МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №1

по дисциплине: Компьютерная графика

тема: «Растровые алгоритмы»

Выполнил: ст. группы ПВ-223

Дмитриев Андрей Александрович

Проверил:

Осипов Олег Васильевич

Белгород 2024 г.

# Содержание:

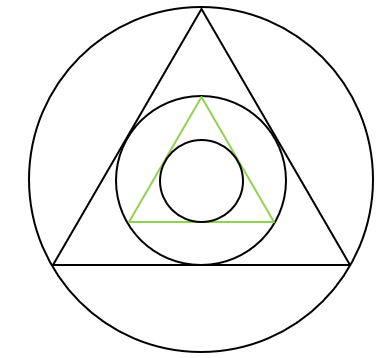
1. Название темы.
2. Цель работы.
3. Постановка задачи.
4. Вывод необходимых геометрических формул для построения изображения.
5. Реализации алгоритмов Брезенхейма для рисования отрезка и окружности.
6. Текст программы для рисования основных фигур.
7. Результат работы программы (снимки экрана).
8. Вывод о проделанной работе.

**Цель работы:** изучение алгоритмов Брезенхейма растеризации графических примитивов: отрезков, окружностей.

**Задачи**:

1. Изучить целочисленные алгоритмы Брезенхейма для растеризации окружности и линии.
2. Разработать алгоритм и составить программу для построения на экране изображения в соответствии с номером варианта (по журналу старосты). В качестве исходных данных взять указанные в таблице №1

Вариант 2:



Для выполнения варианта необходимо представить следующие фигуры:

* Круг
* Треугольник. Также нужно реализовать растрирование отрезков и вращение.

Алгоритм для отрисовки круга предоставлен в теоретической части лабораторной работы:

D(x,y) < 0 – точка внутри окружности, D(x,y) > 0 – вне круга, D(x,y) = 0 – точка лежит на линии круга. Пока x < y, выбираем для закраски из двух пикселей, если D(x,y) > -D(x,y-1), то закрашиваем пиксель (x,y-1), иначе закрашиваем пиксель (x,y). Это справедливо для 1/8 части круга, далее части можно копировать и получить целый круг. Измененный код:

// Рисование окружности  
void drawCircle(PixelCoord p0, int radius, Color color) {  
 int x = 0, y = radius;  
  
 int D = y \* y - (y - 1) \* (y - 1) ;  
 while (x < y)  
 {  
 // Если ближе точка (x, y - 1), то смещаемся к ней  
 if (D > 0) {  
 y--;  
 D -= 4 \* y;  
 }  
 // Перенос и отражение вычисленных координат на все октанты окружности  
  
 setPixel({p0.x + x, p0.y + y}, color);  
 setPixel({p0.x + x, p0.y - y}, color);  
 setPixel({p0.x + y, p0.y + x}, color);  
 setPixel({p0.x + y, p0.y - x}, color);  
 setPixel({p0.x - x, p0.y + y}, color);  
 setPixel({p0.x - x, p0.y - y}, color);  
 setPixel({p0.x - y, p0.y + x}, color);  
 setPixel({p0.x - y, p0.y - x}, color);  
 x++;  
 D += 4 \* x - 2;  
 }  
}

Алгоритм отрисовки линии следует из канонического уравнения прямой . Преобразуем, получим функцию . Для случая, когда находится правее и выше точки и , каждую следующую точку определяем по функции, если , значит , иначе Для остальных случаев меняем местами точки.

Треугольник можно было бы отрисовать, опираясь на начальную и конечную точку области, но решено сделать через *центр* и *расстояние до вершины*. Для построения необходимо найти 3 точки. Первую находим просто – к добавляем *расстояние до вершины*. Для двух других нужно найти *половину стороны* из *расстояния до вершины* (обозначим за ) – это . Теперь от центра  откладываем точки и .

