Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Южно-Уральский государственный университет
(национальный исследовательский университет)»
Школа «Высшая школа электроники и компьютерных наук»
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
Г.И. Радченко
«___» _____2018 г.

Разработка демонстрационного комплекса программ для изучения алгоритмов растеризации

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ - 090301.2018.157 ПЗ ВКР

Руководитель работы
к.т.н., доцент ЭВМ ВШЭКМ
_____ / Е.С. Ярош
«___» _____2018 г.

Автор работы
студент группы КЭ-484
_____ / А.В. Козлова
«___» _____2018 г.

Нормоконтроллер,
ст. преп. ЭВМ ВШЭКМ
_____ / В.В. Лурье
«___» _____2018 г.

Челябинск 2018

Аннотация

Козлова А. В. Разработка демонстрационного комплекса программ для изучения алгоритмов растеризации. - Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)» ВШЭиКН; 2018, 78с., 49 ил., библиогр. список - 17 наименований.

Выпускная квалификационная работа посвящена разработке демонстрационного программного комплекса для изучения алгоритмов растеризации, который визуализирует растровое построение простейших объектов, позволяет прочитать теорию по популярным растровым алгоритмам, изучить их блок-схемы, посмотреть исходный код.

Работа состоит из 5 глав, также есть введение, заключение и библиографический список. В первой главе определены требования к системе, описаны найденные аналоги как отечественные, так и зарубежные, приведено их сравнение. Во второй главе - планирование. В ней выбираются инструменты разработки, проектируются необходимые модели системы. В третьей главе описана архитектура разрабатываемого приложения. В четвертой представлены основные технические решения, особенности реализации. Пятая глава демонстрирует полученные результаты.

					09.03.01.2018.157.00 ПЗ			
Из								
Разраб.	А.В.Козлова			Разработка демонстрационного комплекса программ для изучения алгоритмов растеризации		Лит.	Лист	Листов
Пров.	Е.С.Ярош					Д	3	78
						ФГАОУВО «ЮУрГУ (НИУ)» Кафедра ЭВМ		
Н. контр.	В.В. Лурье							
Утв.								

Введение

На большинстве специальностей, связанных с IT, преподается дисциплина «Компьютерная графика» либо «Алгоритмы и методы представления графической информации», которые являются дисциплинами общепрофессиональной подготовки. Основная задача которых – приобретение обучающимися знаний теоретического и прикладного характера, что помогает понять, каким образом строится изображение на мониторе компьютера, осуществлять разработку и осваивать современные графические системы.

Для наглядности и лучшего усвоения материала необходимо демонстрировать работу алгоритмов. В настоящее время нет единого комплекса примеров алгоритмов, существуют лишь отдельные файлы, написанные на различных языках программирования, для разных операционных систем. Данные файлы необходимо искать, скачивать, на это тратится время, причем не гарантируется безопасность файлов, корректность алгоритмов, а также файлы могут оказаться и вовсе неработоспособными.

Этими обстоятельствами обоснована необходимость разработки демонстрационного комплекса программ для изучения алгоритмов представления графической информации. Так как в современном мире люди постоянно сталкиваются с растровой графикой, целесообразно разработать программный комплекс по растровым алгоритмам.

В проект кроме визуализации алгоритмов входит теоретический материал по ним, блок-схемы, исходные коды алгоритмов. Теоретический материал взят из учебно-методического пособия Методы и средства представления графической информации автора к.т.н. Ярош Е.С., алгоритмы написаны по этому же пособию.

Обозначим цель и задачи работы. Цель: разработать демонстрационный комплекс программ для изучения алгоритмов растеризации.

Задачи:

- определить требования к системе

- провести обзор аналогов
- сравнить отечественные и зарубежные решения,
- выбрать инструменты для разработки
- разработать архитектуру приложения
- выполнить программную реализацию
- протестировать приложение

Основные понятия предметной области

Растровое представление графической информации - при данном методе представления графики используется технология хранения информации о каждом пикселе (pixel- pictureelement).

Растровое изображение - изображение, представляющее собой сетку пикселей - цветных точек (обычно прямоугольных) на мониторе, бумаге и других отображающих устройствах.

Пиксель - комбинированный термин, обозначающий элемент изображения, который является наименьшим элементом экрана монитора, принтера и других графических систем вывода информации. Пиксель является минимальным сегментом растровой строки, которая дискретно управляется системой, образующей изображение. С другой стороны, это координата, используемая для определения горизонтальной и вертикальной пространственной позиции пикселя в пределах изображения.

Связность - возможность соединения двух пикселей растровой линией, т. е. последовательным набором пикселей.

Четырехсвязность - пиксели считаются соседними, если либо их x-координаты, либо их y – координаты отличаются на единицу:

$$|x_1 - x_2| + |y_1 - y_2| \le 1;$$

Восьмисвязность - пиксели считаются соседними, если их x-координаты и y-координаты отличаются не более чем на единицу:

$$|x_1 - x_2| \le 1, \ |y_1 - y_2| \le 1.$$

Коэффициент прозрачности - в описание цвета (RGB) может входить специальный канал, называемый альфа-каналом, который отвечает за прозрачность данного цвета. Таким образом, цвет описывается как ARGB.

1 Обзор и анализ предметной области

1.1 Описание проблемы

В учебном процессе технических специальностей, связанных с информационными технологиями, в рамках общепрофессиональной подготовки рассматриваются способы компьютерного представления графической информации. Это позволяет:

- получить теоретические сведения об используемых алгоритмах представления графической информации;
- при разработке выбирать наиболее подходящие к требованиям предметной области алгоритмы, методы представления графической информации.

Чтобы лучше понять и усвоить материал, нужны наглядные примеры. Проблема заключается в том, что обычно их нет, либо нужно искать и скачивать, рискуя безопасностью компьютера и затрачивая время. При этом найденный пример, визуализирующий определенный алгоритм, может иметь некоторые расхождения (или быть вовсе неправильным) с таким же алгоритмом в учебном пособии, по которому занимаются студенты.

Существует множество графических алгоритмов, с помощью которых осуществляется формирование изображений, наделение их некоторыми свойствами. Очевидно, что в рамках дипломной работы охватить все эти алгоритмы невозможно. Одной из наиболее часто встречающихся является задача растеризации непрерывных объектов: отрезков прямых, эллипсов и окружностей. Поэтому данная работа ограничивается алгоритмами этого класса.

1.2 Определение требований к системе

Исходя из требований заказчика, в приложение необходимо включить наиболее распространенные растровые алгоритмы для непрерывных объектов:

- алгоритм Брезенхема для построения отрезка,
- алгоритм цифрового дифференциального анализатора (далее ЦДА),
- алгоритм Ву,
- алгоритм средней точки для эллипса,
- алгоритм средней точки для окружности.

А также:

- теоретический раздел,
- блок-схемы алгоритмов и исходные коды алгоритмов.

Основные группы пользователей разрабатываемой системы - преподаватели, студенты, другие люди, заинтересованные в данной предметной области. Помимо требований, изложенных выше, можно обозначить для группы преподавателей следующие требования:

- анимация построения вышеперечисленных алгоритмов,
- настройка визуализации - скорость, разрешение графического поля,
- вывод кода алгоритма совместно с его выполнением.

У остальных групп пользователей главное требование - это наглядность и воспринимаемость информации.

1.3 Существующие решения

Поиск программных комплексов для демонстрации всех требуемых алгоритмов оказался безрезультатным. Есть лишь примеры отдельных алгоритмов либо совокупность 2-х и очень редко 3-х примеров, написанные на различных языках для разных операционных систем. Данные алгоритмы можно использовать для обучения, но только после проверки их файлов на безопасность, работоспособность и корректность алгоритма. К тому же, многие из них не наглядны.

1.4 Достоинства и недостатки существующих решений

В таблице 1.1 приведены наиболее характерные найденные в Интернете решения, их достоинства и недостатки. На рисунках 1.1-1.5 представлены скриншоты соответствующих примеров из таблицы 1.1.

Таблица 1.1 – Преимущества и недостатки имеющихся решений

Наименование	Преимущества	Недостатки
1. Визуализация алгоритма Брезенхема (+ часть окружности) (C++ и OpenGL) [1]	Компактность	- Отсутствие настроек визуализации; - Отсутствие теории/пояснений.
2. Визуализация алгоритма By [2]		- gif-формат, следовательно, нет настроек; - Отсутствие теории/пояснений.
3. Визуализация алгоритма ЦДА (+ алгоритм Брезенхема) (C++, QPainter) [3]		- Отсутствуют настройки визуализации; - Отсутствие теории/пояснений.
4. Визуализация растрового представления окружности (C#, Drawing) [4]		- Отсутствие настроек визуализации; - Отсутствие теории/пояснений;

Таблица 1.1 – Преимущества и недостатки имеющихся решений (продолжение)

Наименование	Преимущества	Недостатки
4. Визуализация растрового представления окружности (C#, Drawing) [4]	Компактность	- Очень высокая скорость построения, при которой человеческий глаз воспринимает анимацию как статическую картинку.
5. Визуализация растрового представления эллипса (C#, Drawing) [5]		- Отсутствие теории/пояснений. - Нельзя уменьшить/ увеличить изображение для детального рассмотрения.

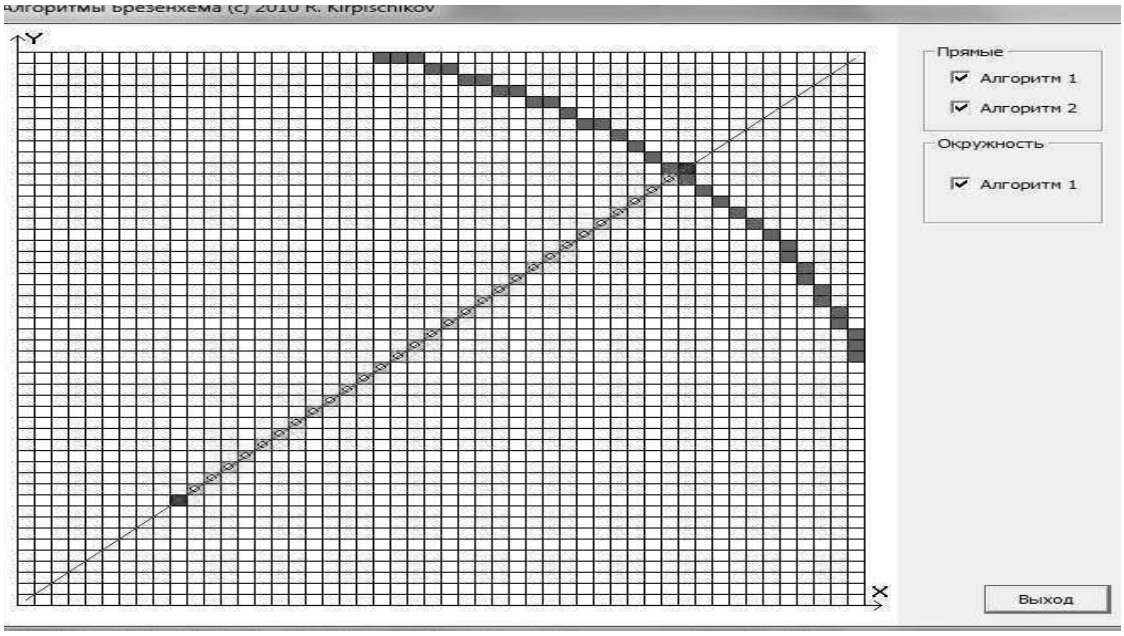

Рисунок 1.1 - Визуализация алгоритма Брезенхема для линии [1]

Рисунок 1.2 - Визуализация алгоритма Ву [2]

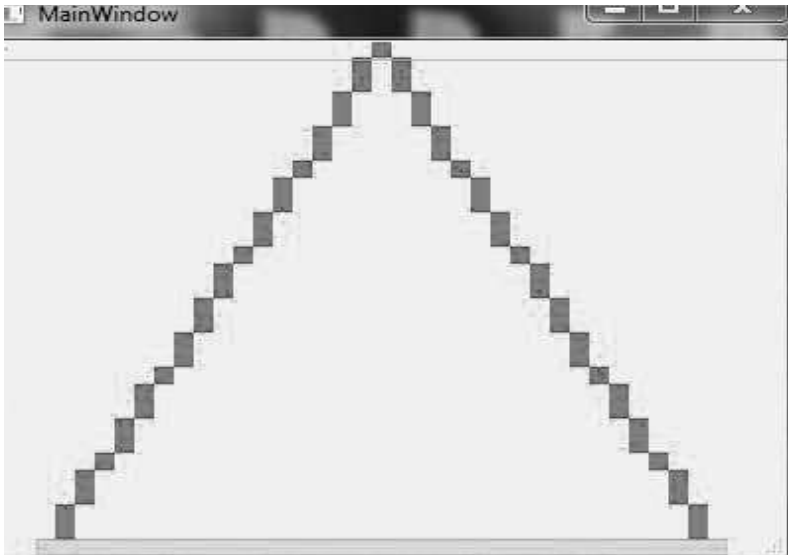

Рисунок 1.3 -Визуализация алгоритма ЦДА [3]

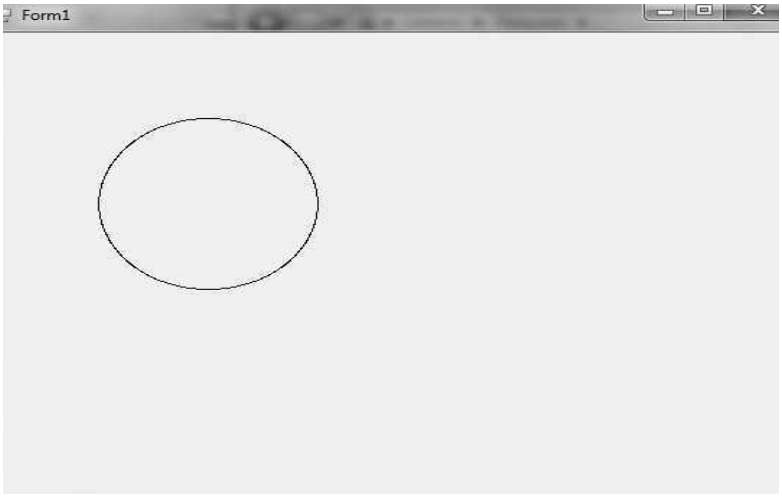

Рисунок 1.4 - Визуализация окружности, построенной по Брезенхему [4]

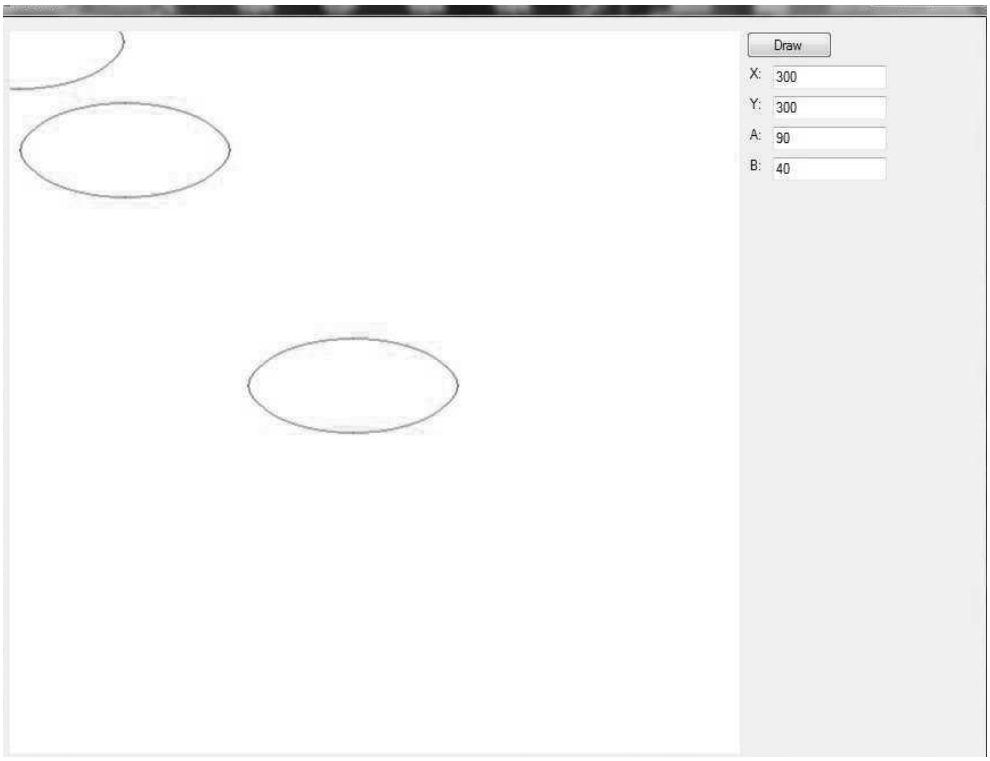

Рисунок 1.5 - Визуализация эллипса, построенного по Брезенхему [5]

1.5 Сравнение отечественных и зарубежных решений и технологий

В предыдущем пункте были описаны найденные решения на просторе рунета. Для формирования более полной картины об аналогах приложения демонстрационного комплекса программ для изучения алгоритмов растеризации произведен поиск похожих решений за рубежом. Результаты поиска сведены в таблицу 1.2.

Таблица 1.2 – Зарубежные решения

Источник	Страна	Алгоритмы	Доп. возможности	Инструменты
Частный сайт о программ-ии, который принадлежит	Германия	- ЦДА - Брезенхем - средняя точка для	- масштабирование - указание позиции курсора - алгоритм	- язык Java - Java graphics

Таблица 1.2 – Зарубежные решения (продолжение 1)

Источник	Страна	Алгоритмы	Доп. возможности	Инструменты
инженеру по разработке ПО в автомобильной промышленности [6].		окружности - средняя точка для эллипса	Брезенхема с целыми и дробными числами	
Публикация статьи «Computer Graphics with OpenGL» от доктора философии компьютерных наук Салхи М. Алзахрани [7].	Малайзия	- ЦДА	отсутствуют	консольное приложение на C++, OpenGL и GLUT
Международный журнал исследований в области науки и технологий (International Journal of Research in Science And Technology) [8].	Индия	- Брезенхем для отрезка	- задание координат отрезка - вычисление промежуточных результатов	язык и средства графики ActionScript

Таблица 1.2 – Зарубежные решения (продолжение 2)

Источник	Страна	Алгоритмы	Доп. возможности	Инструменты
Разработчик архитектуры ПО в Инновационном центре [9].	Индия	- By	- рядом с отрисовкой располагается тот же объект, но без сглаживания - переносимость на другие платформы	MFC (Microsoft Foundation Classes) приложение на C++.

Ниже на рисунках 1.6 - 1.9 приведены скриншоты приложений, найденных зарубежных аналогов.

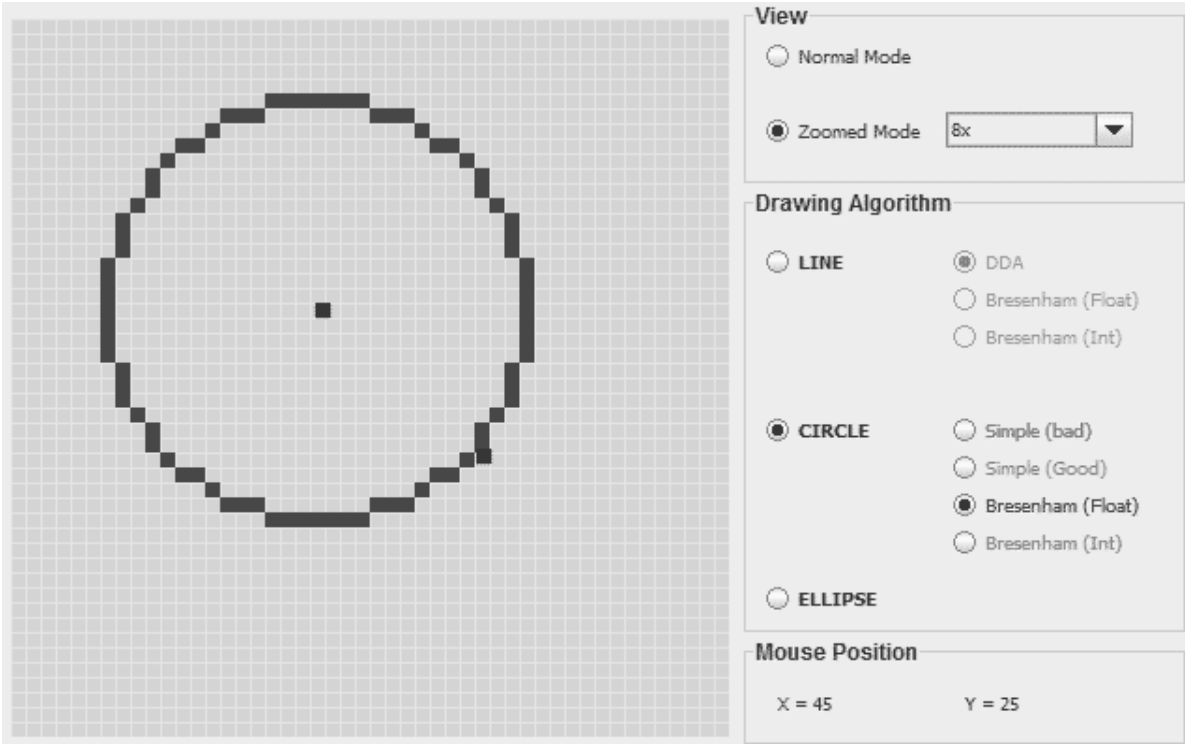

Рисунок 1.6 - Решение из Германии

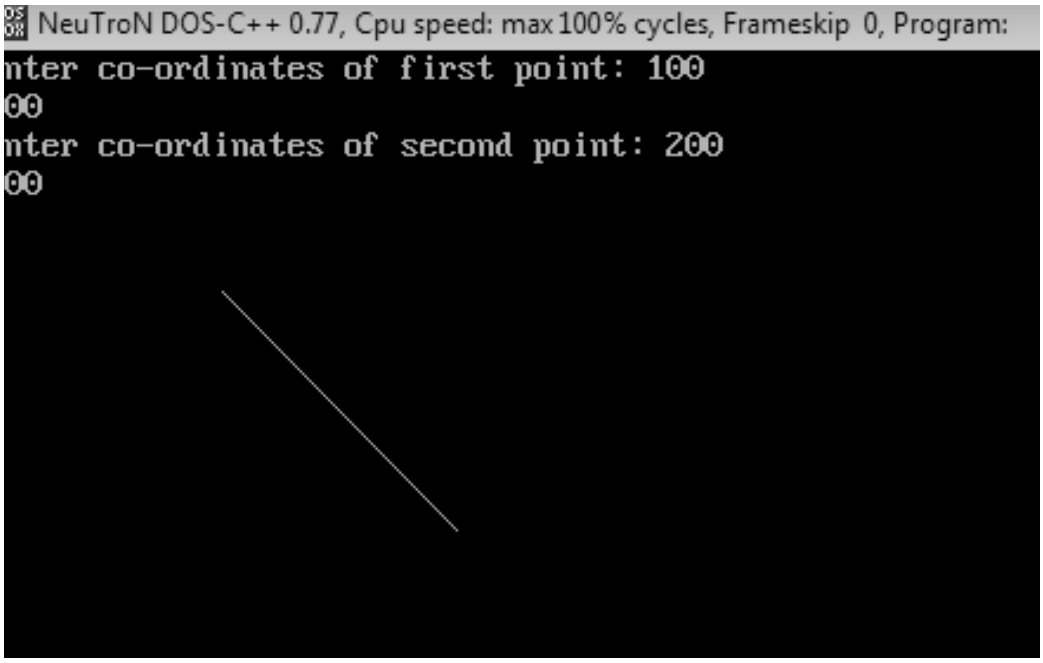

Рисунок 1.7 - Решение из Малайзии

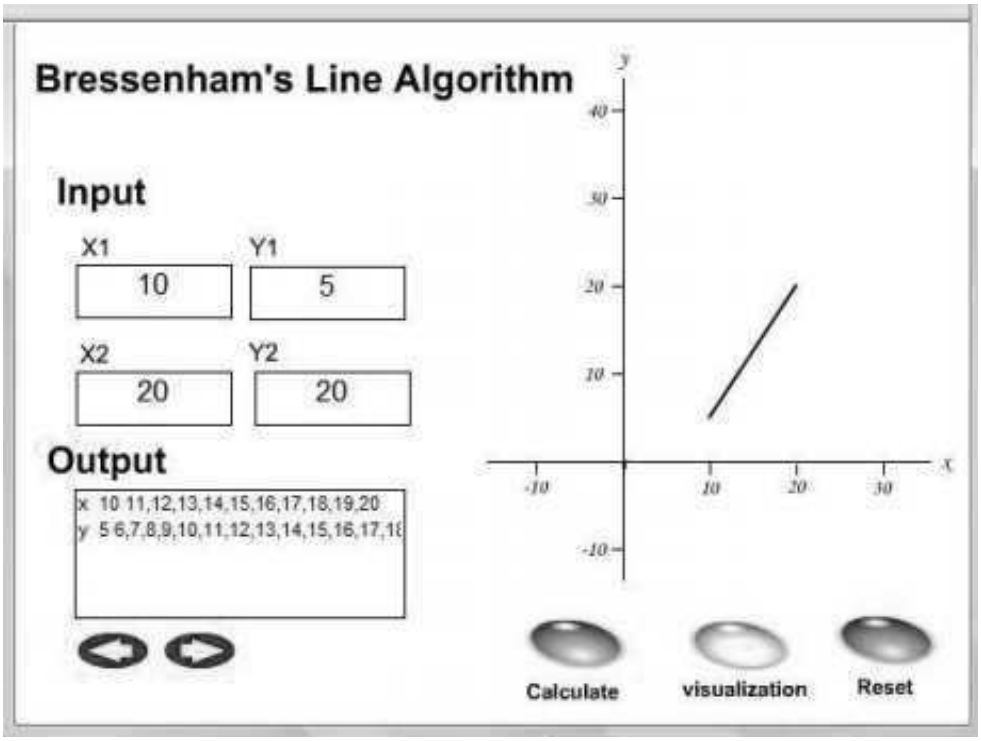

Рисунок 1.8 - Решение из Индии (алгоритм Брезенхема)

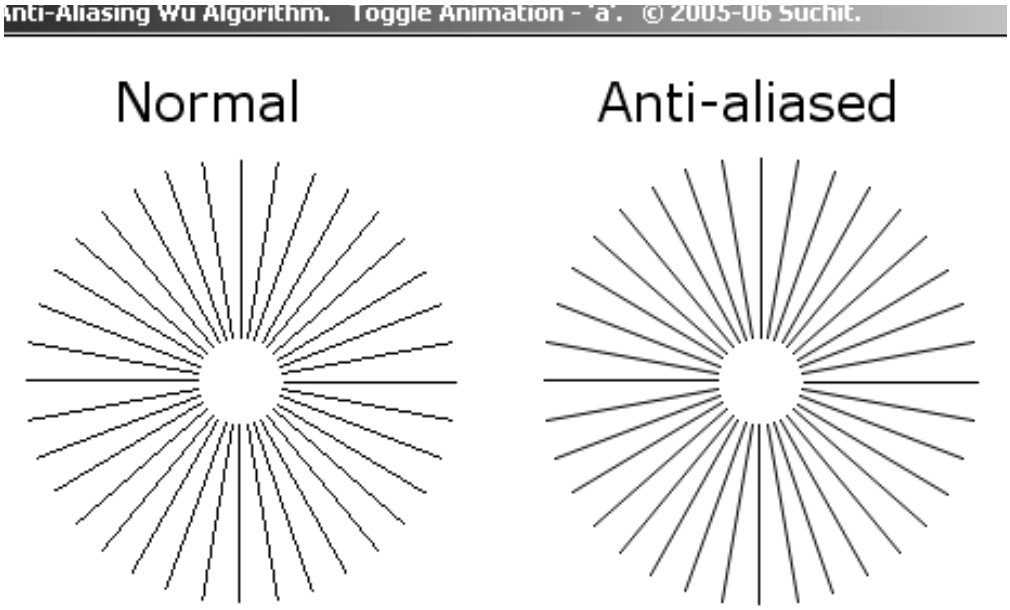

Рисунок 1.9 - Решение из Индии (алгоритм Ву)

Инструменты разработки для отечественных решений из таблицы 1.1 и зарубежных из таблицы 1.2 отличаются многообразием. Следовательно, разработка подобного приложения не привязана к конкретному инструментарию и технологиям. Разработчик выбирает их на свое усмотрение либо исходя из установленных требований (например, кроссплатформенность).

