Инженерная и компьютерная графика
6 семестр (диф.зачет)

Лектор:

Таранцев Игорь Геннадьевич

Доцент ФИТ НГУ, ИАиЭ, «СофтЛаб-НСК»

Создатели курса:

Дебелов Виктор Алексеевич

Валеев Тагир Фаридович

Козлов Дмитрий Сергеевич

Лекция №1

Введение в курс

Минимальные знания, необходимые для решения первой задачи

Сайт курса

http://nsucgcourse.github.io/

Презентации лекций
Материалы к лекциям
Конспект лекций курса
Задачи к практическим занятиям
Результаты практических занятий
Объявления

E-mail курса

nsucgcourse@gmail.com

Обязательно тема письма
FIT 15201 Ivanov Тема – номер группы будет автоматически обработан (он должен быть отделён пробелами)
Культура написания писем, вы не школьники – пишите Здравствуйте, Имя / Имя Отчество… С уважением,…

Цели и задачи курса

Материал курса ориентирован на практическое усвоение: студент должен уметь программировать графические приложения:

- Растеризация и заливка кривых и многоугольников
- Фильтрацию графических изображений
- Визуализацию научных данных
- Построение трёхмерные каркасных и полноценных изображений с учётом локальной освещённости

+ теория

Требования к уровню освоения содержания курса

Студенты должны понимать основные алгоритмы, применяемые в современных графических редакторах и графических программных средствах, их возможности и ограничения.

Студенты должны уметь практически применить полученные знания при необходимости разработки соответствующих графических функций в прикладных программах.

В течение семестра выполняются практические работы (5 заданий) в виде графических приложений

Выполнение указанных видов работ является обязательным для всех студентов, а результаты текущего контроля служат основанием для выставления оценок в ведомость контрольной недели на факультете

Вы не поймёте материал в должной мере до тех пор
пока сами не реализуете излагаемые алгоритмы

Что влияет на оценку

Оценка задачи определяется семинаристом

Оценки по отдельным заданиям

Посещение лекций

По задачам:

- Выполнение требований работы
- Срок сдачи заданий
- Требования семинариста по задачам

Объективность оценки

Задачи оцениваются по пятибалльной шкала

Объективна только 5-ка – студент выполнил всё по плану

Самый большой диапазон у 3-ки

Не надо требовать сравнения себя с другим, если ваша дельта не выводит вас из границ 3-ки.

Нет задачи в указанный срок оценка 0.

Плагиат

Признаками являются:

- Избыточная похожесть программ 2-х студентов (в том числе со студентами прошлых лет)
- Студент не может объяснить как работает его собственная программа или не может поменять её

Плагиаторами объявляются оба

Максимальная оценка 3 за семестр + дополнительные задачи

Посещаемость

- На каждой лекции проводится перекличка.
- О студенте, имеющем 3 пропуска (без уважительной причины) или более (необязательно подряд), будет доложено деканату, как о невыполняющем учебный план.
- За 5 пропусков без письменного допуска из деканата возможен отказ от дальнейшей работы со студентом с выставлением оценки «неудовлетворительно».
- Выявление ложной отметки на лекции приравнивается к 2 пропускам
- Без опозданий

Расписание

Пятница: 12:40-14:15, ауд. 3107
Лекция, присутствие обязательно.

Семинары – обязательность присутствия определяется семинаристом, на семинарах выявляются ошибки других студентов,
можно и нужно показывать предварительное решение

Программа

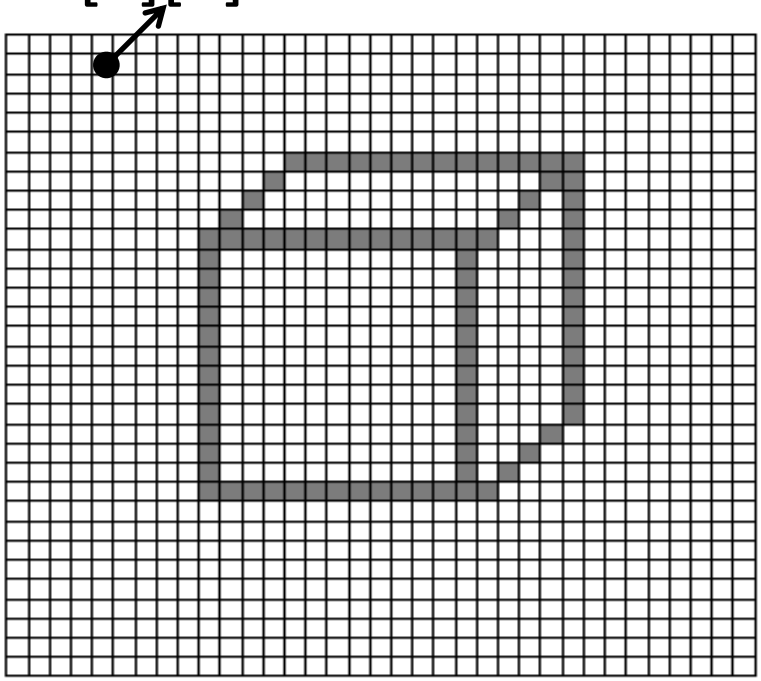
- Пиксельные области
- Растеризация алгебраических кривых
- Алгоритмы клиппирования многоугольников
- Визуализация в научных вычислениях
- Аппроксимация полутонов
- Фильтрация изображений
- Элементы вычислительной геометрии
- Преобразования координат
- Конструирование кривых с локальной модификацией
- Конструирование параметрических поверхностей
- Полигональные сетки
- Фотореалистичные изображения
- Локальная модель освещённости
- Пространственные структуры данных
- Визуализация теней

