Оглавление

[ЛР 3 1](#_Toc177728895)

[ЛР 4 2](#_Toc177728896)

[Виды фигур 3](#_Toc177728897)

[Контрольные вопросы 3](#_Toc177728898)

# ЛР 3

Вывод и аффинные преобразования полигональной модели фигуры (без заливки) на основе алгоритма Брезенхема для отрезка.

Для вершин использовать 2 цвета (поочередно).

Стороны полигона выводятся с интерполяцией цветов.

Ориентации – горизонтальная или вертикальная – задается случайным образом.

Программа содержит параметр, регулирующий ширину линии (ребра полигона).

Рисунок сохраняется в текстовый файл, позволяющий после загрузки воспроизвести изображение.

Программа содержит процедуру, читающую этот файл и воспроизводящую по прочитанным данным рисунок.

Виды преобразований координат: поворот, перенос в центр окна вывода, масштабирование – уменьшение в 2 раза по обеим осям (рис. 1-3); рисунки снабжаются заголовком, информирующем об использованных в программе параметрах.

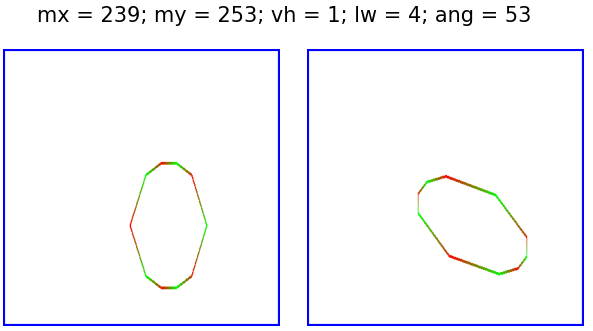


Рис. 1. Поворот

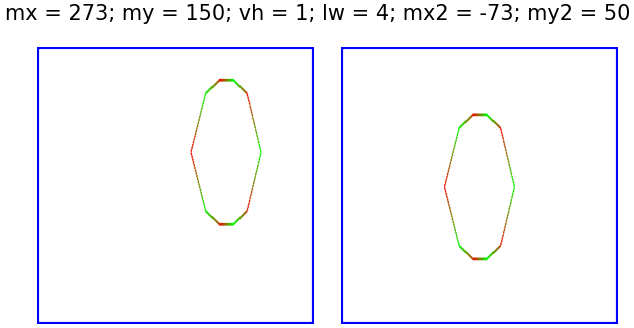


Рис. 2. Перемещение

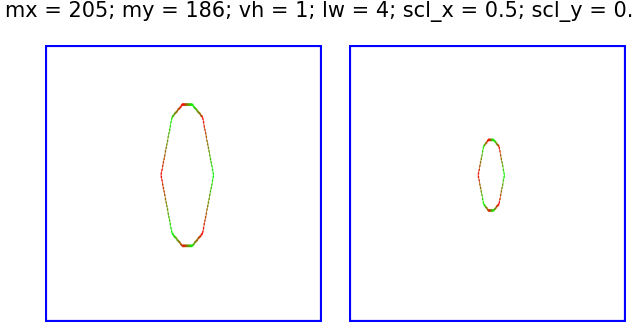


Рис. 3. Масштабирование (сжатие)

Употребляются матрицы аффинных преобразований для однородных координат. Рисунки «до» и «после» преобразований выводятся одновременно.

# ЛР 4

Вывод и заливки с интерполяцией цветов и без нее полигональной модели фигуры на основе алгоритма Брезенхема для отрезка.

В остальном задача та же, что и в ЛР 3, за следующими исключениями:

* вывод выполняется на случайном фоне;
* используется только поворот (рис. 4 и 5).

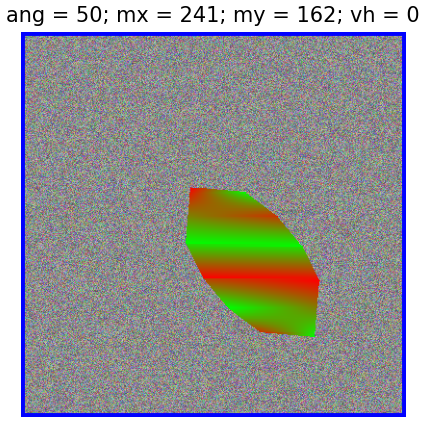


Рис. 4. Заливка с интерполяцией цветов

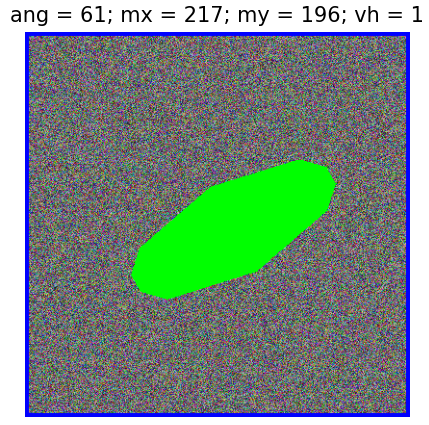


Рис. 5. Заливка без интерполяции цветов

# Виды фигур

* А-05 – эллипс;
* А-13 – парабола;
* А-14, 16 – четырехугольник;
* А-18 – гипербола.