1.6 Разработка нового решения

Для решения проблем, обозначенных в п. 1.1, следует разработать демонстрационный комплекс программ для растровых алгоритмов представления графической информации. Он повысит эффективность изучения растровых алгоритмов, удобство проведения занятий по компьютерной графике, упростит демонстрацию примеров алгоритмов.

Учитывая недостатки имеющихся решений (п. 1.3), следует включить в демонстрационный комплекс программ (далее ДКП):

- все алгоритмы, которые указаны в требованиях п. 1.2;

- теоретический раздел, включающий учебный материал по каждому алгоритму и общую информацию о растровой графике;
- настройки визуализации;
- в дополнение к теоретическому материалу - блок-схемы алгоритмов и исходные коды алгоритмов.

ДКП будет правильным разработать для наиболее распространенной на данный момент операционной системы Windows 7, 32-разрядная, с пакетом обновления SP1 или выше [10].

Для теоретического раздела материал следует взять из учебно-методического пособия автора, к. т.н. Ярош Е.С. Методы и средства представления графической информации, что является надежным достоверным источником. Блок-схемы построить по вышеупомянутому пособию.

Для решения поставленной задачи с точки зрения IT-специалиста необходимо спроектировать программную систему, для реализации выбрать:

- язык программирования;
- графический API;
- технологию программирования.

Правильный выбор инструментов разработки обеспечит баланс между удобством разработки и производительностью готового продукта. После выбора инструментов следуют этапы:

- разработка программного кода исходных алгоритмов для визуализации;
- настройка графического API для воспроизведения визуализации;
- создание теоретического раздела;
- создание блок-схем;
- вывод кода алгоритма;
- создание пользовательского меню;
- создание верхнего меню MenuStrip с настройками;
- обеспечение эргономичности.

1.7 Достоинства и недостатки нового решения

Преимущества ДКП перед найденными наработками из пункта 1.3:

- алгоритмы, перечисленные в пункте 1.4, располагаются в одном месте, что обеспечивает удобство и исключает затраты на поиск;
- в дополнение к заявленным алгоритмам добавлены: 1) алгоритм построения прямой по уравнению, 2) выбор 4- либо 8-связного варианта для алгоритма Брезенхема и алгоритма построения по уравнению (вид 4-связности и 8-связности приведен на рисунке 1.10);
- гарантия работоспособности и безопасности;
- настройка визуализации - выбор разрешения координатной сетки, ввод исходных данных;
- наличие раздела с теоретическим материалом;
- наглядные блок-схемы;
- возможность просмотра основной части исходного кода, которая выводится в ходе выполнения демонстрируемого алгоритма.

Из достоинств ДКП вытекает и его недостаток - требуется больше памяти по сравнению с отдельно взятыми примерами, следовательно, ДКП менее компактен.

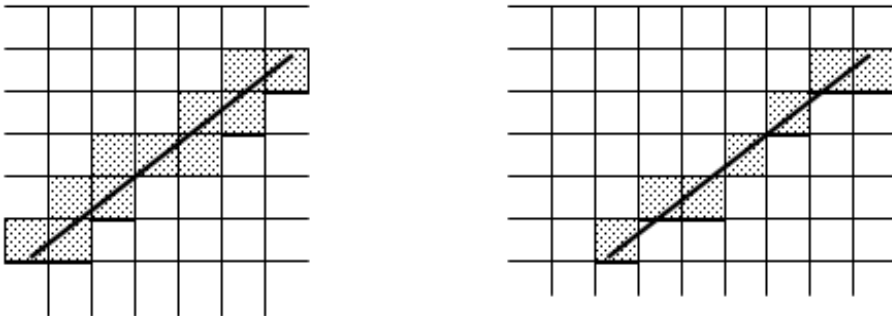

Рисунок 1.10 - 4-связная и 8-связная развертка отрезка

2 Планирование

2.1 Платформа

Для данного проекта выбрана платформа Windows 7x32, исходя из требований заказчика. В настоящий момент Windows является самой распространенной операционной системой [10], к тому же в большинстве учебных заведений на компьютерах установлена именно она. Выбрана 32 - битная архитектура Windows ввиду того, что программы, написанные для 64-разрядной системы, работать на 32-х разрядной не смогут, но ПО под x32 работает как на 32, так и на 64-битной.

2.2 Возможные варианты реализации

Возможные средства реализации ДКП по большей части представляют собой совокупность различных языков программирования с различными графическими API. Так как ДПК - это приложение для Windows, то из языков программирования на данный момент самые распространенные для подобных проектов это C, C++, C#, Java. Для работы с графикой в приоритете такие графические API, как OpenGL, DirectX [11], Vulkan [11]. В зависимости от выбранного языка, программирование может быть структурным, либо объектно-ориентированным. Может использоваться технология многопоточного программирования.

2.3 Обоснование среды разработки

Наиболее рационально использовать среду разработки Visual Studio 2012 (VS). Это бесплатный продукт компании Microsoft, который предоставляет мощный и удобный инструментарий для разработки. С его помощью можно

разрабатывать помимо консольных приложений, приложения с графическим интерфейсом, что необходимо для ДКП. VS не привязана к определенному языку программирования, что позволяет выбрать необходимый язык при создании проекта. Редактор форм VS упрощает создание графического интерфейса. В VS есть возможность подключения сторонних библиотек для расширения функционала. Отладчик позволяет найти ошибки в исходном коде программы, посмотреть, как изменяются значения необходимых переменных, тем самым предоставляя разработчику информацию для контроля правильности вычислений.

2.4 Выбор и обоснование средств разработки

Разрабатываемый продукт должен являться desktop-приложением (запускается в окне на рабочем столе) для Windows 7 SP1 x32, поэтому подходящим является язык программирования C#. Если бы требовалась кроссплатформенность приложения, то выбор был бы между C++ и Java. Преимущество C# перед C – объектно-ориентированность, что удобнее для разработки объемных программ. Также C# ориентирован на безопасность кода по сравнению с C и C++. Из-за простоты языка C# разработка на нем идет быстрее, чем на C++. Из недостатков можно отметить относительно невысокую производительность (медленнее, чем язык C, но сравним с Java). Так как в ДПК не требуется очень быстрых вычислений, то недостаток не является значимым.

Выбранный интерфейс программирования приложения – Windows Form. Он упрощает доступ к элементам интерфейса Windows за счет создания обёртки для существующего Win32 API в управляемом коде, отвечает за графический интерфейс пользователя. Windows Form входит в .NET Framework, поэтому на компьютерах с операционной системой более ранней, чем Windows 7 SP1, нужно перед использованием ДКП устанавливать .NET Framework 3.5.

Для работы с графикой выбрана библиотека OpenGL под враппером OpenTK для совместимости с C#. Выбор основан на том, что:

- работа в ДКП идет лишь с графическими примитивами, не требующими современных эффектов,
- код, написанный с использованием OpenGL, выглядит нагляднее, чем с использованием DirectX или Vulcan, что проще для восприятия пользователям ДКП.

Для подтверждения информации о наглядности OpenGL ниже представлены листинги 2.1 - 2.2 для рисования треугольника с вершинами разных цветов (процесс инициализации пропущен) с использованием OpenGL, DirectX, а также на листинге 2.3 - процесс передачи вершин треугольника в GPU (graphics processing unit - графический процессор) для последующей отрисовки (из-за сложности и объема программного кода процесс отрисовки не приводится) с использованием Vulkan.

Листинг 2.1 - Отрисовка треугольника на OpenGL

```
glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_TRIANGLES);
glColor3d(1,0,0);
glVertex3d(1,2,3);
glColor3d(0,1,0);
glVertex3d(4,5,6);
glColor3d(0,0,1);
glVertex3d(7,8,9);
glEnd();
```

Листинг 2.2 - Отрисовка треугольника на DirectX

```
struct CUSTOMVERTEX { FLOAT x, y, z, rhw; DWORD color;};
CUSTOMVERTEX g_Vertices[] =
{ {1, 2, 3, 1, 0xffff0000},
{4, 5, 6, 1, 0xff00ff00},
{7, 8, 9, 1, 0xff0000ff},};
LPDIRECT3DVERTEXBUFFER8 p_VertexBuffer = NULL;
```

```
d3d_Device->CreateVertexBuffer (3*sizeof(CUSTOMVERTEX),
0, D3DFVF_XYZRHW | D3DFVF_DIFFUSE, D3DPOOL_DEFAULT,
&p_VertexBuffer);
VOID* pVertices;
p_VertexBuffer->Lock (0, sizeof(g_Vertices), (BYTE**)&pVertices, 0);
memcpy (pVertices, g_Vertices, sizeof(g_Vertices));
p_VertexBuffer->Unlock();
d3d_Device->BeginScene ();
d3d_Device->SetVertexShader (D3DFVF_CUSTOMVERTEX);
d3d_Device->SetStreamSource (0, p_VertexBuffer, sizeof(CUSTOMVERTEX));
d3d_Device->DrawPrimitive (D3DFVF_XYZRHW | D3DFVF_DIFFUSE, 0, 1);
d3d_Device->EndScene ();
```

Листинг 2.3 - Передача вершин в GPU для последующей отрисовки на Vulkan

```
VkBuffervk_buffer;
{std::vector<float>source = {
                -0.5f, +0.5f,
                 +0.5f, +0.5f,
               +0.0f, -0.5f  };
VkBufferCreateInfovk_bufferCreateInfo;
{vk_bufferCreateInfo.sType=VkStructureType::VK_STRUCTURE_TYPE_BUFFER_C
REATE_INFO;
vk_bufferCreateInfo.pNext = nullptr;
vk_bufferCreateInfo.flags = 0;
vk_bufferCreateInfo.size = sizeof(float)* source.size();
vk_bufferCreateInfo.usage=
VkBufferUsageFlagBits::VK_BUFFER_USAGE_VERTEX_BUFFER_BI;
vk_bufferCreateInfo.sharingMode =
VkSharingMode::VK_SHARING_MODE_EXCLUSIVE;
 vk_bufferCreateInfo.queueFamilyIndexCount = 0;
```

```
vk_bufferCreateInfo.pQueueFamilyIndices = nullptr;  }
 if(vkCreateBuffer(vk_device, &vk_bufferCreateInfo, nullptr, &vk_buffer) !=
VkResult::VK_SUCCESS){
throwstd::exception("failed to create buffer");}
VkDeviceMemoryvk_deviceMemory; {
 VkMemoryRequirementsvk_memoryRequirements;
 vkGetBufferMemoryRequirements(vk_device, vk_buffer, &vk_memoryRequirements);
VkMemoryAllocateInfovk_memoryAllocateInfo; {
vk_memoryAllocateInfo.sType=
VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
vk_memoryAllocateInfo.pNext = nullptr;
vk_memoryAllocateInfo.allocationSize = vk_memoryRequirements.size;
vk_memoryAllocateInfo.memoryTypeIndex = [&]()   {
for(size_ti = 0;  i<vk_physicalDeviceMemoryProperties.memoryTypeCount; ++i){
auto bit = ((uint32_t)1 <<i);
if((vk_memoryRequirements.memoryTypeBits& bit) != 0) {
if((vk_physicalDeviceMemoryProperties.memoryTypes[i].propertyFlags
&VkMemoryPropertyFlagBits::VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT)
!= 0)
{ returni;}}}
 throwstd::exception("failed to get correct memory type");  }();}
if(vkAllocateMemory(vk_device, &vk_memoryAllocateInfo, nullptr,
&vk_deviceMemory) != VkResult::VK_SUCCESS)
{throwstd::exception("failed to allocate device memory");}
 void* data;
 if(vkMapMemory(vk_device, vk_deviceMemory, 0, sizeof(float)* source.size(),  0,
&data) != VkResult::VK_SUCCESS)
{  throwstd::exception("failed to map device memory"); }
 memcpy(data, source.data(), sizeof(float)* source.size());
```

```
vkUnmapMemory(vk_device, vk_deviceMemory); }
if(vkBindBufferMemory(vk_device, vk_buffer, vk_deviceMemory, 0) !=
VkResult::VK_SUCCESS)
{throwstd::exception("failed to bind buffer memory"); } }
```

2.5 Выбор и обоснование технологии программирования

В программном комплексе используется графика. Построение графических примитивов – ресурсозатратная операция, которая занимает весь основной поток приложения. Из-за этого невозможно параллельно выполнять другие задачи: вывод выполняющегося кода параллельно с графикой невозможен. Ввиду этого выбрана многопоточная технология программирования.

В языке C# предусмотрено несколько инструментов для организации многопоточности. На сегодняшний день основным подходом для разработки многопоточных приложений на C# является TPL (Task Parallel Library) – Библиотека распараллеливания задач [12]. TPL поддерживается начиная с .NET Framework 4. Данная платформа не входит в стандартный комплект программ и компонентов Windows 7, поэтому для многопоточности следует использовать средство .NET Framework 3.5 – класс Thread, на котором основывается вся система многопоточной обработки. Функция рисования графики будет вынесена в отдельный поток приложения.

2.6 Диаграммы UML

UML – это унифицированный язык моделирования, позволяющий строить модели программных систем. Он не зависит от выбранной методологии разработки ПО [13]. Исходя из вида диаграммы UML, модель системы может описывать функционал системы и ее взаимодействие с пользователями, архитектуру приложения; разработчик может писать каркасный код

проектируемого приложения, используя диаграмму классов UML. Также UML облегчает обмен информацией в проектной группе и между заказчиком и исполнителем.

Для отражения предоставляемого функционала системы для разных групп пользователей полезно построить диаграмму прецедентов (диаграмму вариантов использования). Основные группы пользователей разрабатываемого проекта:

- преподаватели компьютерной графики - могут использовать ДКП для проведения лекций/практических занятий по темам растровых алгоритмов;
- студенты, которые изучают на компьютерной графике растровые алгоритмы;
- другие люди, заинтересованные в данной предметной области.

В ДКП функционал для перечисленных выше групп пользователей одинаков, поэтому в диаграмме прецедентов достаточно двух акторов - непосредственно пользователь программы и графический API. На диаграмме использования (см. рисунок 2.1) отражены сценарии работы с системой, которые приводят к желаемому пользователем результату (АГИ - алгоритмы графической информации).

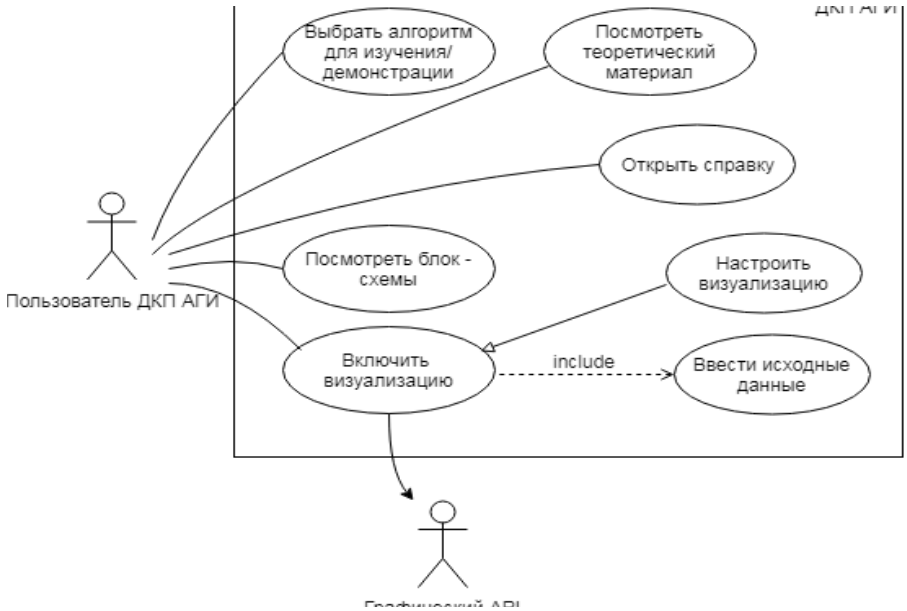

Рисунок 2.1 – Диаграмма использования ДКП (Draw.io)

Для описания классов, методов, атрибутов системы и взаимосвязи между ними на рисунке 2.2 построена диаграмма классов UML.

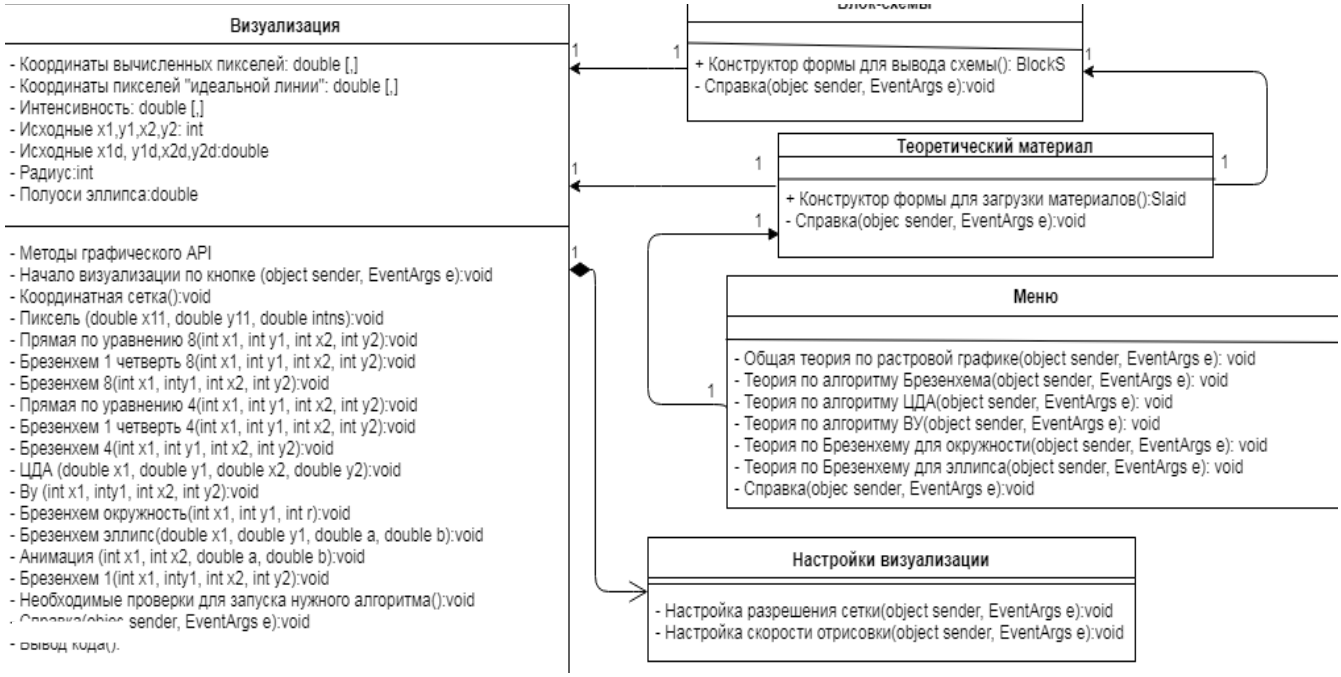

Рисунок 2.2 – Диаграмма классов ДКП (Draw.io)

Взаимосвязи объектов классов на диаграмме:

- ассоциация - объекты одной сущности связаны с объектами другой сущности таким образом, что можно перемещаться от объектов одного класса к другому; классы «Меню» и «Теоретический материал», «Теоретический материал» и «Блок - схемы», «Теоретический материал» и «Визуализация», «Блок-схемы» и «Визуализация;
- композиция - жёсткая зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся в нем классов. Если класс-контейнер будет уничтожен, то всё его содержимое будет также уничтожено; классы «Настройка визуализации» и «Визуализация»; если закрыть визуализатор, то нельзя будет увидеть настройки визуализации.

3 Разработка архитектуры программного комплекса

Теоретический раздел, раздел с блок-схемами алгоритмов и визуализатор - основные модули приложения. Каждый из них выполнен в отдельном окне Windows. Из одного модуля можно перейти в любой другой. Также визуализатор содержит модуль настройки визуализации. Для функционирования графики в визуализатор входят следующие внешние библиотеки: «Tao framework»- для GLUT-функций, «OpenTK» - для OpenGL 4.0 функций. Диаграмма компонентов представлена на рисунке 3.1.