Растровые изображения

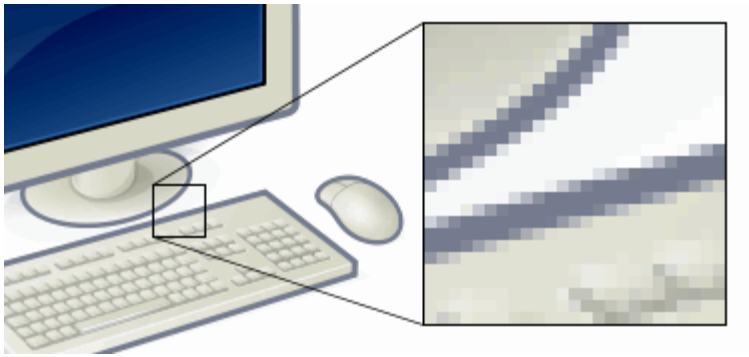
Изображение – это растр = 2D массив пикселей

int FB[n][m] – frame buffer

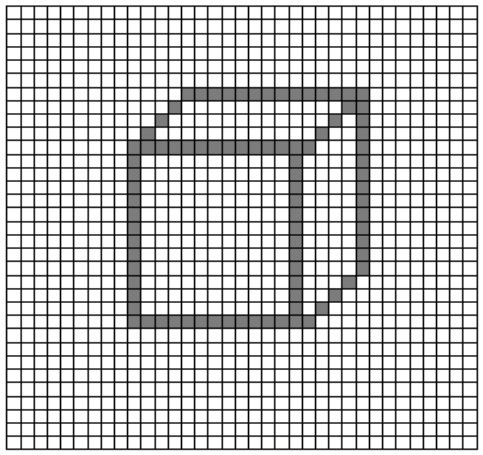
FB[5][2]



