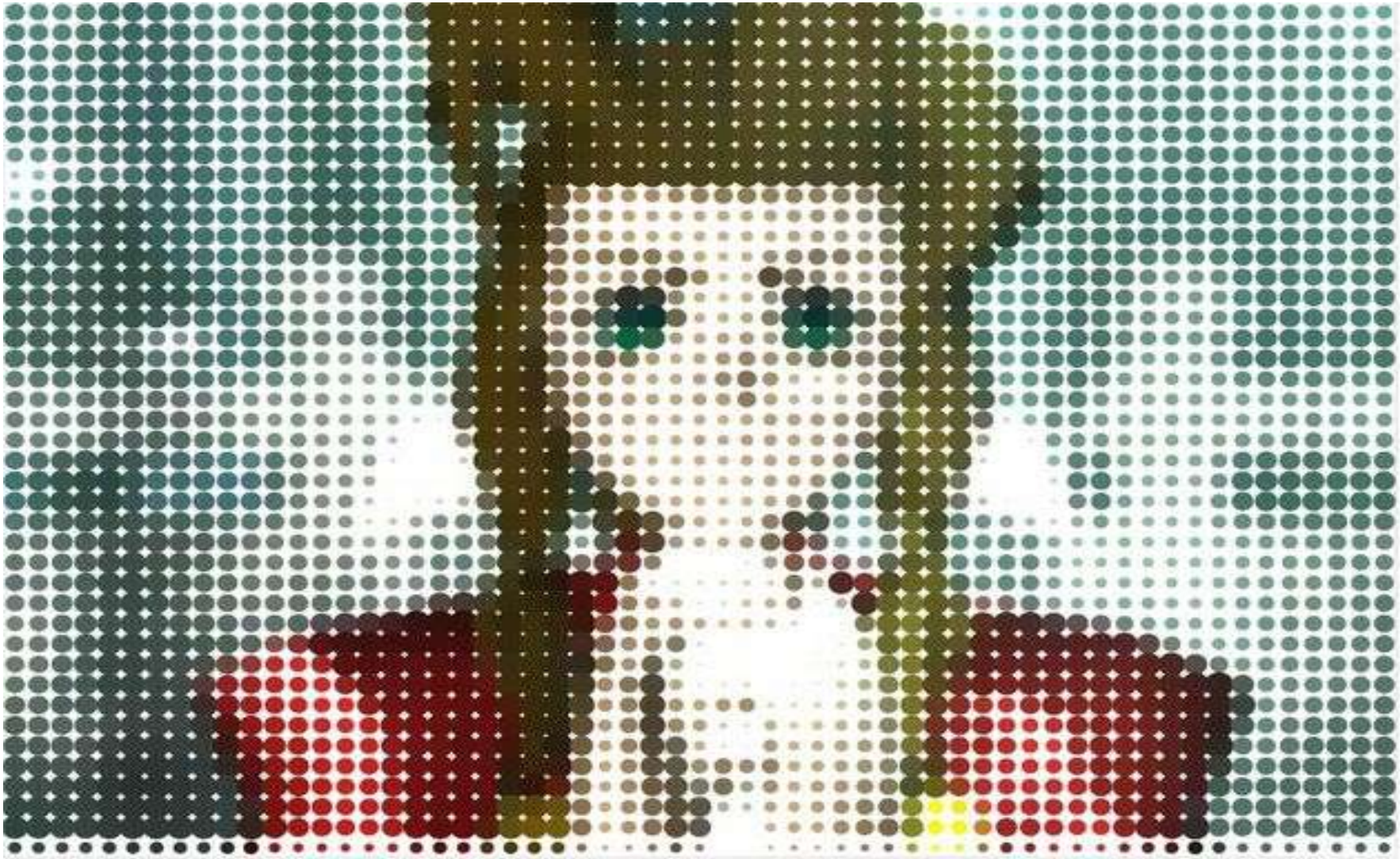
АЛГОРИТМЫ МАШИННОЙ ГРАФИКИ



Элементы растровой графики

А.В. Цыганов 2010

Способы описания изображения

- Растровая графика
- Растр – это порядок расположения точек (растровых элементов) или, другим словами, это матрица ячеек (пикселов).

- Векторная графика
- Изображение строится с помощью математических описаний объектов, окружностей и линий.

Графические форматы Windows & GDI

BMP – фактический стандарт – кроссплатформенный.

GIF – индексированная палитра цветов (в 3 раза меньше RGB), анимация в интернете

JPEG – сжатие с потерями, лучше для градиентный цветов

PNG – 16-bit **ARGB** разложение цветов, где "**A**"(or Alpha) обозначает прозрачность

TIFF – "удобен" для печатной продукции - Photoshop

Распространенные форматы файлов растровой графики

Формат	Макс. число бит/пиксел	Макс. число цветов	Макс. размер изображения, пиксел	Методы сжатия	Кодирование нескольких изображений
BMP	24	16'777'216	65535 x 65535	RLE*	-
GIF	8	256	65'535 x 65535	LZW	+
JPEG	24	16'777'216	65535 x 65535	JPEG	-
PCX	24	16'777'216	65535 x 65535	RLE	-
PNG	48	281'474'976'710'656	2'147'483'647 x 2 147 483 647	Deflation (вариант LZ77)	-
TIFF	24	16'777'216	всего 4'294'967'295	LZW, RLE и другие*	+

* Сжатие выполняется факультативно.

RAW (*raw* — сырой) — формат данных, содержащий необработанные (или обработанные в минимальной степени) данные, что позволяет избежать потерь информации, и не имеющий чёткой спецификации.

В таких файлах содержится полная информация о хранимом сигнале, и она может быть несжатой, сжатой без потерь, или сжатой с потерями.

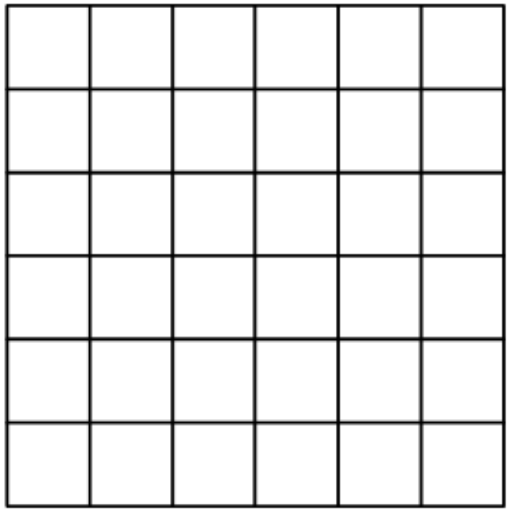
В RAW-файлах цифровых фотоаппаратов обычно содержатся:
«сырые» данные

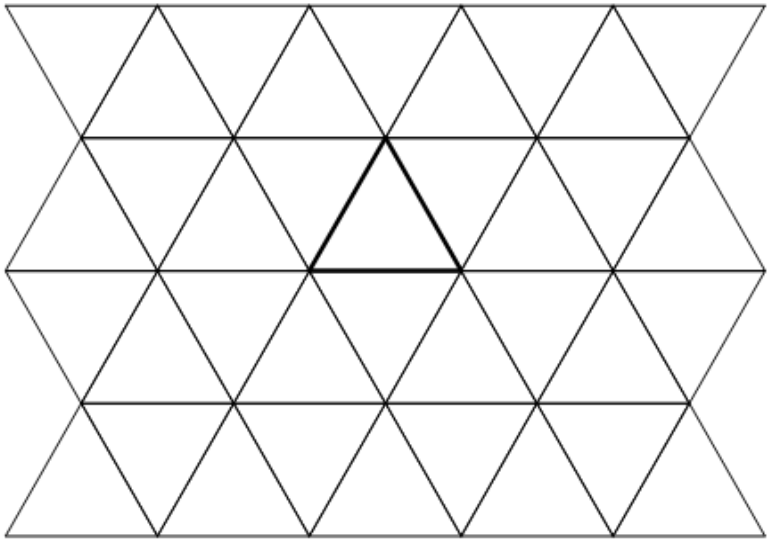
- ✓ метаданные — идентификация камеры
- ✓ метаданные — техническое описание условий съёмки
- ✓ метаданные — параметры обработки по умолчанию
- ✓ один или несколько вариантов стандартного графического представления («**превью**», обычно **JPEG** среднего качества), обработанные по умолчанию.

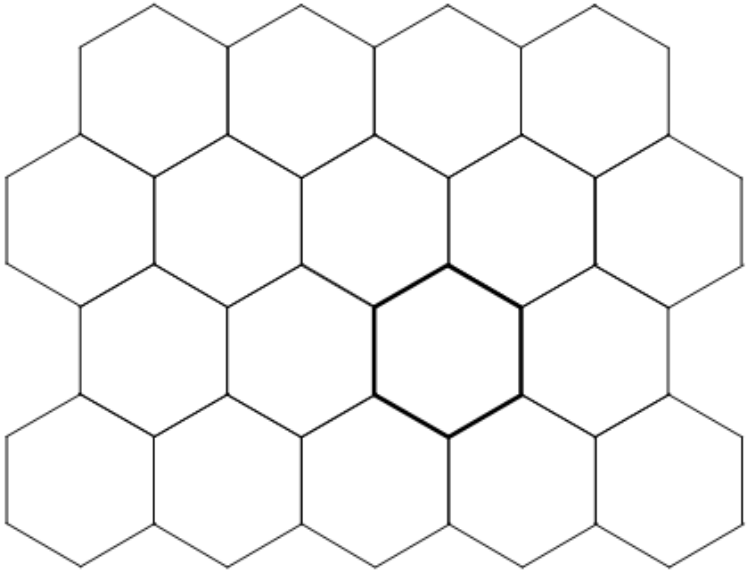
Файлы формата RAW имеют больший объём, чем **JPEG**, но меньший, чем, например, **TIFF**

Виды растров

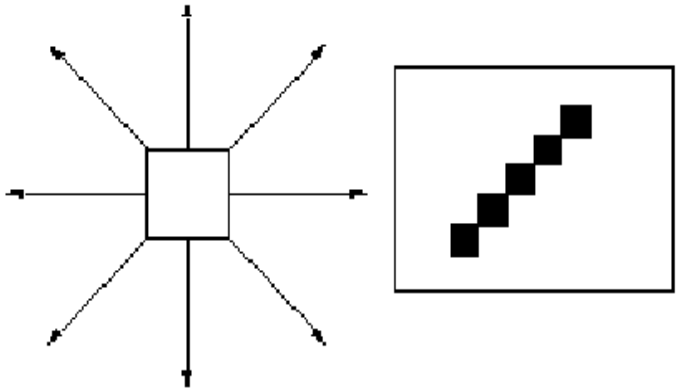
- квадратный
- треугольный
- гексоганальный растры



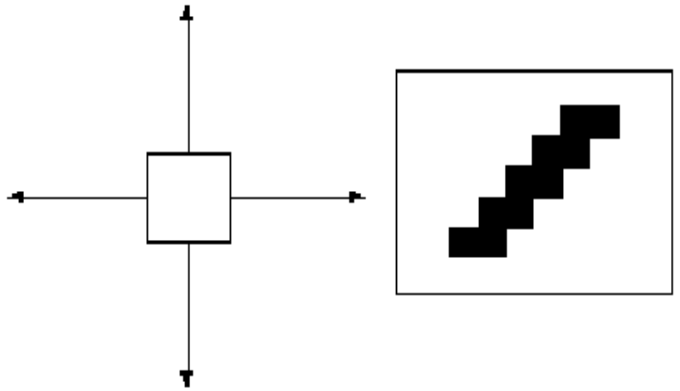




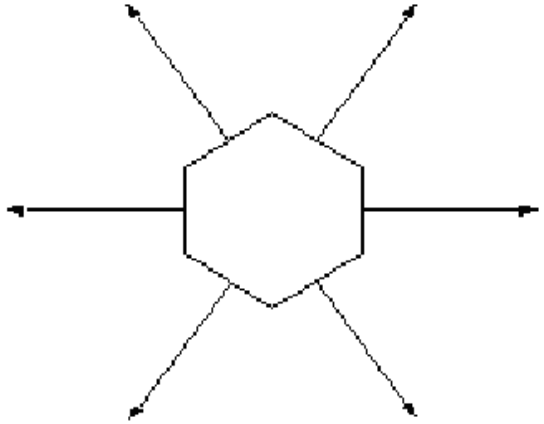
Пример построения линии

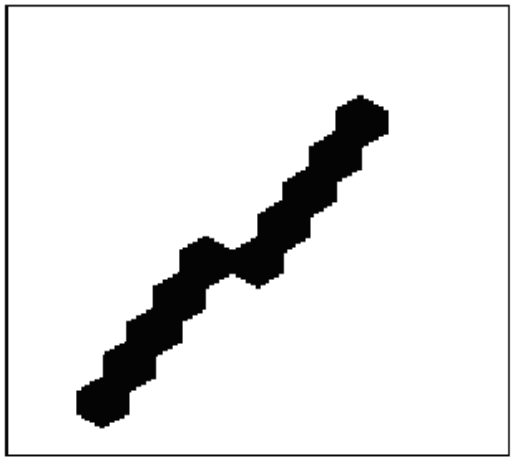


а



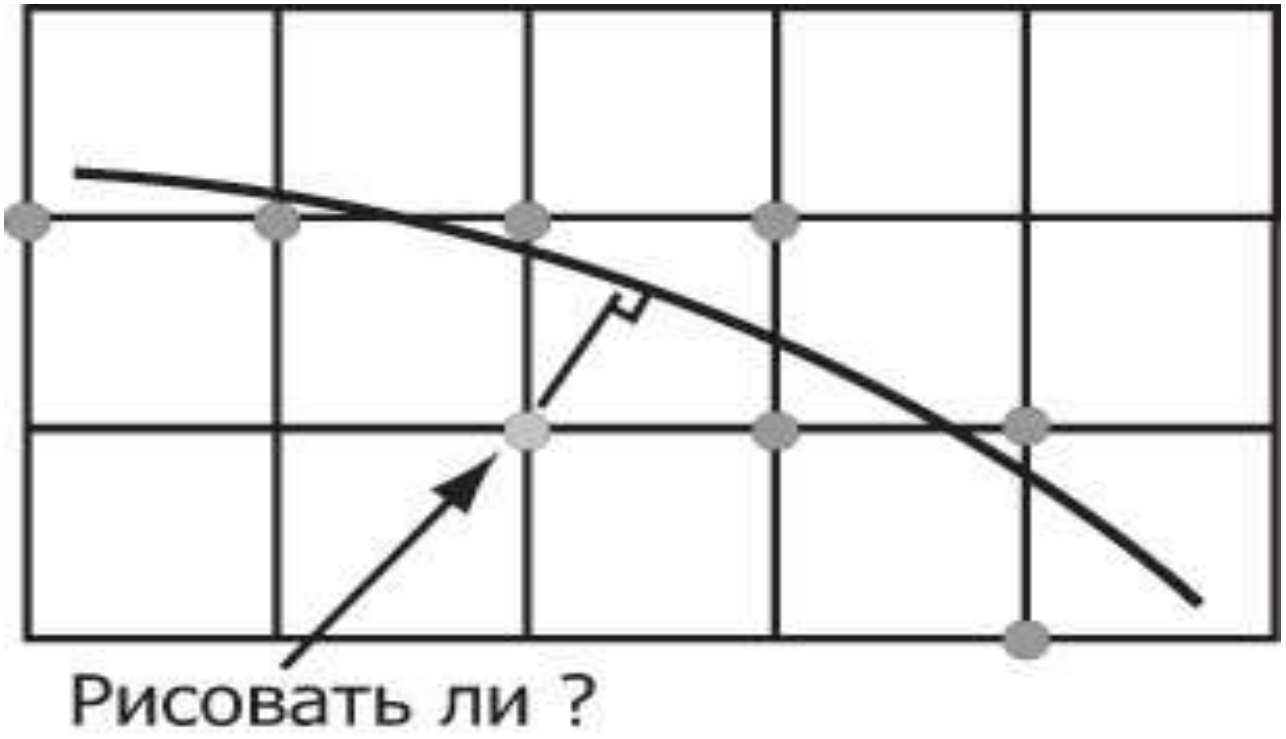
б





Пусть у нас “есть” некоторая кривая, и мы хотим построить ее изображение на растровой решетке.