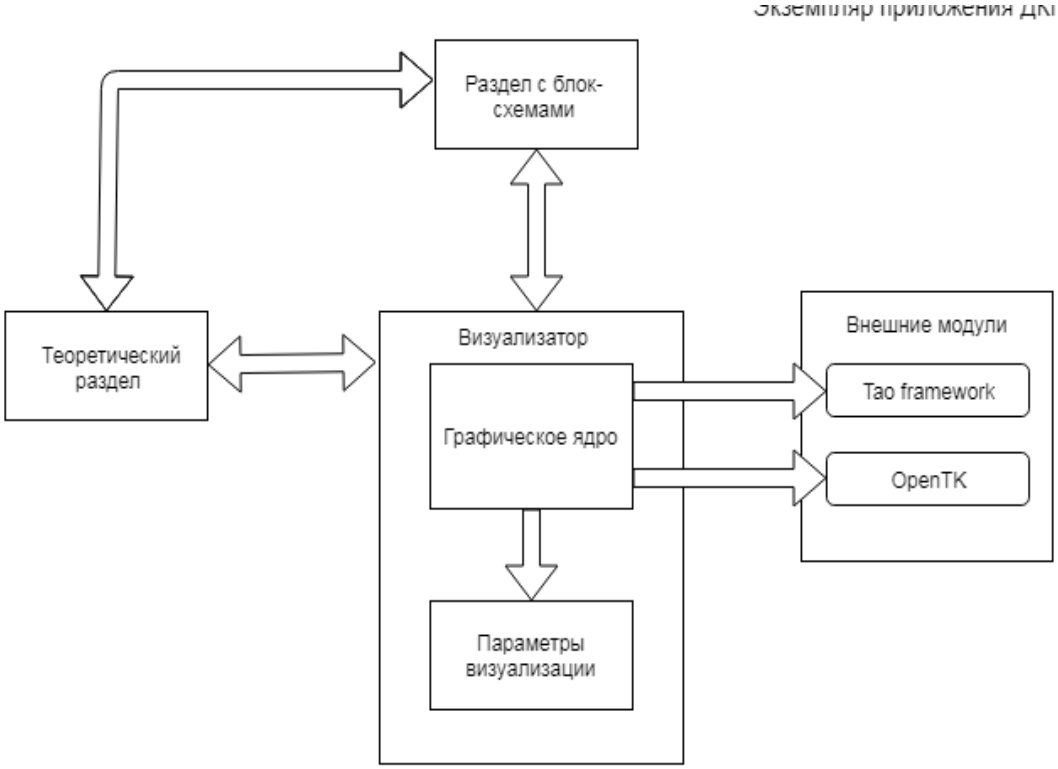

Рисунок 3.1 - Диаграмма компонентов (Draw.io)

OpenGL Utility Toolkit (GLUT) - библиотека утилит для приложений под OpenGL, которая в основном отвечает за системный уровень операций ввода-вывода при работе с операционной системой [14]. Часто использующиеся функции GLUT: создание окна, управление окном, мониторинг ввода с клавиатуры и событий мыши. В данном проекте GLUT необходима для создания

ортографической проекции для двумерного моделирования. Из-за некоторых проблем в файлах OpenTK использовать его GLUT - функции не удалось.

Tao framework является «обёрткой» для OpenGL 2.1.0 [15]. Из-за устаревания Tao framework для рисования было решено использовать более новый и стабильный OpenTK.

4 РАЗРАБОТКА ОСНОВНЫХ ТЕХНИЧЕСКИХ РЕШЕНИЙ

4.1 Растровые алгоритмы

Все растровые алгоритмы в ДКП представлены в 8-связном виде. Для алгоритма отрезка по Брезенхему и алгоритма построения отрезка по уравнению прямой есть варианты 4-связности.

4.1.1 Построение прямой по ее уравнению

Для построения необходимы координаты начала и конца отрезка *A (x1,y1)* и *B (x2,y2)*. Уравнение прямой приведено в формуле (1).

$$y = k * x + b, \quad (1)$$

где *y* - ордината точки прямой;

k - угловой коэффициент прямой;

x - абсцисса точки прямой;

b - свободный член прямой.

Нужно выразить из уравнения коэффициент *b,* подставить его в уравнение прямой и привести подобные слагаемые. Для случая $|k| \leq 1$ уравнение отрезка после преобразований, описанных выше, имеет вид, показанный в формуле (2).

$$y = k * (x - x1) + y1, \quad (2)$$

где $k = \frac{y2-y1}{x2-x1}$;

$x \in [x1,x2]$.

Шаг по *x* всегда 1, инициализируется *y*, ближайший к вычисленному (операция округления Round()) [16]. При $|k| > 1$ шаг будет по оси *y*, и он всегда меньше 1, в данной реализации выбран шаг = 0.35, таким образом, не пропускается ни одной точки. Шаг может быть как положительным, так и отрицательным, это зависит от значений координат начала и конца: если начальная координата (*x* или *y*) больше чем соответствующая конечная, то приращение по данной оси будет

отрицательным. Для правильного приращения введены переменные *sx* и *sy*. Для случая $|k| > 1$ координата *x* рассчитывается по формуле (3).

$$x = \frac{y-y1}{k} + x1, \quad (3)$$

где $y \in [y1,y2]$;

k аналогична случаю в формуле (2).

Такой алгоритм является 8-связным. Блок-схема для 8-связного отрезка представлена на рисунках 4.1, 4.2.

Для организации 4-связности в алгоритм выше был включен массив, хранящий предыдущие координаты. После вычисления очередные значения сравниваются с предыдущими - для 4-связной развертки отрезка лишь одна координата (*x* либо вычисленный *y*) может отличаться от предыдущих, если это условие не выполняется, то между прошлыми координатами (*x1* и *y1*) и новыми (*x2*, *y2*) вставляются еще одни значения (*x*, *y*) , где $x = x2$ и $y = y1$, если $|k| < 1$, либо $x = x1$ и $y = y2$ для остальных случаев. Блок-схема для 4-связного отрезка представлена на рисунках 4.3, 4.4, 4.5.

Недостаток алгоритма построения отрезка по уравнению прямой - вещественные вычисления на целочисленной решетке. Вещественные вычисления более медленны, чем целочисленные.

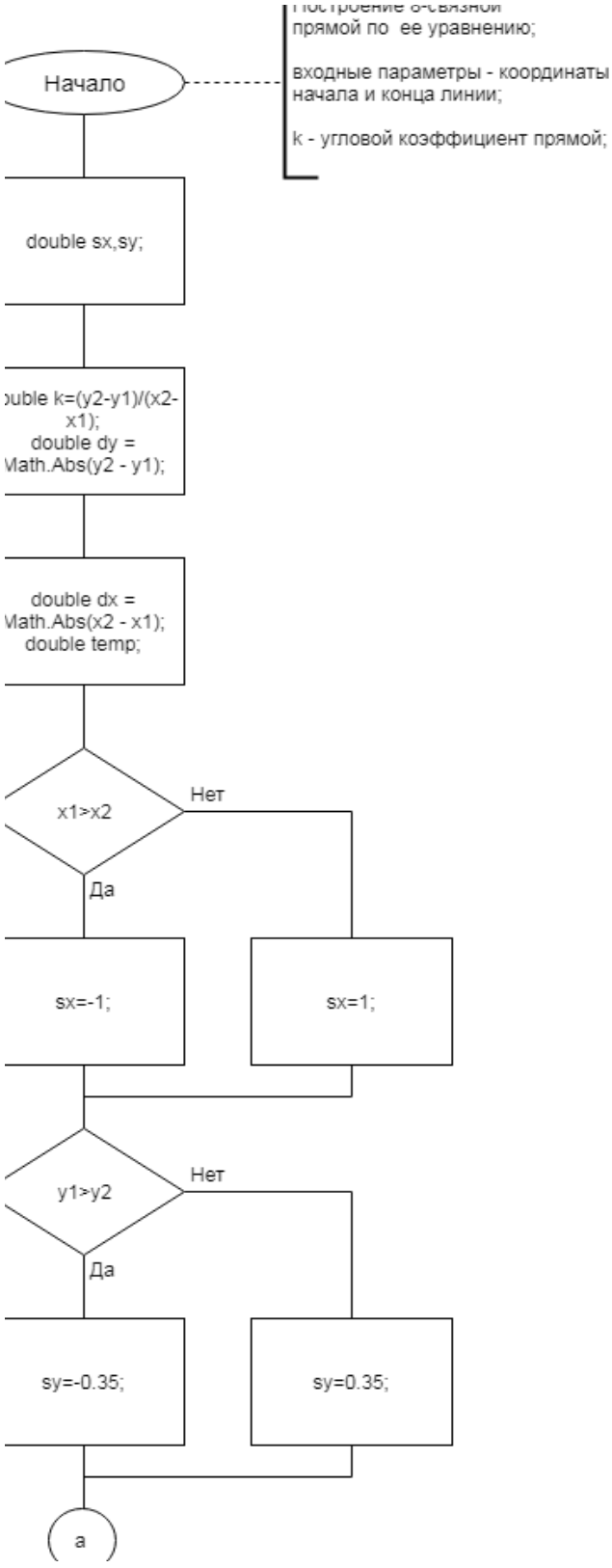

Рисунок 4.1 - Построение 8-связной прямой по уравнению (Draw.io)

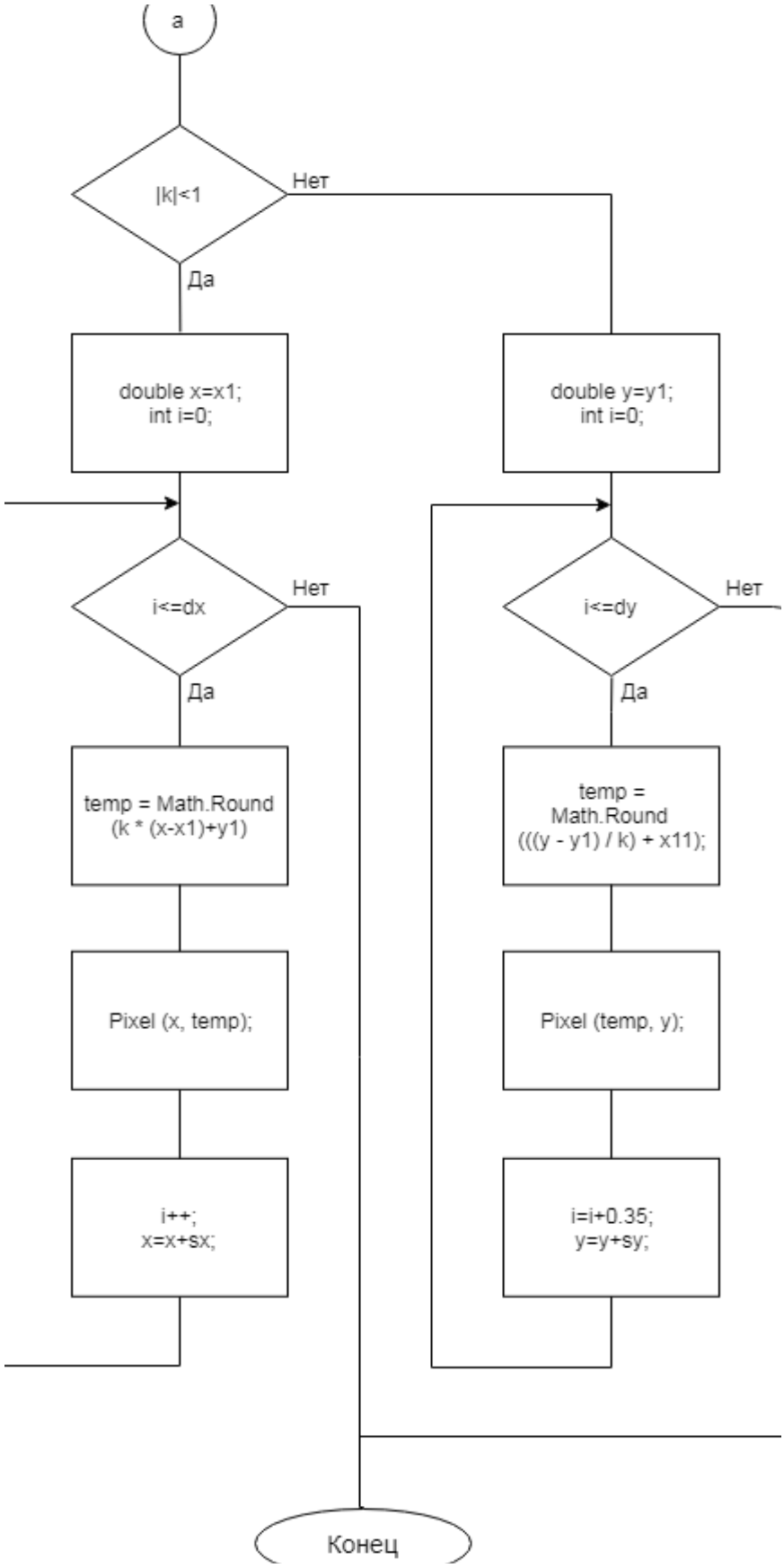

Рисунок 4.2 - Построение 8-связной прямой по уравнению (продолжение)

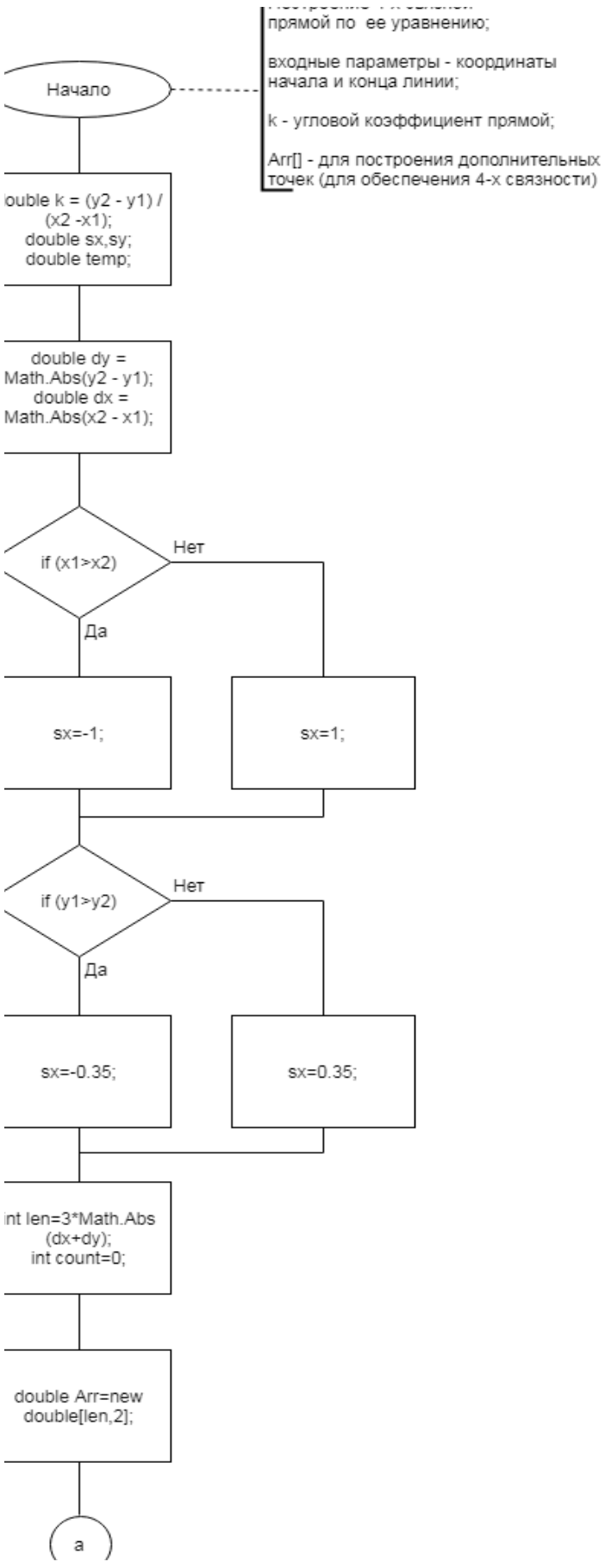

Рисунок 4.3 - Построение 4-связной прямой по уравнению (Draw.io)

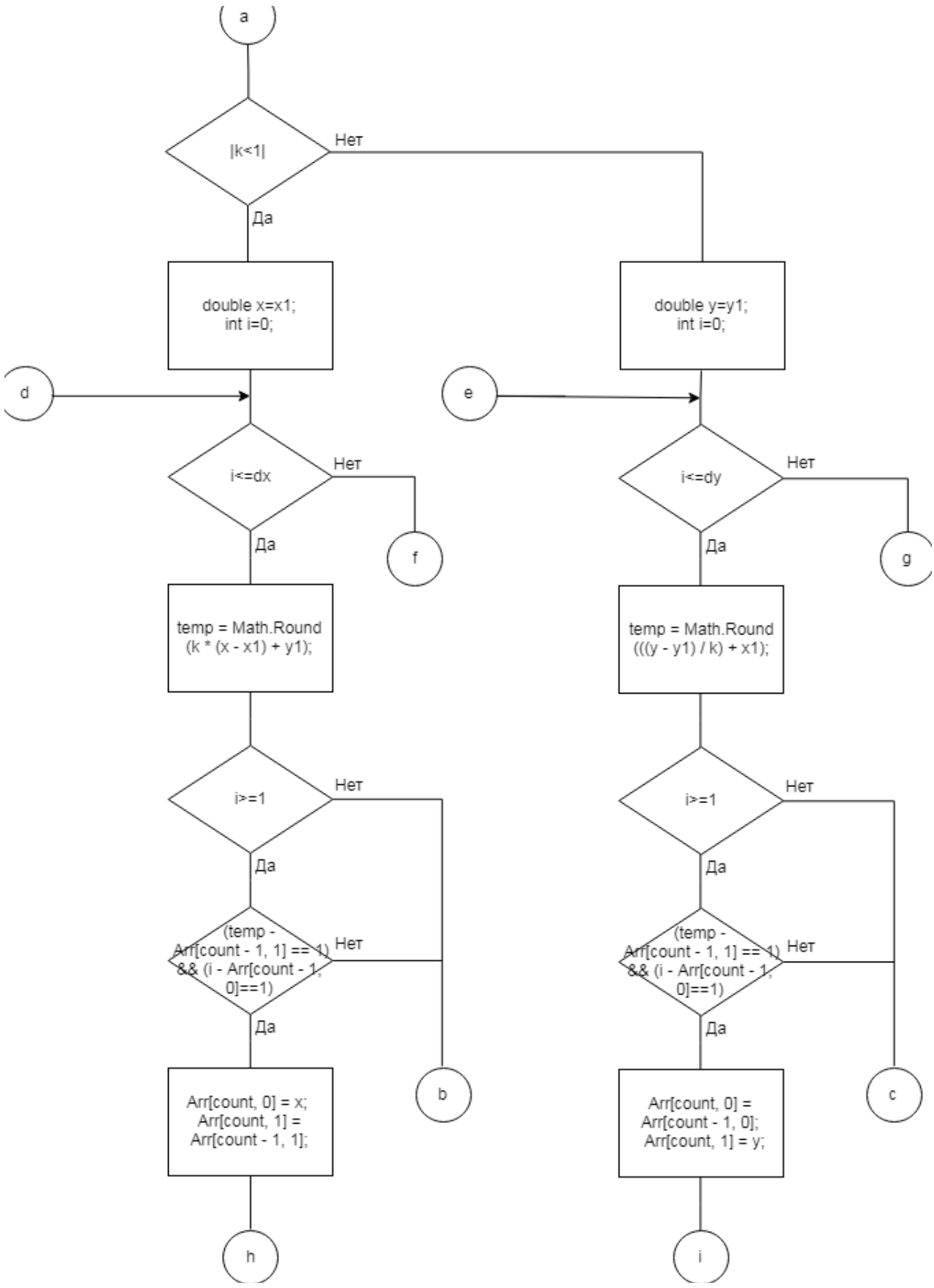

Рисунок 4.4 - Построение 4-связной прямой по уравнению (продолжение 1)

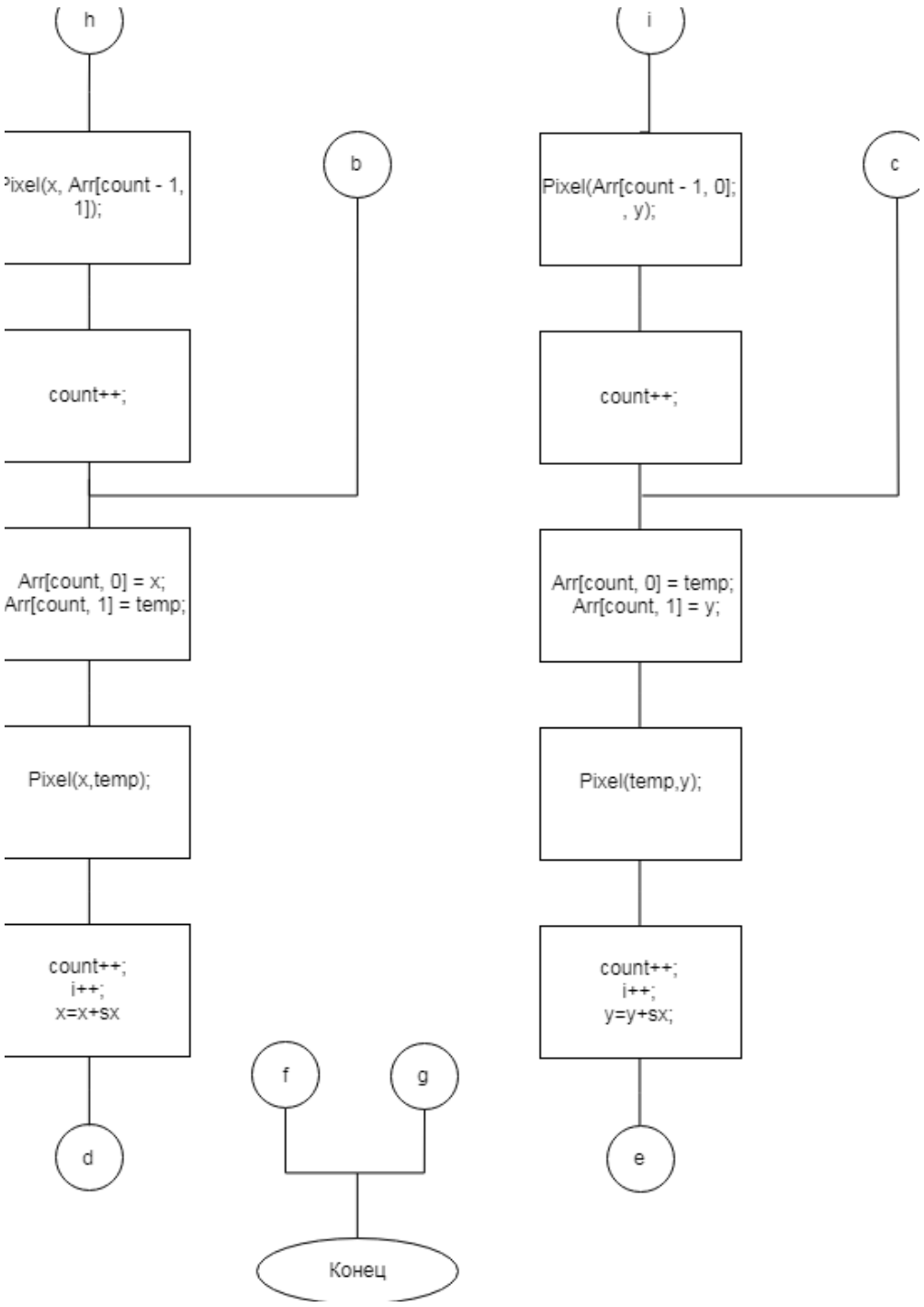

Рисунок 4.5 - Построение 4-связной прямой по уравнению (продолжение 2)

4.1.2 Алгоритм Брезенхема для отрезка

Алгоритм Брезенхема является целочисленным. Необходимы координаты начала и конца отрезка *A (x1,y1)* и *B (x2,y2).* Основная идея алгоритма: координаты предыдущего пикселя известны, для того, чтобы определить, какой пиксель закрашивать следующим, необходимо найти отклонение *d* точной позиции отрезка

от середины между двумя точками-кандидатами на закрашивание, значение d определяет, какой из двух пикселей будет выбран. Если $d > 0.5$, то при увеличении координаты x на 1 увеличивается и y на 1 [16].

Отклонение вычисляется по формуле (4).

$$d = \frac{y2-y1}{x2-x1}. \quad (4)$$

Для избавления от дроби и 0.5 и, следовательно, перехода к целочисленным расчетам необходимо отклонение d и приращение по y домножить на $2 * (x2 - -x1)$. Тогда отклонение d уже будет сравниваться с 0. Если $d > 0$, то инкрементируется и x, и y, а d увеличится согласно формуле (5).

$$d = d + 2 * ((y2 - y1) - (x2 - x1)), \quad (5)$$

иначе y не изменяется, x инкрементируется, а d меняется по формуле (6).

$$d = d + 2 * (y2 - y1). \quad (6)$$

В программной реализации операции умножения на 2 были заменены операцией арифметического сдвига влево (битовый сдвиг вместо умножения на число равное степени двойки работает всегда быстрее), что ускорило вычисления.

Для ДКП были написаны два варианта алгоритма Брезенхема и для 4-связной развертки, и для 8-связной. Один из вариантов - это упрощенный алгоритм Брезенхема. Он может построить отрезок только в первой четверти и для углового коэффициента, лежащего в диапазоне (0;1]. Для 8-связной развертки алгоритм стандартен.

Реализация 4-связного алгоритма имеет следующие особенности: на каждом шаге отклонение d сравнивается не с 0, а с разностью $dx = x2 - x1$, что определяет, когда увеличивать значение x, а когда y, таким образом, в каждой итерации увеличивается лишь одна координата, что и требует 4-связность. При отклонении $d < dx$ инкрементируется x, d увеличивается на $dy \ll 1$. При $d > dx$ увеличивается на 1 y, а d уменьшается на $2 * dx$, чтобы на следующем шаге

инкрементировать *x*. Блок-схема 8-связного упрощенного алгоритма представлена на рисунке 4.6, 4-связного - на рисунке 4.7.