Пиксель (pixel, picture element) мельчайший элемент изображения

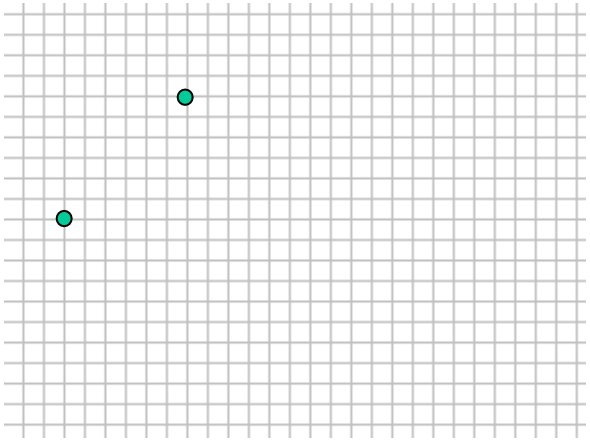


Растровые изображения

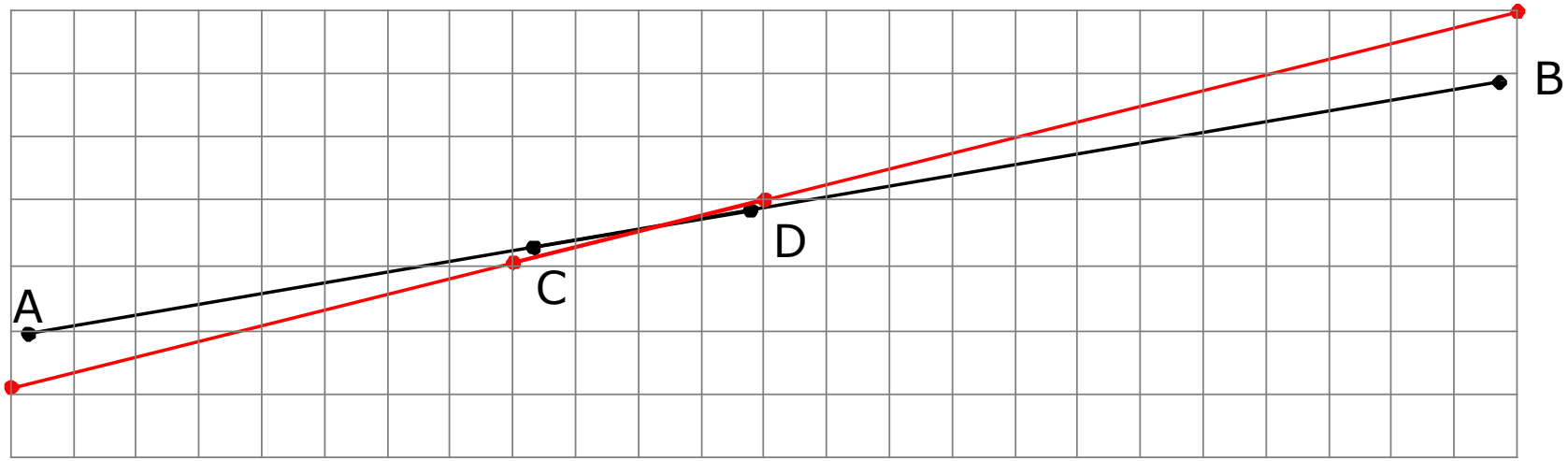


Квадратные пиксели

Узлы сетки – это центры пикселей

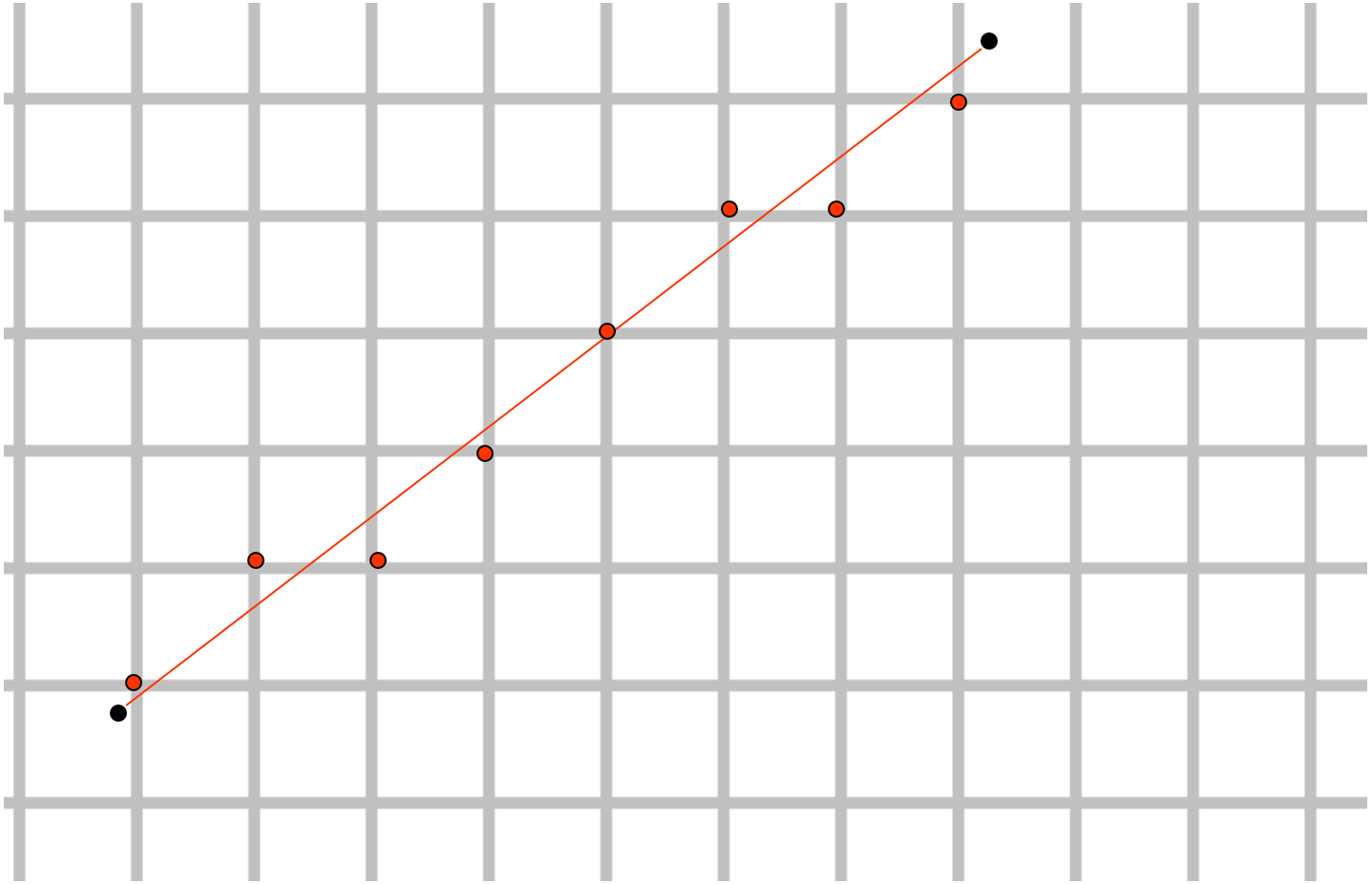


Осторожно – машинная точность

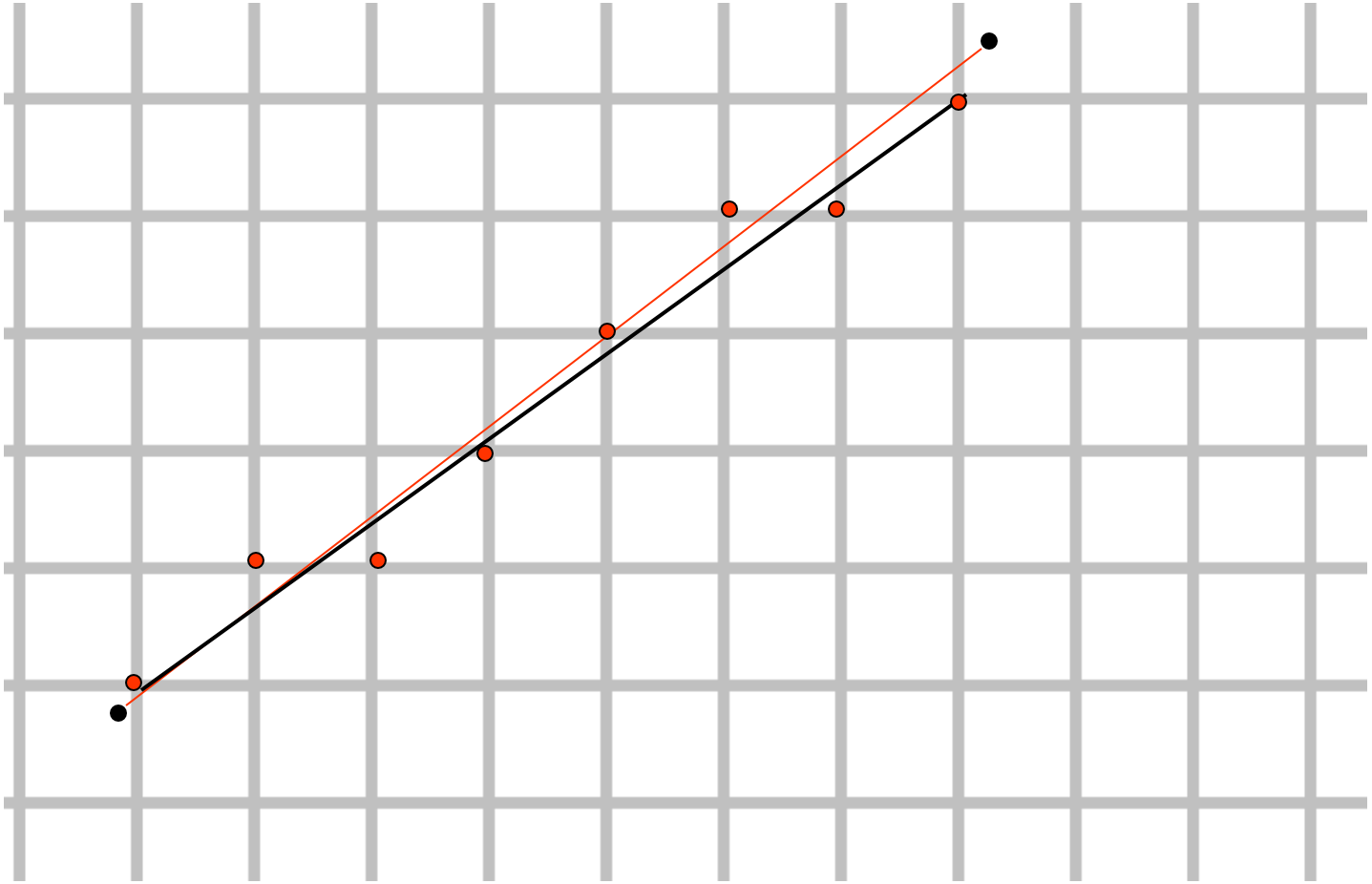


Сетка – это точность представления вещественных чисел. Мы строим прямую **L(A,B)**, проходящую через точки **A** и **B**. На самом деле мы вместо них неявно используем близкие к ним (с машинной точностью) точки. Выбираем еще две точки **C** и **D** **є** **L(A,B)**. Строим прямую **L(C,D)**. Как видим **AB** не принадлежат **L(C,D)**. А теперь представим, что мы работаем на более грубой сетке – на растре экрана или принтера.