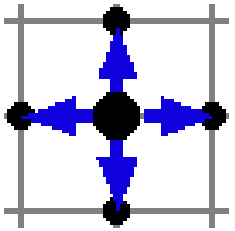
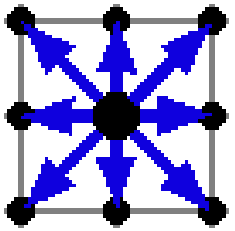
Возникает вопрос: какие из ближайших пикселей следует закрашивать?



Пусть **(x_0, y_0)** - фиксированный пиксель, а **(x, y)** - некоторый другой пиксель на плоскости.

Какое между ними расстояние (“близко” или “далеко”) ?

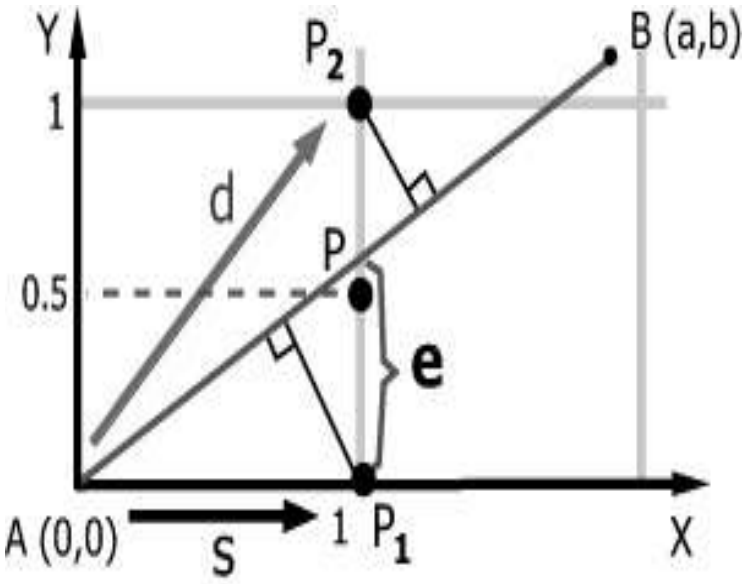
Для определения "близких" пикселей введем понятия **4- и 8-смежности (8-adjacent)** и основанные на них понятия **4- и 8-связности (8-connected)**: когда существует непрерывный путь из 4- или 8-смежных пикселов из **(x_0, y_0)** в **(x, y)**

1. **4-смежность** $\lvert x-x_0\rvert+\lvert y-y_0\rvert\le 1$	
2. **8-смежность** $\lvert x-x_0\rvert\le 1, \lvert y-y_0\rvert\le 1$	

Связность, метрика, смежность, длина, инвариант и т.д.

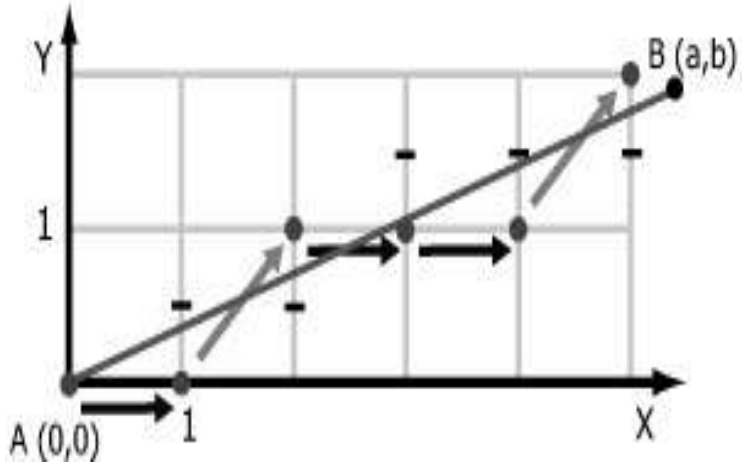
Цифровой дифференциальный анализатор

(**Digital Differential Analyzer**) построение 8-связного отрезка.



Пусть $P_1 = (1, 0)$ и $P_2 = (1, 1)$.

Так как мы закрашиваем “ближайший” из пикселей P_1 или P_2 нам следует вычислить и сравнить расстояния до них.



В силу подобия треугольников достаточно сравнивать гипотенузы вместо катетов, т.е координату **e** пересечения отрезка с прямой **x = 1**

```
// Координаты концов отрезка - (0,0) и (a,b)

  e = b/a;          // Текущая ордината
Δe = b/a;           // Приращение ординаты

// (x,y) - Координаты текущей точки

x = 0; y = 0;
while( x < a ) { plot(x, y);

    if( e > 1/2 )
    { // d : диагональное смещение
      x++; y++;
     // т.к. произошло смещение по y на 1 вверх
      e += Δe - 1; }
     else {
    // s : горизонтальное смещение
      x++;
      e += Δe; } }
```

Растеризация линий.

Инкрементные алгоритмы (incremental algorithms):

алгоритмы построенные только на основе целочисленных сложений и вычитаний, т.е. без умножения и деления

Типичными представителями инкрементных алгоритмов являются **алгоритмы Брезенхэма (Bresenham J.E)**:

- Алгоритм для отрезков, 1965 год (для плоттеров)
- Алгоритм для окружностей, 1977 год

Во всех алгоритмах наличие приращения по **x** или **y** рассчитывается на основе анализа **функции погрешностей**, построенной с различными вариантами на основе цифрового дифференциального анализатора.

Основная идея:

Модифицируем алгоритм DDA следующим образом:

1. **уменьшим везде e на 1/2 , чтобы сравнивать эту величину с 0, а не с дробью;**

2. **домножим e и Δe на 2a:**

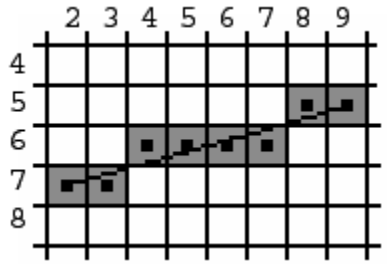
$$e_0 = 2b - a, \qquad \Delta e = 2b$$

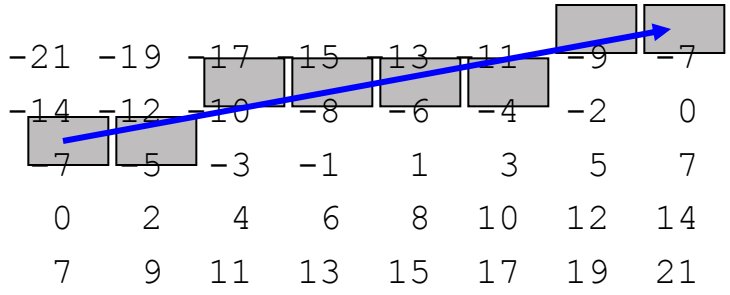
Дальнейшая оптимизация может быть произведена, если заметить, что отрезок **симметричен** относительно прямой, проходящей перпендикулярно ему через его середину; в этом случае можно начинать рисовать сразу с двух концов, что **сократит число итераций цикла в алгоритме вдвое**.

Восьмисвязный алгоритм Брезенхэма для отрезка (x1,y1),(x2,y2)**:**

```
//Bresenham's line-drawing algorithm (число шагов=d+1)
void line_bresenham8(int x1,int y1,int x2,int y2){
int x_Dd=0, y_Dd=0;                                  // функции погрешностей по x и y
int dx=x2-x1; int dy=y2-y1;                          // приращения по x и y на всем отрезке
int incX=0; int incY=0;                              // шаги приращения по x и y =0
if (dx>0) {incX=1; else if (dx!=0) incX=-1;}         // шаг приращения по x
if (dy>0) {incY=1; else if (dy!=0) incY=-1;}         // шаг приращения по y
dx=abs(dx); dy=abs(dy);
dx>dy?  d=dx: d=dy;                                  // выбор max(dx,dy) в качестве тестовой величины
int x=x1, y=y1; setPixel(x,y);                       // установка пикселя в начале отрезка
for (int i=1; i<=d; i++)
 {x_Dd+=dx; y_Dd+=dy;                                // приращение ошибки при перемещении на
 пиксель
 if (x_Dd>=d) {x+=incX; x_Dd-=d;}                    // условие шага по x
 if (y_Dd>=d) {y+=incY; y_Dd-=d;}                    // условие шага по y (+0.5d,-0.5d)!!!
 setPixel(x,y);}} // Dd = Δx·(y-y1) + Δy·(x-x1)  =0
```

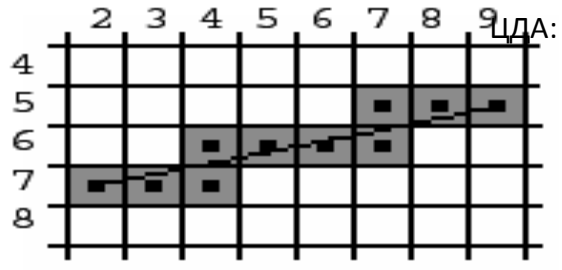
Пример для отрезка (2,7),(9,5):м



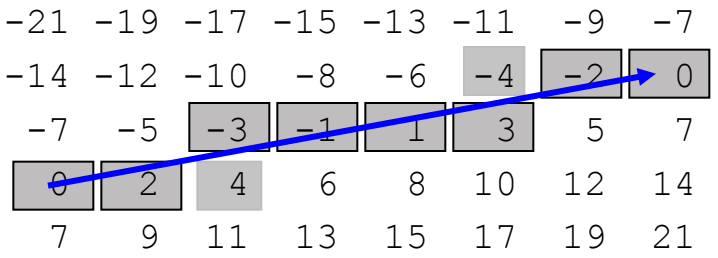


Растеризация линий.
Алгоритмы Брезенхэма

Результат работы 4-связного алгоритма (число шагов=dx+dy+1=10)
На том же множестве значений



ЦДА:

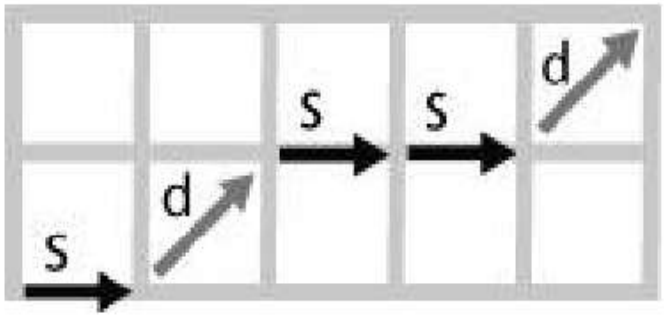


Грубо говоря было “2 – 4 – 2” стало “3 – 4 – 3” !!!!

Алгоритм Кастла-Питвея

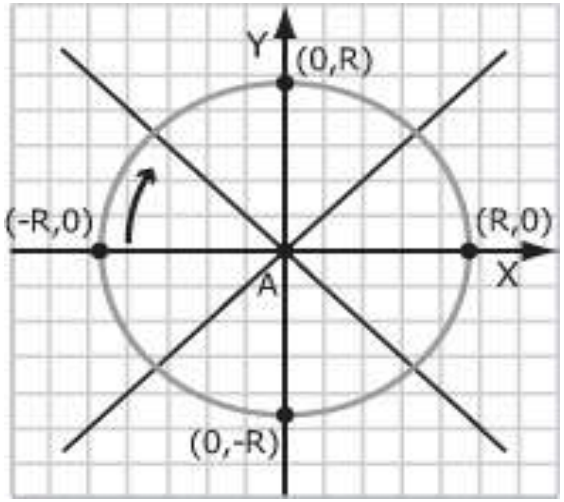
⊕ – конкатенация строк, пример «ssds» ⊕«sddd» = «ssdssddd»
~ – «переворот» строки, пример ~(«ssdds») =«sddss»

```
//Координаты концов (0,0) и (a,b)
y=b;        x=a-b;
m1="s";   m2="d";
while(x != y){
        if (x>y){
                x=x-y;
                m2=m1 ⊕~m2;
        }else{
                y=y-x;
                m1=m2 ⊕~m1;
        }
}
m=m2 ⊕~m1;
```

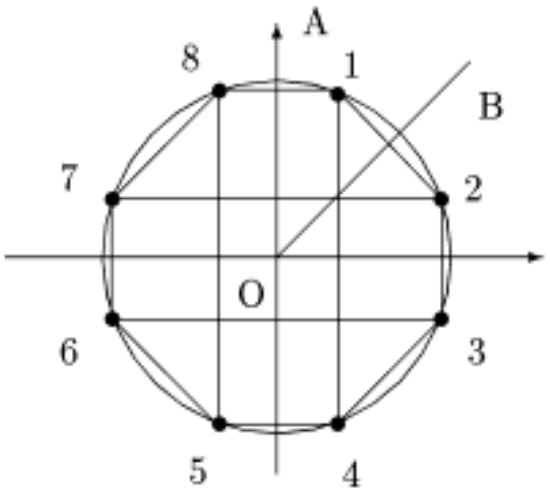


Построение окружностей

Для начала перейдем к канонической системе координат, в которой центр окружности совпадает с началом координат.