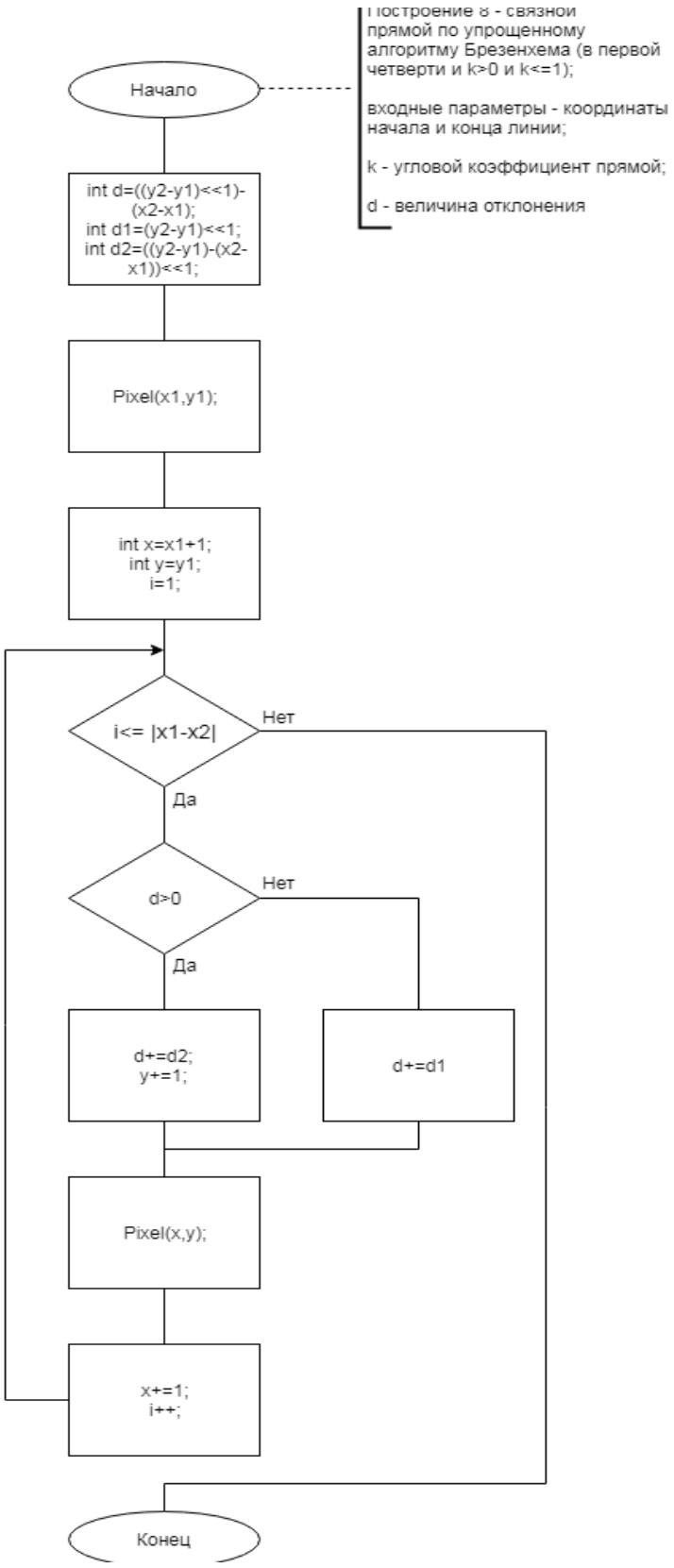

Рисунок 4.6 - Упрощенный 8-связный алгоритм Брезенхема (Draw.io)

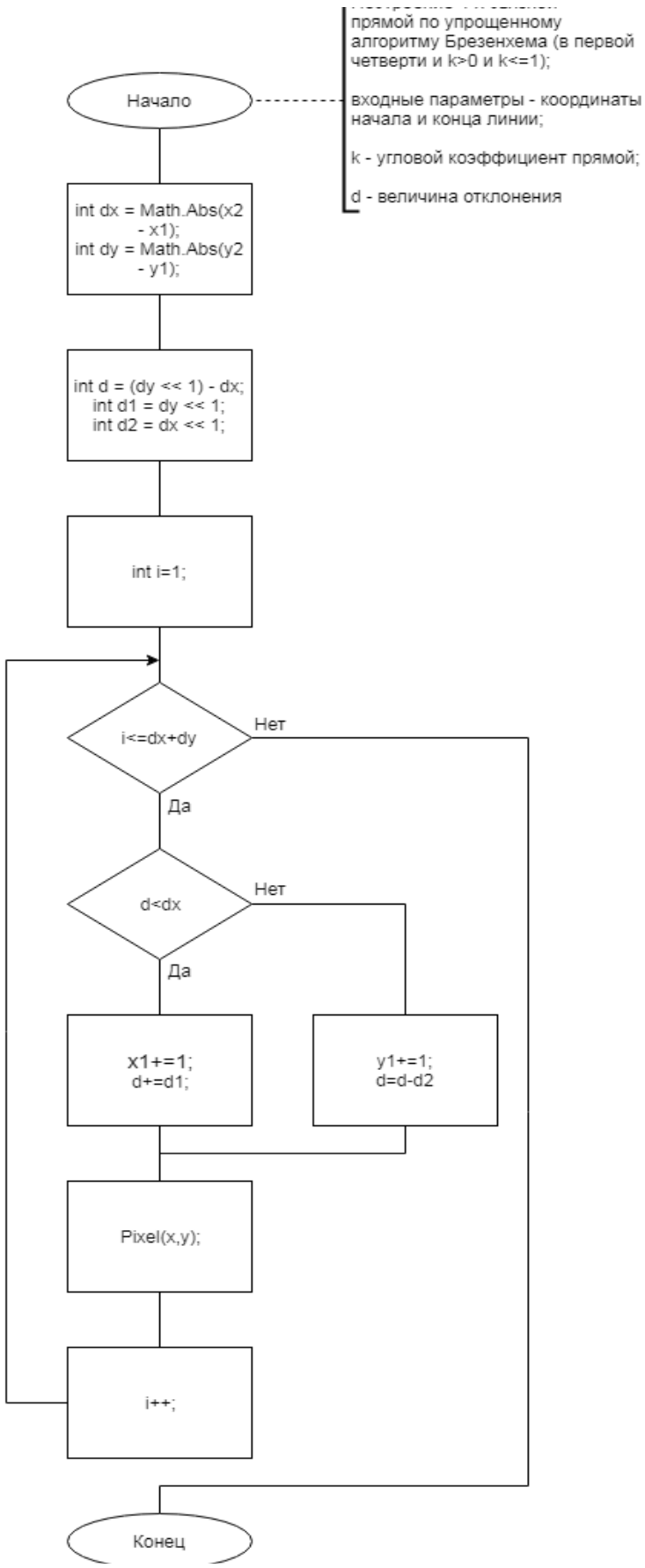

Рисунок 4.7 - Упрощенный 4-связный алгоритм Брезенхема (Draw.io)

Другой вариант алгоритма представляет полную реализацию алгоритма Брезенхема для отрезка - для любой четверти и для любого значения углового коэффициента прямой. Полный вариант алгоритма Брезенхема реализован и в 8-связной развертке, и в 4-связной. Для этого в упрощенные алгоритмы обоих случаев было добавлено:

- вместо инкрементирования специальные переменные, которые определяют сторону отсчета (-1 или +1); сторона отсчета определяется отношением переменных *x1* к *x2, y1* к *y2*, если начальное значение меньше конечного, то +1, иначе -1;
- алгоритм для случая, если угловой коэффициент не соответствует условию $|k| < 1$. В отличие от исходного алгоритма, в формулах для отклонения *d* и его приращения *x* и *y* изменены местами, при этом в 8-связной развертке в каждом шаге цикла вычисляется *y, а x* вычисляется лишь при соблюдении условия $d > 0$; в 4-связной - при $d < dx$ (значения *dx* и *dy* также меняются местами) изменяется *y*, при $d > dx$ изменяется *x*.

Блок-схемы полного алгоритма Брезенхема изображены на рисунках 4.8 - 4.9 для 8-связной развертки, на рисунках 4.10 - 4.11 для 4-связной развертки.

Алгоритм Брезенхема использует целочисленную арифметику и операции побитового сдвига, что значительно улучшает скорость работы по сравнению с построением отрезка по уравнению прямой.

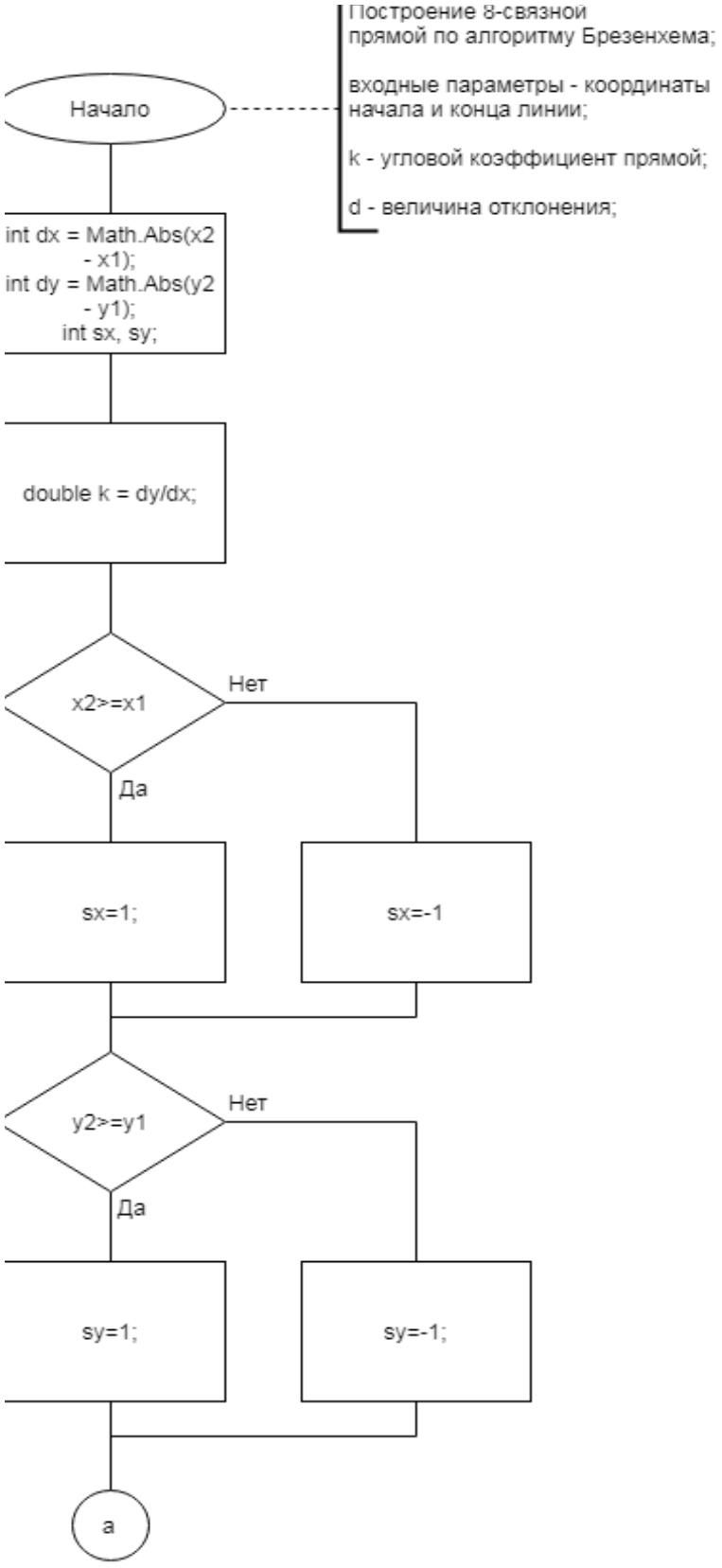

Рисунок 4.8 - Алгоритм Брезенхема 8-связный (Draw.io)

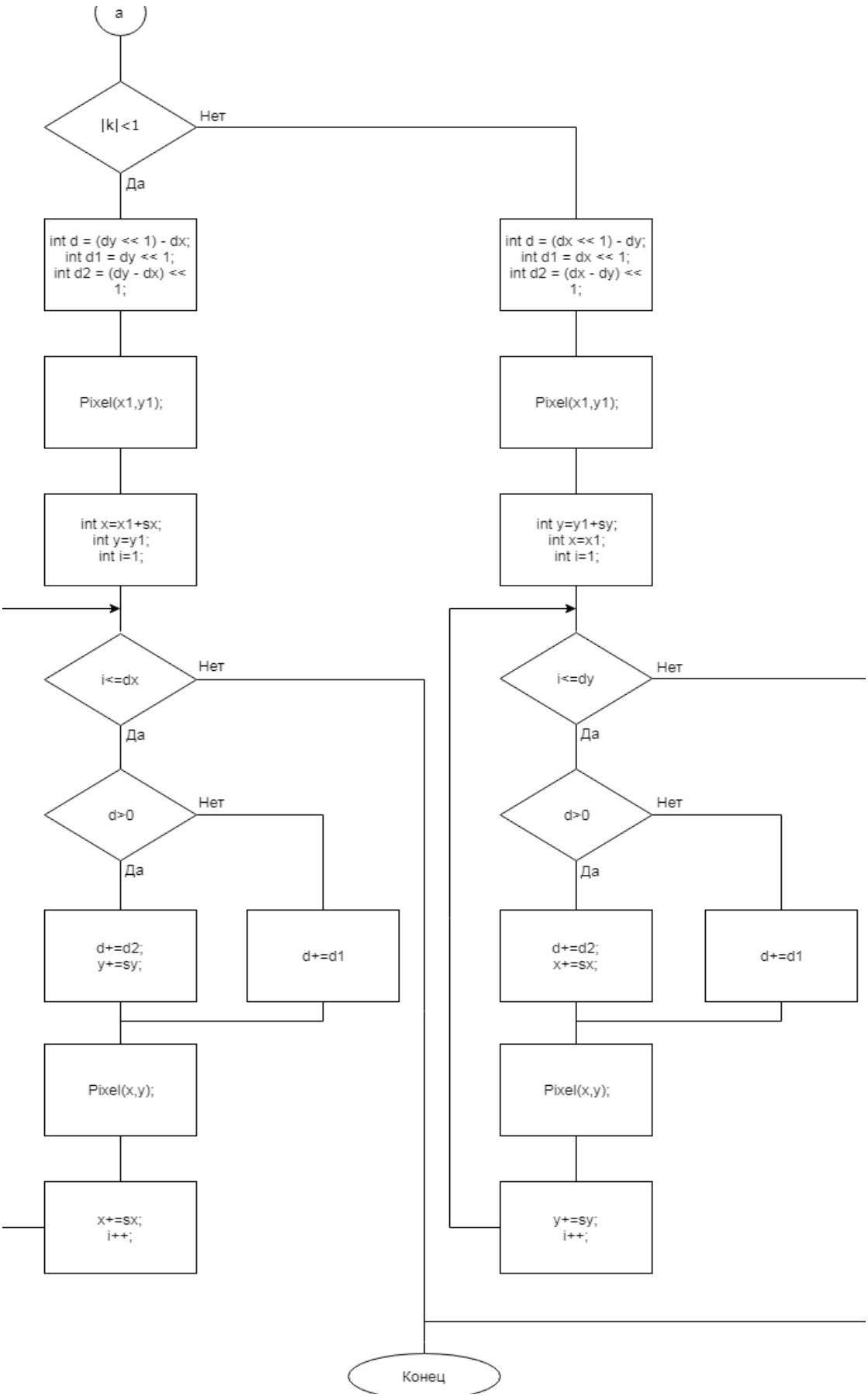

Рисунок 4.9 - Алгоритм Брезенхема 8-связный (продолжение)

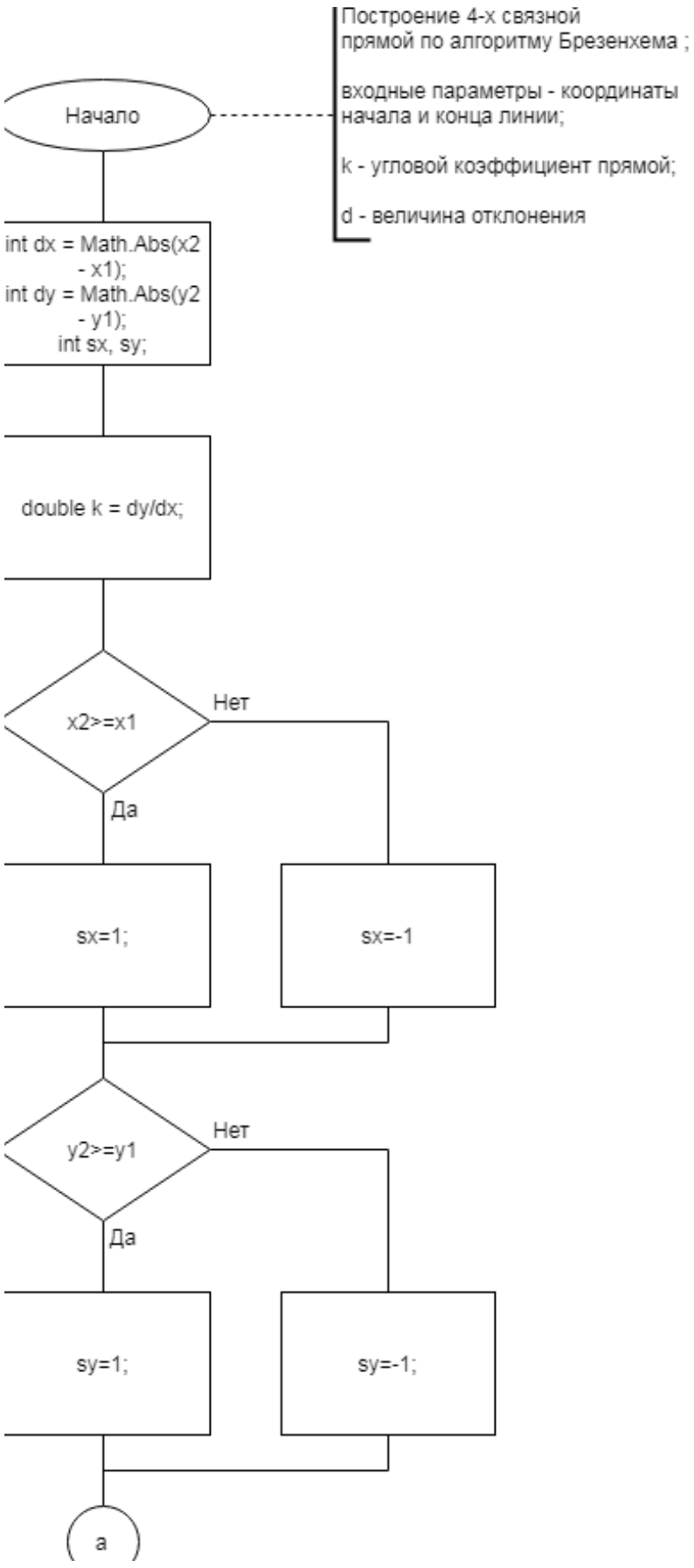

Рисунок 4.10 - Алгоритм Брезенхема 4-связный (Draw.io)

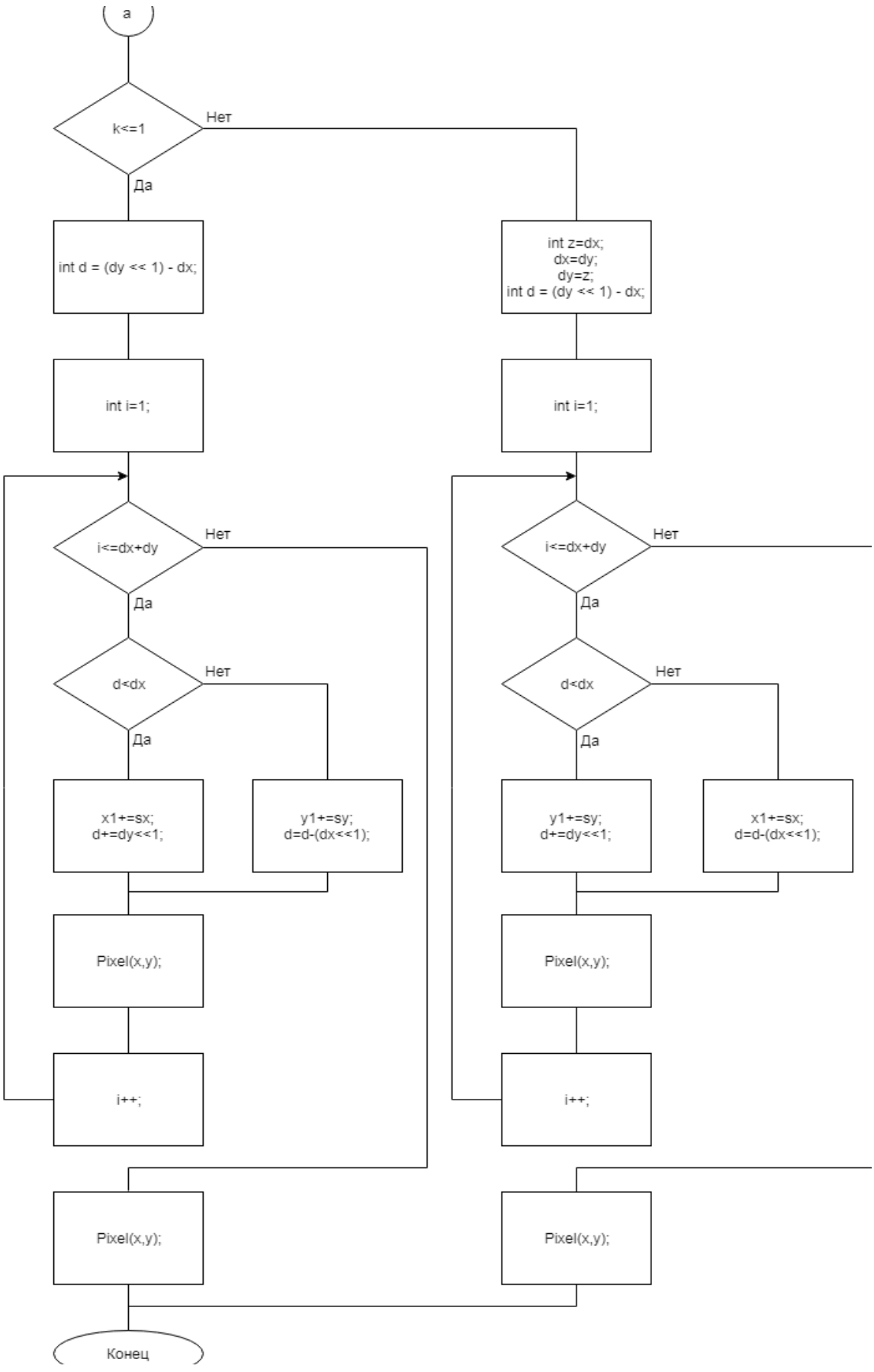

Рисунок 4.11 - Алгоритм Брезенхема 4-связный (продолжение)

4.1.3 Алгоритм цифрового дифференциального анализатора

Данный алгоритм основан на итеративном последовательном вычислении точек (DDA – Digital Differential Analyzer, цифровой дифференциальный анализатор, ЦДА). Своё название алгоритм получил по названию вычислительного устройства, выполняющего суммирование целых чисел и применявшегося некогда для аппаратной реализации генератора векторов. Идея: при единичном приращении по *x* нужно вычислить *y* с помощью дифференциального уравнения прямой с последующем округлением результата до целого [16], см. формулу (7).

$$y_{i+1} = y_i + \Delta y = y_i + (y\_2 - y\_1)/(x\_2 - x\_1)\Delta x = y_i + k\Delta x, \quad (7)$$

где y_i - текущая ордината прямой;

k - тангенс угла наклона;

Δx - приращение x на 1.

При угловом коэффициенте $|k| < 1$ на каждом шаге цикла x инкрементируется, y высчитывается по формуле (7) и приводится к целому с помощью метода Convert.ToInt32. В других случаях - y увеличивается на 1, а x рассчитывается по формуле (8).

$$x = x + \frac{1}{k}. \quad (8)$$

Из-за операции округления накапливаются погрешности, что ощутимо на достаточно длинных отрезках. Обычно погрешности корректируются, в данной реализации показано решение «в лоб» без корректировки.

Алгоритм ЦДА и последующие представлены только в 8-связной развертке. Так как работа происходит с дробными числами, алгоритм ЦДА работает медленнее, чем алгоритм Брезенхема. Блок-схема алгоритма ЦДА на рисунке 4.12.