Чтобы повернуть точку вокруг центра , нужно выполнить преобразование:

Функция отрисовки равностороннего треугольника:

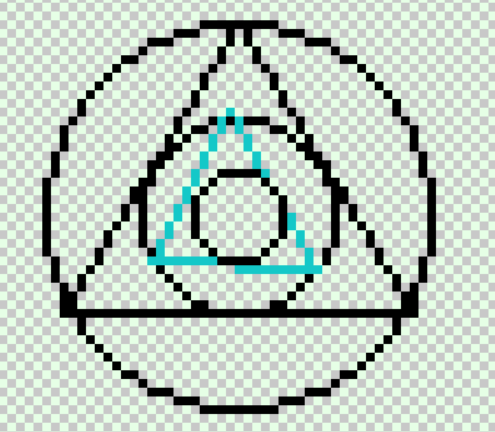
void drawEqualityTriangle(  
 Frame &frame,  
 PixelCoord center,  
 int heightFromCenter,  
 float angle,  
 Color color  
) {  
 float halfSide = heightFromCenter \* cos(M\_PI / 6);  
 PixelCoord vertices[] = {  
 {  
 center.x,  
 center.y - heightFromCenter  
 },  
 {  
 int(center.x - halfSide),  
 center.y + heightFromCenter / 2  
 },  
 {  
 int(center.x + halfSide),  
 center.y + heightFromCenter / 2  
 }  
 };  
  
 for (PixelCoord &p: vertices) {  
 PixelCoord prevP = p;  
 p.x = (prevP.x - center.x) \* cos(angle)  
 - (prevP.y - center.y) \* sin(angle)  
 + center.x;  
 p.y = (prevP.x - center.x) \* sin(angle)  
 + (prevP.y - center.y) \* cos(angle)  
 + center.y;  
 }  
  
 for (int i = 0; i < 3; i++) {  
 int i2 = (i + 1) % 3;  
 frame.drawLine(  
 {  
 int(vertices[i].x + 0.5f),  
 int(vertices[i].y + 0.5f)  
 },  
 {  
 int(vertices[i2].x + 0.5f),  
 int(vertices[i2].y + 0.5f)  
 },  
 color  
 );  
 }  
}

Далее размещаем элементы, относительно друг-друга, они будут меньше в два раза.

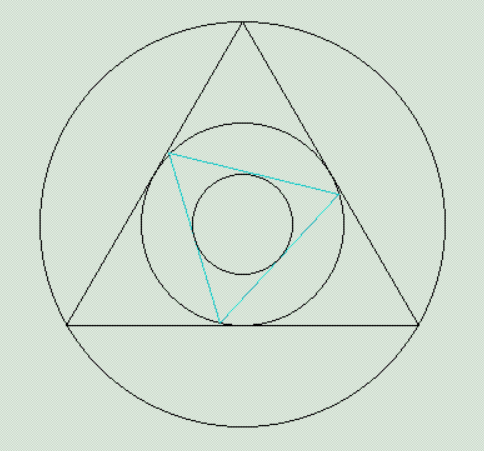
Код полного решения:

void draw(Frame &frame) {  
 Assets::drawChessBg(frame);  
  
 int W = frame.width, H = frame.height;  
  
 float scaledSize = 7.0f / 8 \* ((W < H) ? W - 1 : H - 1) / sqrt(2);  
 if (scaledSize < 1) return;  
  
 float angle = globalAngle;  
 PixelCoord center = {W / 2, H / 2};  
  
 frame.drawCircle(  
 center,  
 scaledSize / 2,  
 staticPartColor  
 );  
  
 drawEqualityTriangle(  
 frame,  
 {frame.width / 2, frame.height / 2},  
 scaledSize / 2,  
 0,  
 staticPartColor  
 );  
  
 frame.drawCircle(  
 center,  
 scaledSize / 4,  
 staticPartColor  
 );  
  
 drawEqualityTriangle(  
 frame,  
 {frame.width / 2, frame.height / 2},  
 scaledSize / 4,  
 -angle,  
 movablePartColor  
 );  
  
 frame.drawCircle(  
 center,  
 scaledSize / 8,  
 staticPartColor  
 );  
}

Внешний вид фигур при низком разрешении:



Внешний вид фигур при высоком разрешении:

****

При низком разрешении заметен дефект. При вращении треугольника, его стороны некорректно прилегают к окружности. Возникает это, вероятно из-за округления вещественных чисел до целых.