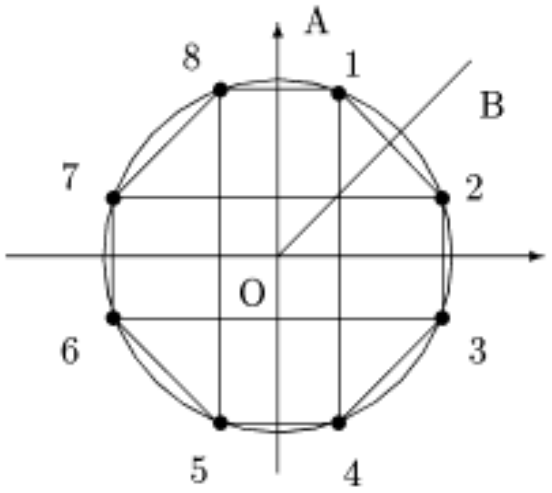


Можно легко понять, что в силу симметрии окружности относительно прямых, разделяющих плоскость на октанты, Достаточно построить растровое представление в одном октанте, а затем с помощью симметрии получить изображения в других октантах



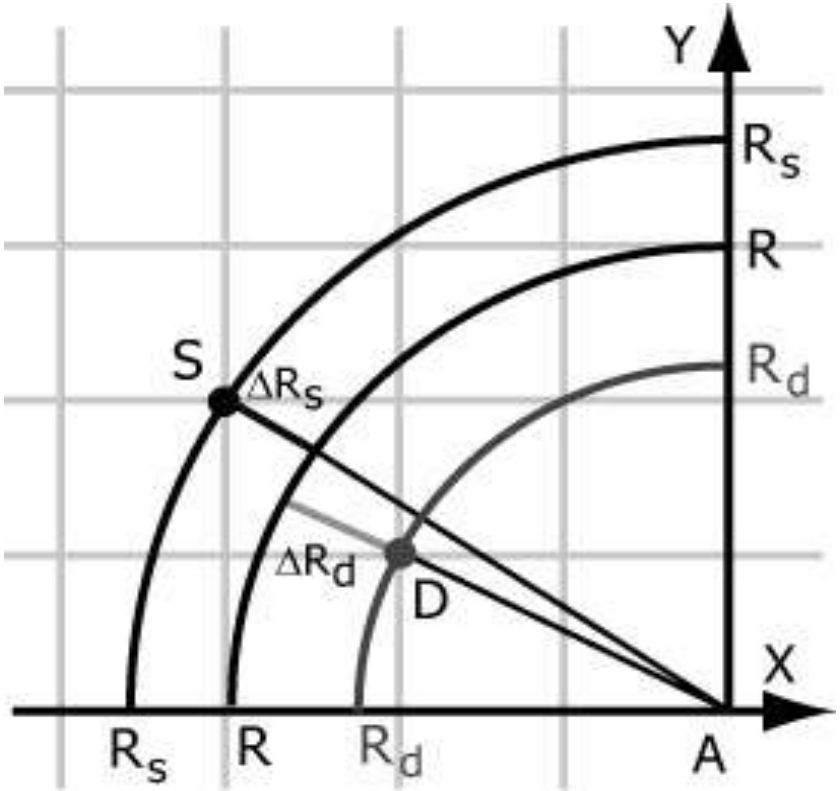
```
/*-------------------------------------------
 * circle8
 * Заносит пикселы окружности по часовой стрелке
 * xc,yc -координаты центра окружности
 * x,y - координаты пиксела
 */

static void circle8(int xc,int yc,int x,int y)
{
   setPixel(xc+x, yc+y);
   setPixel(xc+y, yc+x);
   setPixel(xc+y, yc-x);
   setPixel(xc+x, yc-y);
   setPixel(xc-x, yc-y);
   setPixel(xc-y, yc-x);
   setPixel(xc-y, yc+x);
   setPixel(xc-x, yc+y);
}
```



Алгоритм Брезенхема

Из двух возможных пикселов в 4-м октанте (соответствующих вертикальному и диагональному смещениям, которые обозначаются аналогично прежним ***S*** и ***D***) будем выбирать тот, расстояние **ΔR** от окружности до которого меньше.



Основная идея:

Будем задавать окружность в неявном виде:

$$x^2 + y^2 - R^2 = 0.$$

В этом случае:

$$\Delta R_s = \sqrt{x_s^2 + y_s^2} - R;$$
$$\Delta R_d = R - \sqrt{x_d^2 + y_d^2}.$$

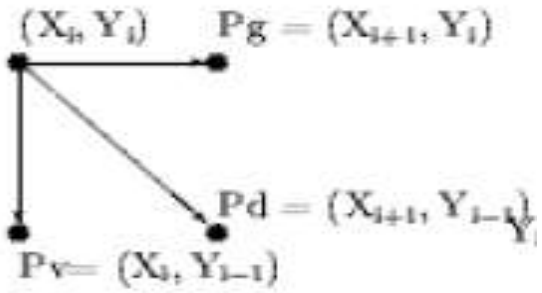
Если из первого вычесть второе, то приближенно ***S*** будет ближе к окружности, чем ***D*** если справедливо неравенство

$$x_s^2 + y_s^2 + x_d^2 + y_d^2 - 2R^2 < 0.$$

в противном случае ***D*** ближе к окружности, чем ***S***

Рассмотрим генерацию 1/8 окружности по часовой стрелке, начиная от точки **X=0**, **Y=R**.

Проанализируем возможные варианты нанесения **i+1**-й точки, после построения **i**-й точки.

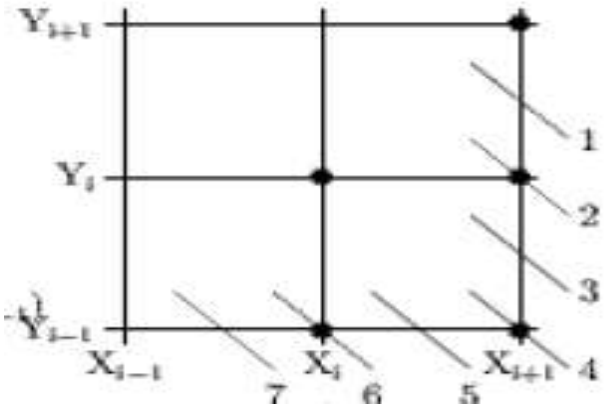


Расстояние от окружности до этих точек

$$|Dg| = |(X+1)^2 + Y^2 - R^2|$$
$$|Dd| = |(X+1)^2 + (Y-1)^2 - R^2|$$
$$|Dv| = |X^2 + (Y-1)^2 - R^2|$$

Выбирается и наносится та точка, для которой это значение минимально.

Выбор пикселей по значению ***Dd***.



здесь линии это возможное точное положение окружности

Если ***Dd < 0***, то диагональная точка внутри окружности. **Это варианты 1-3 (Pg или Pd).**

Если ***Dd > 0***, то диагональная точка вне окружности. **Это варианты 5-7 (Pd или Pv).**

Если ***Dd = 0***, то диагональная точка лежит точно на окружности. **Это вариант 4 - Pd.**

Рассмотрим случаи различных значений ***Dd*** в только что приведенной последовательности.

Случай $Dd < 0$

Здесь в качестве следующего пиксела могут быть выбраны или горизонтальный - **Pg** или диагональный - **Pd**.

Для определения того, какой пиксел выбрать **Pg** или **Pd** составим разность:

$$di=|Dg| - |Dd| =|(X+1)^2 + Y^2 - R^2| - |(X+1)^2 + (Y-1)^2 - R^2|$$

Будем выбирать точку **Pg** при $di> 0$, в противном случае выберем **Pd**.

Для остальных случаев то же самое – проще написать программу:

Начальная инициализация:

$X= 0$

$Y= R$

$Dd = (X+1)^2 + (Y-1)^2 - R^2 = 1 + (R-1)^2 - R^2 = 2*(1 - R)$

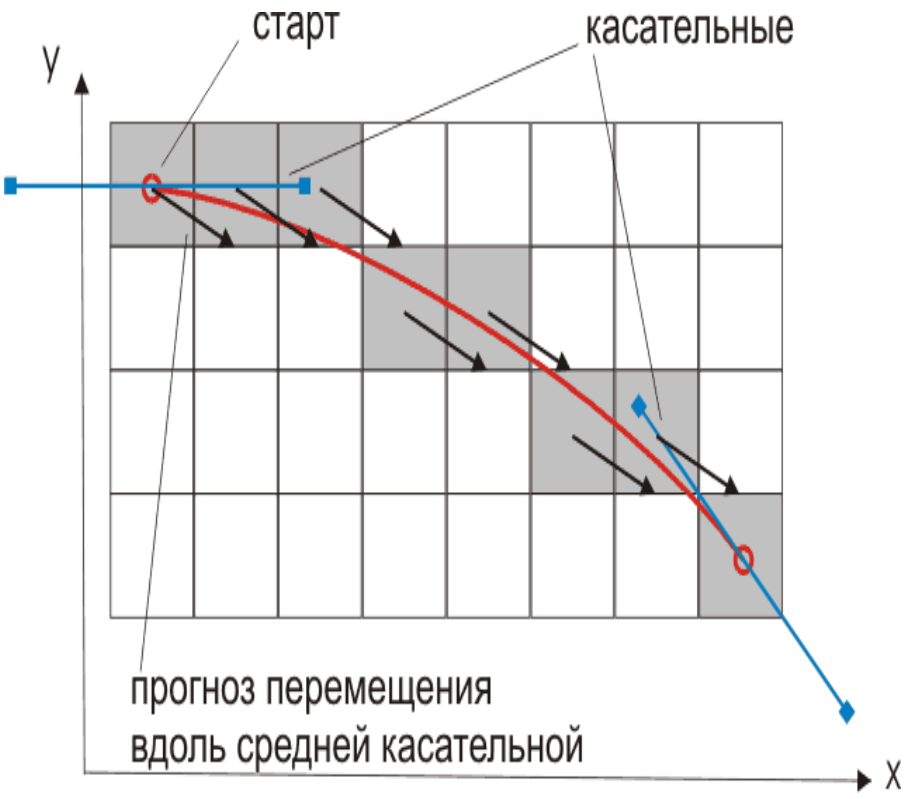
```
/*---------------------------------------------- circleBresenham
// К инкрементному алгоритму можно перейти, если умножать ошибки Dd=1.25-R на 4, исходную
// и далее, или просто отбросить 0.25 и считать Dd целым. Мы отбросили 0.25 и
// считаем Dd целым (int Dd)

 void circleBresenham (xc, yc, r, pixel)
int xc, yc, r, pixel;
{ int x, y, z, Dd;
   x= 0;  y= r;  Dd= 2*(1-r);                          //  начинаем с точки (0,R)
   while (x < y) {
      circle8(xc, yc, x, y, pixel);                   //  построение 8 точек по симметрии
      if (!Dd) goto Pd;
      z= 2*Dd - 1;
      if (Dd > 0) {
         if (z + 2*x <= 0) goto Pd; else goto Pv;
      }
      if (z + 2*y > 0) goto Pd;
Pg: ++x; Dd= Dd + 2*x + 1; continue;                  /* Горизонталь */
 Pd: ++x; --y; Dd= Dd + 2*(x-y+1);   continue;        /* Диагональ */
Pv: --y; Dd= Dd - 2*y + 1;                            /* Вертикаль */
   }
   if (x == y)  circle8 (xc, yc, x, y, pixel);        //  построение 8 точек по симметрии
}
```

Пусть x_i, y_i принадлежит окружности:

$$F(x_i, y_i)=0.$$

Применим метод средней точки (midpoint algorithm, модификация Мичнера), заключающийся в оценке ошибки для наиболее вероятного прогноза движения вдоль **среднего (по интервалу) значения касательной** (=-1/2).



$Dd_i=F(x_i+1, y_i -1/2)=2x_i - y_i + 5/4$

Если $Dd_i<0$, тогда $y_{i+1} = y_i$ и

$Dd_{i+1}=F(x_i+2, y_i -1/2)=4x_i - y_i + 17/4$

$dDd_i= Dd_{i+1}- Dd_i= 2x_i + 3$

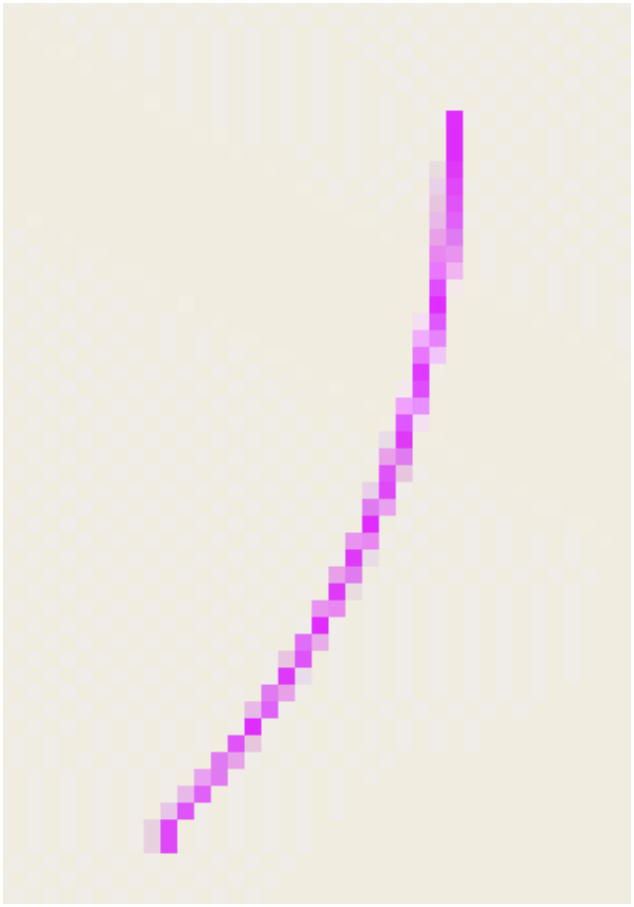
Если $Dd_i>0$, то $y_{i+1} = y_i -1$ и

$Dd_{i+1}=F(x_i+2, y_i -3/2)=4x_i - 3y_i + 25/4$

$dDd_i= Dd_{i+1}- Dd_i= 2(x_i - y_i)+ 5$

Тоновые алгоритмы построения отрезков

Общим недостатком рассмотренных нами ранее алгоритмов является то, что они рисуют отрезки с **неровными, резкими краями**. Для преодоления этого недостатка **Wu Xiaolin** создал алгоритм, рисующий "сглаженный" отрезок.



Алгоритм Ву является одним из методов **антиалиасинга** (иногда это слово пишется, как анти-алиасинг).

Предыдущие алгоритмы рисовали отрезки одним цветом, этот алгоритм закрашивает разные участки отрезка в разные цвета, и за счет этого "сглаживает" неровности.