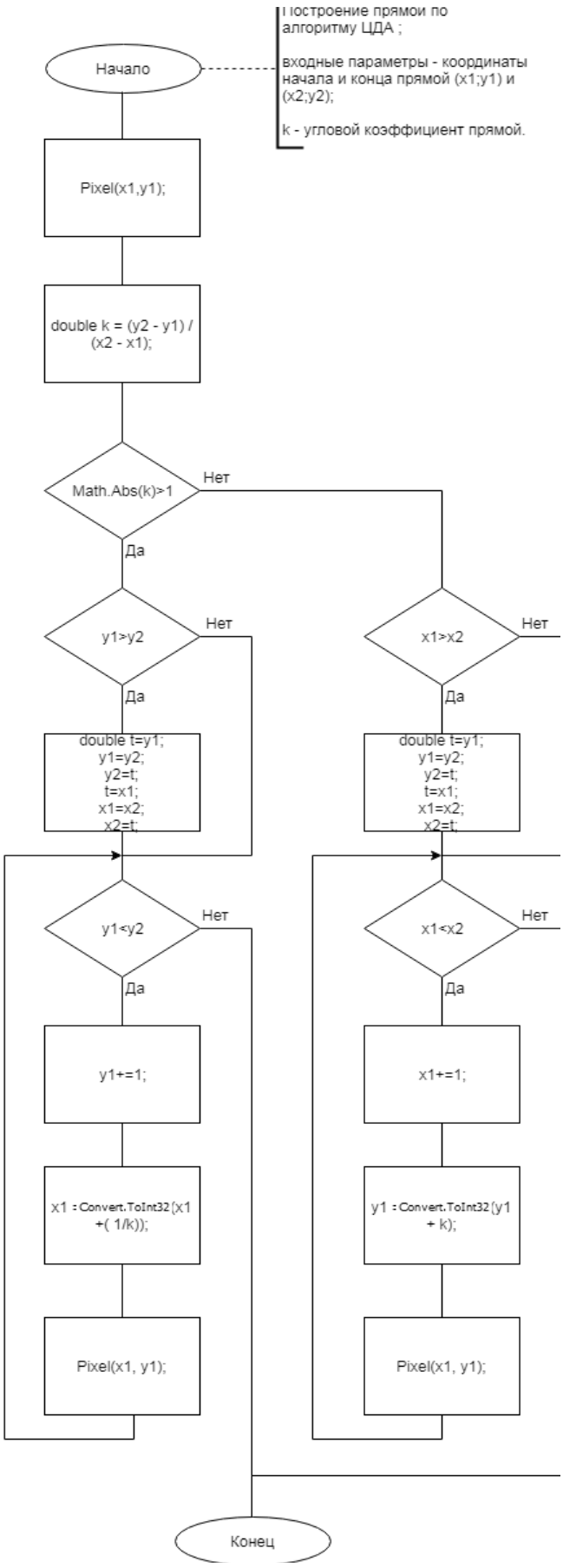

Рисунок 4.12 - Алгоритм ЦДА (Draw.io)

4.1.4 Алгоритм By

Алгоритм By выполняет разложение отрезка в растр со сглаживаем. Значения пикселей подбираются аналогично алгоритму Брезенхема, но на каждой итерации вычисляется не один пиксель, а два - (*x; y)* и (*x*; *y+1)* либо *(x-1;y)* Сглаживание осуществляется за счет распределения интенсивности между этими двумя пикселями: тот, что более близок идеальной линии, более интенсивен. Сумма интенсивностей двух пикселей равна единице. По быстродействию алгоритм By не уступает алгоритму Брезенхема [16].

Для реализации за основу был взят алгоритм Брезенхема (см. пункт 4.1.2). Чтобы определить координаты идеальной линии, в ДКП введена функция koordLine(int, int, int, int). В качестве параметров она принимает координаты начала и конца. В зависимости от углового коэффициента *k* выбирается либо расчет *y* по уравнению прямой и инкрементирование *x,* либо наоборот, где из уравнения прямой уже вычисляется *x*. Вычисленные координаты заносятся в двумерный массив. Для определения близости растровой точки к линии из вычисленных координат растра вычитается соответствующее значение «идеальной» координаты. При $|k| < 1$ вычитается из рассчитанного *y* «идеальный» *y*, в других случаях работа с *x*-координатой. Если при разности получился 0, то значения совпали, и эта растровая точка имеет 100% интенсивности, далее переход к следующему шагу цикла. Если при разности рассчитанного и идеального значения 0 не получился, то на данной итерации цикла образуются две растровые точки. Интенсивность первой будет рассчитываться по формуле (9).

$$(1 - intens) * 100\% , \quad (9)$$

где *intens* - разность между растровой и «идеальной» координатой.

Координаты точки - текущие *x* и *y*. Интенсивность второй растровой точки равна *intens*, а координаты - (*x; y +1)* для $|k| < 1,$ и (*x-1; y)* для остальных условий. На рисунках 4.13 - 4.15 представлена блок-схема алгоритма By.

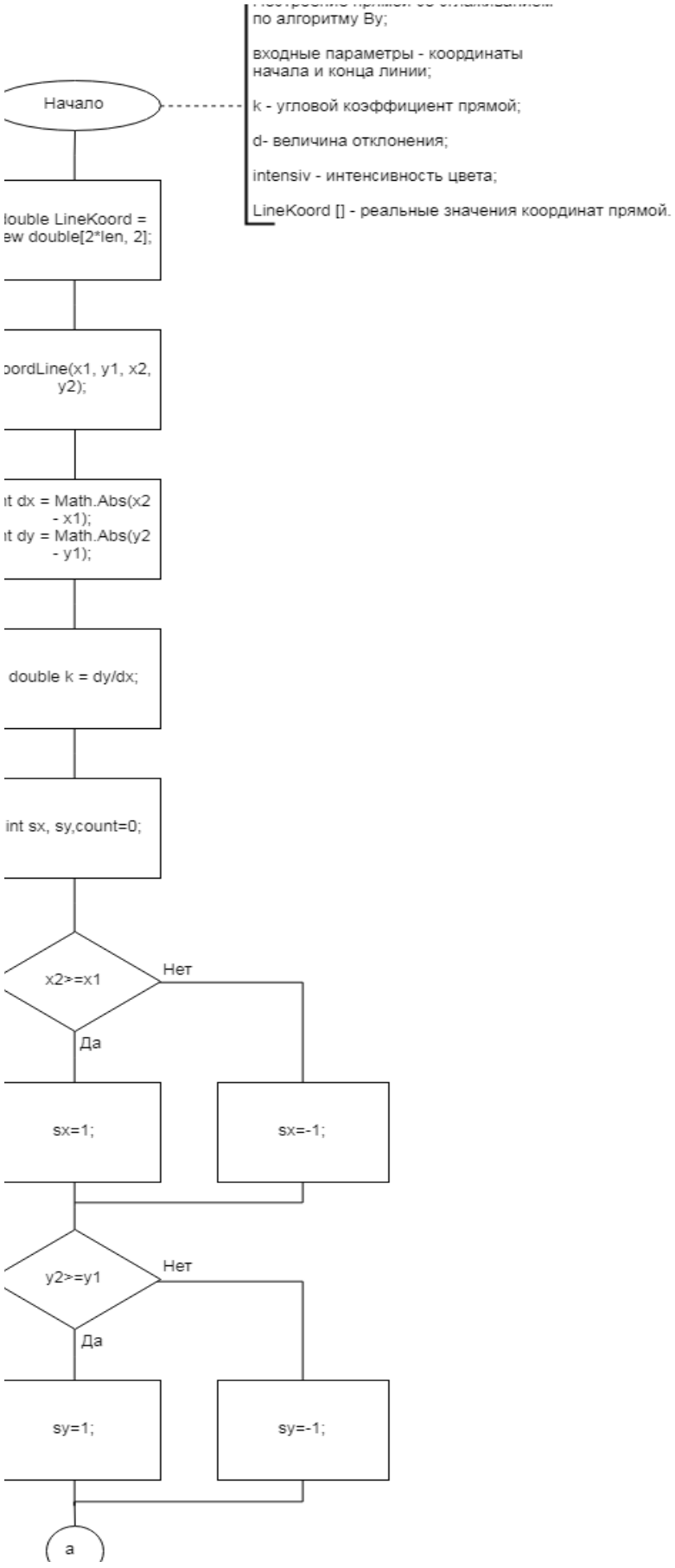

Рисунок 4.13 - Алгоритм By (Draw.io)

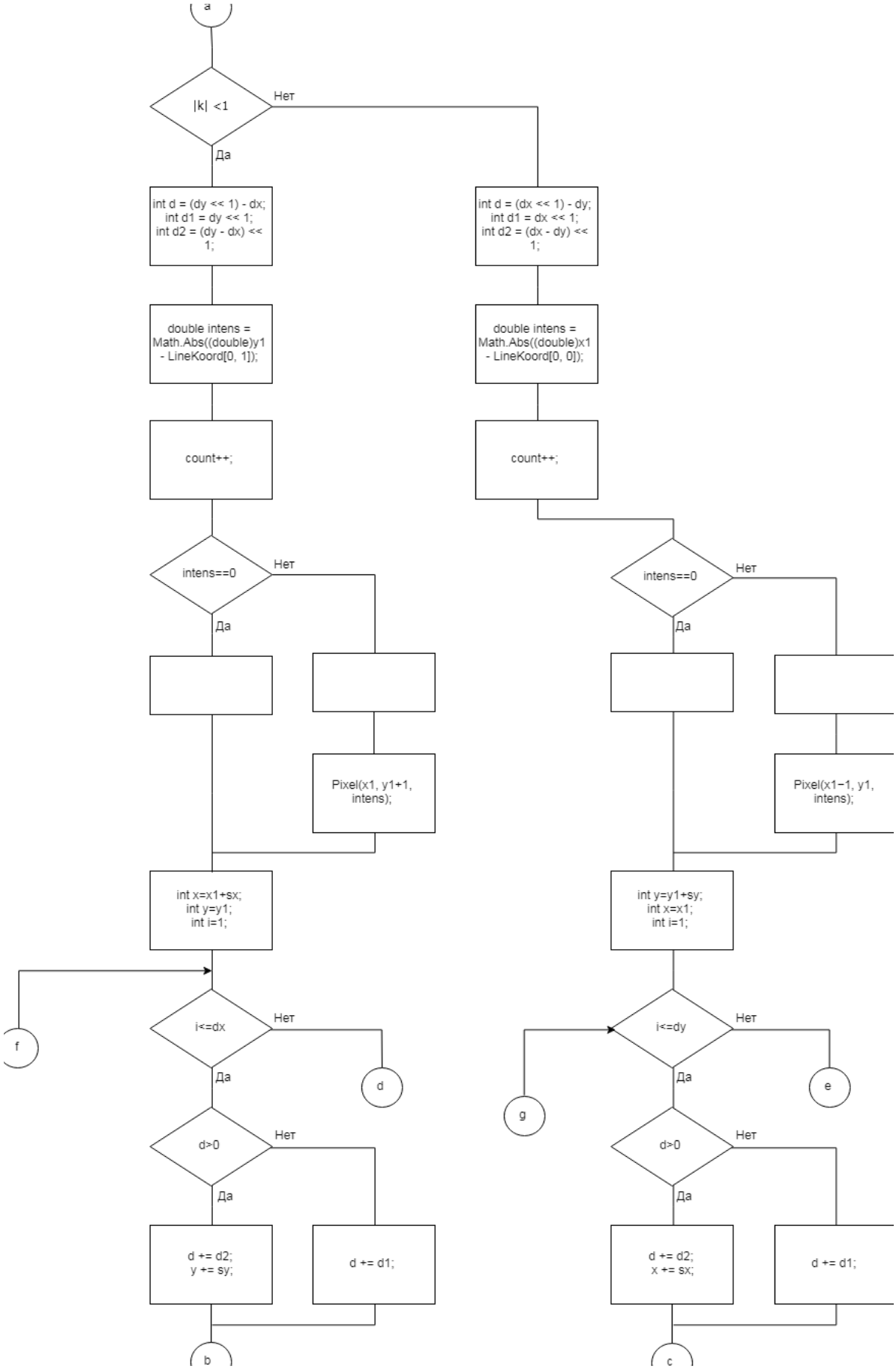

Рисунок 4.14 - Алгоритм By (продолжение 1)

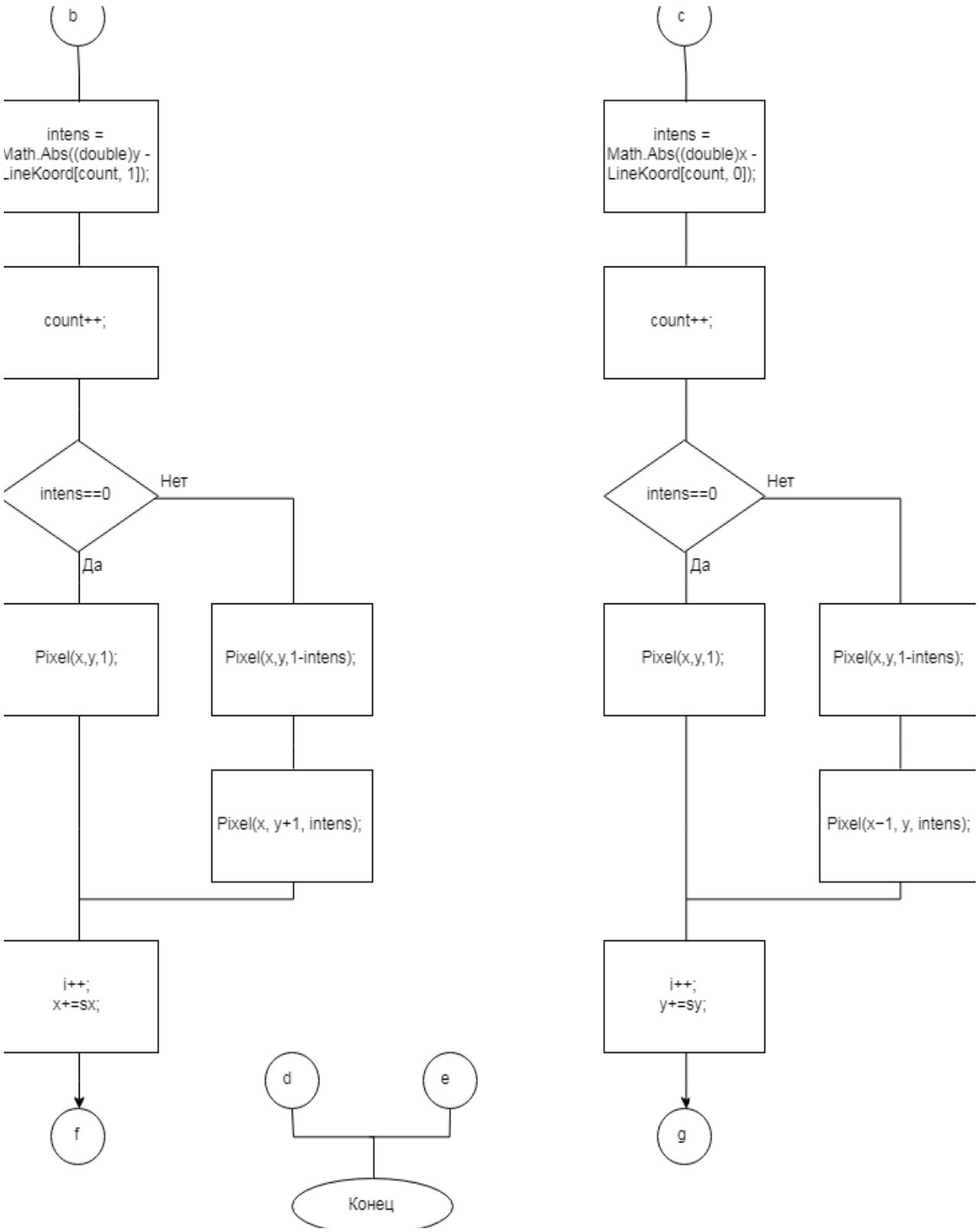

Рисунок 4.15 - Алгоритм By (продолжение 2)

4.1.5 Алгоритм средней точки для эллипса

Итерационный алгоритм построения эллипса аналогичен алгоритму Брезенхема. Для принятия решения о закраске 1-го или 2-го пикселя используется алгоритм средней точки, который выбирает, какой из соседних пикселей ближе к эллипсу, вычисляя, находится ли средняя между пикселями точка внутри эллипса или снаружи. Исходя из значения оценочного параметра *d*, выбирается пиксель по формуле (10). Изначально средняя точка равна $(x_i + 1, y_i - 0.5)$, при этом $x_i = 0$, а $y_i = b$.

$$d = b^2 * p + a^2 * q^2 - a^2 * b^2, \quad (10)$$

где (*q, p*) – координаты средней точки;

a, *b* - полуоси эллипса (для эллипса с центром в начале координат).

Если *d* = 0, то точка лежит на эллипсе, и этот случай можно рассматривать как нахождение точки вне эллипса, так и внутри. Если *d* < 0 - точка внутри эллипса, новые координаты равны $(x_{i+1} = x_i + 1;\ y_{i+1} = y_i)$, приращение по *d* согласно формуле (11).

$$d = d + b^2(2x_i + 3). \quad (11)$$

Если *d* > 0 – точка вне эллипса, тогда координаты $(x_{i+1} = x_i + 1;\ y_{i+1} = y_i - 1)$, приращение по *d* отражено в формуле (12).

$$d = d + b^2(2x_i + 3) + a^2(2 - 2y_i). \quad (12)$$

Так как эллипс имеет непостоянную кривизну, нужно в одной его части задавать единичное приращение по *x* и рассчитывать *y*, а в другой – приращение по *y* и рассчитывать *x*; чтобы определить точку инвертирования, нужно продифференцировать уравнение эллипса и приравнять результат к -1, в итоге получится следующее выражение [16] (см. формулу 13), являющееся тангенсом касательной к эллипсу:

$$\frac{\partial(b^2x^2 + a^2y^2 - a^2b^2)}{\partial x} = 0;\ 2b^2x + 2a^2y\frac{\partial y}{\partial x} = 0;$$

$$\frac{\partial y}{\partial} = \frac{2b^2x}{2a^2y} = -1 \quad (13)$$

Таким образом, переход ко второй части происходит при выполнении условия:

$$a^2 * (y - 0.5) \leq b^2 * (x + 1)$$

Средняя точка во второй части имеет координаты: $(x_i + 0.5, y_i - 1)$, где x_i и y_i - последние вычисленные координаты из первой части. Оценочный параметр *d* рассчитывается в этой точке, далее рассуждения аналогичны первой части. Если параметр $d < 0$ - точка внутри эллипса, координаты равны $(x_{i+1} = x_i + 1;\ y_{i+1} = y_i - 1)$, приращение по *d* рассчитывается по формуле (14).

$$d = d + 2 * b^2 * (x_i + 1) + a^2 * (3 - 2 * y_i). \quad (14)$$

Если $d > 0$ – точка вне эллипса, тогда координаты $(x_{i+1} = x_i;\ y_{i+1} = y_i - 1)$, приращение по *d* определяется формулой (15).

$$d = d + a^2(3 - 2 * y_i). \quad (15)$$

Случай $d = 0$ может рассматриваться как любой из вышеперечисленных, в программной реализации он был отнесен к точке вне эллипса. Построение идет по часовой стрелке, начиная с точки (0, *b*) и заканчивая точкой (*a*, 0).

В ДКП координаты точек рассчитываются в первом квадранте для эллипса с центром (0;0), потом пиксель отражается в остальные квадранты и пиксели параллельно переносятся в заданные пользователем координаты (функция Transfer (int, int, int, int)). При малых значениях *a* и *b* и для тонких эллипсов (большая разница между полуосями) для построения эллипса будет рассчитано относительно мало пикселей, что приведет к ступенчатости, фигура будет выглядеть как многоугольник. Перед построением проверяется, не вырожден ли эллипс в прямую, иначе пользователь получит сообщение об ошибке. Блок-схема алгоритма изображена на рисунках 4.16 - 4.17.

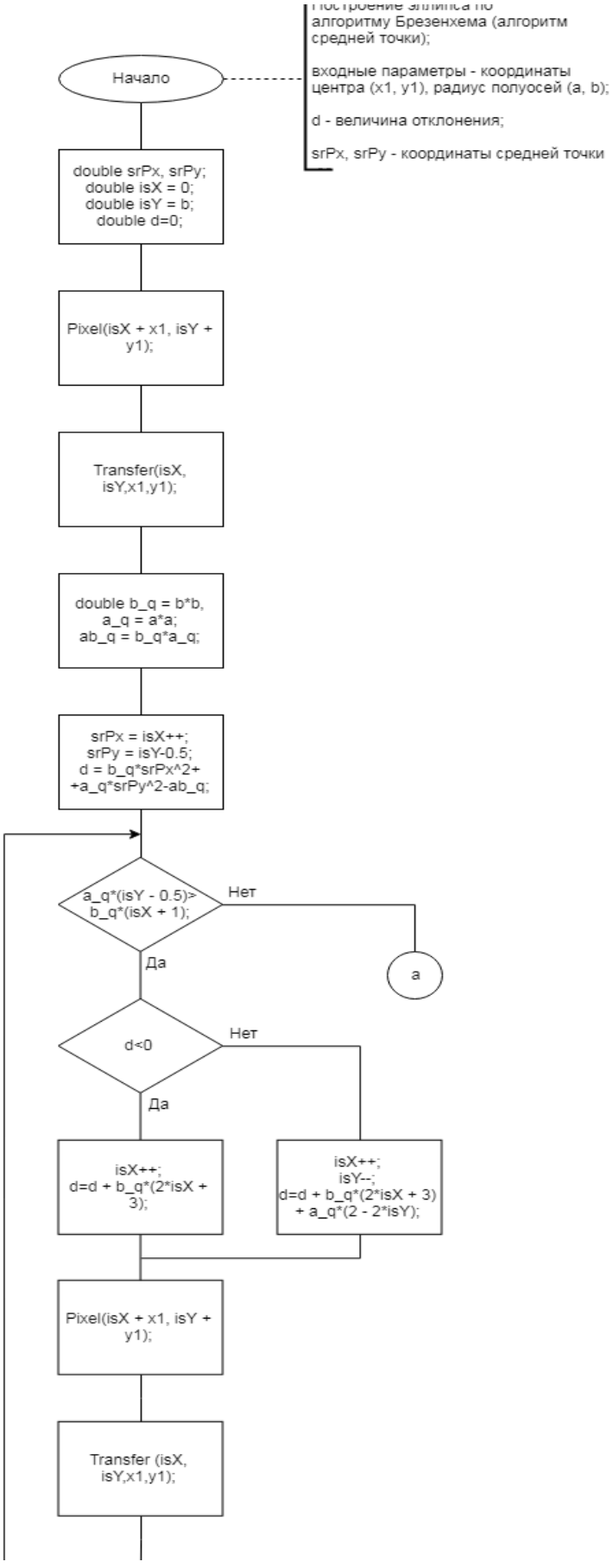

Рисунок 4.16 - Алгоритм средней точки для эллипса (Draw.io)

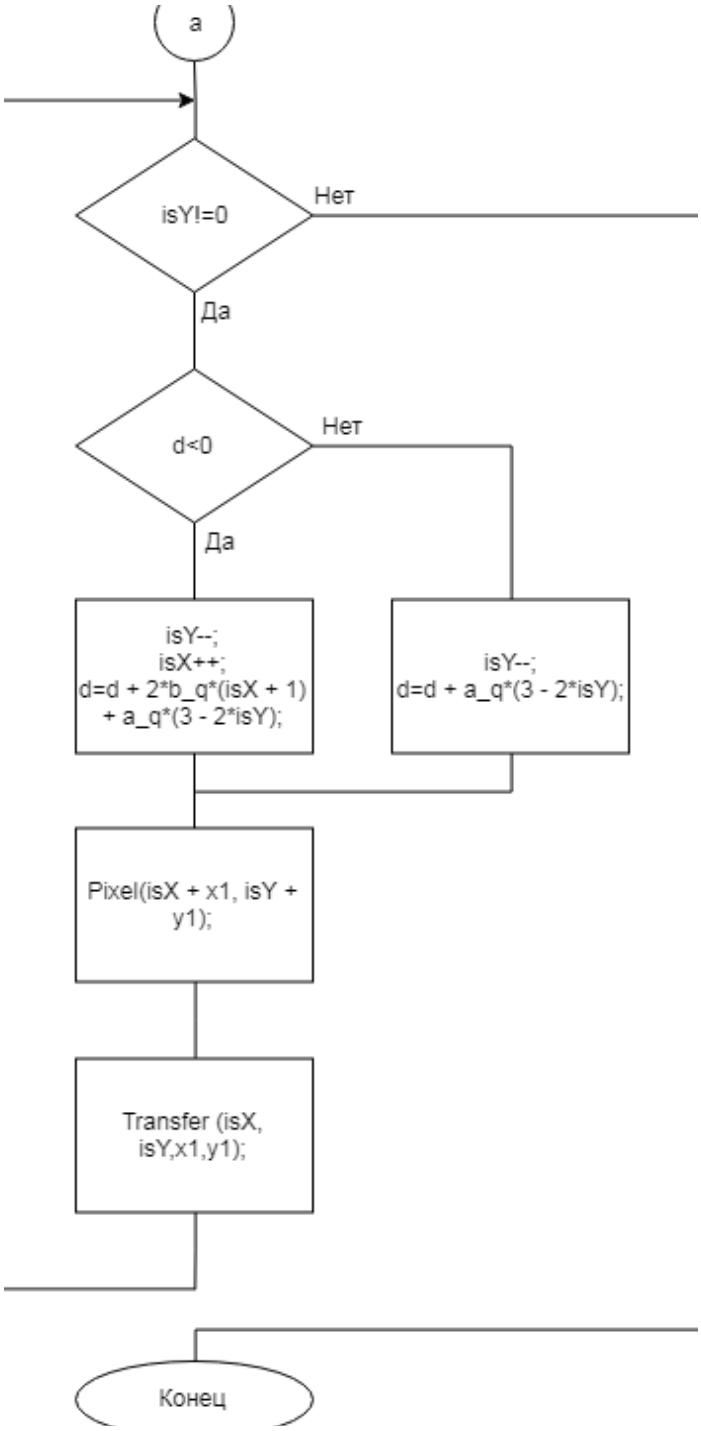

Рисунок 4.17 - Алгоритм средней точки для эллипса (продолжение 1)

4.1.6 Алгоритм средней точки для окружности

Благодаря одинаковой кривизне окружности не нужно менять порядок расчета *x* и *y,* как в случае с эллипсом. Оценочный параметр *d* определяется по формуле (16). Окружность располагается в начале координат. При произвольном расположении окружности нужно выполнить перенос.

$$d = p^2 + q^2 - r^2, \quad (16)$$

где (q,p) – координаты средней точки;

r - радиус окружности.