Код программы:

Frame.h

// Угол поворота фигуры  
float globalAngle = 0;  
  
// Координаты последнего пикселя, который выбрал пользователь  
struct {  
 int x, y;  
} globalClickedPixel = {-1, -1};  
  
struct PixelCoord {  
 int x, y;  
};  
  
// Cтруктура для задания цвета  
typedef struct tagCOLOR {  
 unsigned char RED; // Компонента красного цвета  
 unsigned char GREEN; // Компонента зелёного цвета  
 unsigned char BLUE; // Компонента синего цвета  
 unsigned char ALPHA; // Прозрачность (альфа канал)  
  
 tagCOLOR() : RED(0), GREEN(0), BLUE(0), ALPHA(255) {}  
  
 tagCOLOR(  
 unsigned char red,  
 unsigned char green,  
 unsigned char blue,  
 unsigned char alpha = 255  
 ) : RED(red), GREEN(green), BLUE(blue), ALPHA(alpha) {}  
  
} Color;  
  
template<typename T>  
void swap(T &a, T &b) {  
 T t = a;  
 a = b;  
 b = t;  
}  
  
// Буфер кадра  
class Frame {  
 // Указатель на массив пикселей  
 // Буфер кадра будет представлять собой матрицу, которая располагается в памяти в виде непрерывного блока  
 Color \*pixels;  
  
 // Указатели на строки пикселей буфера кадра  
 Color \*\*matrix;  
  
public:  
  
 // Размеры буфера кадра  
 int width, height;  
  
 Frame(int \_width, int \_height) : width(\_width), height(\_height) {  
 int size = width \* height;  
  
 // Создание буфера кадра в виде непрерывной матрицы пикселей  
 pixels = new Color[size];  
  
 // Указатели на строки пикселей запишем в отдельный массив  
 matrix = new Color \*[height];  
  
 // Инициализация массива указателей  
 for (int i = 0; i < height; i++) {  
 matrix[i] = pixels + i \* width;  
 }  
 }  
  
  
 // Задаёт цвет color пикселю с координатами (x, y)  
 void setPixel(PixelCoord p, Color color) {  
 matrix[p.y][p.x] = color;  
 }  
  
 // Возвращает цвет пикселя с координатами (x, y)  
 Color getPixel(PixelCoord p) {  
 return matrix[p.y][p.x];  
 }  
  
  
 // Рисование окружности  
 void drawCircle(PixelCoord p0, int radius, Color color) {  
 int x = 0, y = radius;  
  
 int D = y \* y - (y - 1) \* (y - 1) ;  
 while (x < y)  
 {  
 // Если ближе точка (x, y - 1), то смещаемся к ней  
 if (D > 0) {  
 y--;  
 D -= 4 \* y;  
 }  
 // Перенос и отражение вычисленных координат на все октанты окружности  
  
 setPixel({p0.x + x, p0.y + y}, color);  
 setPixel({p0.x + x, p0.y - y}, color);  
 setPixel({p0.x + y, p0.y + x}, color);  
 setPixel({p0.x + y, p0.y - x}, color);  
 setPixel({p0.x - x, p0.y + y}, color);  
 setPixel({p0.x - x, p0.y - y}, color);  
 setPixel({p0.x - y, p0.y + x}, color);  
 setPixel({p0.x - y, p0.y - x}, color);  
 x++;  
 D += 4 \* x - 2;  
 }  
 }  
  