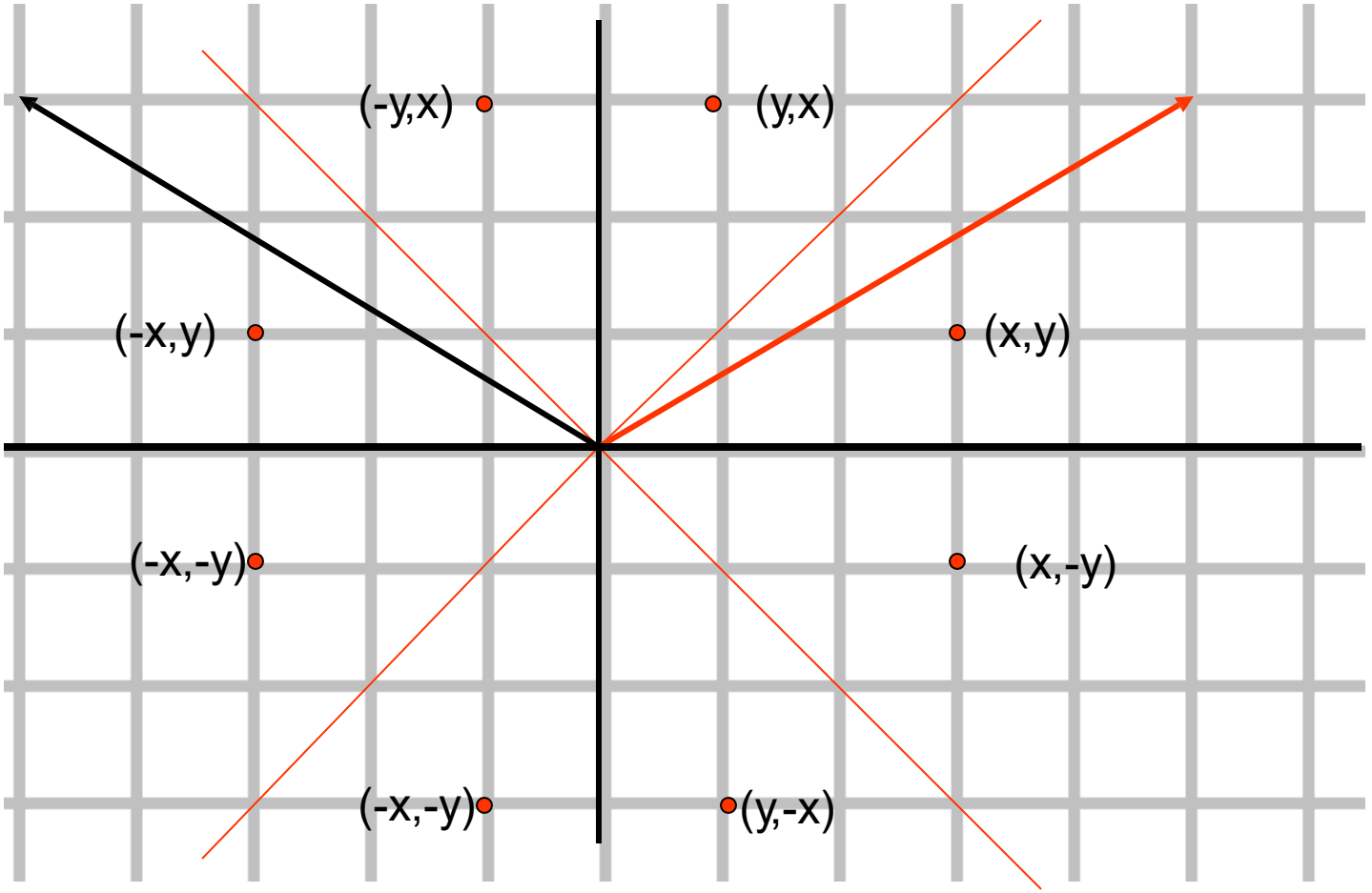
Растеризация отрезков



Растеризация отрезков

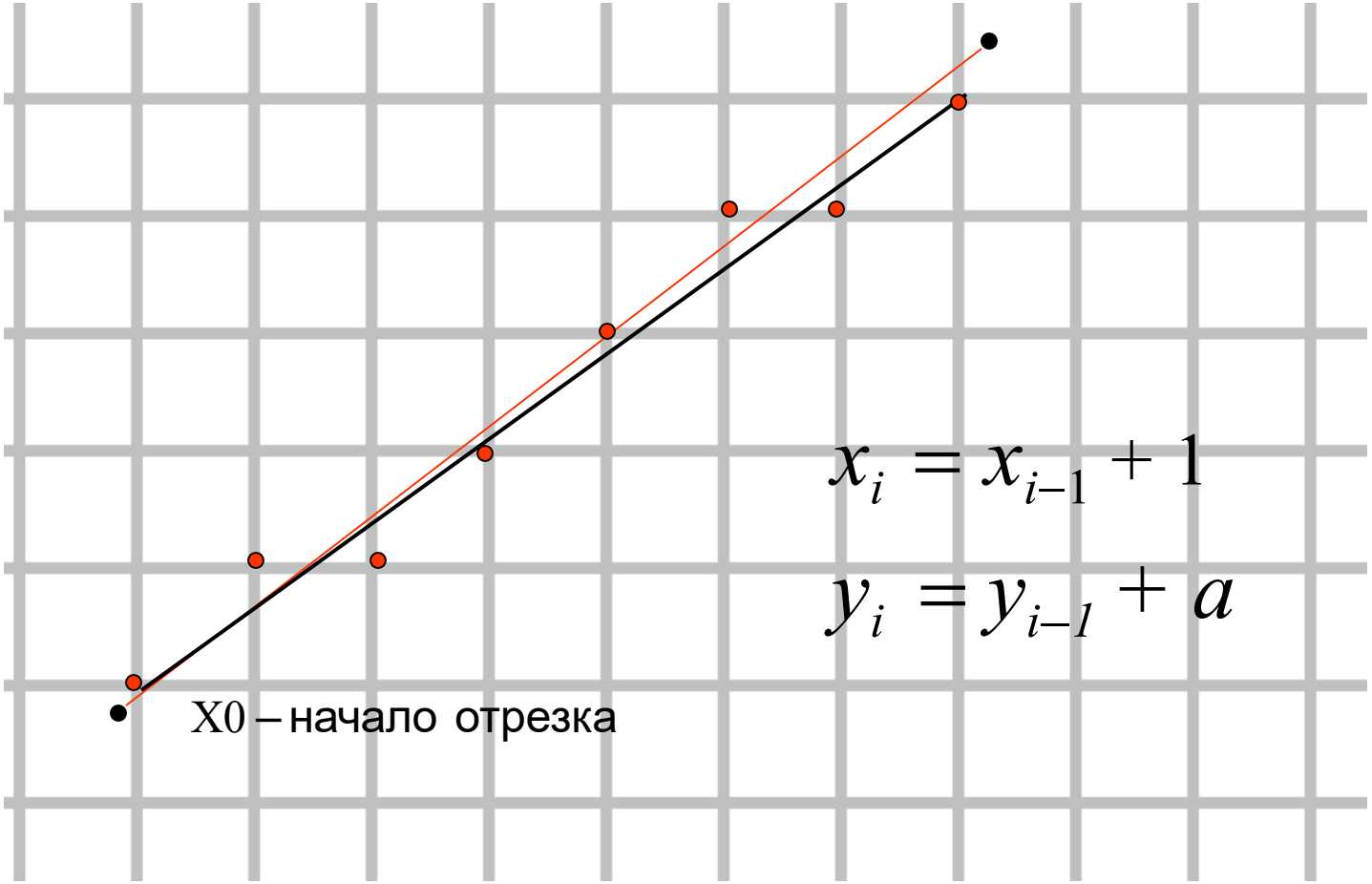