Если $d = 0$, то точка лежит на окружности и этот случай можно рассматривать как нахождение точки вне окружности, так и внутри. Если $d < 0$ - точка внутри окружности, тогда y не изменяется, а d меняется по формуле (17).

$$d = d + 2x_i + 3. \quad (17)$$

Если $d > 0$ – точка лежит вне окружности, y уменьшается на 1, d рассчитывается по формуле (18) [16].

$$d = d + 2(x_i - y_i) + 5. \quad (18)$$

Инкрементирование x производится в обоих случаях. Построение происходит по часовой стрелке и начинается с точки $(0, r)$.

В ДКП пиксель вычисляются для окружности в начале координат, потом переносится в заданные пользователем координаты (функция Translate(int, int, int)) и параллельно отражается в остальные квадранты (функция Transfer(int, int, int, int)). Блок-схема алгоритма изображена на рисунке 4.18.

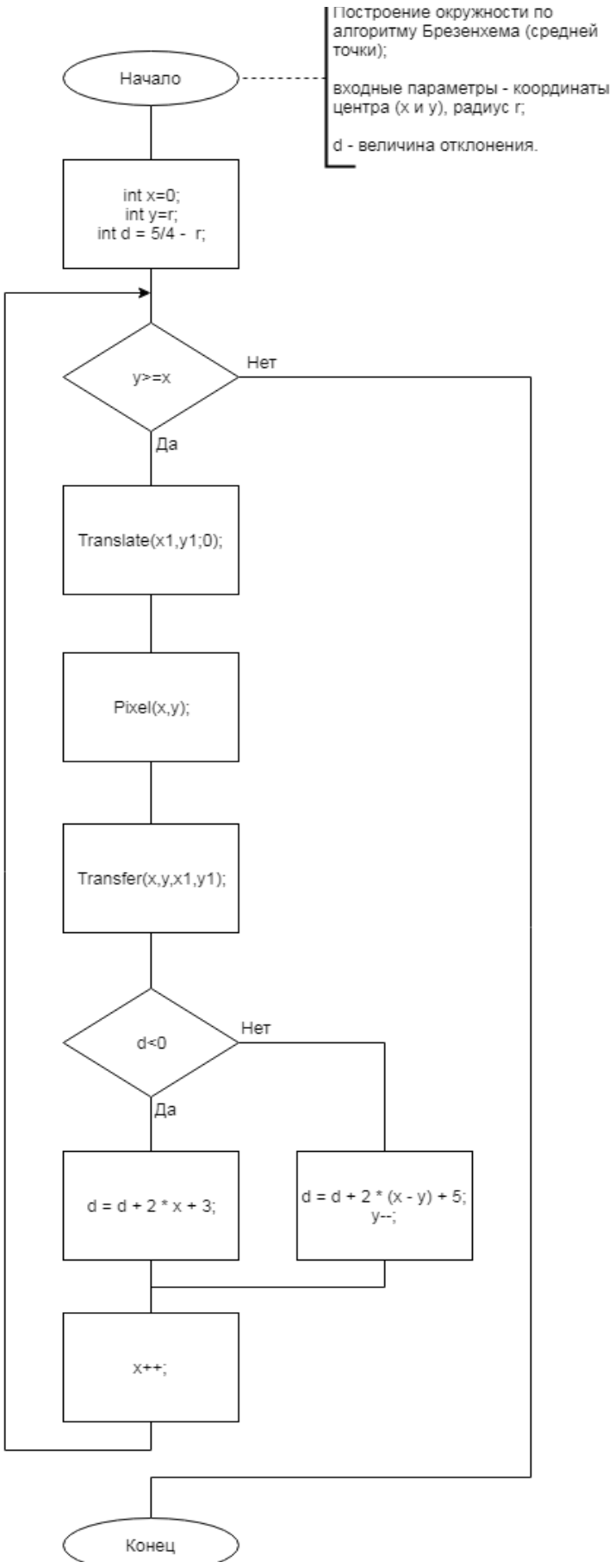

Рисунок 4.18 - Алгоритм средней точки для окружности (Draw.io)

4.2 Визуализация

Визуализатор реализован с помощью элемента glControl и является полем с координатной сеткой, на котором будет происходить построение. Для линий кроме растрового изображения выводятся «идеальные». Сетка реализована с помощью графических примитивов Lines (линии) и Points (точки). Линии в сетке являются осями координат, а точки - разметкой, точки строятся циклически. По умолчанию разрешение сетки 50x50 точек, при желании пользователя разрешение можно изменить в настройках, максимум - 300x300 точек.

Так как ДКП предназначен для изучения алгоритмов растеризации, то в качестве основного элемента визуализации взят рисунок пикселя, который создан с помощью графического примитива QuadStrip (квадрат), исходный код представлен на листинге 4.1.

Листинг 4.1 - Изображение пикселя

```
GL.Begin(PrimitiveType.QuadStrip);
GL.Vertex2(ValuesArray[l, 0] - 0.5, ValuesArray[l, 1] - 0.5);
GL.Vertex2(ValuesArray[l, 0] + 0.5, ValuesArray[l, 1] - 0.5);
GL.Vertex2(ValuesArray[l, 0] - 0.5, ValuesArray[l, 1] + 0.5);
GL.Vertex2(ValuesArray[l, 0] + 0.5, ValuesArray[l, 1] + 0.5);
 GL.End();
```

Пиксель имеет синий цвет (либо оттенки синего для алгоритма Ву), «идеальная» фигура построена красным. Массив ValuesArray - это набор вычисленных координат x и y для рисования. Он введен в качестве защитного механизма координат точек при многопоточности. Несмотря на массив, рисование очередного пикселя происходит сразу же после его вычисления.

Работа с графикой, как наиболее ресурсоемкая операция, вынесена в отдельный поток, также данное решение обеспечивает возможность выводить параллельно с рисованием код алгоритма, иначе визуализация займет весь основной поток. Каждый пиксель рисуется новым потоком. Для этого создан массив Thread[] ArrTh. Количество созданных потоков равно числу необходимых точек для построения. Чтобы не произошло рассинхронизации между

визуализацией и выводом кода алгоритма, основной поток должен дождаться, пока дочерний закончит рисование очередного пикселя, поэтому после каждого запуска рисования идет метод Join(), определенного в классе Thread C#, показано на листинге 4.2.

Листинг 4.2 - Синхронизация дочернего и основного потоков

```
ArrTh[count].Start();
ArrTh[count].Join();
```

Элемент визуализатора glControl находится в основном потоке программы. Для доступа к glControl из дочернего был использован асинхронный делегат. Делегат delegate void MyDelegate() указывает на метод без параметров, в данном случае это метод paint(), в котором происходит рисование пикселя и обновление glControl из дочернего потока, чтобы пользователь смог увидеть результат. Метод Pixel() вызывает BeginInvoke(), чем асинхронно инициализирует paint(), на который указывает делегат. Pixel() вызывается дочерним потоком. Пример описанной конструкции приведен в листинге 4.3.

Листинг 4.3 - Инструменты для рисования в дочернем потоке

```
public delegate void MyDelegate(); //делегат, объявлен до конструктора формы
public void paint() {//рисование «пикселя»
GL.DrawBuffer(DrawBufferMode.Front);
GL.Color3(0.0f, 0.0f, 1.0f); //для paintVu() GL.Color3(0.0f, 0.0f, intensiv[l])
GL.Begin(PrimitiveType.QuadStrip);
GL.Vertex2(ValuesArray[l, 0] - 0.5, ValuesArray[l, 1] - 0.5);
GL.Vertex2(ValuesArray[l, 0] + 0.5, ValuesArray[l, 1] - 0.5);
GL.Vertex2(ValuesArray[l, 0] - 0.5, ValuesArray[l, 1] + 0.5);
GL.Vertex2(ValuesArray[l, 0] + 0.5, ValuesArray[l, 1] + 0.5);
GL.End();
GL.Finish();
...
          l++;        }

public void Pixel()
        {
            if (!byToolStripMenuItem.Checked)
                  {   //асинхронная инициализация
                      BeginInvoke(new MyDelegate(paint)); }
            else {
```

```
                BeginInvoke(new MyDelegate(paintVu));
            } }
private void ObResB(int x1, int y1, int x2, int y2) { //алгоритм для 8-связной прямой по
//уравнению
ArrTh[count] = newThread(Pixel);//запуск рисования в дочернем потоке
ArrTh[count].Start();
ArrTh[count].Join();
....}
```

Анимация построения алгоритмов реализована за счёт использования двойной буферизации и отказа от очистки буфера при выводе каждого пикселя. Суть двойной буферизации - используется два буфера: передний, который видит пользователь, и задний, на котором происходит формирование кадра. Когда кадр сформирован, он меняет предыдущий. Следующая команда позволяет отобразить данные на визуализаторе с переднего буфера:

```
GL.DrawBuffer(DrawBufferMode.Front);
```

При одиночной буферизации изображение формируется «рывками», двойная буферизация позволяет этого избежать [17]. Двойная буферизация включена в OpenTK по умолчанию. Если не очищать буфер, то выводимые кадры накапливаются, что позволяет постепенно строить фигуру. Тем не менее, буфер очищается перед выполнением алгоритма для того, чтобы удалить устаревшие данные. Очистка буфера производится командой:

```
GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
```

Работа с цветом в визуализаторе везде, кроме алгоритма By идет в формате RGB. В алгоритме By для интенсивности пикселя необходима установка прозрачности (интенсивность в терминах OpenGL имеет другой смысл), поэтому нужно включить альфа-канал и использовать цветовой режим RGBA. Перед включением альфа-канала необходимо разрешить смешивание:

```
GL.Enable(EnableCap.Blend).
```

Альфа-канал включается с помощью метода смешивания:

```
GL.BlendFunc(BlendingFactorSrc.SrcAlpha,BlendingFactorDest.OneMinusSrcAlpha);
```

Методы смешивания и разрешения смешивания определены в пространстве имен OpenTK. После этого для задания цвета можно использовать цифры от 0 до 1. Например, если вместо Color3(0.0f, 0.0f, 1.0f) написать Color3(0.0f, 0.0f, 0.5f).

Синий цвет будет прозрачен на 50%. При альфа = 0 - 100% прозрачность, при альфа = 1 - 0 % прозрачности. Для алгоритма By при рисовании очередного пикселя предварительно устанавливается прозрачность:

```
GL.Color3(0.0f, 0.0f, intensiv[j]);
```

В остальном рисование происходит как в функции paint() (см.выше);

При изменении размеров окна данные на визуализаторе сохраняются. Это реализовано обработкой события Resize. Его обрабатывает метод glControl1_Paint (object, PaintEventArgs). Когда у окна изменяют размер, glControl1_Paint очищает визуализатор и запускает его повторную инициализацию с новыми размерами окна, рисует сетку, также в нем вызывается другой метод - Pereris(int, int, int, int), восстанавливающий изображения фигур, если они были на экране до изменения размера. Метод glControl1_Paint показан на листинге 4.4.

Листинг 4.4 - Изменение размера окна

```
private void glControl1_Paint(object sender, PaintEventArgs e)
{
if (!loaded) //loaded флаг для проверки изменения размера окна
return;
GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
if (glControl1.Height == 0)
{
   glControl1.Height = 1; }
GL.Viewport(0, 0, glControl1.Width, glControl1.Height);
GL.MatrixMode(MatrixMode.Projection);
GL.LoadIdentity();
Glu.gluOrtho2D(0.0, (Sinhr.point / 2), 0.0, (Sinhr.point / 2));
GL.MatrixMode(MatrixMode.Modelview);
GL.LoadIdentity();
GL.Translate(0.5, 0.5, 0);

if(prov> 0) {// prov- переменная для проверки наличия нарисованной фигуры
Pereris(x1,x2,a,b); }
setka();
baseEl(x1, y1, x2, y2);
glControl1.SwapBuffers(); }
```

Метод Pereris(int, int, int, int) вызывает Pixel1(), который, благодаря массиву с вычисленными точками, сразу же выводит изображение. Методы glControl1_Paint,

Pereris и Pixel1 работают в основном потоке. Так как для них не требуется анимация, восстановление данных на визуализаторе происходит быстро.

В окне визуализации есть текстовое многострочное поле для вывода исходного кода алгоритма. Поле реализовано классом TextRichBox (принадлежит WindowsForm), который предоставляет элемент управления полем форматированного текста. Вывод текста в элементе выполняется методом AppendText(). Метод при каждом вызове добавляет новую строку в конец поля. Выполнение алгоритма и вывод исходного кода произойдет в том случае, если пользователь корректно задал начальные значения (координаты, выбрал алгоритм) и нажал кнопку «Визуализация», при неподходящих исходных данных откроется сообщение с ошибкой. Исходный код выводится построчно - при выполнении строки алгоритма аналогичная строка появляется в текстовом поле.

4.3 Теоретические разделы

В ДКП есть два теоретических раздела - это непосредственно раздел с теоретическим материалом по алгоритмам и раздел с блок - схемами. В первый раздел включено:

- общие сведения о растровых алгоритмах и представление отрезка по его уравнению;
- алгоритм Брезенхема (отрезок);
- алгоритм ЦДА;
- алгоритм Ву;
- алгоритм средней точки для эллипса;
- алгоритм средней точки для окружности;

Данные для наполнения раздела брались из учебно-методического пособия Методы и средства представления графической информации автора к.т.н. Ярош Е.С., как того требовало задание. Кроме текста в раздел входят графические пояснения, взятые из вышеупомянутого пособия. Вывод материала реализован в

отдельном окне, с помощью класса TextRichBox. Метод класса LoadFile(), загружает в поле необходимый файл формата rtf. Пример загрузки файла в программу приведен на листинге 4.5.

Листинг 4.5 - Загрузка теоретического материала

```
private void Slaid_Load(object sender, EventArgs e)
        {
...
if (брезенхемДляОтрезкаToolStripMenuItem.Checked == true)
                {
inf.LoadFile("Brezenh.rtf");} ... }
```

Важным преимуществом .rtf файла является совместимость с многочисленными операционными системами и приложениями для работы с текстом, что актуально, если пользователь запустит ДКП на другой версии Windows.

В раздел с блок-схемами входят блок-схемы для:

- алгоритма Брезенхема 4-связного;
- алгоритма Брезенхема 8-связного;
- упрощенного алгоритма Брезенхема 4-связного;
- упрощенного алгоритма Брезенхема 8-связного;
- алгоритма построения отрезка по уравнению прямой 4-связного;
- алгоритма построения отрезка по уравнению прямой 8-связного;
- алгоритма ЦДА (здесь и далее только 8-связные);
- алгоритма Ву;
- алгоритма средней точки для эллипса;
- алгоритма средней точки для окружности.

Блок-схемы составлялись по пособию, указанному выше. Вид блок-схем представлен в разделе 4.1 на рисунках 4.1 - 4.16. Блок-схемы в разделе без разделителей, они являются изображением формата .png, которое можно прокручивать скроллбаром. В отличие от теоретического материала, для вывода изображений блок-схем используется класс PictureBox (принадлежит Windows Form). Он предоставляет элемент управления графическим окном Windows для отображения рисунка. Пример кода для загрузки блок-схемы на листинге 4.6.

Листинг 4.6 - Загрузка блок-схем

```
private void общееРешениеToolStripMenuItem_Click(objectsender, EventArgs e)
        {
Sinhr.teor = 21;
title.Text = "Брезенхем для линии";
sv.Text = "4-связность";
Image img = Image.FromFile("brez4.png");
             pictureBox1.Image = img;
             pictureBox1.Visible = true;
pictureBox1.Width = img.Width;
             pictureBox1.Height = img.Height;
           ...
           }
```

4.4 Интерфейс

Программный комплекс оптимизирован для просмотра при разрешении 1366x768. Элементы на формах сгруппированы по своим выполняемым действиям. Дизайн выполнен в нейтральном сером цвете и его оттенках, в формах использовано не более 3-4 цветов, для фона и текста выбраны контрастные цвета. Цвета гармонично сочетаются.

Всего в приложении 5 окон:

- меню;
- теоретический раздел;
- раздел с блок-схемами;
- визуализатор;
- настройка визуализации.

Организована свободная навигация между первыми 4-мя окнами. Настройка разрешения для визуализации доступна лишь из визуализатора. При выборе в меню определенного раздела аналогичный раздел выбирается в теоретическом разделе, в блок-схемах и в визуализаторе, если не было последующих изменений, иначе в качестве изучаемого устанавливается последний выбранный алгоритм из

верхнего меню. Запоминание выбранного алгоритма реализовано использованием статического класса Sinhr в основном файле программы Program.cs. Статический класс доступен из любой формы. У него создано числовое поле teor, которое хранит специальные значения, определяющие выбранный алгоритм, например, если поле teor=2, то выбран 8-связный алгоритм Брезенхема. Данные в поле teor обновляются, как только пользователь выбирает другой алгоритм. При открытии других форм и переходе между ними для предустановки изучаемого алгоритма анализируется значение поля teor. Код статического класса показан на листинге 4.7.

Листинг 4.7 - Статический класс

```
static class Sinhr
{
public static int teor = 0;  }
```

Скорость визуализации регулируется скоростью отрисовки каждого кадра. По умолчанию самая высокая скорость является стандартной. Кроме нее есть еще два варианта:

- замедлить отрисовку каждого кадра на 0.1 секунды;
- замедлить отрисовку каждого кадра на 0.3 секунды;

Замедление осуществляется вызовом метода Sleep() (содержится в пространстве класса Thread) в функции дочернего потока, который выполняет рисование кадра. Sleep() принимает в качестве параметра 100, если выбраны 0.1 секунда и 300, если - 0.3. ДКП сохраняет введенные настройки разрешения сетки и скорости в xml-файле через организацию Properties.Settings, которые при последующих запусках восстанавливаются. В Properties.Settings объявлены три переменных:

- Name - переменная формата «строка» в области приложения, хранит наименование приложения, эту переменная доступна только для чтения;
- Size - целочисленная переменная в области пользователя, ее можно читать и перезаписывать, хранит данные о разрешении координатной сетки;

- Speed - аналогична Size, но хранит данные о выбранной скорости;

На листинге 4.8 представлен фрагмент кода, который устанавливает требуемую скорость отрисовки из Properties.Settings.

Листинг 4.8 - Установка скорости отрисовки кадра

```
public void paint() // либо paintVu()
        {   …
if (Properties.Settings.Default.Speed > 0) {
           Thread.Sleep(Properties.Settings.Default.Speed);
            }
        else
            { if (Sinhr.speed != 0) {
                    Thread.Sleep(Sinhr.speed);}
            } ...}
```

В ДКП для удобства и навигации присутствует верхнее меню MenuStrip. Оно содержит пункты:

- визуализатор (везде кроме визуализатора),
- теория (в блок-схемах и в визуализаторе),
- блок-схемы (везде кроме блок-схем),
- выбор алгоритма (везде, кроме основного меню),
- справка,
- о программе,
- назад (везде, кроме основного меню),
- выход,
- настройка (в визуализаторе).

Помимо основных окон в приложении есть окна сообщений MessageBox. Данные окна вызываются при выборе пунктов верхнего меню «Справка», «О программе», а также при ошибках.

5 Результаты реализации

На рисунках ниже представлены результаты реализации приложения. Программа протестирована путём ввода различных исходных данных. На рисунке 5.1 показано меню приложения, на 5.2 - справка.

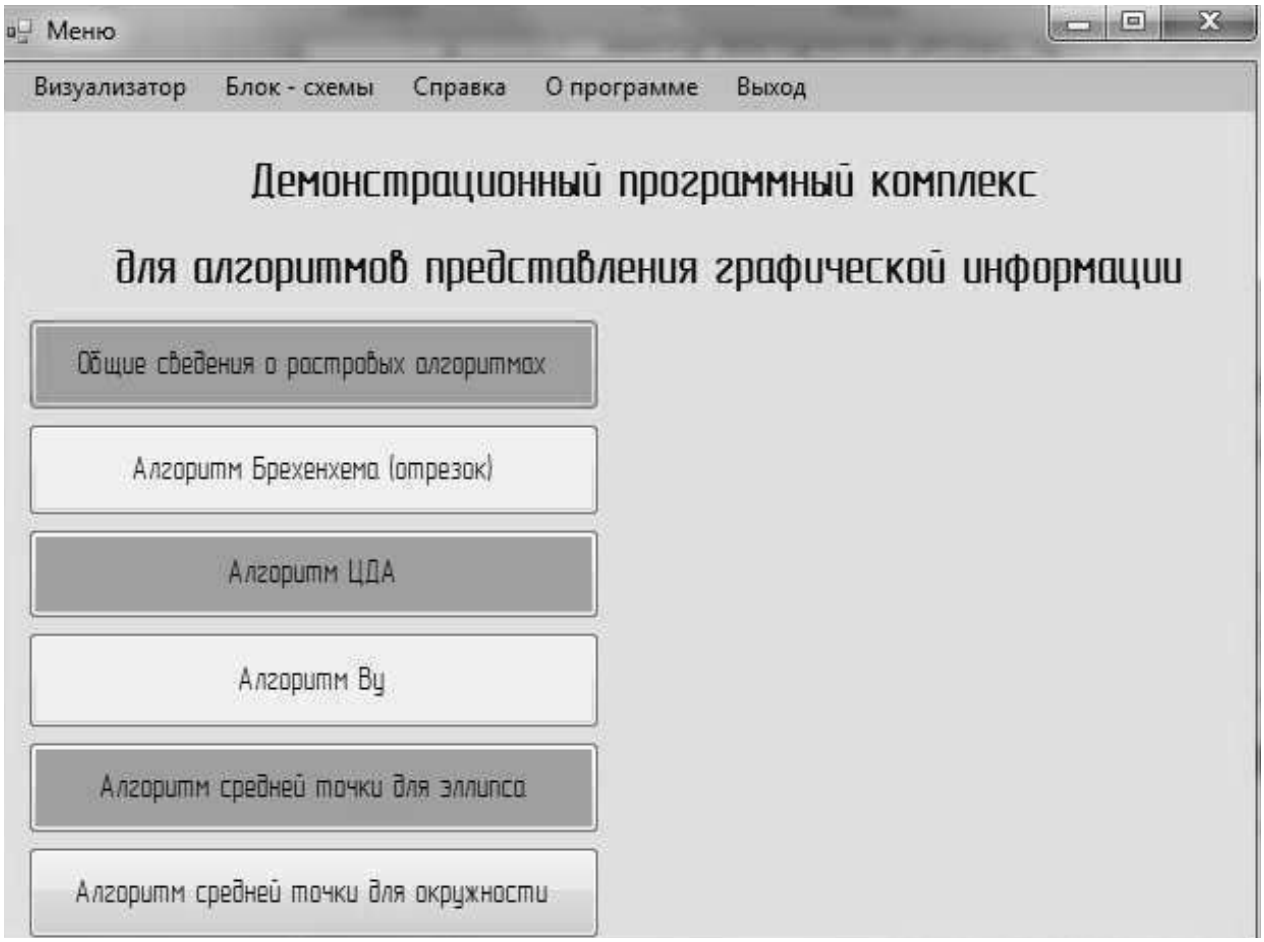

Рисунок 5.1 - меню ДКП

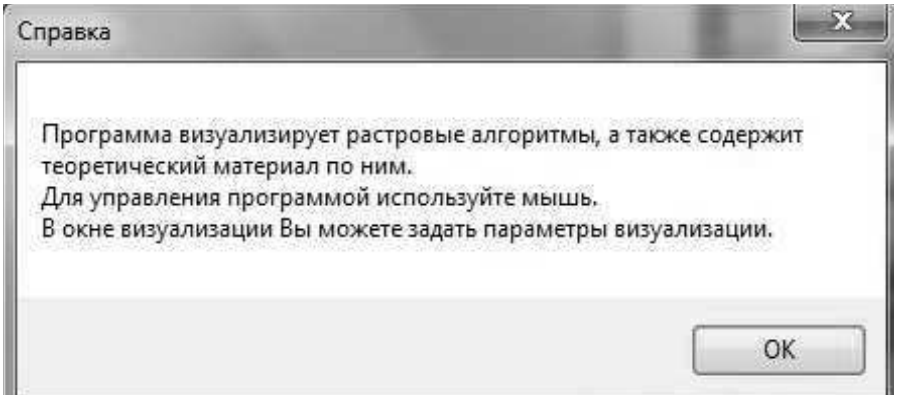

Рисунок 5.2 - Справка приложения

На рисунках 5.3 и 5.4 изображены части материала из теоретического раздела. На 5.3 - общие сведения о растровых алгоритмах, на 5.4 - описание растрового представления эллипса.