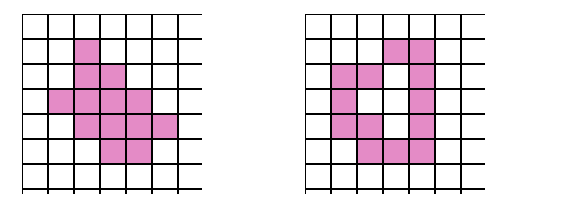
# Контрольные вопросы

1. 4- и 8-связность.

**Определение 4-связной компоненты**: множество чёрных пикселей **P** является ***4-связной компонентой***, если для каждой пары пикселей **pi**и **pj** в **P** существует последовательность пикселей ***pi, ..., pj***, такая, что:  
  
a) все пиксели в последовательности находятся во множестве **P**, т.е. являются чёрными, и  
  
b) каждые два пикселя, которые находятся ***рядом в последовательности***, являются ***4-соседями***

#### **Примеры 4-связных паттернов**

На схемах ниже показаны примеры 4-связных паттернов:

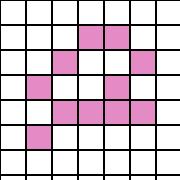


#### **8-связность**

Когда можно сказать, что заданное множество чёрных пикселей ***8-связное***? Во-первых, нам нужно определить понятие ***8-соседа*** (также называемого ***непрямым соседом***):  
  
**Определение 8-соседа**: пиксель **Q** является ***8-соседом*** (или просто ***соседом***) заданного пикселя **P**, если **Q** и **P** имеют общее ребро или вершину. 8-соседи заданного пикселя **P** составляют окрестность Мура этого пикселя.  
  
**Определение 8-связной компоненты**: множество чёрных пикселей **P** является ***8-связной компонентой*** (или просто ***связной компонентой***), если для каждой пары пикселей **pi**и **pj** в **P** существует последовательность пикселей ***pi, ..., pj***, такая, что:  
  
a) все пиксели в последовательности находятся во множестве **P**, т.е. являются чёрными, и  
  
b) каждые два пикселя, которые находятся ***рядом в этой последовательности***, являются ***8-соседями***  
  
**Примечание**: все 4-связные паттерны являются 8-связными, т.е. 4-связные паттерны являются подмножеством множества 8-связных паттернов. С другой стороны, 8-связный паттерн может и не являться 4-связным.

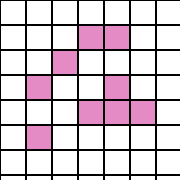
#### **Пример 8-связного паттерна**

На схеме ниже показан паттерн, являющийся 8-связным, но не 4-связным:



#### **Пример паттерна, не являющегося 8-связным:**

На схеме ниже показан пример паттерна, не являющийся 8-связным, т.е. составленный из более чем одной связной компоненты (на схеме показаны три связные компоненты):



1. Обоснование алгоритма Брезенхема для отрезка.

## **Алгоритм Брезенхема.**

Хотя алгоритм Брезенхема был первоначально разработан для цифровых графопостроителей, однако он в равной степени подходит для использования растровыми устройствами с ЭЛТ. Алгоритм выбирает оптимальные растровые координаты для представления отрезка. В процессе работы одна из координат - либо x, либо y (в зависиимости от углового коэффициента) - изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние мы назовем ошибкой.

Алгоритм построен так, что требуется проверить лишь знак этой ошибки. На рис.3.1 это иллюстрируется для отрезка в первом октанте, т.е. для отрезка с угловым коэффициентом, лежащим в диапазоне от 0 до 1. Из рисунка можно заметить, что если угловой коэффициент отрезка из точки (0,0) больше, чем 1/2, то пересечение с прямой x = 1 будет расположено ближе к прямой y = 1, чем к прямой y = 0. Следовательно, точка растра (1,1) лучше аппроксимирует ход отрезка, чем точка (1,0). Если угловой коэффициент меньше 1/2, то верно обратное. для углового кэффициента, равного 1/2, нет какого либо предпочтительного выбора. В данном случае алгоритм выбирает точку (1,1).

|  |
| --- |
|  |

Рис. 3.1. Основная идея алгоритма Брезенхема.