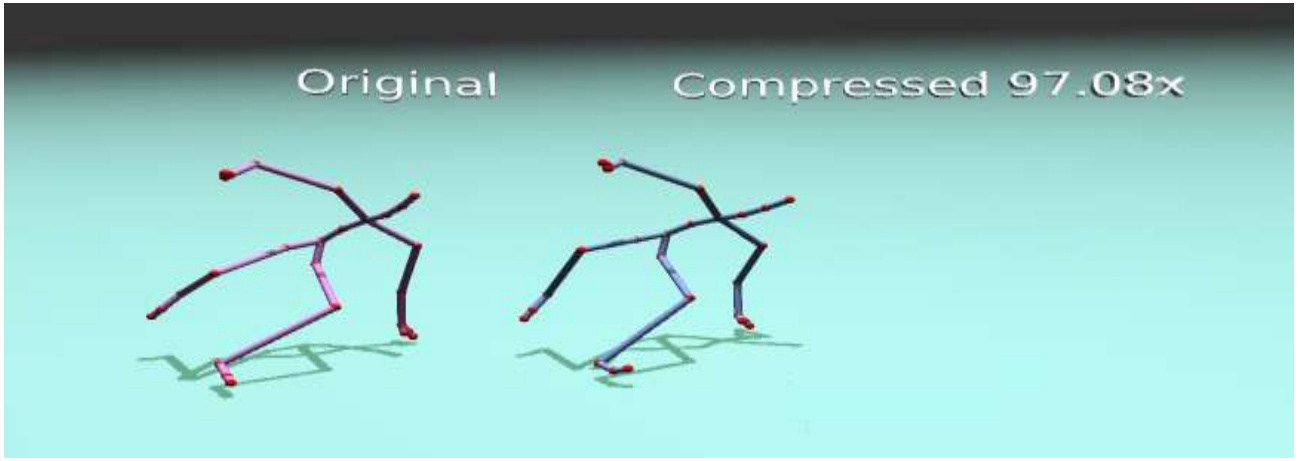
Также следует отметить, что этот алгоритм использует не целые координаты концов отрезка, а вещественные.

В этом есть недостаток (**большее время работы**) но и несколько достоинств.

Например, при изменении координат (движении камеры) отрезок, нарисованный алгоритмом **Брэзенхема**, перемещается резко, скачками.

Отрезок по алгоритму **Ву** будет перемещаться непрерывно.

За счет этого можно обеспечивать плавную анимацию при рисовании движущихся изображений.

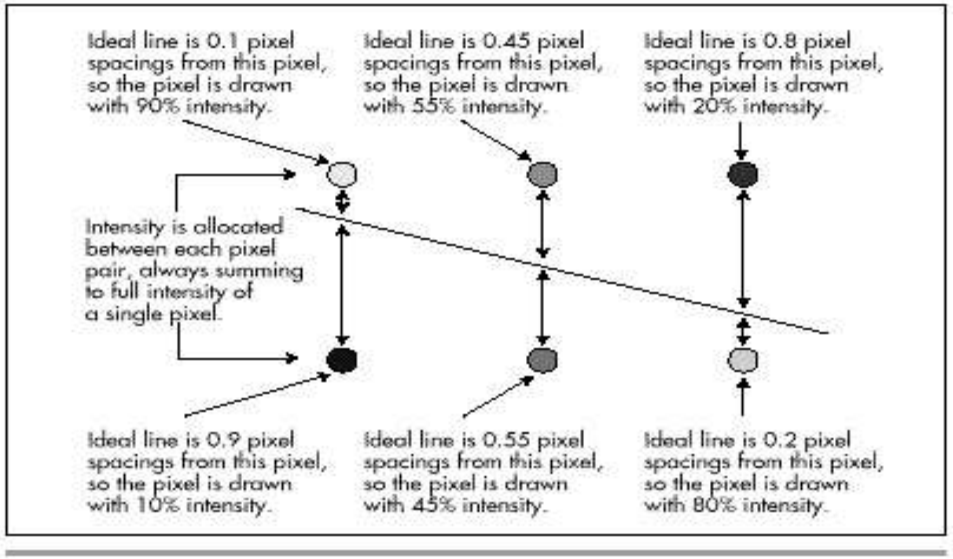


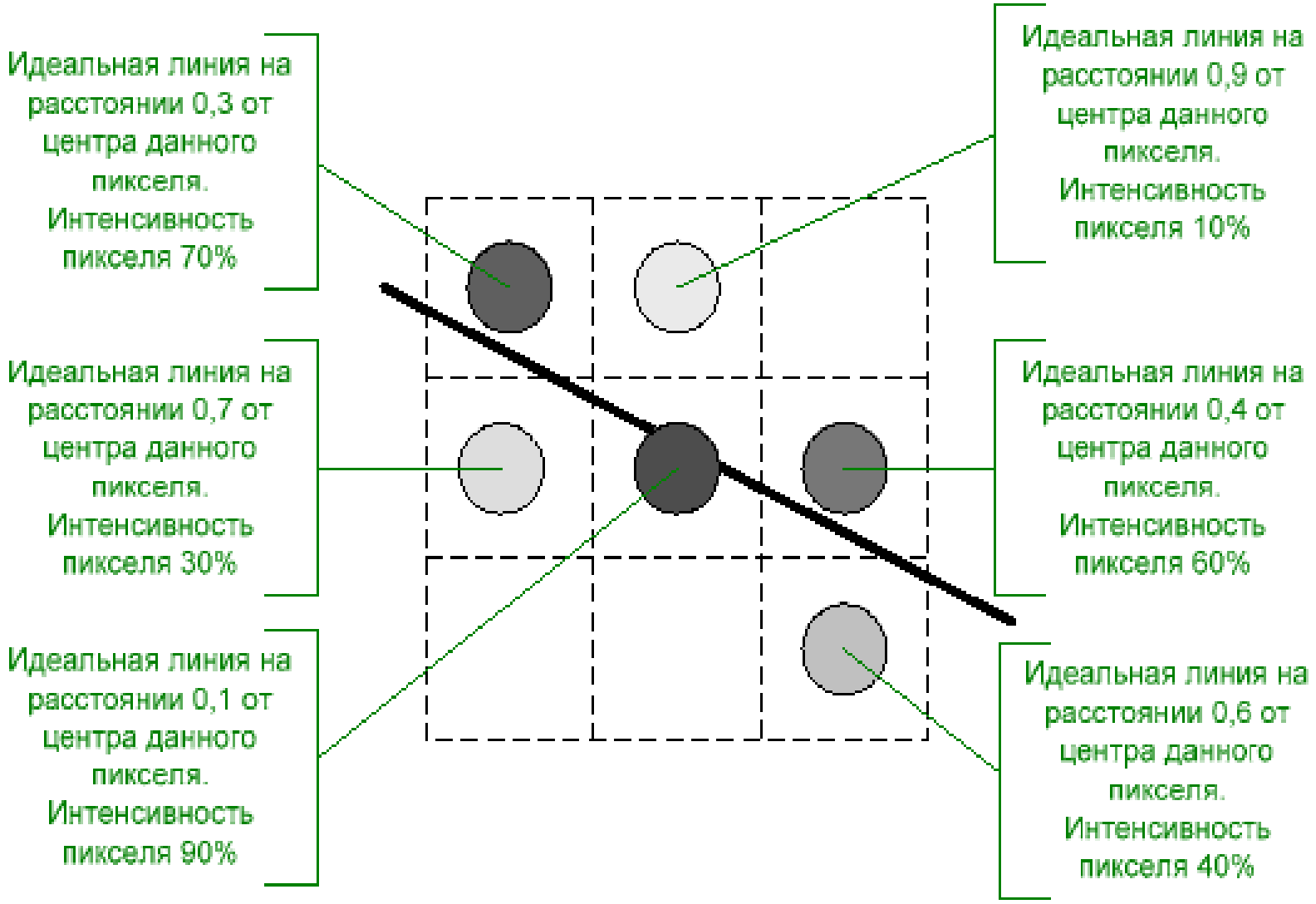
Принцип работы алгоритма

Основная идея алгоритма - работа с **парами пикселей**, между центрами которых проходит наш отрезок.

Здесь пиксели - это квадраты со стороной 1 и центрами, расположенными в узлах целочисленной решетки.

Когда мы говорим "пиксель с координатами (***x,y***)", мы имеем в виду, что его ***центр*** расположен в этой точке.




Идеальная линия на расстоянии 0,3 от центра данного пикселя. Интенсивность пикселя 70%
Идеальная линия на расстоянии 0,9 от центра данного пикселя. Интенсивность пикселя 10%
Идеальная линия на расстоянии 0,7 от центра данного пикселя. Интенсивность пикселя 30%
Идеальная линия на расстоянии 0,4 от центра данного пикселя. Интенсивность пикселя 60%
Идеальная линия на расстоянии 0,1 от центра данного пикселя. Интенсивность пикселя 90%
Идеальная линия на расстоянии 0,6 от центра данного пикселя. Интенсивность пикселя 40%

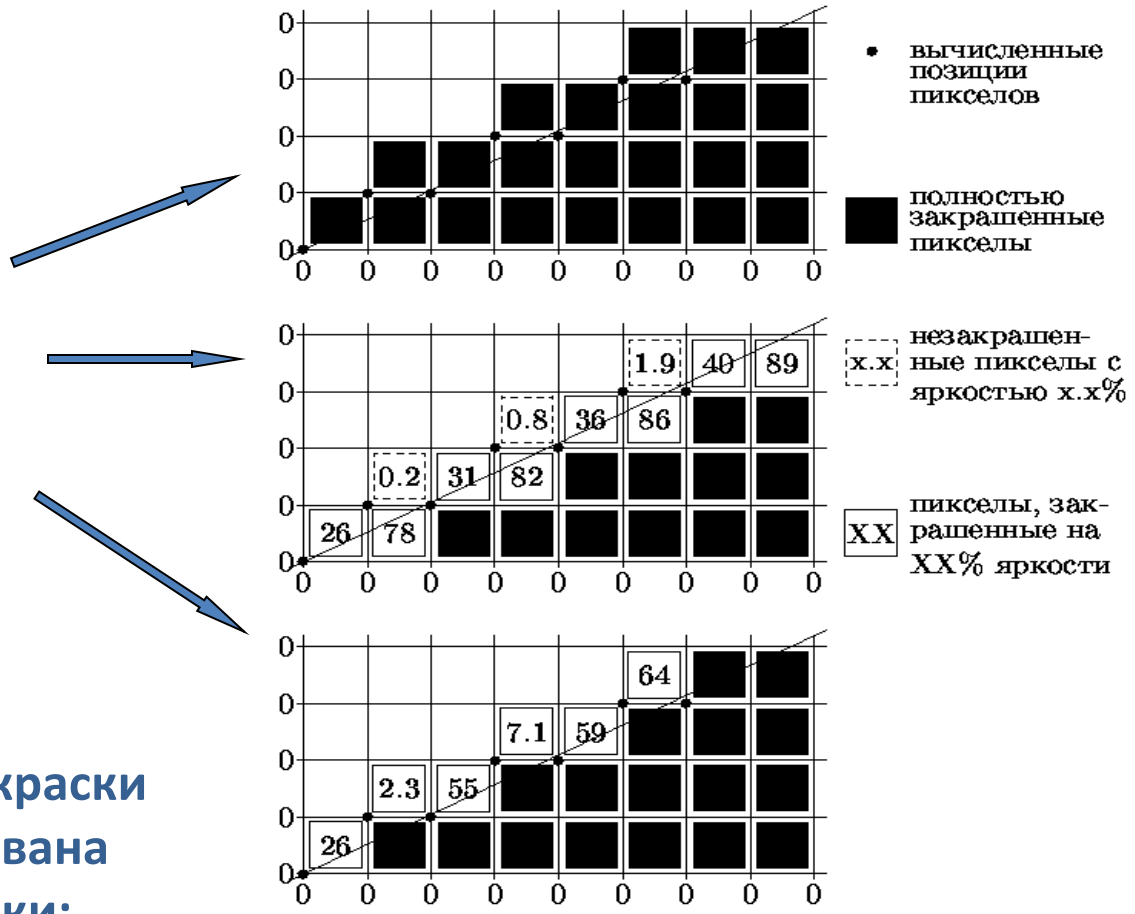
```
#include <stdafx.h>
#include "wucircle.h“
void drawwucircle(int cx, int cy, int r)
{
  int x;    int y;    double t;    double d;    int j;    int kx;    int ky;    int lastx;   .//  хулиганство
   x = r;    lastx = r;    y = 0;    t = 0;
 for(j = 0; j <= 3; j++)
   {
      kx = j%2*2-1;
      ky = j/2%2*2-1;
      setpixel(kx*x+cx, ky*y+cy, double(1));
      setpixel(kx*y+cx, ky*x+cy, double(1));
   }
   while(x>y)
   {
      y = y+1;
      d = ap::iceil(sqrt(double(r*r-y*y)))-sqrt(double(r*r-y*y));
      if( d<t )
      {
         x = x-1;
      }
      if( x<y )
      {          break;       }
      if( x==y&&lastx==x )
      {          break;        }
      for(j = 0; j <= 3; j++)
      {
         kx = j%2*2-1;
         ky = j/2%2*2-1;
         setpixel(kx*x+cx, ky*y+cy, 1-d);
         setpixel(kx*y+cx, ky*x+cy, 1-d);
         if( x-1>=y )
         {
            setpixel(kx*(x-1)+cx, ky*y+cy, d);
            setpixel(kx*y+cx, ky*(x-1)+cy, d);
         }
      }
      t = d;
      lastx = x;
   }
}
```

Растеризация фигур.

Модификация алгоритма Брезенхэма со сглаживанием границы

Модификация растеризации линии по Брезенхэму с целью сглаживания границы:

- Полная закраска
- Неполная закраска 8-связный вариант
- Неполная закраска, 4-связный вариант



Для оценки процента закраски (p) может быть использована величина функции ошибки:

p=100·(d-err)/d

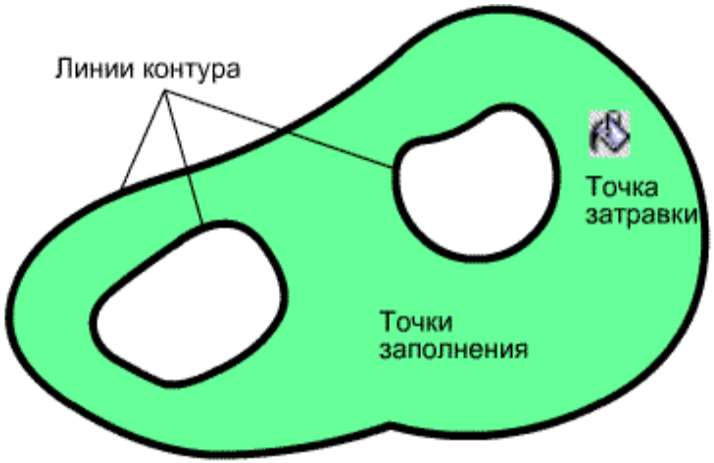
Алгоритмы закрашивания.

Алгоритмы вывода фигур

Фигура (region) -плоский геометрический объект, состоящий из **линий контура** и **точек заполнения**, содержащихся внутри контура.

Поэтому две задачи:

1) **нарисовать контур**;
2) **построить точки заполнения**.



В выводе точек заполнения различают:

1) **закрашивание** (от внутренней точки - затравки) до границы заданного цвета, либо закрашивание пикселов, принадлежащих заданному цветовому диапазону; ~pixel-defined regions
2) **закрашивание контура**, заданного математически ~ symbolic-defined regions.