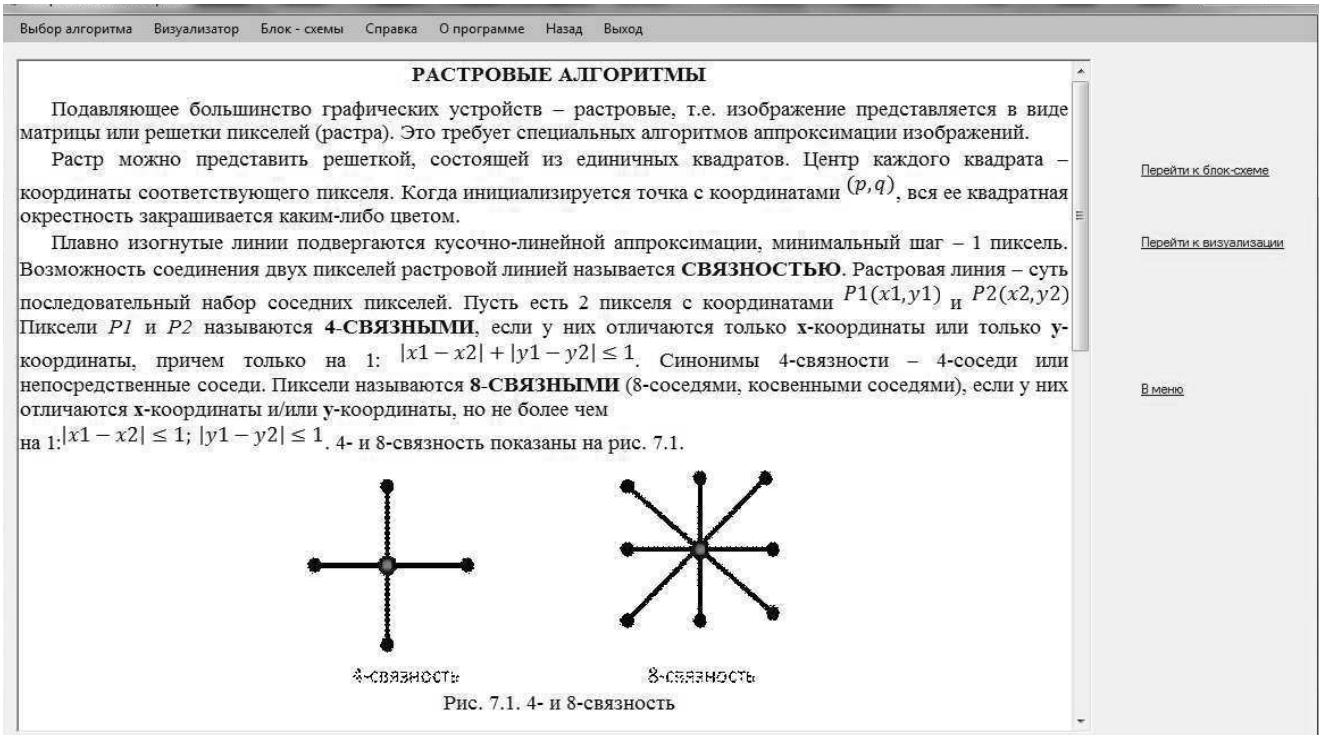

Рисунок 5.3 - Общие сведения о растровых алгоритмах

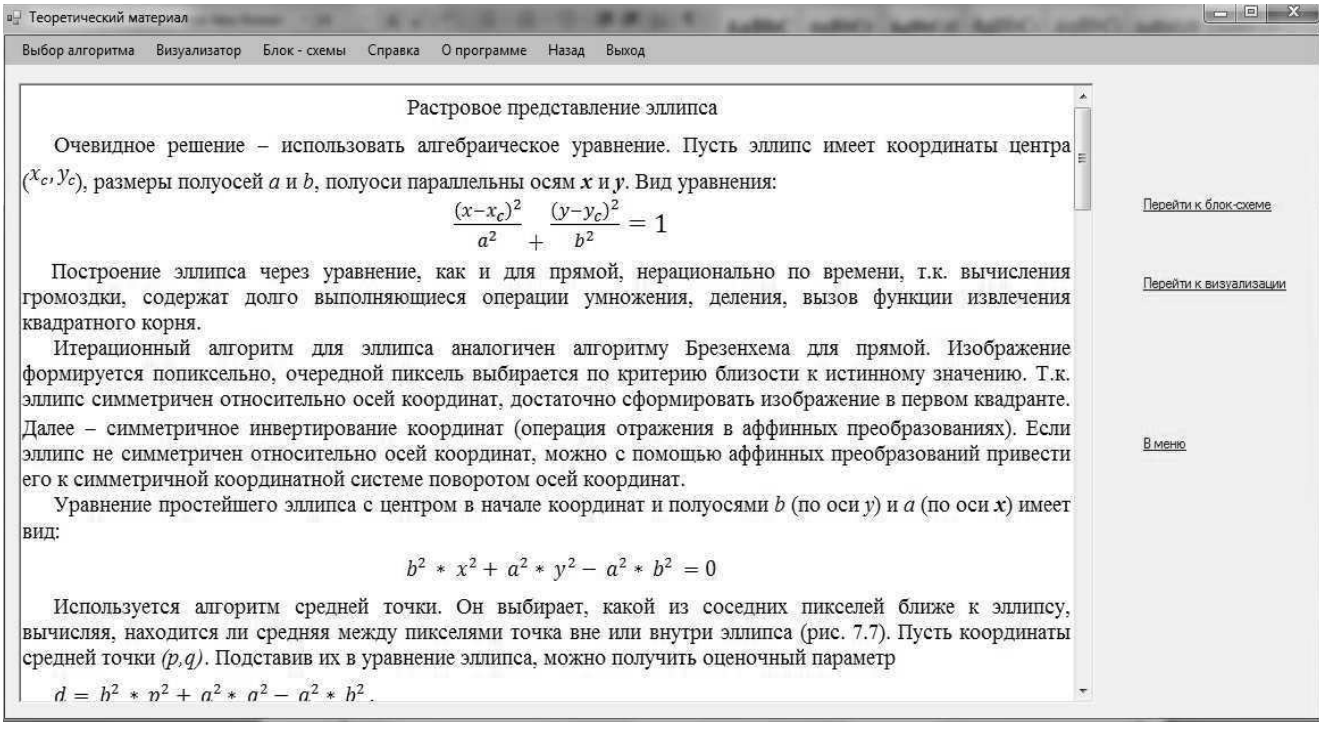

Рисунок 5.4 - Растровое представление эллипса

На рисунках 5.5 и 5.6 показан вид и функционирование раздела с блок-схемами. Приведены некоторые блок-схемы алгоритмов. На 5.5 - алгоритм By, на 5.6 - алгоритм средней точки для окружности.

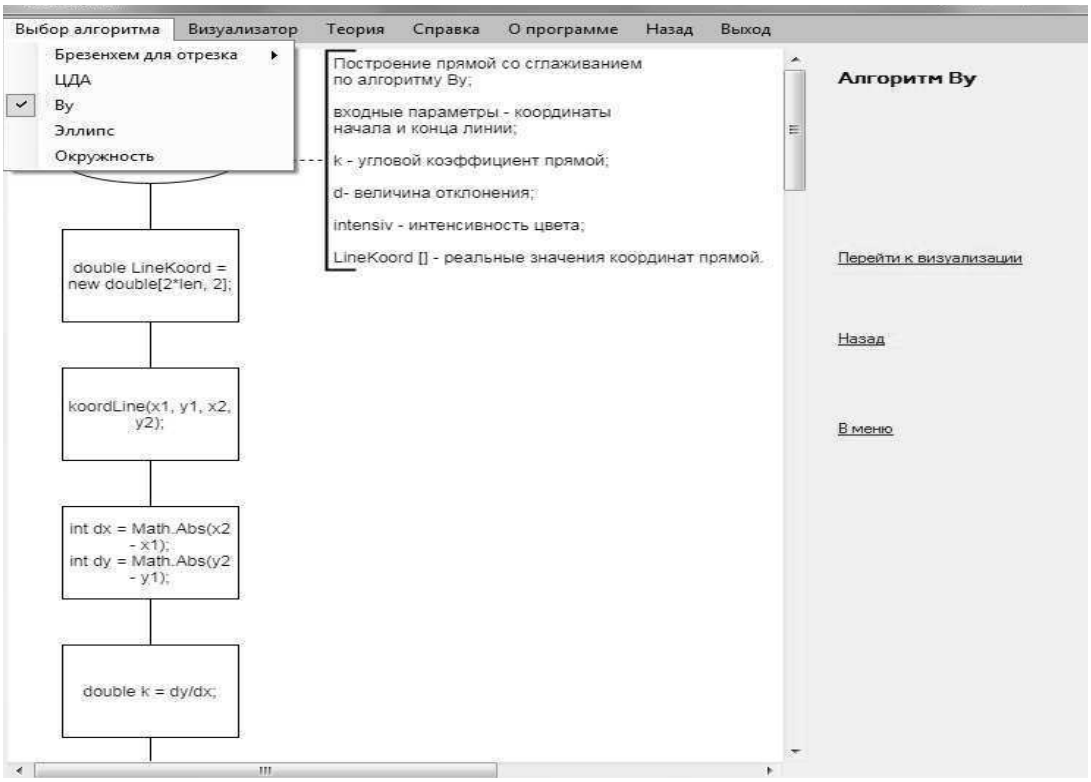

Рисунок 5.5 - Алгоритм By в разделе «Блок-схемы»

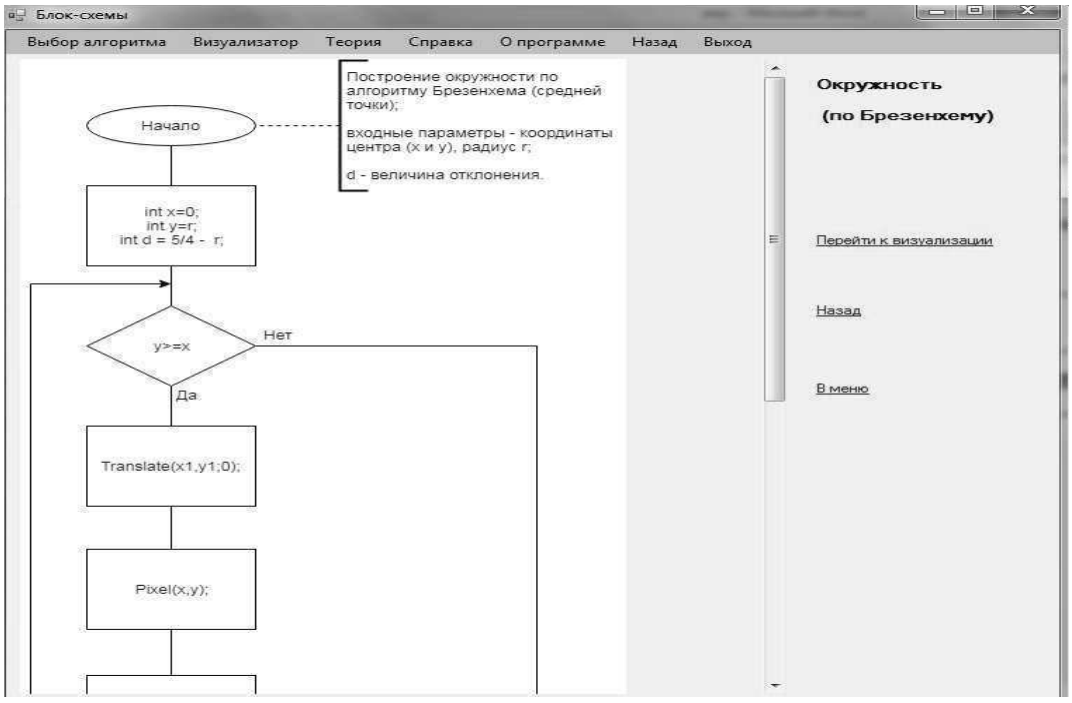

Рисунок 5.6 - Алгоритм средней точки для окружности в разделе «Блок-схемы»

Рисунки 5.7 - 5.14 демонстрируют функционирование основного модуля приложения - визуализатора. На рисунке 5.7 - построение 8-связной прямой по ее уравнению, разрешение = 150. На рис. 5.8 - процесс построения 4-связной линии по уравнению, расширение сетки 100.

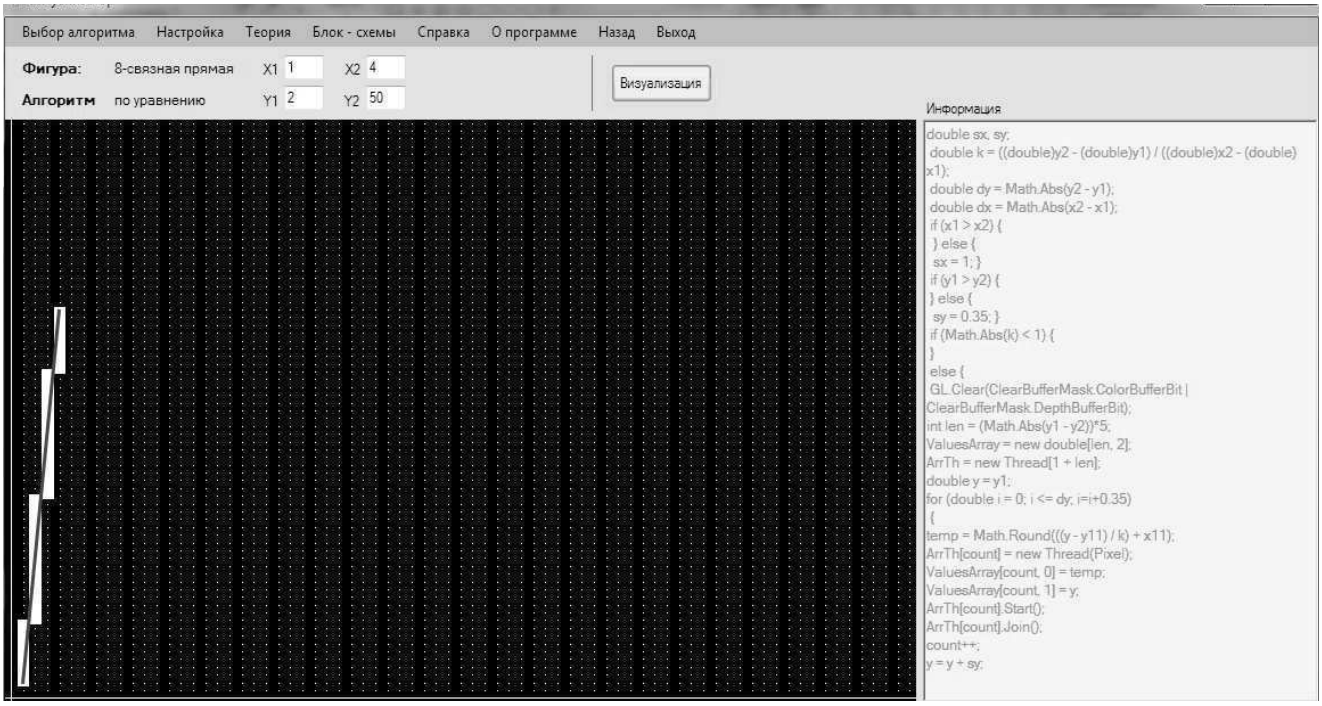

Рисунок 5.7 - Визуализация построения 8-связной прямой по ее уравнению

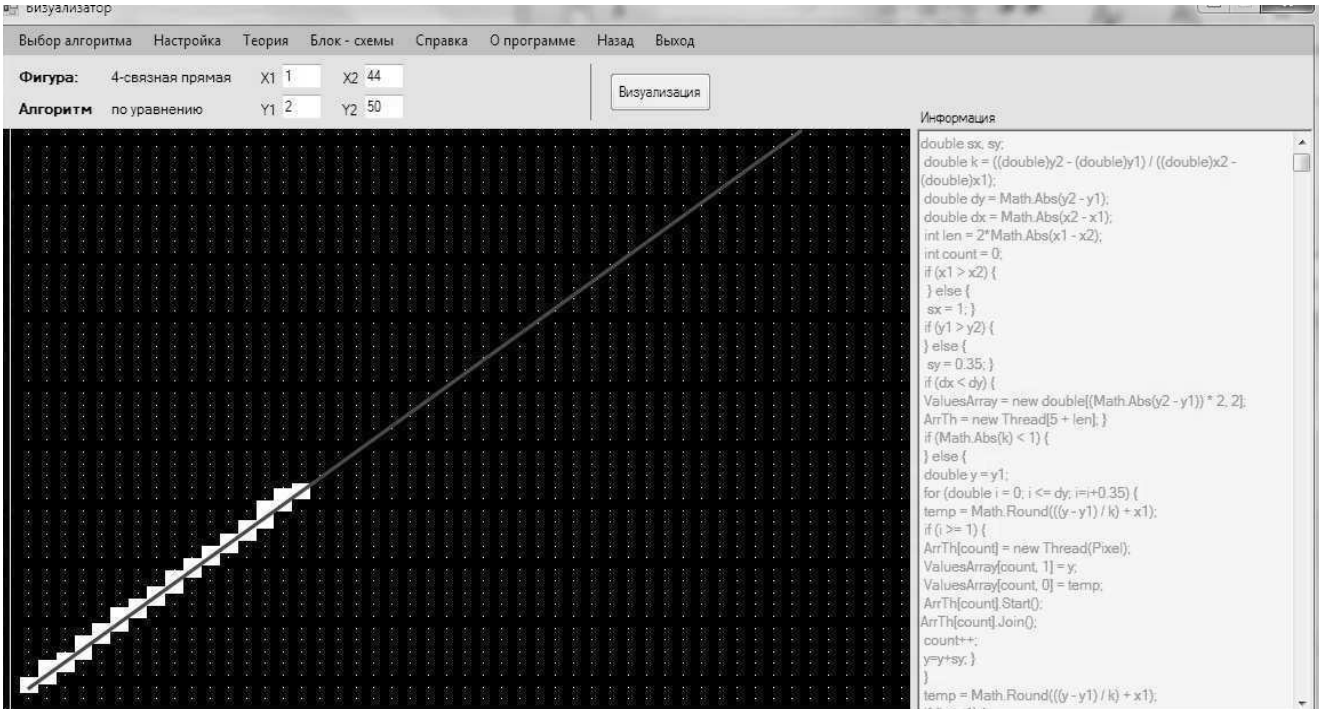

Рисунок 5.8 - Процесс построения 4-связной прямой по ее уравнению

Рисунки 5.9 и 5.10 показывают растеризацию 8-связной и 4-связной прямой по алгоритму Брезенхема. Для 4-связного выбрано разрешение 60 точек, для 8-связного - 100 точек.

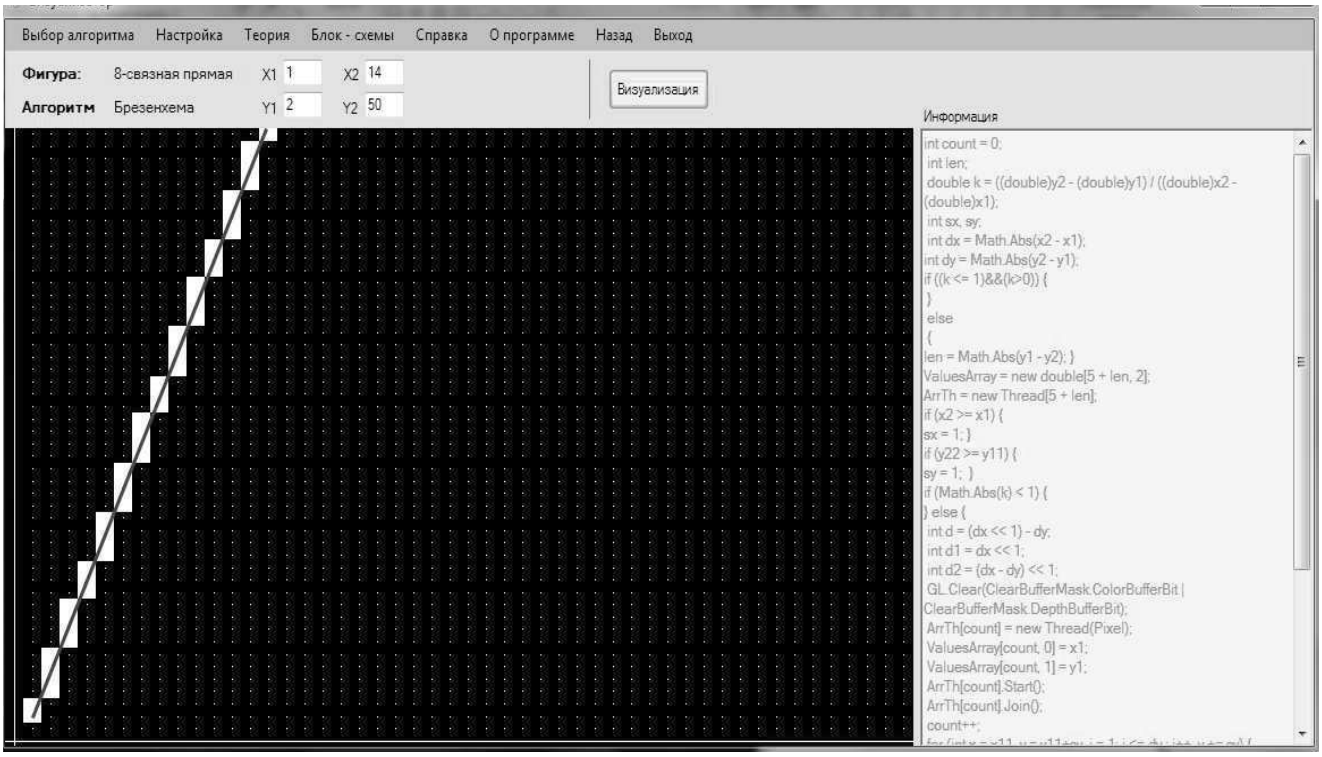

Рисунок 5.9 - Визуализация построения 8-связной прямой по Брезенхему

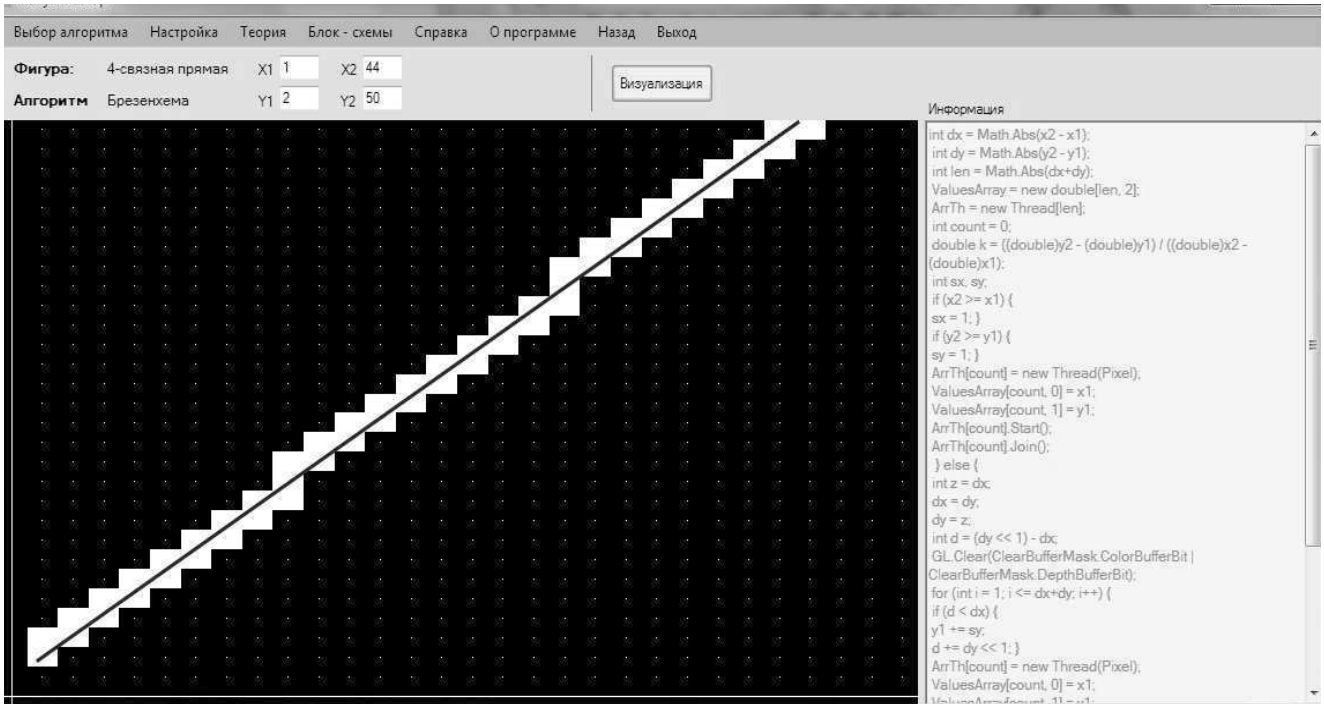

Рисунок 5.10 - Визуализация построения 4-связной прямой по Брезенхему

На рисунке 5.11 представлен скриншот визуализации алгоритма ЦДА без корректировки погрешности, на 5.12 - визуализация алгоритма By. У данных алгоритмов и далее выбрано стандартное разрешение сетки - 100 точек.

