2D- и 3D-случаев, ориентированных как на программную, так и аппаратную реализацию. Рассмотрим простой и эффективный алгоритм отсечения отрезков границами произвольного прямоугольника. Вся плоскость вывода разбивается прямыми, образующими прямоугольник на девять подплоскостей; каждой из них присваивается четырехбитный код, в котором единица

- в нулевом бите означает, что конец отрезка лежит левее окна;
- в первом бите - выше окна;
- во втором бите - правее окна;
- в третьем бите - ниже окна.

Например, если один конец отсекаемого отрезка лежит в правом верхнем относительно окна углу экрана, он получает код этого угла равный 0110 по схеме:

0011	0010	0110
0001	0000	0100
1000	1000	1100

3 Разработка алгоритма

Общая идея алгоритма состоит в том, чтобы определить положение вершины относительно “экрана”, и если вершина лежит вне него, то необходимо провести отсечение части прямой.

Схема алгоритма приведена в приложении А, листинг процедур, реализующих данный алгоритм приведён в приложении Б, экранные формы программы находятся в приложении В.

4 Вывод

В ходе данной лабораторной работы был изучен алгоритм отсечения прямой на плоскости Сазерленда-Коэна. Главным преимуществом этого алгоритма является быстрота его выполнения: в худшем случае прямая будет отсечена за 4 шага, тогда как, например, проход по всем координатам линии и проверка принадлежности точки “экрану” займёт значительно больше времени.