Затравочный пиксел для фигур заданных цветовой границей или цветовым диапазоном, как правило, вносится пользователем.

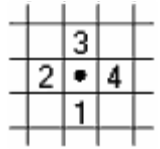
Внутренний пиксел для фигур заданных математически, при необходимости, может быть определен либо на основе решения задачи пересечения фигуры с тестовой горизонталью, либо вычислением суммарного угла при обходе всех вершин фигуры лучем из тестируемой точки (=360 градусов).

Алгоритмы закрашивания до цвета границы.

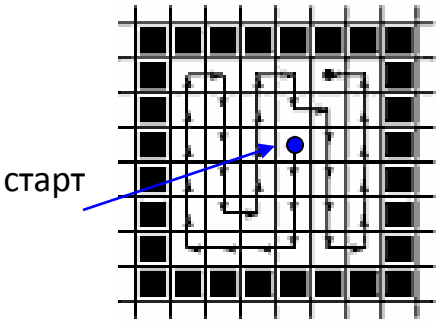
Простейший рекурсивный алгоритм

Для 8-связной границы нельзя применять 8-связный алгоритм закрашивания, но можно 4-связный.

```
void FillPixel(int x, int y, int borderC, int fillC)
{ // 4-связный вариант
int c=getPixel(x,y);                    // цвет пиксела
if ((c!=borderC)&&(c!=fillC))           // не граничный и не закрашен?
                {setPixel(x,y,fillC);   // закрасить и «осмотреться»:
                FillPixel(x+1,y,borderC,fillC);   //Рекурсия
                FillPixel(x,y+1,borderC,fillC);   //
                FillPixel(x-1,y,borderC,fillC);   //
                FillPixel(x,y-1,borderC,fillC); } //
}
```



Порядок перебора соседних пикселов

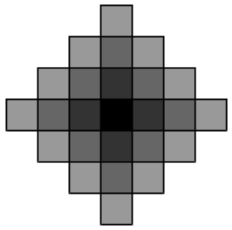


Пример обхода области

Алгоритм работает пока стек рекурсии не пуст. Имеет низкую эффективность

1) требует большой глубины рекурсии ~ 4n pix (при стеке в 64K рекурсивным алгоритмом можно закрасить квадрат 57×57 пикселов);

2) каждый уже отрисованный пиксел тестируется еще 3 раза.



Волновой алгоритм

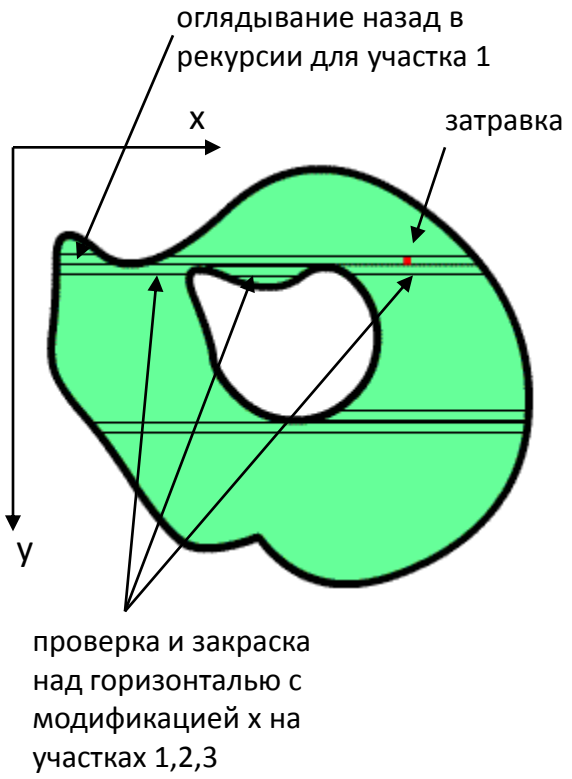
Более рационален **волновой** алгоритм (см. рис). Для хранения координат пикселов текущего ромбовидного фронта волны использует динамичекие массивы до 10000 элементов.

Алгоритмы закрашивания до цвета границы.

Закрашивание линиями

На каждом шаге закрашивается горизональная линия между пикселами контура и обнаруживаются незакрашенные пикселы над и под линией. **Рекурсивный, но глубина рекурсии пропорциональна лишь числу пикселов в линии**. В случае создания стека достаточно хранить 1 пиксел на каждый закрашиваемый участок.

```
int lineFill(int x, int y,int dy,int preXL, int preXR)
{int xl=x, xr=x; int c;
//левая и правая граница текущей горизонтали
do {xl--; c=getPixel(xl,y):} while (c!=BORDER);
do {xr++; c=getPixel(xr,y):} while (c!=BORDER);
xl++; xr--;
Line(xl,y,xr,y,fillC); //закрашиваем горизонталь
//Рекурсии: 1)проверка и закраска над горизонталью
for (x=xl; x<=xr; x++) {c=getPixel(x,y+dy);
   if (c!=BORDER) x=lineFill(x,y+dy,dy,xl,xr);}
// 2)оглянулись и закрасили левее предыдущей левой
// и правее предыдущей правой границ
for (x=xl; x<=preXL; x++) {c=getPixel(x,y-dy);
   if (c!=BORDER) x=lineFill(x,y-dy,-dy,xl,xr);}
for (x=preXR; x<=xr; x++) {c=getPixel(x,y-dy);
   if (c!=BORDER) x=lineFill(x,y-dy,-dy,xl,xr);}
return xr; }
```



Построчное заполнение контура полигона, заданного списком вершин (polygon-defined region)

Алгоритм "XY"

Свойство топологии контура: ***любая прямая пересекает контур четное число раз.***

1. Найти $\min\{y_n\}$ и $\max\{y_n\}$ по всем N вершинам P_n
2. Выполнить цикл по y от ymin до ymax:

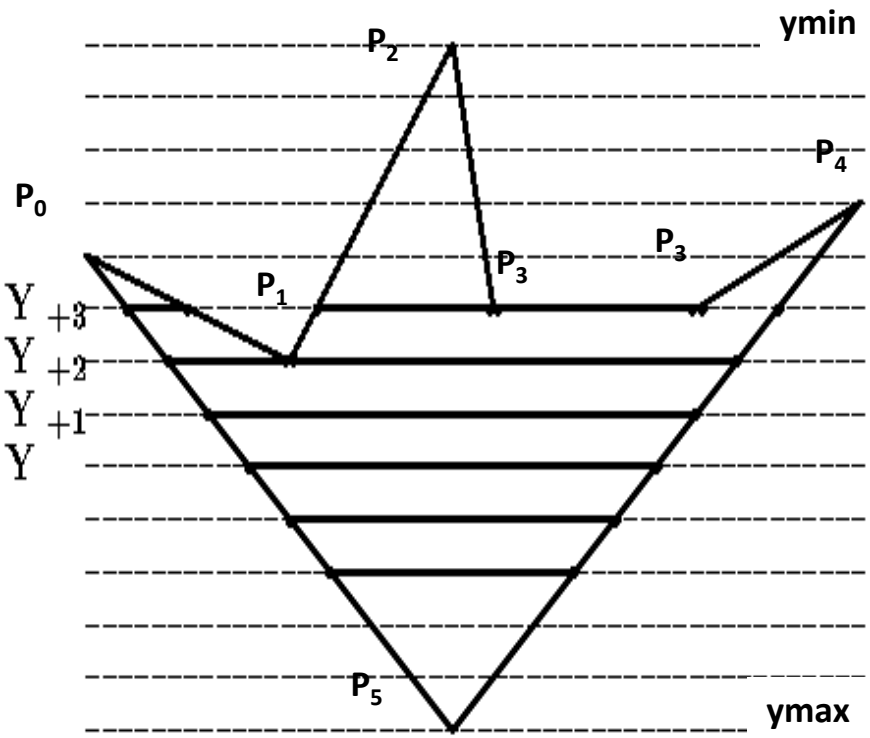
{ 2.1. Расчет множества $\{x_k\}$ x-координат пересечения всех отрезков границы с горизонталью y

$x_k = x_n + [(x_{n+1} - x_n)\cdot(y - y_n)]/(y_{n+1} - y_n)$

2.2. Сортировка $\{x_k\}$ по возрастанию

2.3. Вывод отрезков $\{(x_0,y)-(x_1,y), (x_2,y)-(x_3,y), \ldots, (x_{2m},y)-(x_{2m+1},y)\}$ цветом заливки

}



Число тактов работы алгоритма: $N_{тактов} = (y_{max} - y_{min}) \cdot N_{горизонтали} = (y_{max} - y_{min}) \cdot (N+1)$.

Свойство топологии контура можно также использовать **для поиска полигона по указанной внутренней точке**.

Моделирование кривых.

Сплайны

Параметрические кривые

P(t) =(x(t) , y(t)) – точка параметрической кривой, ***t*** - параметр

v(t)=dP(t)/dt – вектор скорости

$L(t_0,u)=P(t_0) + u\, v(t_0)$ - уравнение касательной

$n(t_0)=v_\perp(t_0)=(-dy(t_0)/dt, dx(t_0)/dt)$ – нормаль к кривой

Рациональные параметрические формы

Каждая из координат определена как отношение (ratio) двух полиномов:

$$P(t) = \frac{P_0(1-t)^2 + 2wP_1t(1-t) + P_2t^2}{(1-t)^2 + 2wt(1-t) + t^2}$$

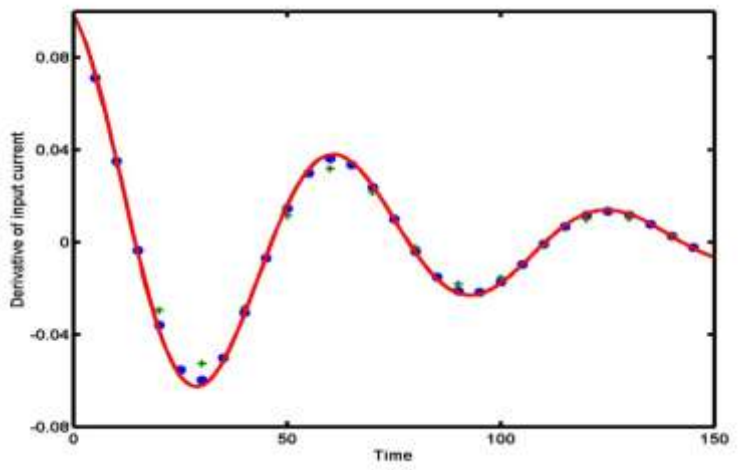
При ***w<1*** – эллипс, при ***w=1*** – парабола, при ***w>1*** – гипербола, ограниченные ломаной $P_0P_1P_2$.

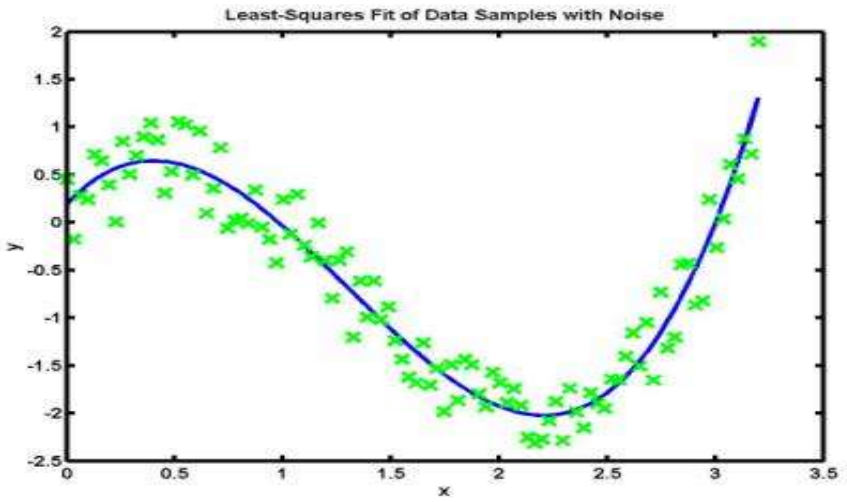
Параметрические кривые, а также поверхности, применяются для описания более сложных, чем плоские, форм. Они находят широкое применение в графическом и промышленном дизайне.

В CG нас будут интересовать две задачи:

Интерполяция - построение кривой, проходящей через контрольные точки и обладающей некими дополнительными свойствами (например гладкостью);

Аппроксимация - приближение кривой (не обязательно проходит точно через данные точки, но удовлетворяет некоторому заданному свойству относительно этих точек).



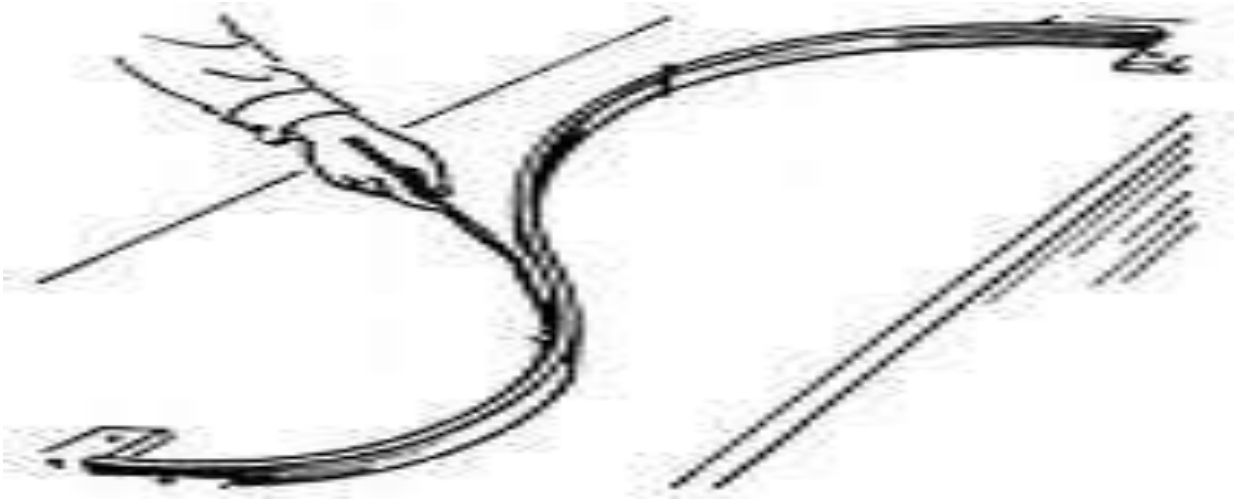


Интерполяция сплайнами

Сплайн - кусочный многочлен степени **n** с непрерывной производной степени **m** в точках соединения сегментов.

Далее нас будут интересовать самые распространенные **кубические сплайны**. (почему кубические?)

Понятие сплайна пришло из кораблестроения, где сплайном называли гибкую линейку (веревку), закрепив которую в нужных местах, добивались плавной формы кривой.



Интерактивное конструирование кривых.

