Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Оренбургский государственный университет»

Кафедра геометрии и компьютерных наук

С.А. Герасименко, Т.А. Фомина

# КОМПЬЮТЕРНАЯ ГРАФИКА

Рекомендовано к изданию Редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» в качестве методических указаний для студентов, обучающихся по программам высшего образования по направлению подготовки 01.03.02 Прикладная математика и информатика и специальности 10.05.01 Компьютерная безопасность

Оренбург
2016

УДК 004.92 (076.5)
ББК 32.973.26-018.2я7
Г 37

Рецензент – кандидат технических наук, доцент И.В. Влацкая

**Герасименко, С.А.**
Г 37 Компьютерная графика: методические указания к лабораторным работам / С.А. Герасименко, Т.А. Фомина; Оренбургский гос. ун-т. – Оренбург: ОГУ, 2016. – 41с.

Методические указания содержат материал, предназначенный для выполнения лабораторных работ по курсу «Компьютерная графика».

Методические указания предназначены для студентов, обучающихся по направлению подготовки 01.03.02 Прикладная математика и информатика и специальности 10.05.01 Компьютерная безопасность, и составлены в соответствии с утвержденными рабочими программами дисциплины «Компьютерная графика». Материалы могут быть полезны преподавателям высших учебных заведений, ведущим соответствующую дисциплину, и изучающим эту дисциплину студентам.

УДК 004.92 (076.5)
ББК 32.973.26-018.2я7

# Содержание

# Введение

Компьютерная или машинная графика – это вполне самостоятельная область человеческой деятельности, со своими проблемами и спецификой. Компьютерная графика – это и новые эффективные технические средства для проектировщиков, конструкторов и исследователей, и программные системы и машинные языки, и новые научные, учебные дисциплины, родившиеся на базе синтеза таких наук как аналитическая, прикладная и начертательная геометрии, программирование для ЭВМ, методы вычислительной математики и т. п.

Машина наглядно изображает такие сложные геометрические объекты, которые раньше математики даже не пытались изобразить. Основными задачами машинной графики являются ввод (считывание) графической информации в ЭВМ, вывод ее из ЭВМ (формирование изображений), а также определенного рода переработка информации в компьютере. Таким образом, основные задачи машинной геометрии или, как говорят, автоматизированного геометрического моделирования и конструирования – синтез в ЭВМ и анализ геометрических объектов, решение задач геометрического характера.

В методических указаниях изложены теоретические сведения о базовых алгоритмах компьютерной графики для генерации простых фигур (отрезка и окружности), заполнения многоугольников, отсечения невидимых линий и двумерных преобразованиях фигур. Представлены формальные описания этих алгоритмов на простом и доступном псевдокоде.

Практикум состоит из шести лабораторных работ. Каждая лабораторная работа содержит цель, необходимый для выполнения заданий теоретический материал, практическую часть и список литературы.

Материалы могут быть полезны преподавателям высших учебных заведений, ведущим соответствующую дисциплину, и изучающим эту дисциплину студентам.

# 1 Лабораторная работа №1. Алгоритмы построения отрезков. Цифровой дифференциальный анализатор

## 1.1 Цель работы

Изучить алгоритм цифрового дифференциального анализатора вычерчивания отрезков.

## 1.2 Теоретическая часть

Один из методов разложения отрезка в растр состоит в решении дифференциального уравнения, описывающего этот процесс. Для прямой линии имеем

$$\frac{dy}{dx} = const \text{ или } \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}.$$

Решение представляется в виде

$$y_{i+1} = y_i + \Delta y,$$

$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1}\Delta x, \quad (1)$$

где $x_1, y_1, x_2, y_2$ — концы разлагаемого отрезка;

$y_i$ — начальное значение для очередного шага вдоль отрезка.

Фактически уравнение (1) представляет собой рекуррентное соотношение для последовательных значений вдоль нужного отрезка. Этот метод, используемый для разложения в растр отрезков, называется цифровым дифференциальным анализатором (ЦДА). В простом ЦДА большее из приращений либо $\Delta x$, либо $\Delta y$ выбирается в качестве единицы растра. Ниже приводится алгоритм ЦДА, работающий во всех квадрантах, на псевдокоде.

Предполагается, что концы отрезка $(x_1, y_1)$ и $(x_2, y_2)$ не совпадают.

*Integer* — функция преобразования вещественного числа в целое.

Примечание: Во многих реализациях функция *Integer* означает взятие целой части, т.е. *Integer*(*-8,5*) = *-9*, а не *-8*. В алгоритме используется именно такая функция.

*Sign* — функция, возвращающая *-1, 0, 1* для отрицательного, нулевого и положительного аргумента соответственно.

Аппроксимируем длину отрезка.

***if*** $abs(x_2 - x_1) \geq abs(y_2 - y_1)$ ***then***

$Длина = abs(x_2 - x_1)$

***else***

$Длина = abs(y_2 - y_1)$

***end if***

Полагаем большее из приращений $\Delta x$ или $\Delta y$ равным единице растра.

$\Delta x = (x_2 - x_1)/Длина$

$\Delta y = (y_2 - y_1)/Длина$

Округляем величины, а не отбрасываем дробную часть. Использование знаковой функции делает алгоритм пригодным для всех квадрантов.

$x = x_1 + 0,5 * Sign(\Delta x)$

$y = y_1 + 0,5 * Sign(\Delta y)$

Начало основного цикла.

*i=1*

***while*** *(i* $\leq$ *Длина)*

*Plot (Integer(x), Integer(y))*

$x = x + \Delta x$

$y = y + \Delta y$

*i=i+1*

***end while***

***finish***

### 1.3 Практическая часть

Задание:

1) Реализовать в виде процедуры CDA алгоритм «Цифровой дифференциальный анализатор».

Предполагается, что:

- $(x_1, y_1)$ — начальная точка отрезка;
- $(x_2, y_2)$ — конечная точка отрезка;
- *IntRe(x)* — функция, преобразующая вещественные числа в целые;
- *Sign(x)* — функция, возвращающая *-1, 0, 1* для отрицательного, нулевого и положительного аргументов соответственно;
- *Plot(x,y)* — функция, отображающая на экране точку с координатами *(x, y)*.

2) Отобразить на экране с помощью полученной процедуры несколько отрезков.

3) Изменить процедуру CDA, используя разные способы преобразования типов.

4) Сравнить полученные результаты (использовать в разных процедурах различный цвет отображаемых отрезков).

### 1.4 Литература

1) [2] с 48-53;
2) [4] с 53;
3) [5] с 96.

# 2 Лабораторная работа №2. Обобщенный алгоритм Брезенхема построения отрезков

## 2.1 Цель работы

Изучить алгоритм Брезенхема формирования растрового представления произвольного отрезка прямой.

## 2.2 Теоретическая часть

Обобщенный целочисленный алгоритм Брезенхема позволяет генерировать произвольным образом ориентированные отрезки, используя при этом только целочисленную арифметику и операции сравнения. Отрезок вычерчивается пошагово (от пикселя к пикселю) от начала $P_1$ – с целочисленными координатами $(x_1 , y_1)$ – до конца $P_2$ – с целочисленными координатами $(x_2 , y_2)$.