Приложение А
(обязательное)
Блок-схемы алгоритмов

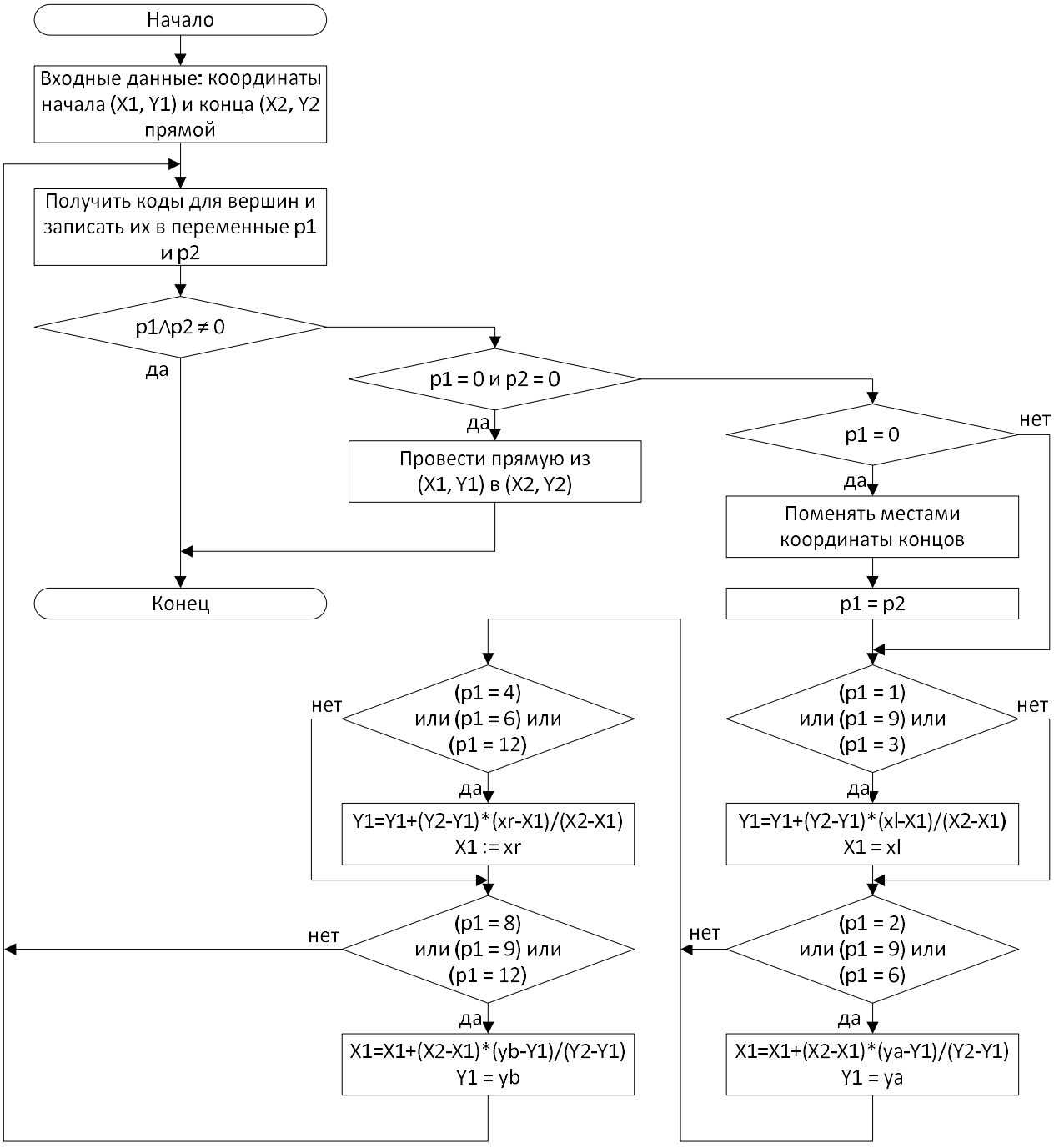

Рисунок А.1 – Схема алгоритма Сазерленда-Коэна

Приложение Б
(обязательное)
Листинг процедур

```
unsigned CohenSutherlandMask(pii r1, pii r2, int x, int y) {
	int rx1 = r1.first, ry1 = r1.second,
		rx2 = r2.first, ry2 = r2.second;
	if (x < rx1 && y < ry1) {
		return 0x9;
	}
	else if (x >= rx1 && x <= rx2 && y < ry1) {
		return 0x8;
	}
	else if (x > rx2 && y < ry1) {
		return 0xA;
	}
	else if (x > rx2 && y >= ry1 && y <= ry2) {
		return 0x2;
	}
	else if (x > rx2 && y > ry2) {
		return 0x6;
	}
	else if (x >= rx1 && x <= rx2 && y > ry2) {
		return 0x4;
	}
	else if (x < rx1 && y > ry2) {
		return 0x5;
	}
	else if (x < rx1 && y >= ry1 && y <= ry2) {
		return 0x1;
	}
	else {
		return 0x0;
	}
}

void CohenSutherland(ren_ptr ren, std::pair<pii, pii> rect, const
std::vector<std::pair<pii, pii>> lines) {
	int rx1 = rect.first.first, rx2 = rect.second.first,
		ry1 = rect.first.second, ry2 = rect.second.second;
	if (rx1 > rx2) {
		std::swap(rx1, rx2);
	}
	if (ry1 > ry2) {
		std::swap(ry1, ry2);
	}
	for (const auto& line : lines) {
		int x1 = line.first.first, y1 = line.first.second,
			x2 = line.second.first, y2 = line.second.second;

		unsigned p1, p2;
		p1 = CohenSutherlandMask({ rx1, ry1 }, { rx2, ry2 }, x1, y1),
		p2 = CohenSutherlandMask({ rx1, ry1 }, { rx2, ry2 }, x2, y2);
		do {
			if ((p1 & p2) != 0) {
				break;
			}
			if (p1 == 0 && p2 == 0) {
				//DrawLineBresenhem8(ren, x1, y1, x2, y2);
				SDL_RenderDrawLine(ren.get(), x1, y1, x2, y2);
				break;
			}
			else {
				if (p1 == 0) {
```

```
                        std::swap(x1, x2);
                        std::swap(y1, y2);
                        std::swap(p1, p2);
                    }

                    // left
                    if ((p1 & 0x1) != 0) {
                    y1 = (int) y1 + 1.0 * (y2 - y1) * (rx1 - x1) / (x2 - x1);
                    x1 = rx1;
                    }
                    //top
                    else if ((p1 & 0x8) != 0) {
                    x1 = (int) x1 + 1.0 * (x2 - x1) * (ry1 - y1) / (y2 - y1);
                    y1 = ry1;
                    }
                    //right
                    else if ((p1 & 0x2) != 0) {
                    y1 = (int) y1 + 1.0 * (y2 - y1) * (rx2 - x1) / (x2 - x1);
                    x1 = rx2;
                    }
                    //bottom
                    else if ((p1 & 0x4) != 0) {
                    x1 = (int) x1 + 1.0 * (x1 - x1) * (ry2 - y1) / (y2 - y1);
                    y1 = ry2;
                    }
                }
                p1 = CohenSutherlandMask({ rx1, ry1 }, { rx2, ry2 }, x1, y1),
                p2 = CohenSutherlandMask({ rx1, ry1 }, { rx2, ry2 }, x2, y2);
            } while (true);
        }
}
```