Не все отрезки проходят через точки растра. Подобная ситуация иллюстрируется рис.3.2, где отрезок с тангенсом угла наклона 3/8 сначала походит через точку растра (0,0) и последовательно пересекает три пиксела. Также иллюстрируется вычисление ошибки при представлении отрезка дискретными пикселами.

Параметры:

dx = x1 – x0; dy = y1 – y0; k = dy / dx

Ограничения:

первый квадрант;

x1 > x0; - y1 > y0; 0 ≤ k ≤ 1.

Вариант 1 (тривиальный).

x = x0

пока x ≤ x1:

y = [kx + b] # Целая часть числа

Залить(x, y)

x = x + 1

Вариант 2 (без умножения на k, контроль ci).

x = x0

y = y0

ci = 0

пока x ≤ x1:

Залить(x, y)

x = x + 1

ci = ci + k

если ci > 0.5:

y = y + 1

ci = ci - 1 # см. пояснения ниже

Пояснения:

(1) yi = kxi + b – ci (см. рис. 2)

(2) yi+1 = kxi+1 + b – ci+1 # xi+1 = xi + 1

xi+1 – xi = 1

yi+1 – yi = 1 (ci > 0.5)

(2) – (1): 1 = k – ci+1 + ci

ci+1 = ci + k – 1 (см. выше перед "если" ci = ci + k)

Вариант 3 (замена ci > 0.5 на d > 0).

k = dy / dx # Из d > 0 следует, что d = 2ci – 1

x = x0

y = y0

d = –1

пока x ≤ x1:

Залить(x, y)

x = x + 1

d = d + 2k # см. пояснения ниже

если d > 0:

y = y + 1

d = d – 2 # см. пояснения ниже

Пояснения:

а)

ci = ci + k

2ci = 2ci + 2k

2ci – 1 = 2ci – 1 + 2k, то есть d = d + 2k

б)

ci = ci - 1

2ci = 2ci - 2

2ci - 1 = 2ci - 1 - 2, то есть d = d - 2

Вариант 4 (целочисленный) - алгоритм Брезенхема.

k = dy # Прежний k = dy / dx умножаем на dx

Тогда вместо:

d = –1 имеем d = –dx

d = d – 2 имеем d = d – 2dx

d = d + 2k имеем d = d + 2dy

Алгоритм:

x = x0 # k = dy не нужен

y = y0

d = –dx

dx2 = dx + dx

dy2 = dy + dy

пока x ≤ x1:

Залить(x, y)

x = x + 1

d = d + dy2

если d > 0:

y = y + 1

d = d – dx2

1. Обобщение алгоритма Брезенхема для отрезка.

Входные данные:

x0, y0 – координаты начала отрезка;

x1, y1 – координаты конца отрезка;

vp – массив значений пикселей окна вывода формы (w, h),

где w, h - соответственно ширина и высота окна.

vp = 255.

Выходные данные:

vp – заполненный массив значений пикселей.

Алгоритм:

1. Начало.

2. vp = 255 # Белый цвет

3. steep = abs(y1 - y0) > abs(x1 - x0) # Крутизна

4. Если steep: # Обмен X, Y, если угол наклона отрезка более 45º

x0, y0 = y0, x0 # Обмен

x1, y1 = y1, x1 # Обмен

5. Если x0 > x1: # Приводим к базовой форме алгоритма, в которой x0 < x1

x0, x1 = x1, x0 # Обмен

y0, y1 = y1, y0 # Обмен

6. dx = x1 – x0; dy = abs(y1 – y0)

7. dx2 = dx + dx; dy2 = dy + dy

8. d = –dx

9. y\_step = 1 Если y0 < y1 Иначе –1 # Шаг по Y

10. y = y0; x = x0

11. Пока x ≤ x1:

Если steep: # Нужен обмен

xp, yp = y, x # Обмен

Иначе:

xp, yp = x, y

vp[yp, xp] = 0 # Черный цвет

d = d + dy2

Если d > 0:

y = y + y\_step # 1 или -1

d = d – dx2

x = x + 1

12. Вывод vp.

13. Останов.

1. Интерполяция цветов при выводе отрезка (два варианта).

Берем R.

R1, R2 - известны. R3 - ?

R3 = (1 - t) \* R1 + t \* R2,

где t = d13 / d12

Вариант 2. Цвет - массив.

dx = x1 - x0

d\_clr = (clr1 - clr0) / dx

x = x0

clr = clr0

В цикле:

x += dx

clr += d\_clr

1. Заливка полигона с интерполяцией цветов и без нее.
2. Алгоритм заливки невыпуклого многоугольника.