8-симметрия



$$y = ax + b, \quad b = 0, \quad 0 \le a \le 1 \quad => \quad y = ax$$

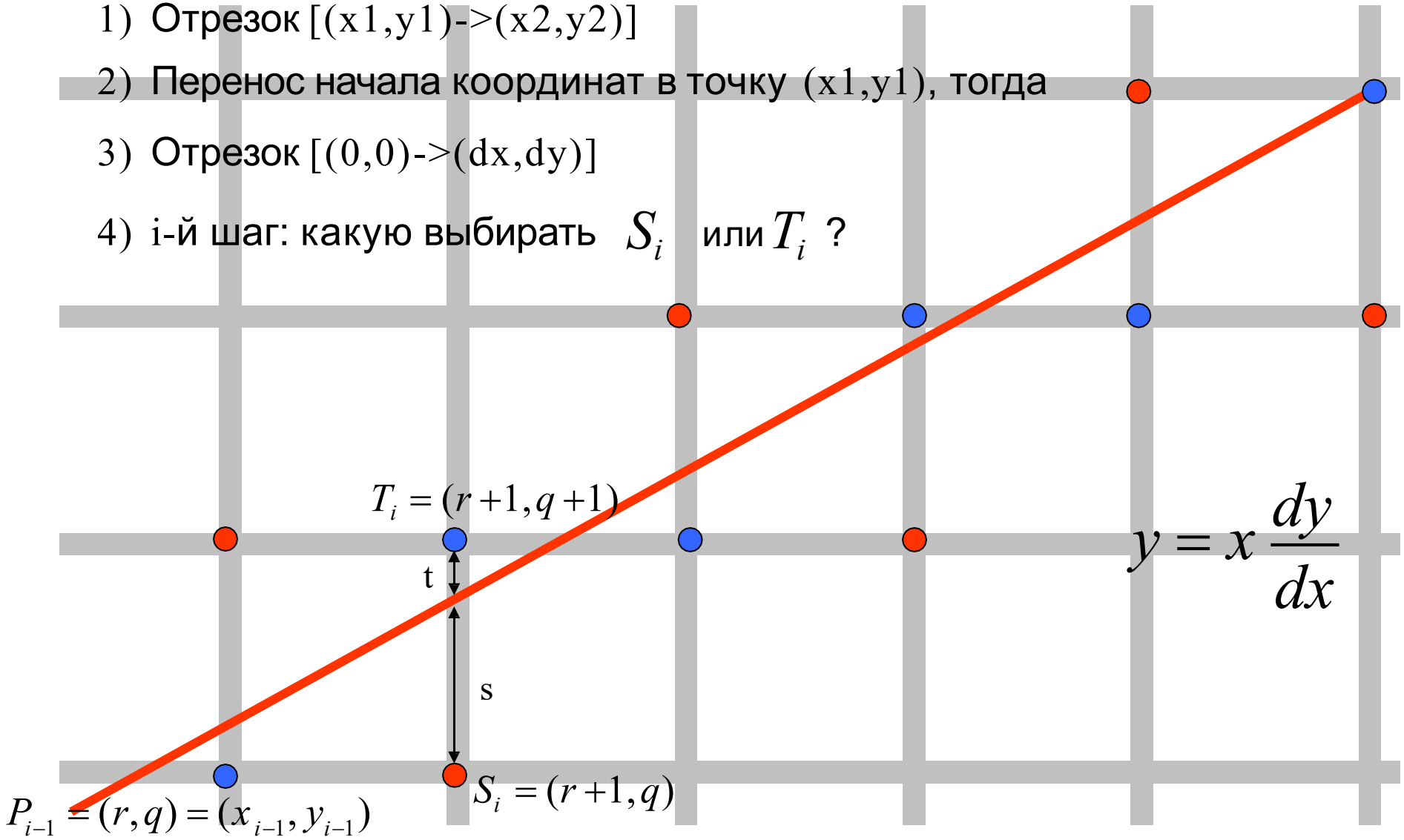