Приложение В
Экранные формы программы

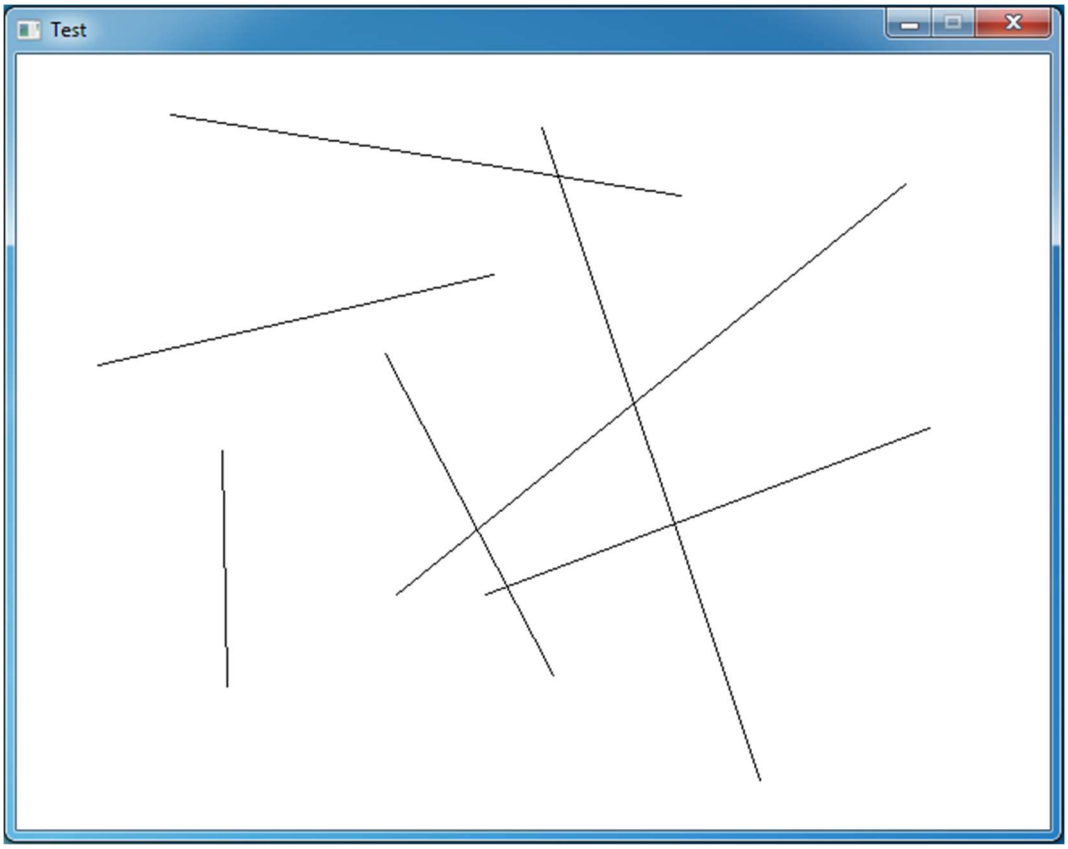

Рисунок В.1 – Основной экран программы до применения алгоритма

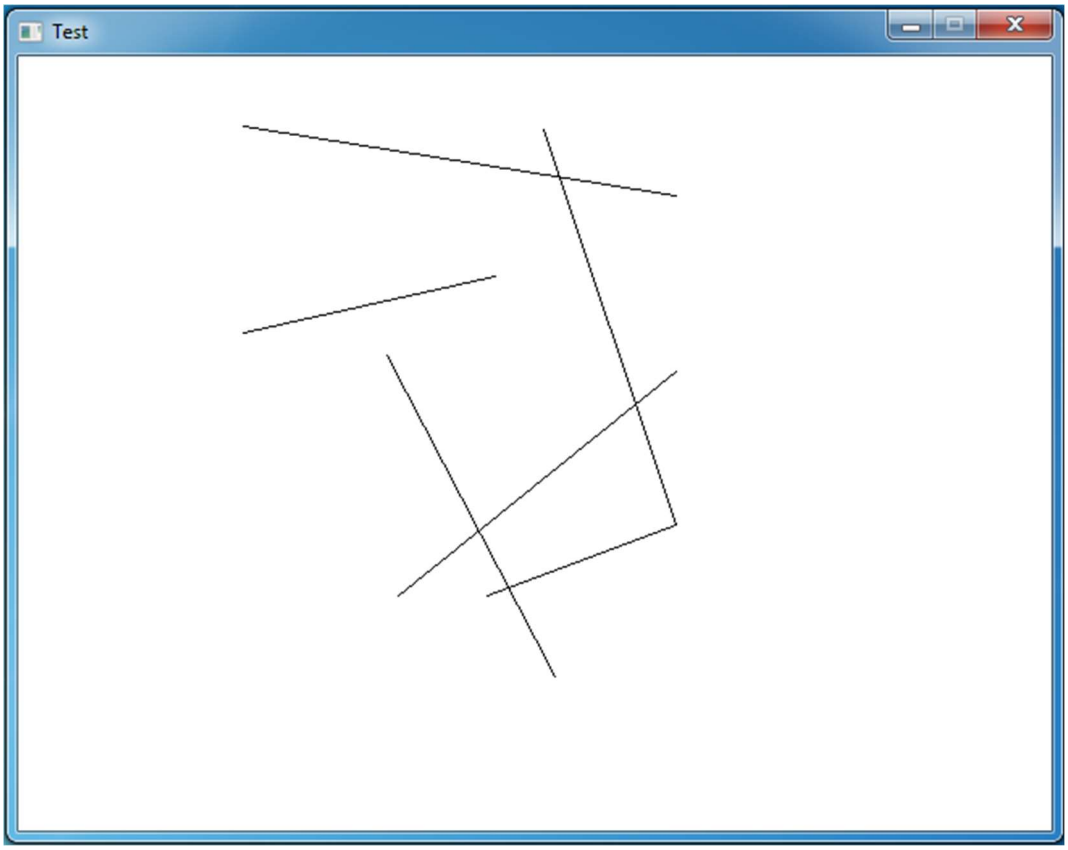

Рисунок В.2 – Основной экран программы после применения алгоритма