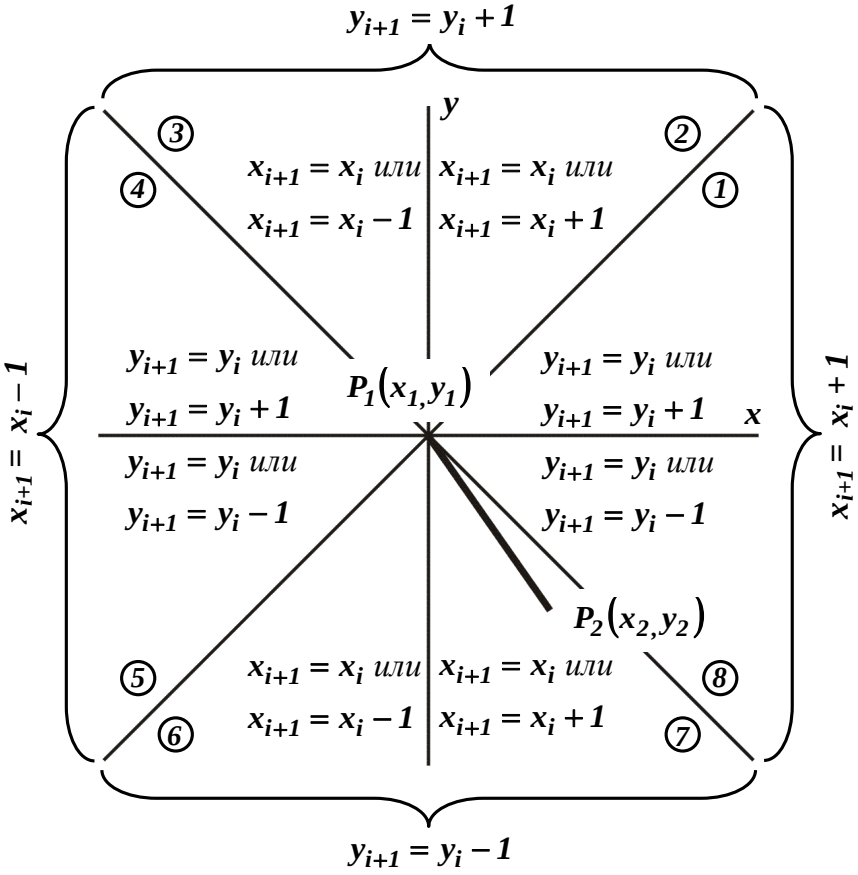


Рисунок 1 – Изменения координат в зависимости от ориентации отрезка

Ориентацию отрезка определяет положение точки $P_2\ (x_2\ ,\ y_2)$ относительно начала координат, которое условно совмещается с точкой $P_1\ (x_1\ ,\ y_1)$ (рисунок 1). На каждом шаге в зависимости от величины тангенса угла наклона отрезка (углового коэффициента) единичное приращение одной из координат задается безусловно. Необходимость единичного изменения другой координаты определяется по знаку рассчитываемой на каждом шаге величины *e*, которая оценивает расстояние между действительным положением отрезка и ближайшими по данной координате адресуемыми точками растра: изменение происходит только в случае, когда $e \geq 0$; при этом величина *e* корректируется и принимает отрицательное значение. Если модуль углового коэффициента не больше 1 (1, 4, 5 и 8 октанты), *x* изменяется на единицу, а *y* – либо не изменяется, либо изменяется на единицу (рисунок 1), в зависимости от знака ошибки. При модуле углового коэффициента больше 1 (2, 3, 6 и 7 октанты), наоборот, *y* безусловно изменяется на единицу, а знак ошибки используется для принятия решения об изменении величины *x*. Знак же приращений координат *x* и *y* зависит от конкретной ориентации отрезка (рисунок 1). Так, например, при генерации отрезка $P_1P_2$ (рисунок 1), расположенного в 7-ом октанте, на каждом шаге *y* безусловно получает приращение –1, а *x* либо изменяется на +1, либо остается неизменным.

Общий целочисленный алгоритм Брезенхема построения отрезков на псевдокоде выглядит следующим образом. Предполагается, что концы отрезка $(x_1, y_1)$ и $(x_2, y_2)$ не совпадают все переменные считаются целыми. Функция *Sign* возвращает *-1, 0, 1* для отрицательного, нулевого и положительного аргумента соответственно.

Инициализация переменных:

$x = x_1$

$y = y_1$

$\Delta x = abs(x_2 - x_1)$

$\Delta y = abs(y_2 - y_1)$

$s_1 = Sign(x_2 - x_1)$

$s_2 = Sign(y_2 - y_1)$

Обмен значений $\Delta x$ и $\Delta y$ в зависимости от углового коэффициента наклона отрезка:

***if*** $\Delta y > \Delta x$ ***then***

*Врем* = $\Delta x$

$\Delta x = \Delta y$

$\Delta y$ = *Врем*

*Обмен = 1*

***else***

*Обмен = 0*

***end if***

Инициализация *e* с поправкой на половину пикселя:

$e = 2\Delta y - \Delta x$

Начало основного цикла:

***for*** *i = 1* ***to*** $\Delta x$

*Plot(x, y)*

***while*** $\left(\bar{e} \geq 0\right)$

***if*** *Обмен = 1* ***then***

$x = x + s_1$

***else***

$y = y + s_2$

***end if***

$e = e - 2\Delta x$

***end while***

***if*** *Обмен = 1* ***then***

$y = y + s_2$

***else***

$x = x + s_1$

***end if***

$e = e + 2\Delta y$

***next*** *i*

***finish***

### 2.3 Практическая часть

Задание:

1) Реализовать в виде процедуры *Brez* общий алгоритм Брезенхема. Предполагается, что:

- $(x_1, y_1)$ — начальная точка отрезка;
- $(x_2, y_2)$ — конечная точка отрезка;
- *Sign(x)* — функция, возвращающая *-1, 0, 1* для отрицательного, нулевого и положительного аргументов соответственно;
- *Plot(x, y)* — функция, отображающая на экране точку с координатами *(x, y)*.

2) Отобразить на экране с помощью полученной процедуры несколько отрезков.

3) Сравнить результаты построения отрезков процедурой *Brez* и процедурами CDA (с разными округлениями) с помощью наложения друг на друга отрезков разного цвета.

### 2.4 Литература:

1) [2] с 54-62;
2) [4] с 56;
3) [5] с 100;
4) [6] с 157.

# 3 Лабораторная работа №3. Алгоритм Брезенхема построения окружностей

## 3.1 Цель работы

Изучить алгоритм Брезенхема формирования растрового представления окружности.

## 3.2 Теоретическая часть

Данный алгоритм также является целочисленным. В соответствии с ним полагается, что генерируется окружность с целочисленным радиусом $R$ и центром в точке ($x = 0$, $y = 0$) (что всегда можно сделать, связав координаты $x$ и $y$ с координатной сеткой растра простым преобразованием координат). Изначально пошагово строится четверть окружности в первом квадранте. Причем ее построение начинается в точке ($x = 0$, $y = R$) и осуществляется в направлении по часовой стрелке.