Кривая Безье (Bezier Curve)

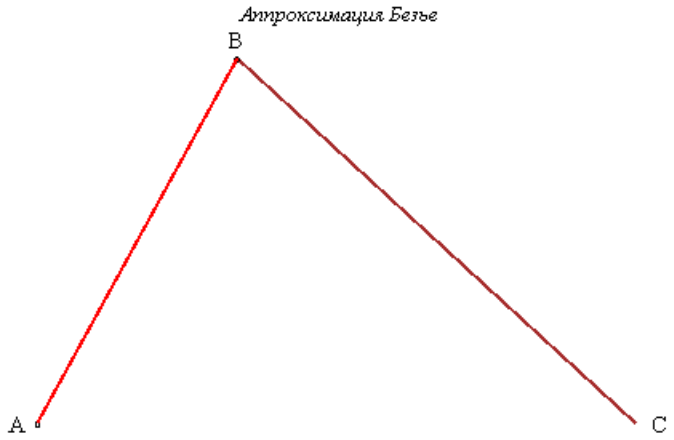
Кривые Безье были разработаны Пьером Безье (Pierre Bezier) и независимо от него Полем де Кастельжо (Кастельо, Paul de Casteljau) примерно в 1962 году. Эти кривые были включены в системы геометрического проектирования (CAGD) двух автомобильных компаний: "Рено" и "Ситроен".

Алгоритм де Кастельжо (геометрический)

Метод de Casteljau основан на разбиении отрезков, соединяющих исходные точки в отношении **t** (значение параметра), а затем в рекурсивном повторении этого процесса для полученных отрезков.

$$P_0^1 = (1-t)P_0 + tP_1 \quad P_1^1 = (1-t)P_1 + tP_2$$

$$P(t) = P_0^2 = (1-t)P_0^1 + tP_1^1 = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$



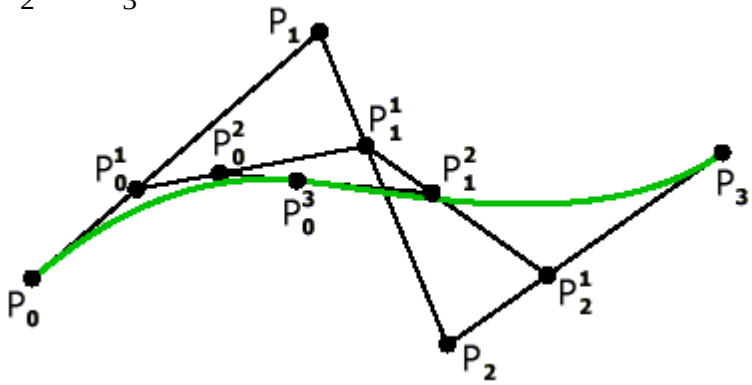
Кривая Безье (Bezier Curve)

Теперь постоим по алгоритму де Кастельжо кривую Безье с 4 опорными точками – начинается все с ломаной Безье.

$$P_0^1 = (1-t)P_0 + tP_1 \quad P_1^1 = (1-t)P_1 + tP_2 \quad P_2^1 = (1-t)P_2 + tP_3$$

$$P_0^2 = (1-t)P_0^1 + tP_1^1 = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

$$P_1^2 = (1-t)P_1^1 + tP_2^1 = (1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3$$



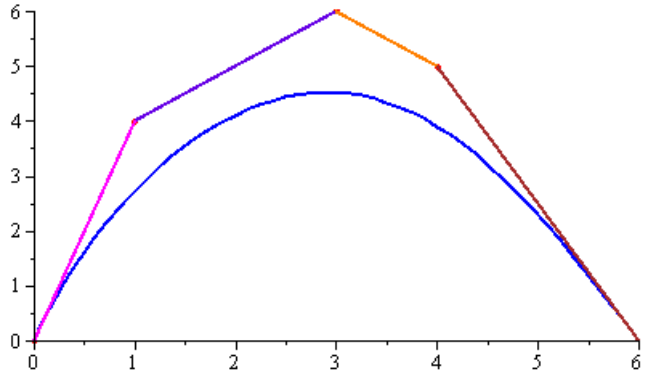
$$P_0^3(t) = (1-t)P_1^2 + tP_2^2 = (1-t)^2 P_0^1 + 2t(1-t)P_1^1 + t^2 P_2^1 =$$

$$(1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3$$

итоговое уравнение кривой

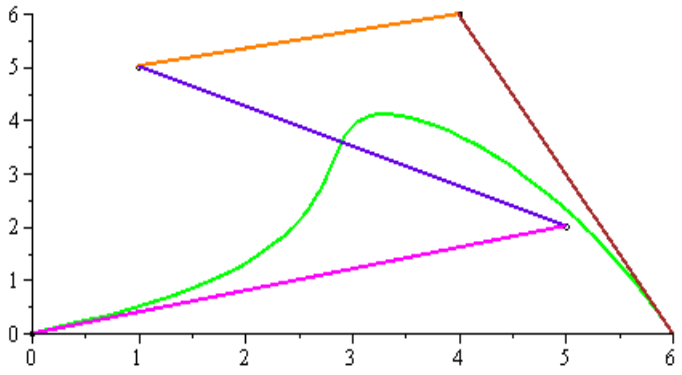
Кривая Безье (Bezier Curve)

параметрическое описание через полиномы Бернштейна



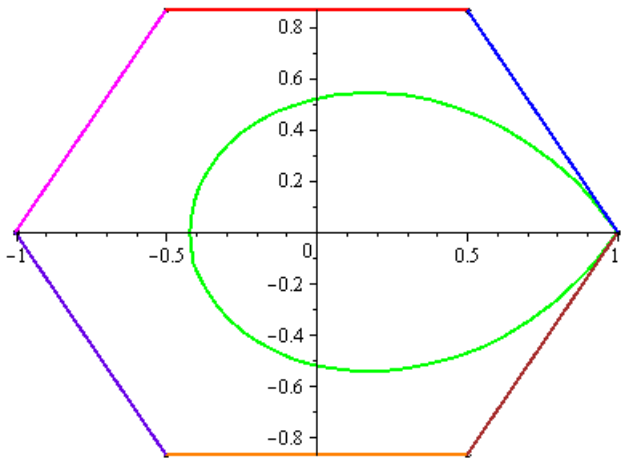
$$x = 4t^4 - 8t^3 + 6t^2 + 4t$$

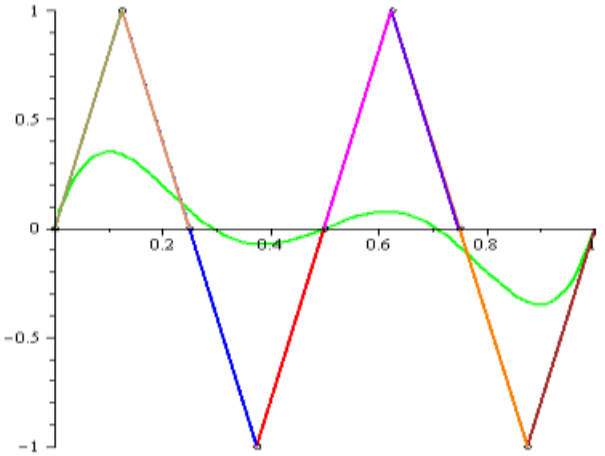
$$y = -4t^3 - 12t^2 + 16t$$



$$x = -24t^4 + 64t^3 - 54t^2 + 20t$$

$$y = -2t^4 - 12t^3 + 6t^2 + 8t$$





Кривая Безье. Многочлены Бернштейна

Запишем общее аналитическое представление для кривой Безье с ***n+1*** опорной точкой: ,

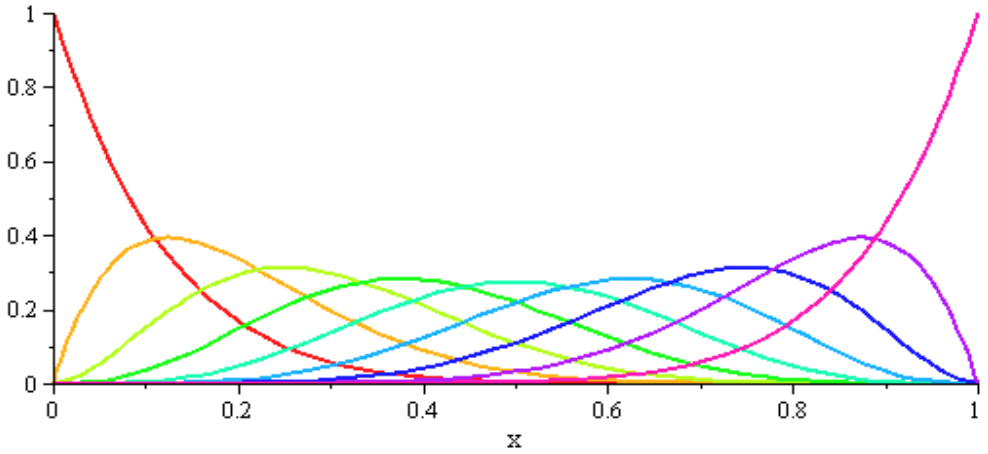
$$P^N(t) = \sum_{i=0}^{N} B_i^N(t) \cdot P_i, \text{ где} \qquad \boldsymbol{B_i^N(t) = C_i^N\, t^i\, (1-t)^{N-i}}$$

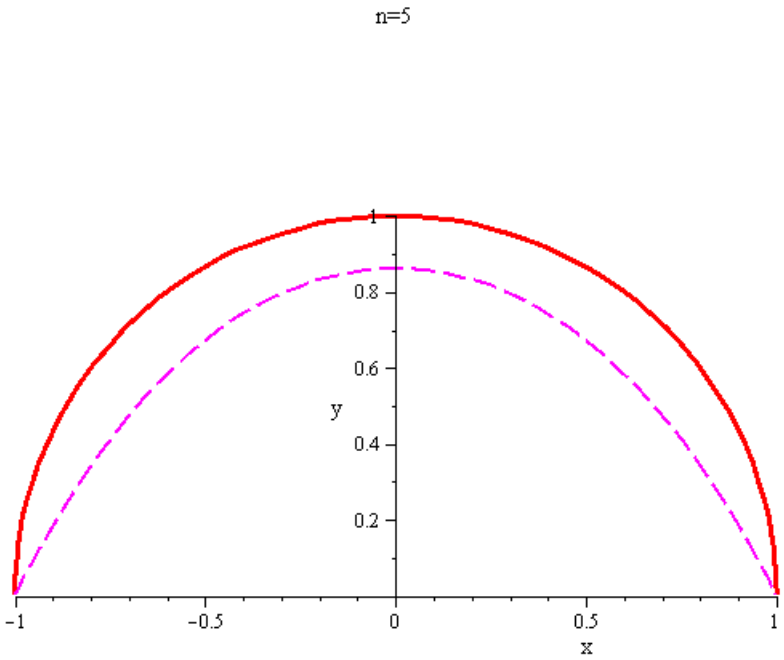
$$\boldsymbol{C_i^N = \frac{N!}{i!\,(N-i)!}}$$ - биномиальные коэффициенты

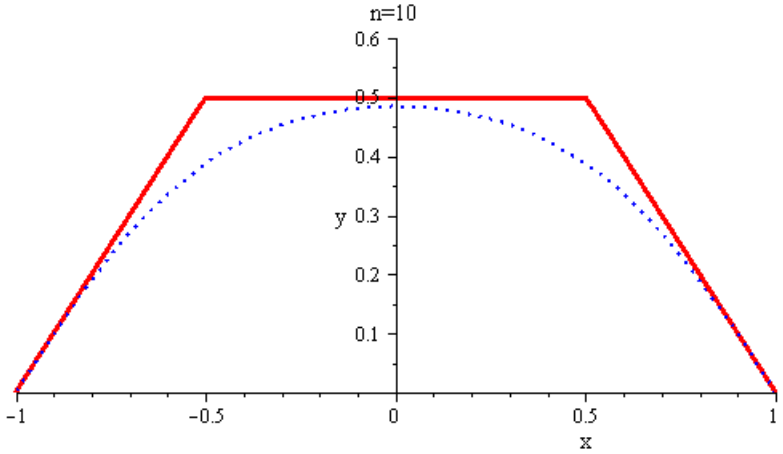
многочлены Бернштейна ***N*** степени, **образующие разложение единицы**

$$B_0^3(t) = (1-t)^3$$
$$B_1^3(t) = 3(1-t)^2 t$$
$$B_2^3(t) = 3(1-t)\ t^2$$
$$B_3^3(t) = t^3$$

$$\sum_{k=0}^{3} B_k^3(t) = 1$$







1. **вычисления проходят очень быстро**
2. **погрешность на концах интервала минимальна – т.е. интервалы хорошо “склеиваются”**
3. **происходит сглаживание, а не для аппроксимации исходных данных - поэтому в CG это не “данные”, а управляющие точки**
4. **для того, чтобы добиться более точной аппроксимации необходимо сильно увеличивать степень полиномов.**

Свойства кривых Безье

- Инвариантность относительно аффинных преобразований
- Инвариантность относительно линейных замен параметризации $t = \frac{u-a}{b-a}$
- Кривая Безье принадлежит выпуклой оболочке опорных точек (*следует из геометрического способа построения*)
- Кривая Безье проходит через P_0 и P_N
- Симметричность: если рассматривать контрольные точки в противоположном порядке, то кривая не измениться;
- Степень многочлена, представляющего кривую в аналитическом виде на 1 меньше числа опорных точек;
- Векторы касательных в точках P_0 и P_N коллинеарны P_0P_1 и $P_{N-1}P_N$ соответственно - “тангенциальные ручки”.

