Министерство образования Республики Беларусь
Учреждение образования “Белорусский государственный университет информатики и радиоэлектроники”

Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий
Дисциплина: Графический интерфейс интеллектуальных систем

Отчёт к лабораторной работе №1

Выполнил: Гафаров М.С.
Группа: 221702
Проверил: Жмырко А.В.

Минск 2024

Лабораторная работа №1

Алгоритмы построения отрезков

Цель: изучить основные алгоритмы разложения отрезков в растр

Задание: разработать элементарный графический редактор, реализующий построение отрезков с помощью алгоритма ЦДА, целочисленного алгоритма Брезенхема и алгоритма Ву. Вызов способа генерации отрезка задаётся из пункта меню и доступно через панель инструментов “Отрезки”. В редакторе кроме режима генерации отрезков в пользовательском окне должен быть предусмотрен отладочный режим, где отображается пошаговое решение на дискретной сетке.

Теоретические сведения:

Разложение в растр – процесс определения пикселей, наилучшим образом аппроксимирующих заданный отрезок.

Общие требования к алгоритмам рисования отрезков:

1. Отрезки должны выглядеть прямыми, начинаться и заканчиваться в заданных точках.
2. Яркость должна быть постоянной и не зависеть от длины и наклона.
3. Условие к быстродействию алгоритма.

В данной лабораторной работе реализованы 3 алгоритма разложения отрезка в растр: алгоритм ЦДА, алгоритм Брезенхема, алгоритм Ву.

Программная реализация:

Алгоритм ЦДА:

Начальные данные для алгоритма:

```
def Signum(argument):  2 usages
    if argument > 0:
        return 1
    elif argument == 0:
        return 0
    else:
        return -1


def DDA(f_coord, s_coord):  3 usages
    length = max(abs(float(s_coord[0]) - float(f_coord[0])), abs(float(s_coord[1]) - float(f_coord[1])))
    dx = (float(s_coord[0]) - float(f_coord[0])) / length
    dy = (float(s_coord[1]) - float(f_coord[1])) / length
    x1 = float(f_coord[0]) + 0.5 * Signum(dx)
    y1 = float(f_coord[1]) + 0.5 * Signum(dy)
    return length, dx, dy, x1, y1
```

Основной цикл алгоритма и построения графика:

```
def graphic(length, dx, dy, x1, y1):  1 usage
    x_values = []
    y_values = []
    for i in range(0, int(length)):
        x = x1 + dx * i
        y = y1 + dy * i
        x_values.append(x)
        y_values.append(y)
        print(x, y)

    plt.figure(figsize=(12, 10))
    plt.plot(x_values, y_values, marker='o', label='точки DDA')
    plt.title('Алгоритм DDA')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid(True)
    plt.legend()
    plt.show()
```

Алгоритм Брезенхема:

Начальные данные и основной цикл алгоритма:

```python
def brezenhem(f_coord, s_coord):  3 usages
    x1 = int(f_coord[0])
    y1 = int(f_coord[1])
    dx = int(s_coord[0]) - x1
    dy = int(s_coord[1]) - y1
    x, y = x1, y1
    if dx >= dy:
        e = 2*dy - dx
    elif dy > dx:
        e = 2*dx - dy
    x_values = []
    x_values.append(x1)
    y_values = []
    y_values.append(y1)
    if dx >= dy:
        for i in range(1, dx+1):
            if e >= 0:
                y += 1
                e -= 2*dx
            x += 1
            e += 2*dy
            x_values.append(x)
            y_values.append(y)
    elif dy > dx:
        for i in range(1, dy+1):
            if e >= 0:
                y += 1
                e -= 2*dy
            x += 1
            e += 2*dx
            x_values.append(x)
            y_values.append(y)
    return x_values, y_values
```

Построение графика:

```python
def graphic(x_values, y_values):  1 usage
    for i in range(len(x_values)):
        plt.plot([x_values[i], x_values[i] + 1], [y_values[i], y_values[i]], linestyle='-.', color='blue')
        plt.plot([x_values[i], x_values[i] + 1], [y_values[i] + 1, y_values[i] + 1], linestyle='-.', color='blue')
        plt.plot([x_values[i], x_values[i]], [y_values[i], y_values[i] + 1], linestyle='-.', color='blue')
        plt.plot([x_values[i] + 1, x_values[i] + 1], [y_values[i], y_values[i] + 1], linestyle='-.', color='blue')

    lower_left_x = x_values[::2]
    lower_left_y = y_values[::2]
    lower_left_x.append(x_values[-1] + 1)
    lower_left_y.append(y_values[-1] + 1)
    plt.plot(lower_left_x, lower_left_y, color='red', linestyle='-', linewidth=2, label='отрезок')

    plt.title("Отрезок по алгоритму Брезенхема")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid(True)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()
```

Алгоритм By:

Начальные данные и основной цикл алгоритма:

```python
def vu(f_coord, s_coord):  3 usages
    x0, y0 = int(f_coord[0]), int(f_coord[1])
    x1, y1 = int(s_coord[0]), int(s_coord[1])
    dx, dy = x1 - x0, y1 - y0
    values = []
    if abs(dx) >= abs(dy):
        if x0 > x1:
            x0, x1 = x1, x0
            y0, y1 = y1, y0
        k = dy / dx       #угловой коэффициент
        for x in range(x0, x1 + 1):
            temp_list1 = []
            temp_list2 = []
            y = y0 + k*(x-x0)
            y_int = int(y)
            y_dec = y - y_int
            In_bot = 1 - y_dec
            In_top = y_dec
            temp_list1.append(x)
            temp_list1.append(y_int)
            temp_list1.append(In_bot)
            if x+1 <= x1 or y+1 <= y1:
                temp_list2.append(x)
                temp_list2.append(y_int + 1)
                temp_list2.append(In_top)
                values.append(temp_list1)
                values.append(temp_list2)
            else:
                values.append(temp_list1)
```

```
            values.append(temp_list1)
    else:
        if y0 > y1:
            x0, x1 = x1, x0
            y0, y1 = y1, y0
        k = dx / dy       #угловой коэффициент
        for y in range(y0, y1 + 1):
            temp_list1 = []
            temp_list2 = []
            x = x0 + k*(y-y0)
            x_int = int(x)
            x_dec = x - x_int
            In_bot = 1 - x_dec
            In_top = x_dec
            temp_list1.append(x_int)
            temp_list1.append(y)
            temp_list1.append(In_bot)
            if x+1 <= x1 or y+1 <= y1:
                temp_list2.append(x_int + 1)
                temp_list2.append(y)
                temp_list2.append(In_top)
                values.append(temp_list1)
                values.append(temp_list2)
            else:
                values.append(temp_list1)
    return values
```

Построение графика:

```
def graphic(values):  1 usage
    plt.figure(figsize=(8, 8))
    x_start, y_start = values[0][0], values[0][1]
    x_end, y_end = values[-1][0], values[-1][1]
    plt.plot([x_start, x_end], [y_start, y_end], color='red', linewidth=1.5, label='Отрезок')
    for value in values:
        x, y, intensity = value
        plt.fill(
            [x, x + 1, x + 1, x],
            [y, y, y + 1, y + 1],
            color='black',
            alpha=intensity
        )
    plt.axis('equal')
    plt.grid(visible=True, color='lightgray', linestyle='--', linewidth=0.5)
    plt.show()
```

Вывод: в результате лабораторной работы был спроектирован элементарный графический редактор для построения отрезков. Были изучены и запрограммированы такие алгоритмы построения отрезков, как алгоритм ЦДА, алгоритм Брезенхема и алгоритм Ву.