При таких условиях и нахождении в некоторой текущей точке растра с координатами адресуемой точки ($x_i$, $y_i$) есть только три варианта выбора очередного пикселя (рисунок 2): по горизонтали вправо (направление $m_H$), по диагонали вниз и вправо (направление $m_D$) и по вертикали вниз (направление $m_V$).

В качестве критериев перехода используются следующие величины:

$$\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2;$$

$$\delta = 2(\Delta_i + y_i) - 1;$$

$$\delta' = 2(\Delta_i - x_i) - 1.$$

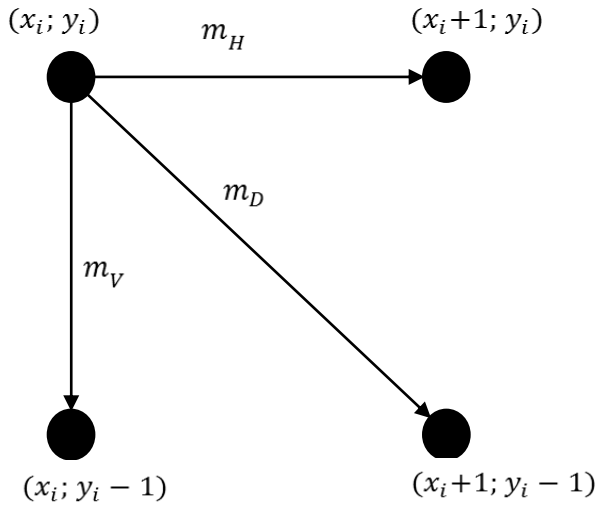


Рисунок 2 – Выбор очередного пикселя

Согласно алгоритму Брезенхема:

1) если $\Delta_i < 0$:
   - при $\delta \leq 0$ выбираем направление $m_H$, т.е. пиксель $(x_i + 1, y_i)$;
   - при $\delta > 0$ выбираем направление $m_D$, т.е. пиксель $(x_i + 1, y_i - 1)$;
2) если $\Delta_i > 0$:
   - при $\delta' \leq 0$ выбираем направление $m_D$, т.е. пиксель $(x_i + 1, y_i - 1)$;
   - при $\delta' > 0$ выбираем направление $m_V$, т.е. пиксель $(x_i, y_i - 1)$;
3) если $\Delta_i = 0$ выбираем направление $m_D$, т.е. пиксель $(x_i + 1, y_i - 1)$.

Для реализации пошагового алгоритма при переходе к очередному текущему пикселю можно использовать следующие выражения:

1) при шаге в горизонтальном направлении $m_H$:

   $x_{i+1} = x_i + 1;$

   $y_{i+1} = y_i;$

   $\Delta_{i+1} = \Delta_i + 2x_{i+1} + 1.$

2) при шаге в диагональном направлении $m_D$:

   $x_{i+1} = x_i + 1;$

$y_{i+1} = y_i - 1;$

$\Delta_{i+1} = \Delta_i + 2x_{i+1} - 2y_{i+1} + 2.$

3) при шаге в вертикальном направлении $m_V$:

$x_{i+1} = x_i;$

$y_{i+1} = y_i - 1;$

$\Delta_{i+1} = \Delta_i - 2y_{i+1} + 1.$

Для построения полной окружности полученную в первом квадранте четверть окружности следует симметрично отобразить относительно осей $x$ и $y$, а также относительно начала координат, т.е. активизировать не только выбранные пиксели $(x_i, y_i)$, но и симметричные им пиксели $(x_i, -y_i)$, $(-x_i, y_i)$ и $(-x_i, -y_i)$.

Ниже представлен составленный с учетом изложенных пояснений алгоритм Брезенхема для генерации окружности на псевдокоде. Кроме описанных выше величин в нем присутствуют переменные $a$, $b$ – целочисленные координаты центра окружности в сетке растра.

Инициализация переменных:

$x_i = 0$

$y_i = R$

$\Delta_i = 2(1 - R)$

*Предел=0*

**1** $Plot(x_i + a, y_i + b)$

$Plot(-x_i + a, -y_i + b)$

$Plot(-x_i + a, y_i + b)$

$Plot(x_i + a, -y_i + b)$

***if*** $(y_i <= Предел)$ ***then*** *4*

Выделение случая 1 или 2, 4 или 5, или 3.

***if*** $(\Delta_i < 0)$ ***then*** *2*

***if*** $(\Delta_i > 0)$ ***then*** *3*

***if*** $(\Delta_i = 0)$ ***then*** *20*

Определение случая 1 или 2.

**2** $\delta = 2\Delta_i + 2y_i - 1$

***if*** $(\delta <= 0)$ ***then*** *10*

***if*** $(\delta > 0)$ ***then*** *20*

Определение случая 4 или 5.

**3** $\delta = 2\Delta_i + 2x_i - 1$

***if*** $(\Delta_i \le 0)$ ***then*** *20*

***if*** $(\Delta_i > 0)$ ***then*** *30*

Выполнение шагов:

шаг $m_H$

**10** $x_i = x_i + 1$

$\Delta_i = \Delta_i + 2x_i + 1$

***go to 1***

шаг $m_D$

**20** $x_i = x_i + 1$

$y_i = y_i - 1$

$\Delta_i = \Delta_i + 2x_i - 2y_i + 2$

***go to 1***

шаг $m_v$

**30** $y_i = y_i - 1$

$\Delta_i = \Delta_i - 2y_i + 1$

***go to 1***

**4** ***finish***

### 3.3 Практическая часть

Задание:

1) Реализовать в виде процедуры *BrezCir* алгоритм Брезенхема построения окружности.

Предполагается, что центр окружности находится в точке $(0, 0)$ и ее радиус равен $R$, где $R$ — целое число.

2) Написать процедуру построения окружности с центром в точке $(a, b)$, радиуса $R$, цветом $C$, координаты центра, радиус и цвет передавать в процедуру в виде параметров. Построить несколько окружностей разного цвета.

3) Написать процедуры построения различных четвертей и половинок окружности.

4) Реализовать в виде процедуры *BrezArc* алгоритм построения дуги окружности (при построении дуги использовать алгоритм Брезенхема построения окружности).

Предполагается, что центр окружности, частью которой является дуга, находится в точке $(a, b)$, ее радиус равен $R$, цвет $C$, начальный угол *BeginAngle*, конечный угол *EndAngle*. Все числа целые, углы заданы в градусах.

Отобразить на экране с помощью полученной процедуры несколько дуг окружностей.

### 3.4 Литература:


1) [2] с 63-72;
2) [4] с 60;
3) [5] с 102;
4) [6] с 161.

# 4 Лабораторная работа №4. Алгоритмы заполнения с затравкой

## 4.1 Цель работы

Изучить алгоритмы закрашивания замкнутых контуров.

## 4.2 Теоретические сведения

### 4.2.1 Простой алгоритм заполнения с затравкой

В алгоритмах, относящихся к группе методов «затравочного заполнения», один пиксель внутри закрашиваемого контура – затравка – должен быть заранее известен. Начиная с этого пикселя, алгоритмы находят и закрашивают все другие пиксели области, ограниченной контуром, до обнаружения границ. В процессе работы в качестве новой затравки назначаются, по тем или иным соображениям, другие пиксели области.