Растеризация отрезков



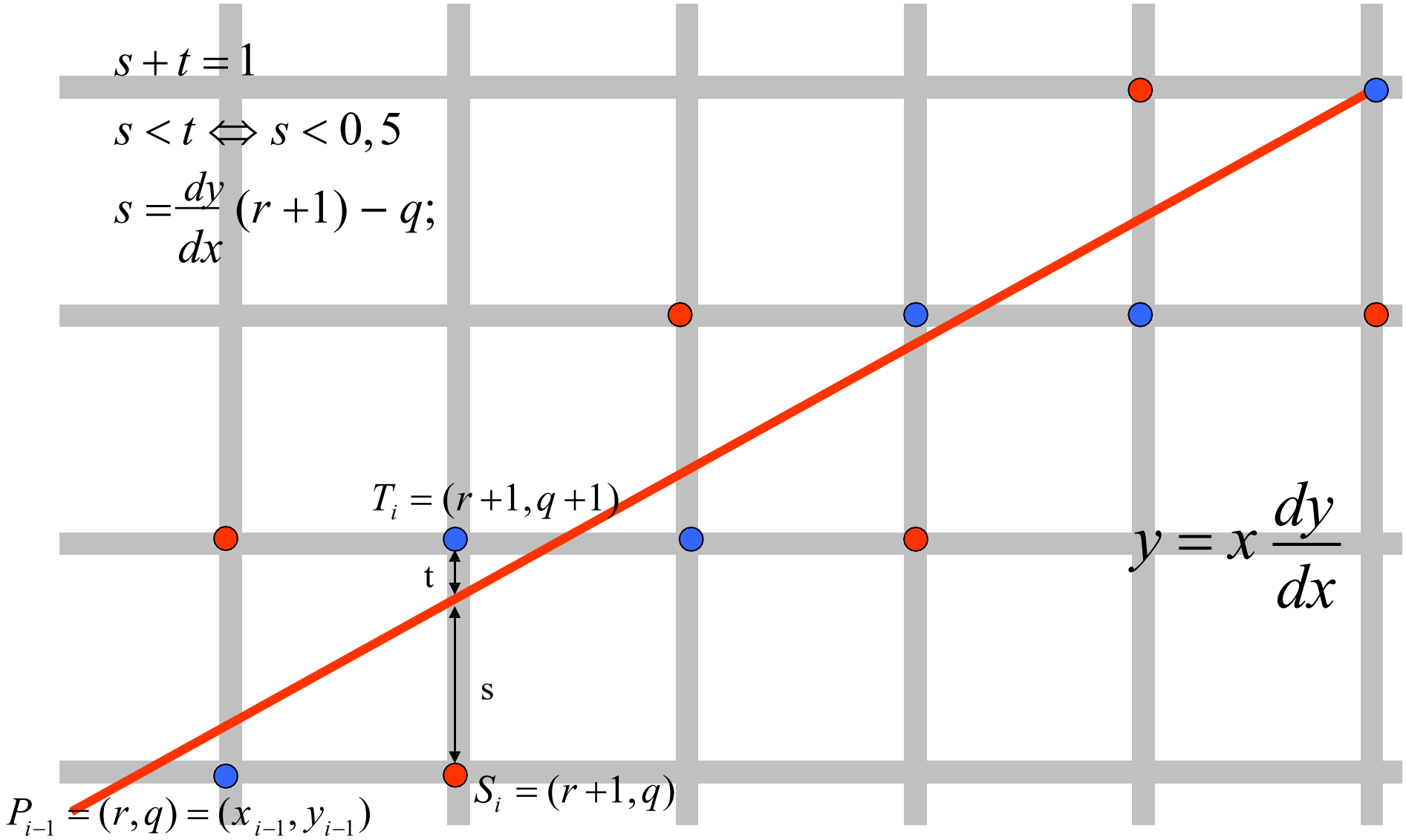
Делаем пересчет в растровую систему координат $\Delta x = 1$