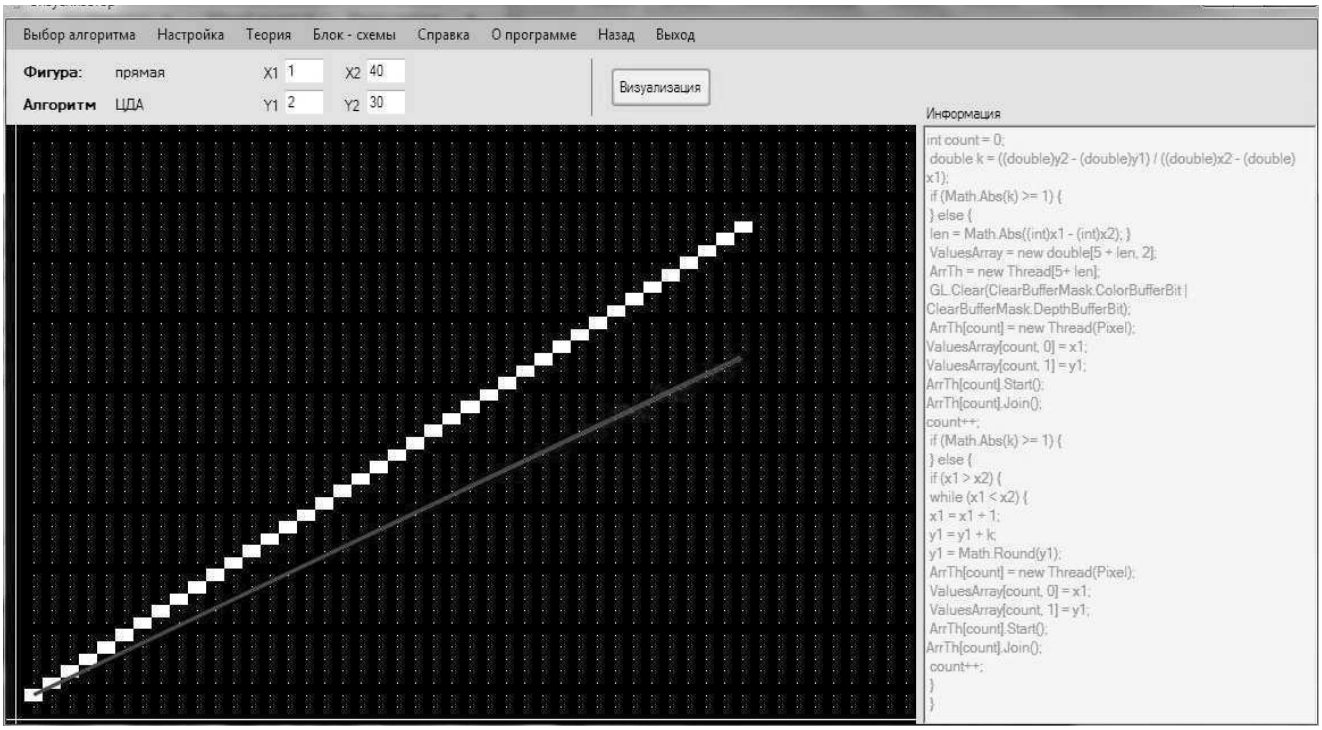

Рисунок 5.11 - Визуализация построения прямой по алгоритму ЦДА

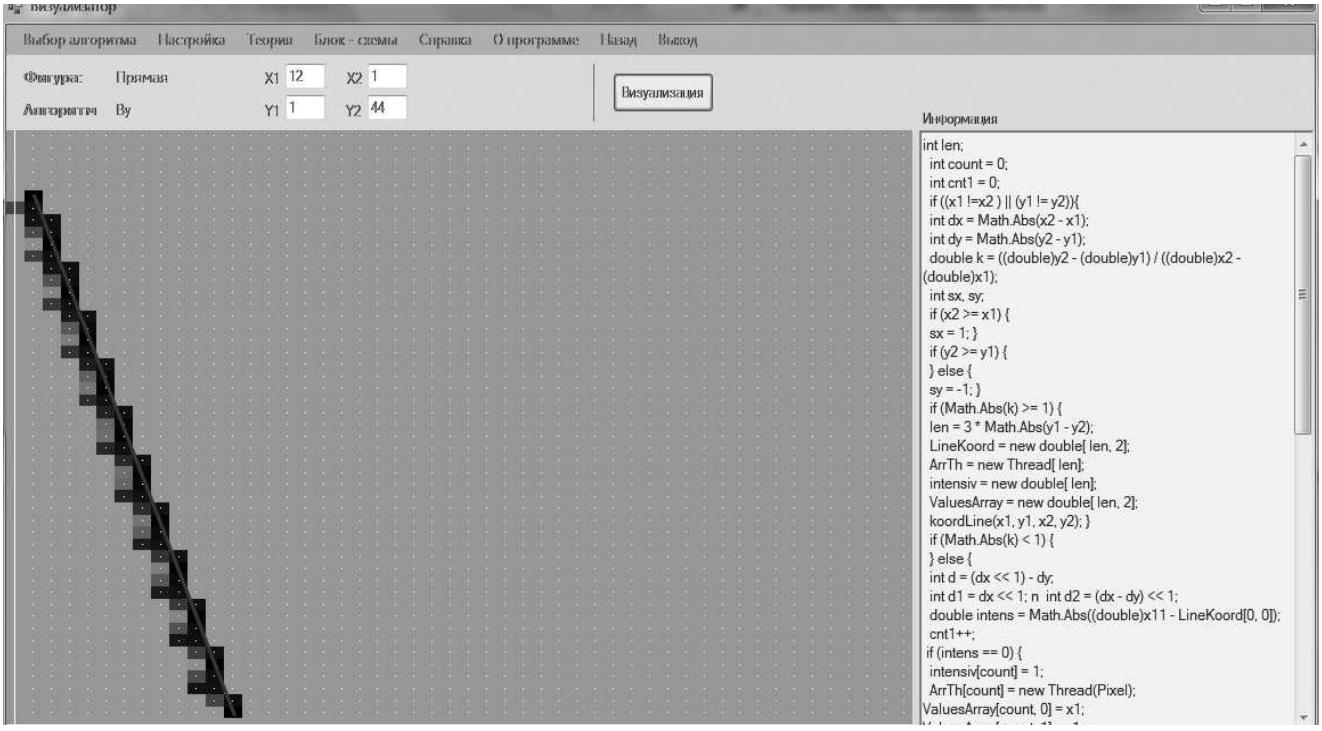

Рисунок 5.12 - Визуализация построения прямой по алгоритму By

На рисунке 5.13 изображена визуализация окружности, на 5.14 - растеризация эллипса по алгоритму средней точки.

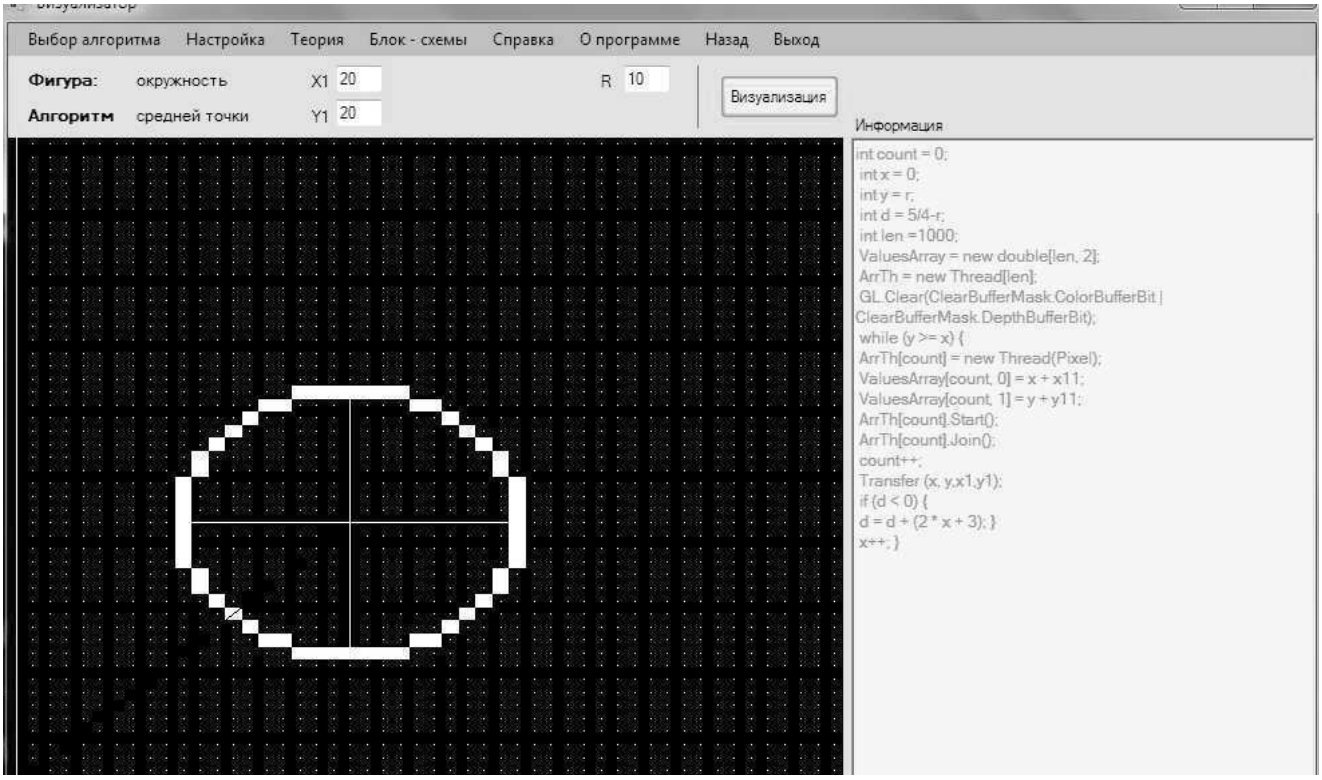

Рисунок 5.13 - Визуализация построения окружности по алгоритму средней точки

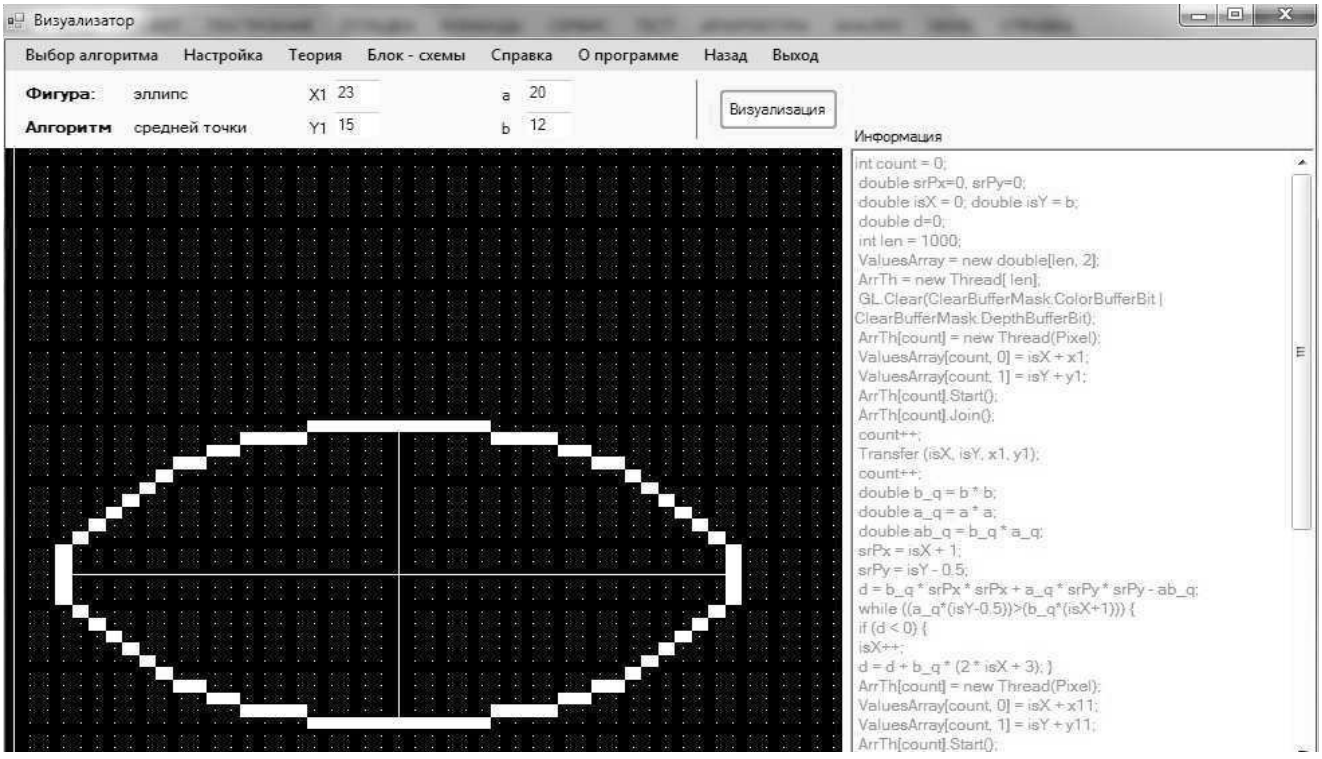

Рисунок 5.14 - Визуализация построения эллипса по алгоритму средней точки

На рисунке 5.15 показан пример окна сообщения с ошибкой при вводе отрицательных значений. Рисунок 5.16 демонстрирует вид окна с настройками.

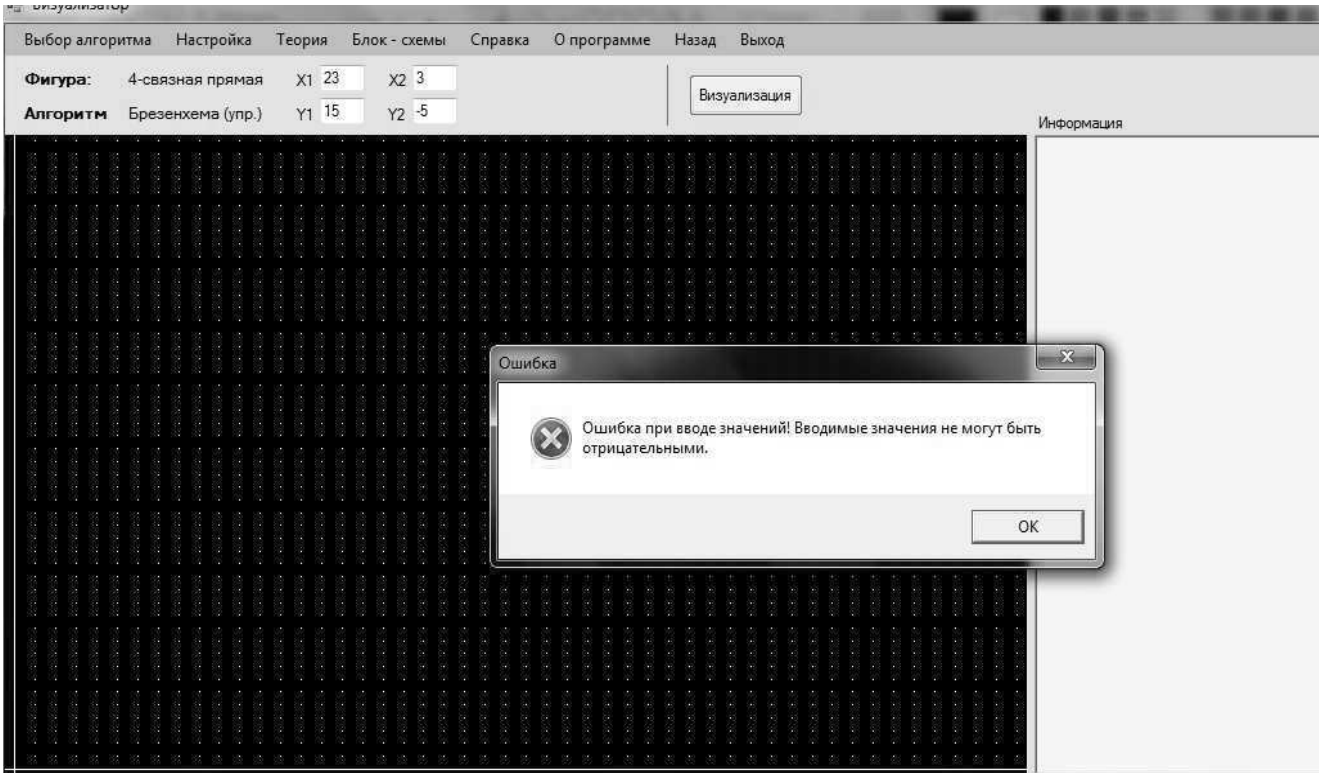

Рисунок 5.15 - Сообщение об ошибке

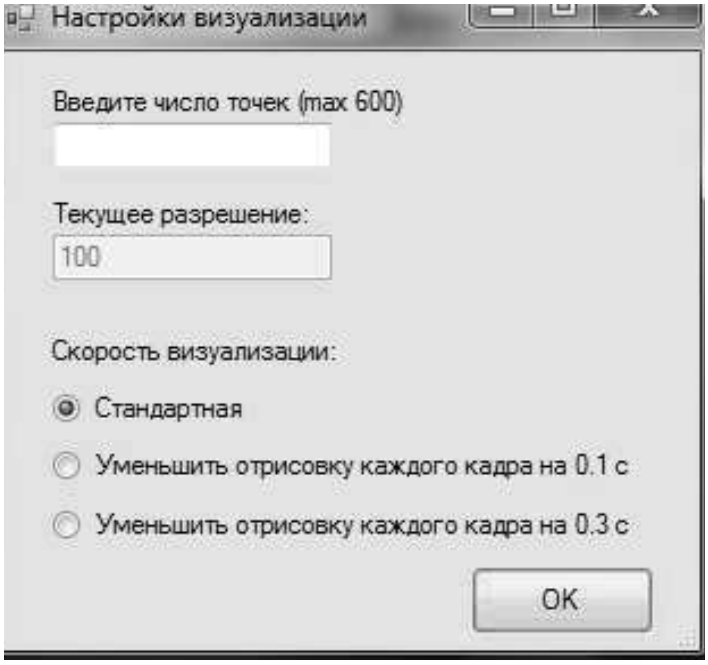

Рисунок 5.16 - Окно с настройками

Заключение

В рамках выпускной квалификационной работы было разработано оконное приложение - «Демонстрационный комплекс программ для изучения алгоритмов растеризации» для Windows 7 x32 SP1.

В ходе выполнения работы решены такие задачи, как:

1) обоснована актуальность разработки;
2) определены требования к системе.
3) проведен обзор аналогов приложения;
4) выбраны инструменты разработки;
5) спроектирована программная система;
6) реализовано само приложение.

В приложение были включены наиболее распространенные растровые алгоритмы:

- алгоритм построения прямой по ее уравнению, вариант 4-связности и 8-связности,
- алгоритм Брезенхема, вариант 4-связности и 8-связности,
- алгоритм ЦДА,
- алгоритм Ву,
- алгоритм средней точки для эллипса;
- алгоритм средней точки для окружности;

А также исправлены такие недостатки найденных аналогов, как:

- отсутствие настроек визуализации,
- отсутствие пояснений,
- отсутствие наглядности построения,
- недостоверность алгоритмов.

На основании учебно-методического пособия Ярош Е.С «Методы и средства представления графической информации» разработаны исходные коды перечисленных выше алгоритмов. Для каждого алгоритма была реализована

анимация построения с выводом выполняемого кода за счёт многопоточного программирования.

Предусмотрено задание и сохранение настроек визуализации (скорость, разрешение сетки), которые в последующем запуске программы восстанавливаются. Реализован ввод исходных данных. При некорректных данных обрабатывается исключение, пользователь получает информационное окно с указанием ошибки. Для алгоритмов построения прямых сделана отрисовка «идеальной» линии.

Кроме модуля визуализации разработаны модули:

- теоретический материал;
- графический материал - блок-схемы.

Модули используют данные из учебного методического пособия Ярош Е.С «Методы и средства представления графической информации». Организована навигация по всему приложению и создан эргономичный интерфейс.

Демонстрационный комплекс программ написан на языке программирования C#. Для работы с графикой использовались внешние библиотеки - OpenTK (OpenGL 4.0) и Tao Framework. Разработка приложения закончена и передана заказчику.

Некоторые перспективы развития: данный продукт можно доработать до универсального учебного пособия по машинной графике, добавив многие другие темы для изучения и визуализации (например, закрашивание, отсечение отрезков, удаление невидимых линии и т.д.).

Библиографический список

1 Визуализация алгоритма Брезенхема. - https://www.twirpx.com/file/620172/.

2 Визуализация алгоритма Ву. - https://ru.wikipedia.org/wiki/Алгоритм_Ву#/media/File:LineXiaolinWu.gif.

3 Исследования алгоритмов генерации отрезков: визуализация несимметричного алгоритма ЦДА и алгоритма Брезенхема. - https://www.twirpx.com/file/598259/.

4 Компьютерная графика: алгоритм Брезенхема для генерации окружности. - http://grafika.me/node/28.

5 Компьютерная графика: алгоритм Брезенхема генерации эллипса. - http://grafika.me/node/661.

6 Частный сайт разработчика программного обеспечения из Германии. - http://www.sunshine2k.de/java.html#bresenham.

7 Статья «Computer Graphics with OpenGL» С. М. Алзахрани. - http://www.c2learn.com/lecture_notes/Graphics/Graphics%20with%20OpenGL/7%20LineDDA_Salha%20Alzahrani.pdf.

8 Международный журнал исследований в области науки и технологий. - http://www.ijrst.com/images/short_pdf/1426575098_kislay_anand.pdf.

9 Проект разработчика архитектуры программного обеспечения из Инновационного центра в Индии. - https://www.codeproject.com/Articles/13360/Antialiasing-Wu-Algorithm.

10 TADVISER: Операционные системы (мировой рынок). - http://www.tadviser.ru/index.php/Статья:Операционные_системы_(мировой_рынок).

11 Сравнительный анализ DirectX 11, OpenGL и Vulkan - http://www.hardwareluxx.ru/index.php/artikel/hardware/grafikkarten/37520-directx-11-vulkan-opengl.html.

12 Шилдт, Г. C# 4.0 полное руководство/ Г. Шилдт - М.: Бином, 2011. - 754 с.

13 ИНТУИТ: что такое The UML - https://www.intuit.ru/studies/courses/1007/229/lecture/5952?page=2.

14 OpenGL - официальный сайт. Информация о GLUT. - https://www.opengl.org/resources/libraries/glut/.

15 Tao Framework. - https://ru.wikipedia.org/wiki/Tao_Framework.

16 Ярош, Е.С. Методы и средства представления графической информации: учебное пособие/ Е.С. Ярош. - Челябинск: Издательский центр ЮУрГУ, 2017. - 230 с.

17 Горнаков, С.Г. Инструментальные средства программирования и отладки шейдеров в DirectX и OpenGL/ С.Г. Горнаков - СПб.: БХВ-Петербург, 2005. - 240 с.