Гранично-определенная область характеризуется тем, что все пиксели на ее границе имеют определенный цвет, причем ни один пиксель самой области не может иметь этот цвет. Четырехсвязной называется область, которую можно заполнить, переходя от пикселя к пикселю в четырех направлениях: влево, вправо, вверх и вниз.

Простой алгоритм заполнения с затравкой и стеком можно представить в следующем виде:

1) поместить затравочный пиксель в стек;
2) пока стек не пуст;
3) извлечь пиксель из стека;
4) присвоить пикселю требуемое значение;
5) для каждого из соседних к текущему 4-связных пикселов проверить: является ли он граничным пикселом или не присвоено ли уже пикселу требуемое

значение. Проигнорировать пиксел в любом из этих двух случаев. В противном случае поместить пиксел в стек.

Ниже представлен простой алгоритм заполнения с затравкой на псевдокоде.

Затравка (*x, y*) выдает затравочный пиксель.

*Push* – процедура, которая помещает пиксел в стек.

*Pop* – процедура, которая извлекает пиксел из стека.

*Пиксель*(*x, y*) = *Затравка*(*x, y*)

Инициализируем стек:

*Push Пиксел (x, y)*

***while*** *(стек не пуст)*

Извлекаем пиксель из стека:

*Pop Пиксел (x, y)*

***if*** $(Пиксел(x,y) <> Нов\ значение)$ ***then***

$Пиксел(x,y) = Нов\ значение$

***end if***

Проверим, надо ли помещать соседние пиксели в стек:

***if*** $(Пиксел(x+1,y) <> Нов\ значение)$ ***and***

$(Пиксел(x+1,y) <> Гран\ значение)$ ***then***

$Push\ Пиксел(x+1,y)$

***if*** $(Пиксел(x,y+1) <> Нов\ значение)$ ***and***

$(Пиксел(x,y+1) <> Гран\ значение)$ ***then***

$Push\ Пиксел(x,y+1)$

***if*** $(Пиксел(x-1,y) <> Нов\ значение)$ ***and***

$(Пиксел(x-1,y) <> Гран\ значение)$ ***then***

$Push\ Пиксел(x-1,y)$

***if*** $(Пиксел(x,y-1) <> Нов\ значение)$ ***and***

$(Пиксел(x,y-1) <> Гран\ значение)$ ***then***

$Push\ Пиксел(x,y-1)$

***end if***

***end while***

### 4.2.2 Построчный алгоритм заполнения с затравкой

В построчном алгоритме заполнения размер стека минимизируется за счет хранения только одного затравочного пиксела для любого непрерывного интервала на сканирующей строке. Данный алгоритм применим к гранично-определенным областям. Область может быть как выпуклой, так и не выпуклой, а также может содержать дыры.

Работу построчного алгоритма заполнения с затравкой для гранично-определенной четырехсвязной области схематично можно описать так:

1) поместить координаты затравочного пикселя в стек;

Пока стек не пуст:

2) извлечь данные о пикселе из стека; если ему уже присвоено требуемое значение цвета – проигнорировать, если нет, тогда:

– присвоить пикселю требуемое значение цвета;

– заполнить интервал с текущим пикселем вправо и влево от него вдоль сканирующей строки до обнаружения границ;

– переменным $x_{лев}$ и $x_{прав}$ присвоить значения горизонтальных координат адресуемых точек соответственно крайнего левого и крайнего правого пикселей интервала;

3) в диапазоне $x_{лев} \le x \le x_{прав}$ проверить строки, расположение непосредственно над и под текущей строкой; если в них есть еще не заполненные интервалы (т.е. не все пиксели граничные или уже заполненные), то в указанном диапазоне данные о крайнем правом пикселе на каждом интервале поместить в стек.

При инициализации алгоритма в стек помещается единственный затравочный пиксел, работа завершается при опустошении стека.

Ниже представлена запись рассматриваемого алгоритма на псевдокоде.

*Затравка (x, y)* выдает затравочный пиксель.

*Pop* — процедура, которая извлекает пиксел из стека.

*Push* – процедура, которая помещает пиксел в стек инициализируем стек.

*Push Затравка (x, y)*

***while*** *(стек не пуст)*

Извлекаем пиксел из стека и присваиваем ему новое значение.

***Pop*** *Пиксел (x, y)*

$Пиксел(x,y) = Нов\ значение$

Сохраняем *x*-координату затравочного пиксела:

$Врем\_x = x$

Заполняем интервал справа от затравки:

$x = x + 1$

***while*** $(Пиксел(x,y) <> Гран\ значение)$

$Пиксел(x,y) = Нов\ значение$

$x = x + 1$

***end while***

Сохраняем крайний справа пиксель:

$x_{прав} = x - 1$

Восстанавливаем x-координату затравки:

$x = Врем\_x$

Заполняем интервал слева от затравки:

$x = x - 1$

***while*** $(Пиксел(x,y) <> Гран\ значение)$

$Пиксел(x,y) = Нов\ значение$

$x = x - 1$

***end while***

Сохраняем крайний слева пиксел:

$x_{лев} = x + 1$

Восстанавливаем x-координату затравки

$x = Врем\_x$

Проверим, что строка выше не является ни границей многоугольника, ни уже полностью заполненной; если это не так, то найти затравку, начиная с левого края подынтервала сканирующей строки

$x = x_{лев}$

$y = y + 1$

**while** $(x < x_{прав})$

Ищем затравку на строке выше

$Флаг = 0$

**while** $\left( \left(Пиксел(x,y) <> Гран\ значение\right)\ \textbf{and}\right.$

$\left.\left(Пиксел(x,y) <> Нов\ значение\right)\right)\ \textbf{and}\ \left(x < x_{прав}\right)$

**if** $(Флаг = 0)$ **then** $Флаг = 1$

$x = x + 1$

**end while**

Помещаем в стек крайний справа пиксел

**if** $(Флаг = 1)$ **then**

**if** $\left(x = x_{прав}\right)$ **and** $\left(Пиксел(x,y) <> Гран\ значение\right)$ **and**

$\left(Пиксел(x,y) <> Нов\ значение\right)$ **then**

**Push** $Пиксел\ (x, y)$

**else**

**Push** $Пиксел\ (x - 1, y)$

**end if**

$Флаг = 0$

**end if**

Продолжим проверку, если интервал был прерван

$X_{вход} = x$

**while** $\left( \left(Пиксел(x,y) = Гран\ значение\right)\ \textbf{or}\right.$

$\left.\left(Пиксел(x,y) = Нов\ значение\right)\right)\ \textbf{and}\ \left(x < x_{прав}\right)$

$x = x + 1$

**end while**

Удостоверимся, что координата пиксела увеличена

***if*** $(x = X_{вход})$ ***then***

$x = x + 1$

***end while***

Далее проверим, что строка ниже не является ни границей многоугольника, ни уже полностью заполненной.

Эта часть алгоритма совершенно аналогична проверке для строки выше, за исключением того, что вместо $y = y + 1$ надо подставить $y = y - 2$.