B-сплайны - (bell or beta or basic – все три разные)

Уравнение для beta-сплайнов:

$$P(t) = b_0(t)P_0 + b_1(t)P_1 + b_2(t)P_2 + b_3(t)P_3$$

$$b_0(t) = 2\beta_1^3(1-t)^3 / \delta$$

$$b_1(t) = (2\beta_1^3 t(t^2 - 3t + 3) + 2\beta_1(\beta_1 + 1)(t^3 - 3t + 2) + \beta_2(t^3 - 3t^2 + 1)) / \delta$$

$$b_2(t) = 2(\beta_1^2 t^2(-t + 3) + \beta_1 t(-t^2 + 3) + \beta_2 t^2(-2t + 3) + (-t^3 + 1)) / \delta$$

$$b_3(t) = 2t^3 / \delta$$

$$\delta = 2\beta_1^3 + 4\beta_1^2 + 4\beta_1 + \beta_2 + 2$$

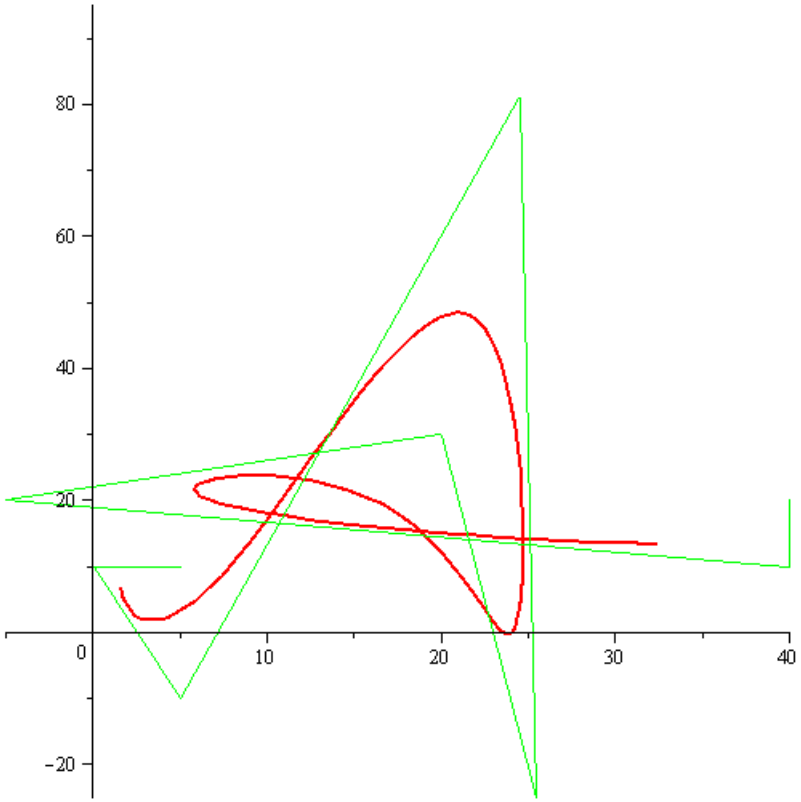
где $\beta_1 > 0$ – параметр скоса (смещения), $\beta_2 \geq 0$ – параметр натяжения.

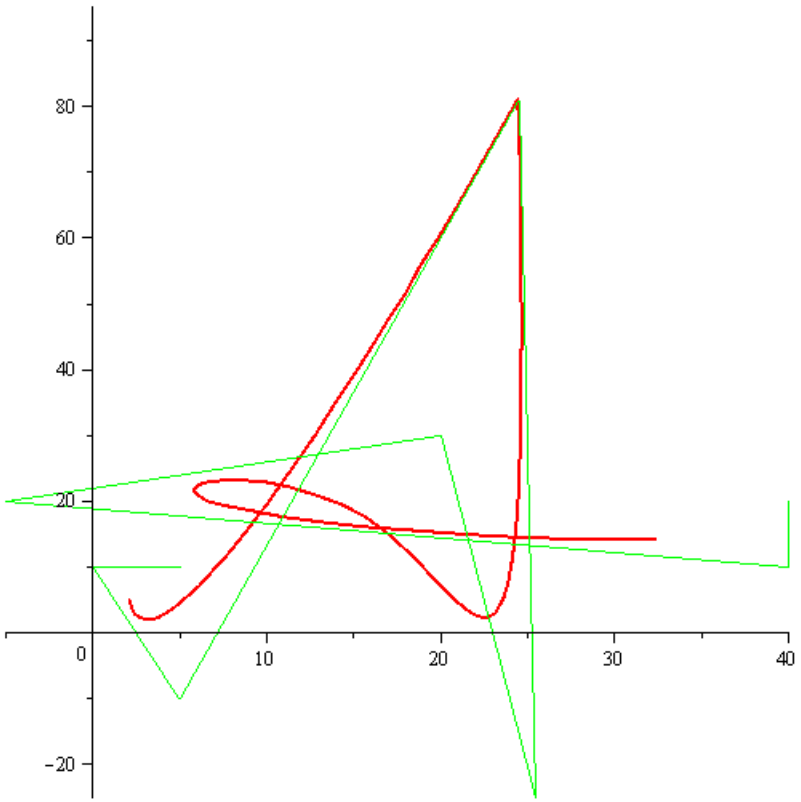
Свойства:

не проходит ни через одну из базовых точек, но начальная (и конечная) точки обязательно лежат в треугольнике образованном 3 начальными (конечными) базовыми точками ➔ базовый набор точек дополняется 2 начальными (конечными) точками

- проходит внутри выпуклой оболочки, заданной опорными точками
- дважды геометрически непрерывна

Зачем это надо – больше возможностей при малом повышении затрат





Простое изменение весов точек приводит к появлению “острых углов”

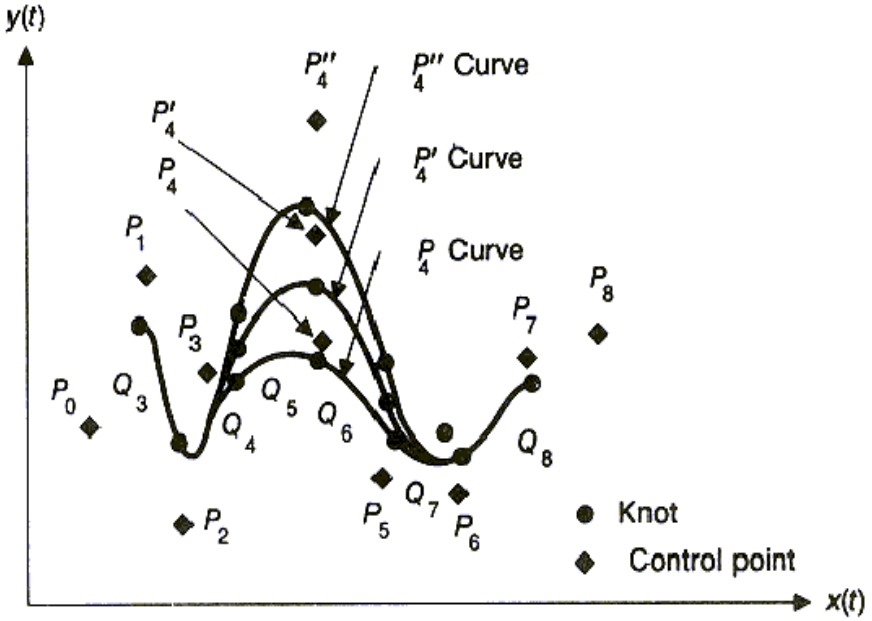
B-сплайны и NURBS

В общем случае **B-сплайн** состоит из нескольких сплайновых сегментов, каждый из которых определен как **набор управляющих точек.**

Коэффициенты многочлена будут зависеть только от управляющих точек на рассматриваемом сегменте кривой.

Этот эффект называется локальным управлением, поскольку перемещение управляющей точки будет влиять не на все сегменты кривой.

Рисунок показывает, как управляющая точка P_4 локально влияет на форму кривой.



Если координаты (x, y, z) точки кривой представлены в виде рациональной дроби,

$$x = \frac{X(t)}{W(t)},\ y = \frac{Y(t)}{W(t)},\ z = \frac{Z(t)}{W(t)}$$

то говорят, что B-сплайн рациональный, иначе – нерациональный.

Существуют 4 типа B-сплайнов:
- **равномерные нерациональные**;
- **неравномерные нерациональные**;
- **равномерные рациональные**;
- **неравномерные рациональные**.

Последний, наиболее общий тип B-сплайнов, и называют NURBS.

Неоднородные рациональные B-сплайны (Non-Uniform Rational B-Spline - NURBS)

Неоднородный означает, что различные области объектов NURBS (кривых или поверхностей) обладают различными свойствами (весами), значения которых не равны между собой.

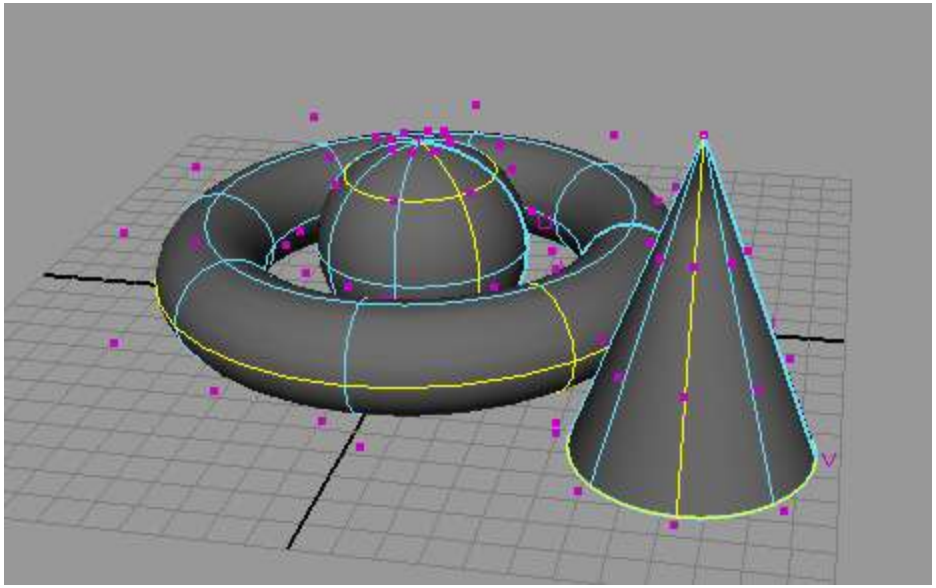
Большинство современных систем компьютерной анимации поддерживают моделирование с использованием NURBS.

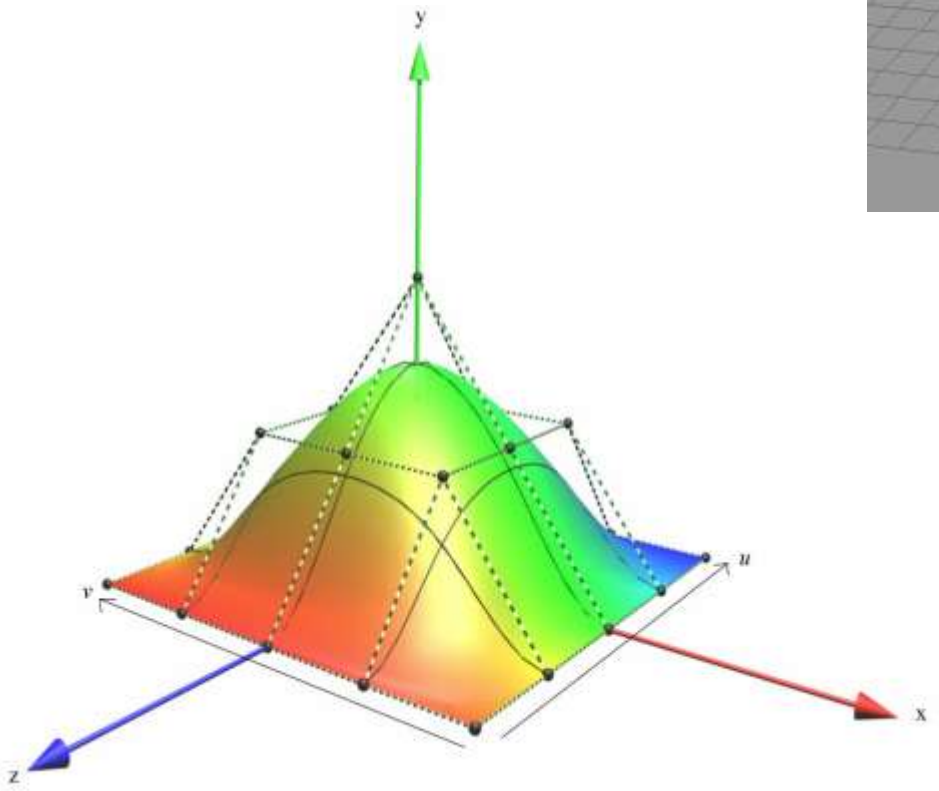
Моделирование на основе неоднородных рациональных B-сплайнов обладает следующими преимуществами перед другими методами:

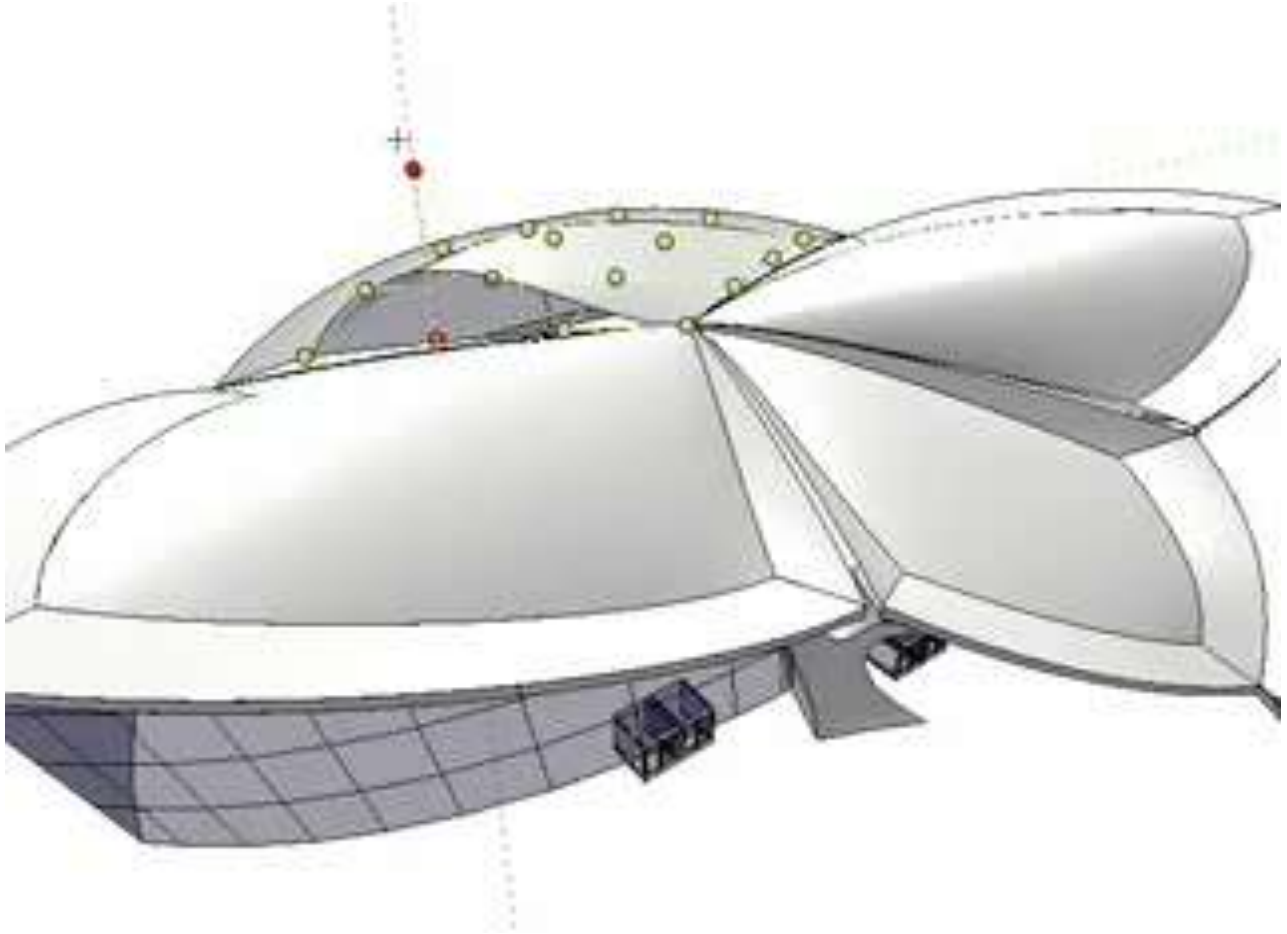
✓ с помощью NURBS проще имитировать поверхности природных объектов или объектов, поверхности которых имеют сложным образом искривленные профили