 // Рисование отрезка  
void drawLine(PixelCoord p1, PixelCoord p2, Color color) {  
 int x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y;  
  
 int dy = y2 - y1, dx = x2 - x1;  
 if (dx == 0 && dy == 0) {  
 matrix[y1][x1] = color;  
 return;  
 }  
  
 int D = dy \* x1 - dy \* x2 - dx \* (y1 - y2);  
  
 if (abs(dx) > abs(dy)) {  
 if (x2 < x1) {  
 // Обмен местами точек (x1, y1) и (x2, y2)  
 swap(x1, x2);  
 swap(y1, y2);  
 dx = -dx;  
 dy = -dy;  
 }  
  
 int D = dy \* (x1 - x2) - dx \* (y1 - y2);  
  
 int y, p = 0;  
 for (int x = x1; x <= x2; x++) {  
 y = (p + D) / dx + y1;  
 p += dy;  
 matrix[y][x] = color;  
 }  
 } else {  
 if (y2 < y1) {  
 // Обмен местами точек (x1, y1) и (x2, y2)  
 swap(x1, x2);  
 swap(y1, y2);  
 dx = -dx;  
 dy = -dy;  
 }  
  
 int x, p = 0;  
 for (int y = y1; y <= y2; y++) {  
 x = (p + D) / dy + x1;  
 p += dx;  
 matrix[y][x] = color;  
 }  
 }  
}  
  
 ~Frame() {  
 delete[]pixels;  
 delete[]matrix;  
 }  
  
};

Painter.h

class Painter\_Var2 {  
public:  
 void draw(Frame &frame) {  
 Assets::drawChessBg(frame);  
  
 int W = frame.width, H = frame.height;  
  
 float scaledSize = 7.0f / 8 \* ((W < H) ? W - 1 : H - 1) / sqrt(2);  
 if (scaledSize < 1) return;  
  
 float angle = globalAngle;  
 PixelCoord center = {W / 2, H / 2};  
  
 frame.drawCircle(  
 center,  
 scaledSize / 2,  
 staticPartColor  
 );  
  
 drawEqualityTriangle(  
 frame,  
 {frame.width / 2, frame.height / 2},  
 scaledSize / 2,  
 0,  
 staticPartColor  
 );  
  
 frame.drawCircle(  
 center,  
 scaledSize / 4,  
 staticPartColor  
 );  
  
 drawEqualityTriangle(  
 frame,  
 {frame.width / 2, frame.height / 2},  
 scaledSize / 4,  
 -angle,  
 movablePartColor  
 );  
  
 frame.drawCircle(  
 center,  
 scaledSize / 8,  
 staticPartColor  
 );  
 }  
  
private:  
 Color movablePartColor = {20, 200, 200};  
 Color staticPartColor = {0, 0, 0};  
  
 void drawEqualityTriangle(  
 Frame &frame,  
 PixelCoord center,  
 int heightFromCenter,  
 float angle,  
 Color color  
 ) {  
 float halfSide = heightFromCenter \* cos(M\_PI / 6);  
 PixelCoord vertices[] = {  
 {  
 center.x,  
 center.y - heightFromCenter  
 },  
 {  
 int(center.x - halfSide),  
 center.y + heightFromCenter / 2  
 },  
 {  
 int(center.x + halfSide + 1),  
 center.y + heightFromCenter / 2  
 }  
 };  
  
 for (PixelCoord &p: vertices) {  
 PixelCoord prevP = p;  
 p.x = (prevP.x - center.x) \* cos(angle)  
 - (prevP.y - center.y) \* sin(angle)  
 + center.x;  
 p.y = (prevP.x - center.x) \* sin(angle)  
 + (prevP.y - center.y) \* cos(angle)  
 + center.y;  
 }  
  
 for (int i = 0; i < 3; i++) {  
 int i2 = (i + 1) % 3;  
 frame.drawLine(  
 {  
 int(vertices[i].x + 0.5f),  
 int(vertices[i].y + 0.5f)  
 },  
 {  
 int(vertices[i2].x + 0.5f),  
 int(vertices[i2].y + 0.5f)  
 },  
 color  
 );  
 }  
 }  
};

**Вывод:** в ходе работы изучены алгоритмы Брезенхейма растеризации графических примитивов: отрезков, окружностей. С помощью отрезков были построены и анимированы многоугольники. Собраны в изображение требуемое по варианту.