***end while***

***finish***

### 4.3 Практическая часть

Задание:

1) Реализовать в виде процедуры «Простой алгоритм заполнения с затравкой».

*Push* – процедура, которая помещает координаты пикселя в стек.

*Pop* – процедура, которая извлекает координаты пикселя из стека.

Нарисовать на экране с помощью процедуры построения отрезков замкнутый многоугольник и закрасить его с помощью полученной процедуры.

2) Реализовать в виде процедуры «Построчный алгоритм заполнения с затравкой». Нарисовать на экране с помощью процедуры построения отрезков замкнутый многоугольник и закрасить его с помощью полученной процедуры.

### 4.4 Литература:


1) [2] с 110-118;
2) [3] с 64;
3) [4] с 95;
4) [5] с 109;
5) [6] с 165, 173.

# 5 Лабораторная работа №5. Алгоритм двумерного внутреннего отсечения Сазерленда-Коэна

## 5.1 Цель работы

Освоить методы отсечений двумерных объектов с использованием алгоритма Сазерленда-Коэна.

## 5.2 Теоретические сведения

Данный алгоритм реализует отсечение отрезков координатно-ориентированным прямоугольником, границы которого: левая, правая, нижняя и верхняя – задаются координатами соответственно $x_л$ , $x_п$ , $y_н$ и $y_в$ (рисунок 3). В алгоритме используются четырехразрядные (битовые) коды, определяющие расположение концов отрезков относительно границ отсекающего окна. При формировании кода конца отрезка с координатами *(x, y)* в первый (крайний правый) бит заносится *1*, если $x < x_л$, во второй – если $x > x_п$, в третий – если $y < y_н$, в четвертый – если $y > y_в$. В остальных случаях в соответствующие биты заносится *0*. Таким образом, вся координатная плоскость разбивается на девять областей, в каждой из которых концу отрезка присваивается уникальный код (рисунок 3).

При внутреннем отсечении признаком полной видимости отрезка являются нулевые коды обоих его концов (как для отрезка *ab* на рисунке 3). Признак тривиальной невидимости отрезка – отличное от нуля побитовое логическое произведение концевых кодов (так, у отрезка *cd*, расположенного целиком левее отсекающего окна, с кодами концов *0101* и *0001* такое произведение будет равно *0001*). Вместе с тем, при полностью нулевом побитовом логическом произведении концевых кодов отрезок может оказаться частично видимым (как отрезки *ef* и *gh*) или полностью невидимым (как отрезок *ij*). Для подобных отрезков при выявлении их возможных видимых частей используются результаты определения точек пересечения бесконечной прямой линии, проведенной через отрезок, с бесконечными прямыми линиями, проведенными через ребра отсекающего окна.

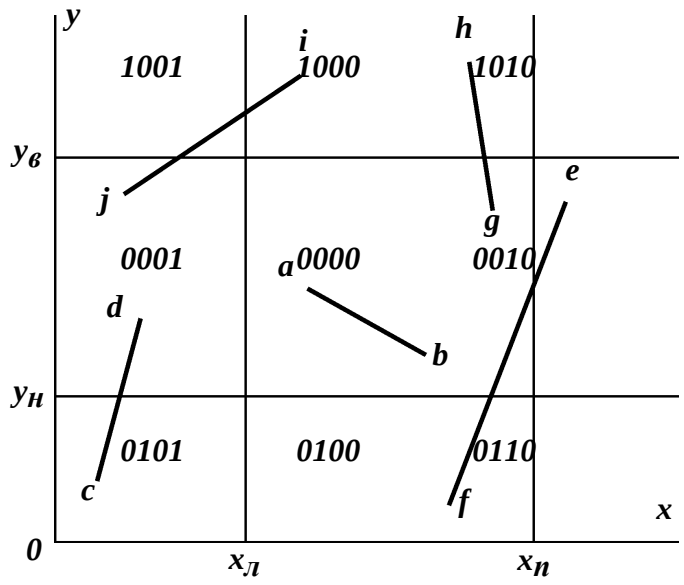


Рисунок 3 – Коды концов отрезков

Для отрезка с началом в точке $P_1(x_1, y_1)$ и концом в точке $P_2(x_2, y_2)$, если он не вертикален $(m = (y_2 - y_1)/(x_2 - x_1) \neq \infty)$, такие точки пересечения на прямых, проведенных через левое и правое ребра, будут иметь координаты соответственно:

$$x = x_л,\ y = m(x_л - x_1) + y_1;$$

$$x = x_п,\ \ y = m(x_п - x_1) + y_1.$$

Если отрезок не горизонтален $(m \neq 0)$, координаты аналогичных точек пересечения на прямых, проведенных через нижнее и верхнее ребра, будут равны соответственно:

$$x = (y_н - y_1)/m + x_1,\ \ y = y_н;$$

$$x = (y_в - y_1)/m + x_1,\ \ y = y_в.$$

Ключом к алгоритму Сазерленда – Коэна является информация о том, что одна из концевых точек отрезка лежит вне окна. Поэтому тот отрезок, который

заключен между этой точкой и точкой пересечения, можно отвергнуть как невидимый. Фактически это означает замену исходной концевой точки на точку пересечения.

Последовательность действий в алгоритме двумерного внутреннего отсечения Сазерленда-Коэна следующая:

1) Для каждого отрезка $P_1P_2$, определить, не является ли он полностью видимым или может быть тривиально отвергнут как невидимый.

2) Если $P_1$ вне окна, то продолжить выполнение, иначе поменять $P_1$, и $P_2$ местами.

3) Заменить $P_1$ на точку пересечения $P_1P_2$ со стороной окна.

Ниже приведен алгоритм на псевдокоде. В нем используются следующие обозначения:

Окно – массив 1х4, содержащий координаты $(x_л, x_п, y_н, y_в)$ сторон окна.

$P_1P_2$ – концевые точки отрезка с координатами $(P_1x, P_1y)$ и $(P_2x, P_2y)$ соответственно.

$T_{1код}$, $T_{2код}$ – массивы 1х4, содержащие коды точек $P_1$ и $P_2$.

*Флаг* – индикатор координатной ориентации отрезка. Если *Флаг = –1*, то отрезок вертикальный, если *Флаг = 0*, то горизонтальный.

*Видимость* – признак видимости отрезка равный: «нет», «частично», «да», если отрезок соответственно: полностью невидим, видим частично или полностью видим.

$T_{код}$ – массив 1х4, содержащий коды концевой точки.

$T_{1код}$, $T_{2код}$ – массивы 1х4, содержащие коды первой и второй концевых точек соответственно.

*Сумма* – сумма всех элементов массива $T_{код}$.

Инициализация Флаг.

*Флаг = 1*

Проверка вертикальности и горизонтальности отрезка.

***if*** $(P_2x - P_1x = 0)$ ***then***

*Флаг = –1*

***else***

Вычисление наклона.