✓ NURBS-модели обеспечивают лучшее качество визуализации закругленных краев объектов благодаря разбиению на грани, выполняемому с использованием аналитических выражений

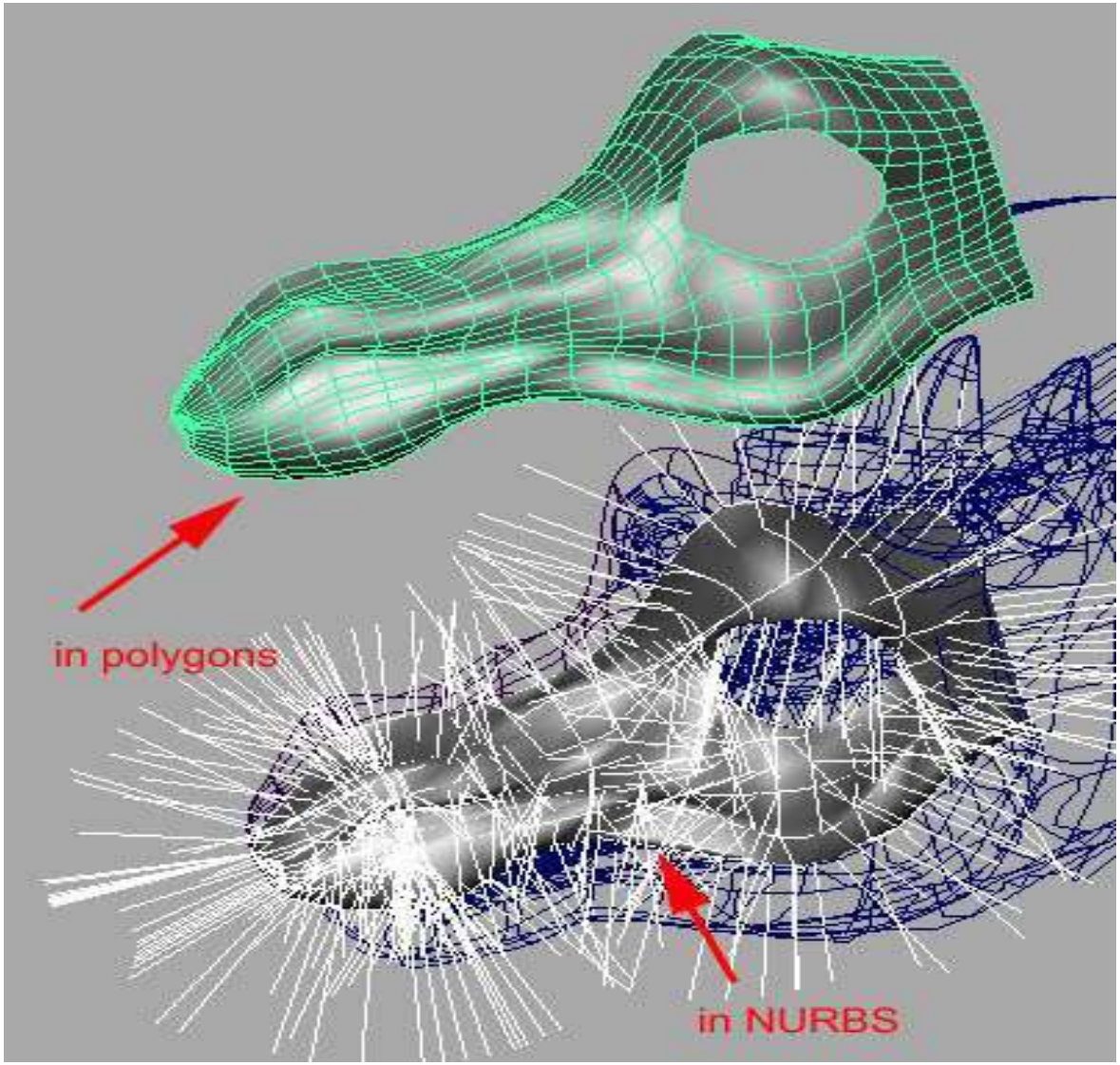
Пример поверхности и сетки







Полигоны и NURBS



B-сплайны и NURBS.

Математические выражения для кривой и поверхности

$$P(t)=\frac{\sum_{i=0}^{p} B_{i,n}(t)P_i w_i}{\sum_{i=0}^{p} B_{i,n}(t)w_i}$$ - кривая;

$$S(u,v)=\frac{\sum_{i=0}^{p}\sum_{j=0}^{q} B_{i,n}(u)B_{j,m}(v)P_{i,j} w_{i,j}}{\sum_{i=0}^{p}\sum_{j=0}^{q} B_{i,n}(u)B_{j,m}(v)w_{i,j}}$$ - поверхность,

базовая функция $B_{i,n}$ определена рекурсивно **формулами Кокса-де Бура**:

$$B_{i,n}(t):\quad B_{i,0}(t)=\begin{cases}1, & t_i \le t < t_{i+1}\\ 0, & \text{иначе}\end{cases}\qquad \forall k>0,\ B_{i,k}(t)=\frac{t-t_i}{t_{i+k}-t_i}B_{i,k-1}(t)+\frac{t_{i+k+1}-t}{t_{i+k+1}-t_{i+1}}B_{i+1,k-1}(t)$$

$w_{i,j}$ – вес, ассоциированный с управляющей точкой $P_{i,j}$,

$1 \le k \le n$, $n=p-1$ – степень полиномов,

p – порядок B-сплайна и число сегментов поддержки,

$p+1$ – число узлов поддержки, принято, что 0/0=0.

$$\sum_{i=0}^{p} B_{i,n}(t)=1$$ - разложение единицы при каждом **n**

B-сплайны. NURBS

Множество стыковочных функций

Пример

Стыковочная кусочная кривая g(t) второй степени 3 порядка - с поддержкой на интервале [0,3]:

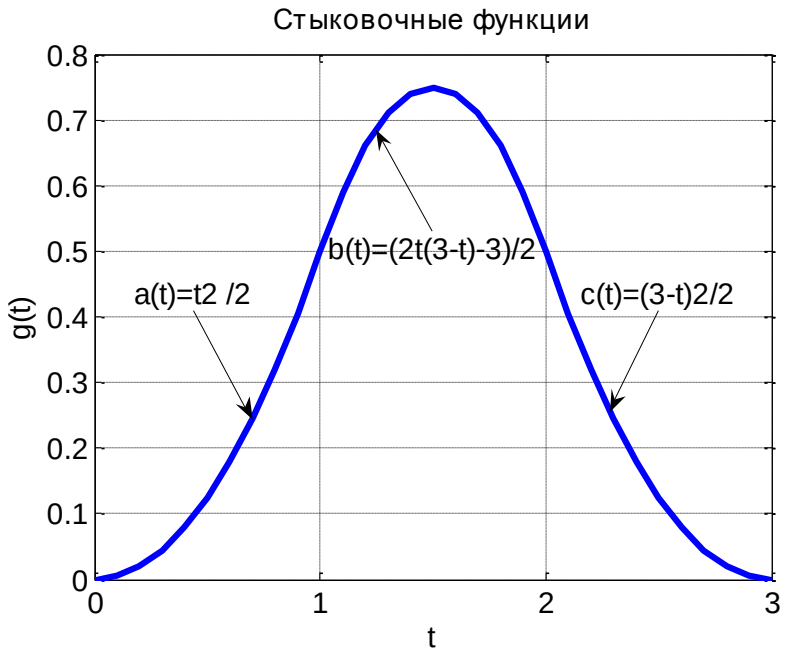
$a(t)=t^2/2, \quad 0<t<1;$

$b(t)=(2t(3-t)-3)/2, \quad 1<t<2; \quad (4)$

$c(t)=(3-t)^2/2, \quad 2<t<3$

Св-во: $a(t)+b(t)+c(t)=[t^2+2t(3-t)-3+(3-t)^2]/2=3$

➔ Нормировка →равномерный рациональный B-сплайн



Неоднородный рациональный B-сплайн (NURBS) 3 степени на открытом интервале [0,1]:

$$P(t)=\frac{w_0P_0(1-t)^3+3w_1P_1t(1-t)^2+3w_2P_2t^2(1-t)+w_3P_3t^3}{w_0(1-t)^3+3w_1t(1-t)^2+3w_2t^2(1-t)+w_3t^3}$$

B-сплайны. NURBS

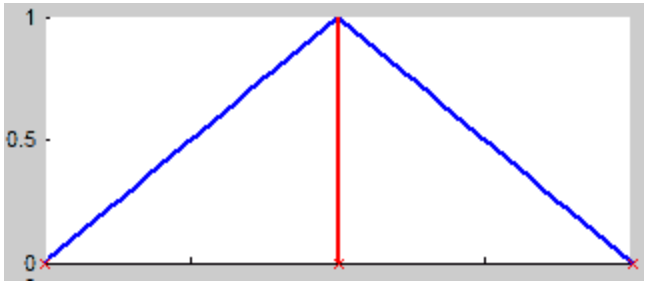
Множество стыковочных функций $B_{i,n}$. Рекурсия Кокса-де Бура

$B_{i,n}(t)$ -стыковочные функции, определяющие число и степень влияния узлов поддержки;
i – номер первого узла поддержки,
n – степень полиномов, *n*+2 – число узлов.

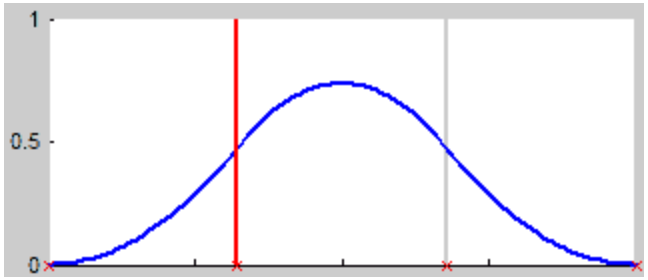
$B_{i,n}(t)$: $B_{i,0}(t) = \begin{cases} 1, & t_i \le t < t_{i+1} \\ 0, & \text{иначе} \end{cases}$ → локальность

$$\forall k > 0,\ B_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} B_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(t)$$

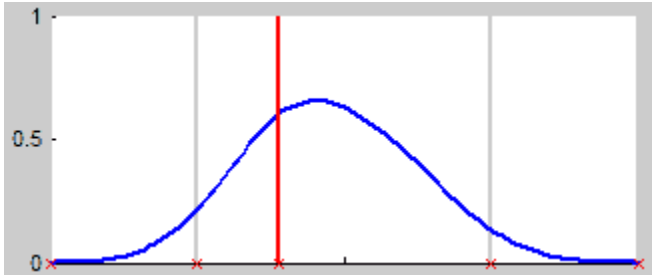
$1 \le k \le n$. **Свойство:** $\sum_{i=0}^{p} B_{i,n}(t) = 1$



степень=1, число узлов = 3, интервал [0,2]



степень=2, число узлов = 4 , интервал [0,3]



степень=3, число узлов = 5 , интервал [0,4]

NURBS имеет все преимущества Безье-сплайнов, а также следующие:

- порядок усложнения не высок
- инвариантность относительно проективных преобразований;
- возможность локального управления кривизной сплайна;
- наличие весов для управляющих точек: еще более гибкие.

B-сплайны и NURBS.

Сетки узлов

Возможные сетки узлов:

1) Равномерные:

[0 1 2 3 4] – ненормированная, [0 0.25 0.5 0.75 1] – нормированная.

Равномерные последовательности узлов порождают периодические равномерные функции базиса, для которых

$$B_{i,n}(t) = B_{i-1,n}(t-1) = B_{i+1,n}(t+1)$$

2) Открытые равномерные:

У открытого равномерного вектора узлов, число одинаковых узлов на концах равно порядку B-сплайна:

p=2	[0 0 1 2 3 4 4],	[0 0 1/4 1/2 3/4 1 1]
p=3	[0 0 0 1 2 3 3 3],	[0 0 0 1/3 2/3 1 1 1]
p=4	[0 0 0 0 1 2 2 2 2],	[0 0 0 0 1/2 1 1 1 1]

Если число вершин многоугольника равно порядку базиса, то базисные функции B-сплайна сводятся к базису Бернштейна, а B-сплайн к сплайну Безье.

3) Неравномерные ……

Источники

1. Порев В.Н. Компьютерная графика. –СПб.: БХВ-Петербург, 2002. –432с.
2. Хилл Ф. OpenGL. Программирование компьютерной графики. – С.Пб: Питер, 2002. 1088с.
3. Роджерс Д., Адамс Дж. Математические основы машинной графики: Пер. с англ. - М.: Мир, 2001. - 604с.
4. Шикин Е.В., Боресков А.В. Компьютерная графика. Полигональные модели. –М.: ДИАЛОГ-МИФИ, 2001.-464с.
5. Майкл Ласло. Вычислительная геометрия и компьютерная графика на С++: Пер. с англ. –М.: «БИНОМ», 1997. –304с.
6. http://graphics.sc.msu.su/courses/ - курсы "Введение в компьютерную графику" Баяковского Ю.М. и Шикина Е.В. для ф-та ВМиК МГУ, а также материалы к курсам и материалы лаборатории Graphics & Media Lab при МГУ
7. The home page of the Graphics Research Group at Harvard University. Computer Graphics courses http://www.eecs.harvard.edu/graphics/#classes
8. Stanford Computer Graphics Laboratory. Courses in Graphics http://graphics.stanford.edu/courses/
9. Фень Юань. Программирование графики для Windows. -М.: 2004.
10. Юджин Олафсен, Кенн Скрайбнер, К.Дэвид Уайт.MFC и Visual C++ 6.0. Энциклопедия программиста. – М.: .2003.
11. Хилл Ф. OpenGL. Программирование компьютерной графики. – С.Пб: Питер, 2002. 1088с.
12. Поляков А.Ю., Брусенцев В.А. Программирование графики: GDI+ и DirectX. – СПб.: БХВ-Петербург, 2005. -368с.
13. А.Игнатенко. Геометрическое моделирование сплошных тел: On-line журнал "Графика и мультимедиа", 2003, http://cgm.graphicon.ru
14. Splines //en.wikipedia.org
15. An Interactive Introduction to Splines: Bezier, B-spline, NURBS, and many other spline curves and surfaces with interactive 2D Java applets and VRML. www.ibiblio.org/e-notes/Splines/Intro.htm