$Наклон = (P_2y - P_1y)/(P_2x - P_1x)$

***if*** *(Наклон = 0)* ***then***

$Флаг = 0$

***end if***

Для каждой стороны окна:

***for*** *i = 1* ***to*** *4*

***call*** *Коэн ($P_1P_2$, Окно; Видимость)*

***if*** *(Видимость = да)* ***then*** *2*

***if*** *(Видимость = нет)* ***then*** *3*

Проверка пересечения отрезка и стороны окна.

***if*** *($T_{1код}(5 - i) = T_{2код}(5 - i)$)* ***then*** *1*

Проверка нахождения $P_1$ вне окна; если $P_1$ внутри окна, то поменять $P_1$ и $P_2$ местами.

***if*** *($T_{1код}(5 - i) = 0$)* ***then***

$Раб = P_1$

$P_1 = P_2$

$P_2 = Раб$

***end if***

Поиск пересечений отрезка со сторонами окна. Выбор соответствующей подпрограммы вычисления пересечения. Контроль вертикальности отрезка.

***if*** *(Флаг < > –1) и ($i \le 2$)* ***then***

$P_1y = Наклон * (Окно_i - P_1x) + P_1y$

$P_1x = Окно_i$

***else***

***if*** *(Флаг < > 0)* ***then***

***if*** *(Флаг < > -1)* ***then***

$P_1x = (1/Наклон) * (Окно_i - P_1y) + P_1x$

***end if***

$P_1y = Окно_i$

***end if***

***end if***

1 ***next*** *i*

Начертить видимый отрезок

2 *Draw* $P_1P_2$

3 ***finish***

*Подпрограмма определения видимости отрезка*

***subroutine*** *Коэн* ($P_1$, $P_2$, *Окно; Видимость*)

Вычисление кодов концевых точек отрезка.

***call*** *Конец* ($P_1$, *Окно;* $T_{1код}$, *Сумма1*)

***call*** *Конец* ($P_2$, *Окно;* $T_{2код}$, *Сумма2*)

Предположим, что отрезок частично видим.

*Видимость = частично*

Проверка полной видимости отрезка.

***if*** *(Сумма 1 = 0) и (Сумма 2 = 0)* ***then***

*Видимость = да*

***else***

Проверка тривиальной невидимости отрезка.

***call*** *Логическое* ($T_{1код}$, $T_{2код}$, *Произвед*)

***if*** *(Произвед < > 0)* ***then***

*Видимость = нет*

***end if***

Отрезок может оказаться частично видимым.

***return***

*Подпрограмма вычисления кодов концевой точки отрезка*

***subroutine*** *Конец* (*P, Окно;* $T_{код}$, *Сумма*)

Вычисление кодов концевой точки.

***if*** ($P_x < x_л$) ***then*** $T_{код}(4) = 1$ ***else*** $T_{код}(4) = 0$

***if*** *($P_x > x_п$)* ***then*** *$T_{код}(3) = 1$* ***else*** *$T_{код}(3) = 0$*

***if*** *($P_y < y_н$)* ***tnen*** *$T_{код}(2) = 1$* ***else*** *$T_{код}(2) = 0$*

***if*** *($P_y > y_в$)* ***then*** *$T_{код}(1) = 1$* ***else*** *$T_{код}(1) = 0$*

Вычисление суммы.

*Сумма = 0*

***for*** *(i = 1)* ***to*** *4*

*Сумма = Сумма + $T_{код}(i)$*

***next*** *i*

***return***

*Подпрограмма вычисления логического произведения:*

***subroutine*** *Логическое ($T_{1код}$, $T_{2код}$; Произвед)*

*Произвед* – сумма побитовых логических произведений.

*Произвед = 0*

***for*** *(i = 1)* ***to*** *4*

*Произвед = Произвед + Целая часть (($T_{1код}(i) + T_{2код}(i))/2$)*

***next*** *i*

***return***

### 5.3 Практическая часть

Задание:

1) Реализовать в виде программы алгоритм двумерного внутреннего отсечения Сазерленда-Коэна.

*Окно* – массив 1х4, содержащий координаты $x_л$, $x_п$, $y_н$, $y_в$.

$P_1$, $P_2$, $P$ – начало и конец отрезка с координатами соответственно ($x_1$, $y_1$) и ($x_2$, $y_2$), а также концевая точка, обрабатываемая подпрограммой *Конец*.

*Код1*, *Код2*, *Код* – массивы 1х4, содержащие коды соответственно начала отрезка, конца отрезка и точки, обрабатываемой подпрограммой *Конец*.

*Сумма1*, *Сумма2*, *Сумма* – суммы всех элементов соответственно массивов *Код1*, *Код2*, *Код*.

*Лог_произв* – сумма побитовых логических произведений кодов концов отрезка.

*Видимость* – признак видимости отрезка, принимающий значения *Да*, *Нет* и *Возможно* в случаях, когда отрезок полностью виден, безусловно невидим и возможно частично видим соответственно.

*Флаг* – индикатор ориентации отрезка, принимающий значения *-1* (при вертикальности отрезка) или *0* (в горизонтальном случае).

### 5.4 Литература:


1) [2] с 143-157;
2) [3] с 49;
3) [4] с 83;
4) [6] с 197.

# 6 Лабораторная работа №6. Двумерные преобразования. Однородные координаты на плоскости

## 6.1 Цель работы

Научиться выполнять двумерные преобразования графических объектов, базирующихся на матричных операциях с данными.

## 6.2 Теоретические сведения

В компьютерной графике, наряду с соответствующими алгоритмами рисования, важно иметь математический аппарат, позволяющий редактировать изображение, т.е. осуществлять его преобразование (модификацию) в соответствии с требованиями решаемой задачи. Графические объекты (точки, отрезки прямых, плоские и объемные фигуры) нужно уметь масштабировать, поворачивать, симметрично отражать, перемещать и т.п. Подобные действия и их комбинации обычно реализуются с помощью матричных операций с данными.

### 6.2.1 Простейшие преобразования точек, отрезков и многоугольников

Точка *P* на плоскости однозначно определяется двумя своими координатами *(x, y)*. В соответствие ей можно поставить матрицу-строку размером 1×2 вида

$$[P] = [x \quad y].$$

Следует заметить также, что точка может задаваться и соответствующей матрицей-столбцом размером 2×1. В любом случае матрицу, определяющую положение точки, часто называют *координатным вектором* или *вектором положения*.

Большинство из элементарных преобразований по отношению к точке можно реализовать путем умножения матрицы $[P]$ на матрицу общего преобразования размером 2×2 вида:

$$[T]=\begin{bmatrix} a & b \\ c & d \end{bmatrix}: [P][T]=[x \quad y]\begin{bmatrix} a & b \\ c & d \end{bmatrix}=[(ax+cy) \quad (bx+dy)]=[x^* \quad y^*]=[P^*],$$

где $x^*=ax+cy$, $y^*=bx+dy$ – координаты точки $P^*$, являющейся результатом преобразования точки $P$.

Входящие в матрицу общего преобразования коэффициенты $a$, $b$, $c$, $d$ влияют на соответствующие преобразования (таблица 1).

Таблица 1 – Матрицы простейших преобразований точек

| Преобразование | Матрица |
|---|---|
| 1 | 2 |
| Тождественное преобразование, поворот на 0° или 360° | $[T]=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ |
| Локальное масштабирование в $a$ раз по оси $x$ и (при $a < 0$) отражение относительно оси $y$ | $[T]=\begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix}$ |
| Локальное масштабирование в $d$ раз по оси $y$ и (при $d < 0$) отражение относительно оси $x$ | $[T]=\begin{bmatrix} 1 & 0 \\ 0 & d \end{bmatrix}$ |
| Пропорциональное (равномерное) локальное масштабирование в $a = d$ раз | $[T]=\begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}$ |
| Симметричное отражение относительно оси $y$ | $[T]=\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ |
| Симметричное отражение относительно оси $x$ | $[T]=\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ |
| Симметричное отражение относительно точки начала координат, поворот на 180° | $[T]=\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$ |
| Сдвиг вдоль оси $y$ пропорционально координате $x$ | $[T]=\begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix}$ |

Продолжение таблицы 1

| 1 | 2 |
|---|---|
| Сдвиг вдоль оси $x$ пропорционально координате $y$ | $[T]=\begin{bmatrix}1 & 0\\ c & 1\end{bmatrix}$ |
| Поворот на 90° | $[T]=\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}$ |
| Поворот на 180° | $[T]=\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}$ |
| Поворот на 270° | $[T]=\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}$ |
| Поворот на произвольный угол $\theta$ | $[T]=\begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix}$ |
| Отражение относительно прямой $y = x$ | $[T]=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$ |
| Отражение относительно прямой $y = -x$ | $[T]=\begin{bmatrix}0 & -1\\ -1 & 0\end{bmatrix}$ |

Рассмотренные выше преобразования точки можно обобщить на аналогичные преобразования отрезков прямых и многоугольников.

Отрезок прямой линии определяется координатами двух его концов: $A$ ($x_1$, $y_1$) и $B$ ($x_2$, $y_2$) или двумя координатными векторами:

$$[A]=[x_1 \quad y_1] \text{ и } [B]=[x_2 \quad y_2].$$

Более компактно можно задать отрезок матрицей размером 2×2 вида

$$[L]=\begin{bmatrix}x_1 & y_1\\ x_2 & y_2\end{bmatrix}.$$

Аналогичным образом многоугольник может быть представлен $M$×2 матрицей (где $M$ – число вершин многоугольника), в строке которой в строгой последовательности занесены координаты вершин, т.е. матрицей вида:

$$[F]=\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \dots & \dots \\ x_M & y_M \end{bmatrix}.$$

Используя 2×2 матрицу общего преобразования $[T]$, с отрезками и многоугольниками можно осуществлять преобразования, аналогичные тем, которые рассмотрены ранее по отношению к точкам.

Замечания:

1) отрезок прямой линии преобразуется в отрезок другой прямой линии;

2) все точки преобразованного отрезка непосредственно соответствуют всем точкам исходного отрезка;

3) пересекающиеся отрезки преобразуются в пересекающиеся отрезки, причем (как следствие предыдущего пункта) точка пересечения преобразованных отрезков соответствует точке пересечения исходных отрезков;

4) результатом преобразования параллельных отрезков будут параллельные отрезки.

Преобразование точки, находящейся в начале координат, с использованием матрицы преобразования дает однозначный результат:

$$[P][T]=[0 \quad 0]\begin{bmatrix} a & b \\ c & d \end{bmatrix}=[0 \quad 0]=[P]$$

т.е. начало координат инвариантно относительно данного преобразования. Это ограничивает возможности модификации графических объектов: нельзя, например, осуществить перемещение объекта с сохранением размеров, повернуть объект относительно произвольной точки, отразить относительно произвольной линии и т.п. Однако такие ограничения устраняются при использовании однородных координат.

### 6.2.2 Однородные координаты

Ранее было замечено, что использование двумерных координатных векторов, отображающих точки на плоскости, в совокупности с матрицей общего преобразования $[T]$ размером 2×2 накладывает ряд ограничений на модификацию объектов. Существенно расширить возможности модификации позволяет использование однородных координат для отображения точек и, соответственно, матрицы общего преобразования $[T]$ размером 3×3.

Однородные координаты координатного вектора $[x \quad y]$ представляют собой тройку

$$[x' \quad y' \quad h],$$

где $x = x'/h$, $y = y'/h$, $h$ – некоторое вещественное число (случай $h = 0$ является особым).

Таким образом, координатному вектору $[x \quad y]$ на физической плоскости $xy$ соответствует бесконечно много наборов однородных координат вида $[hx \quad hy \quad h]$. Вместе с тем, имеется лишь один набор однородных координат с $h = 1$ т.е. вида $[x \quad y \quad 1]$, для представления вектора $[x \quad y]$. В компьютерной графике для отображения точек используются однородные координаты именно такого вида.

Матрицу преобразования для двумерных однородных координат в общем виде можно представить так:

$$[T] = \begin{bmatrix} a & b & p \\ c & d & q \\ m & n & s \end{bmatrix}.$$

Условно ее можно разбить на четыре части

$$[T]=\left[\begin{array}{cc:c} a & b & p \\ c & d & q \\ \hdashline m & n & s \end{array}\right],$$

входящие в нее коэффициенты $a$, $b$, $c$, $d$, $m$ и $n$ влияют на соответствующие преобразования, а именно: коэффициенты $a$, $b$, $c$ и $d$ задают операции локального масштабирования, сдвига, отражения и поворота; $m$ и $n$ - перемещения вдоль координатных осей; $p$ и $q$ – проецирование в однородных координатах; $s$ - общее масштабирование. Теперь каждая точка плоскости, в том числе начало координат, может быть преобразована (таблица 2).

Однородные координаты преобразованных точек всегда имели вид $[x^* \quad y^* \quad 1]$, т.е. число $h$ тождественно принимало единичное значение. Геометрически это можно трактовать как ограничение преобразований физической плоскостью $h = 1$ в трехмерном пространстве $xyh$. В компьютерной графике используют однородные координаты только такого вида. Поэтому, когда какое-либо преобразование приводит к результату с $h \neq 1$ (и, кстати, $h \neq 0$), этот результат нормализуют, т.е. приводят к указанному виду путем деления трех составляющих однородных координат на величину $h$.

Таблица 2 – Матрицы двумерных преобразований в однородных координатах

| Преобразование | Матрица |
|---|---|
| 1 | 2 |
| Тождественное преобразование, поворот на 0° или 360° | $[T]=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Локальное масштабирование в $a$ раз по оси $x$ и (при $a < 0$) отражение относительно оси $y$ | $[T]=\begin{bmatrix} a & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Локальное масштабирование в $d$ раз по оси $y$ и (при $d < 0$) отражение относительно оси $x$ | $[T]=\begin{bmatrix} 1 & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |

Продолжение таблицы 2

| 1 | 2 |
|---|---|
| Пропорциональное (равномерное) локальное масштабирование в $a = d$ раз | $[T]=\begin{bmatrix} a & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$ при $a = d$ |
| Симметричное отражение относительно оси $y$ | $[T]=\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Симметричное отражение относительно оси $x$ | $[T]=\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Симметричное отражение относительно точки начала координат, поворот на 180° | $[T]=\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Сдвиг вдоль оси $y$ пропорционально координате $x$ (на $bx$) | $[T]=\begin{bmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Сдвиг вдоль оси $x$ пропорционально координате $y$ (на $cy$) | $[T]=\begin{bmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Поворот на 90° | $[T]=\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Поворот на 270° | $[T]=\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Поворот на произвольный угол $\theta$ | $[T]=\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Отражение относительно прямой $y = x$ | $[T]=\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |

Продолжение таблицы 2

| | |
|---|---|
| Отражение относительно прямой $y = -x$ | $[T]=\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ |
| Перемещение вдоль оси $x$ на $m$ | $[T]=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & 0 & 1 \end{bmatrix}$ |
| Перемещение вдоль оси $y$ на $n$ | $[T]=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & n & 1 \end{bmatrix}$ |
| Проецирование в однородных координатах (если $h = px + qy + 1 \neq 1$ и $h \neq 0$, результат необходимо нормализовать путем деления всех однородных координат на $h$) | $[T]=\begin{bmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{bmatrix}$ |
| Общее масштабирование в $1/s$ раз (если $h = s \neq 1$, и $h \neq 0$ результат необходимо нормализовать путем деления всех однородных координат на $h$) | $[T]=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix}$ |

### 6.2.3 Комбинированные (сложные) преобразования

При модификации графического объекта часто требуется не одно простое его преобразование, а несколько последовательно проведенных преобразований. В этом случае исходную матрицу $[X]$ (составленную из векторов положения точки, концов отрезка или вершин многоугольника) можно умножить на матрицу первого преобразования, полученный результат – на матрицу второго преобразования и т.д. Конечный результат будет определяться выражением

$$[X^*] = [X][T_1][T_2]\ldots[T_N],$$

где $[T_n]$ – матрица $n$-го преобразования, $n = 1, 2, \ldots, N$.

Замечания:

1) Операция умножения матриц не является коммутативной (в общем случае $[A][B] \neq [B][A]$), и следует строго соблюдать порядок (последовательность) выполнения преобразований (операций умножения).

2) Для матриц справедлив ассоциативный закон: $[A]([B][C]) = ([A][B])[C]$. Поэтому для достижения того же результата можно поступить иначе: отдельные преобразования предварительно комбинировать (или конкатенировать), т.е. рассчитать матрицу полного преобразования $[T] = [T_1][T_2]...[T_N]$, и затем применить ее для преобразования исходной матрицы – $[X\ *] = [X][T]$.

Теперь имея более мощный, чем ранее, математический аппарат можно реализовать сложные преобразования, например поворот вокруг произвольной точки или отражение относительно произвольной прямой.

Последовательность действий при реализации поворота вокруг произвольной точки может быть следующей:

1) точка, относительно которой совершается поворот, перемещается в начало координат (одновременно с этим параллельно перемещается и объект преобразования);

2) выполняется требуемый поворот объекта;

3) точка центра вращения возвращается на исходное место (параллельно перемещается и объект преобразования).

Применительно к однородным координатам исходной точки $[x \quad y \quad 1]$ подобная операция поворота вокруг точки с координатами ($m$, $n$) на произвольный угол $\theta$ реализуется следующими матричными преобразованиями:

$$[x^* \quad y^* \quad 1] = [x \quad y \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix}.$$

Рассмотрим теперь более сложное преобразование – отражение объекта относительно произвольной прямой линии, причем не проходящей через начало

координат. Используемый теперь подход позволяет осуществить это, выполнив следующие действия:

1) переместить линию и объект вдоль осей *x* и *y* таким образом, чтобы линия прошла через начало координат;

2) повернуть линию и объект вокруг начала координат до совпадения линии с одной из координатных осей;

3) отразить объект относительно данной координатной оси;

4) повернуть объект вокруг начала координат на тот же угол, что и во втором пункте, но в обратном направлении;

5) переместить объект вдоль осей *x* и *y* на те же расстояния, что и в первом пункте, но в обратных направлениях.

В матричном виде полное преобразование, представляющее собой комбинацию пяти преобразований, можно записать так:

$$[T] = [T_1][T_2][T_3][T_2]^{-1}[T_1]^{-1},$$

где $[T_1]$ – матрица перемещения,

$[T_2]$ – матрица поворота вокруг начала координат,

$[T_3]$ – матрица отражения относительно какой-либо координатной оси,

$[T_2]^{-1}$ и $[T_1]^{-1}$ – матрицы, обратные матрицам соответственно $[T_2]$ и $[T_1]$.

Таким образом, сложная операция разбивается на простейшие и задается произведением соответствующих матриц преобразования, причем порядок, в котором перемножаются матрицы, существенно определяет результат.

### 6.3 Практическая часть

Задание:

1) Написать программу, выполняющую над заданным треугольником, простейшие двумерные преобразования:

а) тождественное преобразование,

b) локальное масштабирование,

c) симметрию относительно осей координат,

d) симметрию относительно начала координат, сдвиг вдоль оси OX пропорционально координате y,

e) сдвиг вдоль оси OУ пропорционально координате x,

f) поворот на заданный угол вокруг начала системы координат,

g) симметрию относительно прямых $y = x$ и $y = -x$, перемещение (параллельный перенос) на заданный вектор,

h) проецирование в однородных координатах,

i) общее масштабирование.

С помощью простейших преобразований реализовать поворот относительно произвольной точки и симметрию относительно произвольной прямой.

При работе программы на экран должны выводиться оси координат. Заданный и преобразованный треугольники выводить разным цветом.

2) Изменить программу так, чтобы она могла выполнять эти преобразования для произвольного многоугольника, заданного своими вершинами. Проверить работу полученной программы, используя единичный квадрат.

**6.4 Литература:**

1) [1] с 76-114;

2) [3] с 36;

3) [5] с 63;

4) [6] с 181.

# Список использованных источников

1 Роджерс, Д. Математические основы машинной графики/ Д. Роджерс, Дж. Адамс — М.: Мир, 2001. — 604 с.

2 Роджерс, Д. Алгоритмические основы машинной графики/ Д. Роджерс — М.: Мир, 1989. — 512с.

3 Перемитина, Т.О. Компьютерная графика: учебное пособие / Т.О. Переметина. – Томск: Эль Контент, 2012, 144 с. Режим доступа: http://biblioclub.ru/index.php?page=book_view&book_id=208688

4 Иванов, Д.В. Алгоритмические основы растровой машинной графики: учебное пособие / Иванов Д.В., Карпов А.С., Кузьмин Е.П., Лемпицкий В.С., Хропов А.А. – М.: Интернет-Университет Информационных Технологий, 2007, 256 с. Режим доступа: http://biblioclub.ru/index.php?page=book_view&book_id=233998

5 Порев, В.Н. Компьютерная графика/ В.Н. Порев – СПб: БХВ - Петербург, 2005. – 432 с.

6 Шикин, Е.В. Компьютерная графика. Полигональные модели. [Электронный ресурс]: учебно-справочное издание / Е.В. Шикин, А.В. Боресков. - М.: Диалог-МИФИ, 2005. – 462с. - ISBN: 5-86404-139-4. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=89300