Алгоритм Брезенхэма

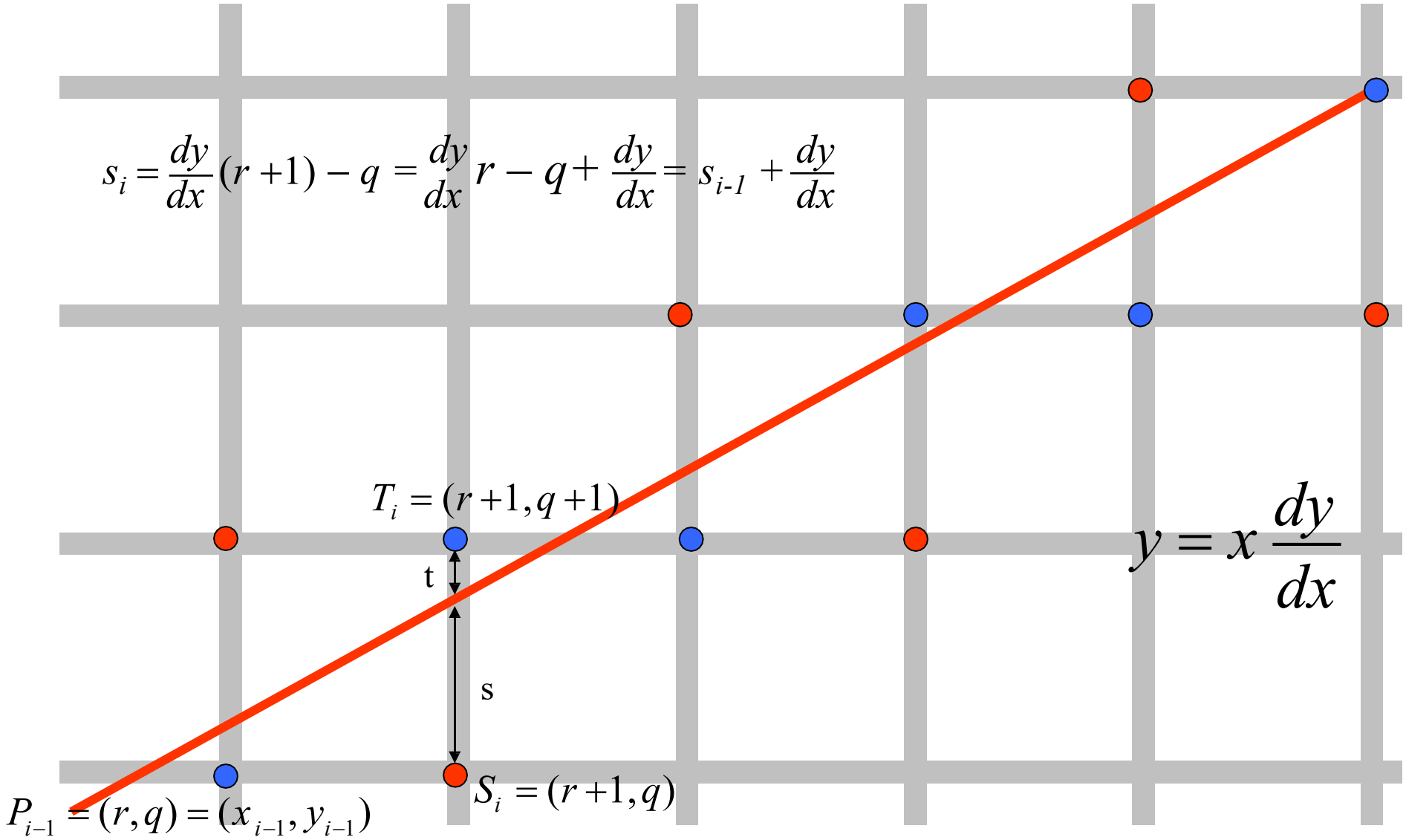
1) Отрезок [(x1,y1)->(x2,y2)]
2) Перенос начала координат в точку (x1,y1), тогда
3) Отрезок [(0,0)->(dx,dy)]
4) i-й шаг: какую выбирать S_i или T_i ?



Алгоритм Брезенхэма



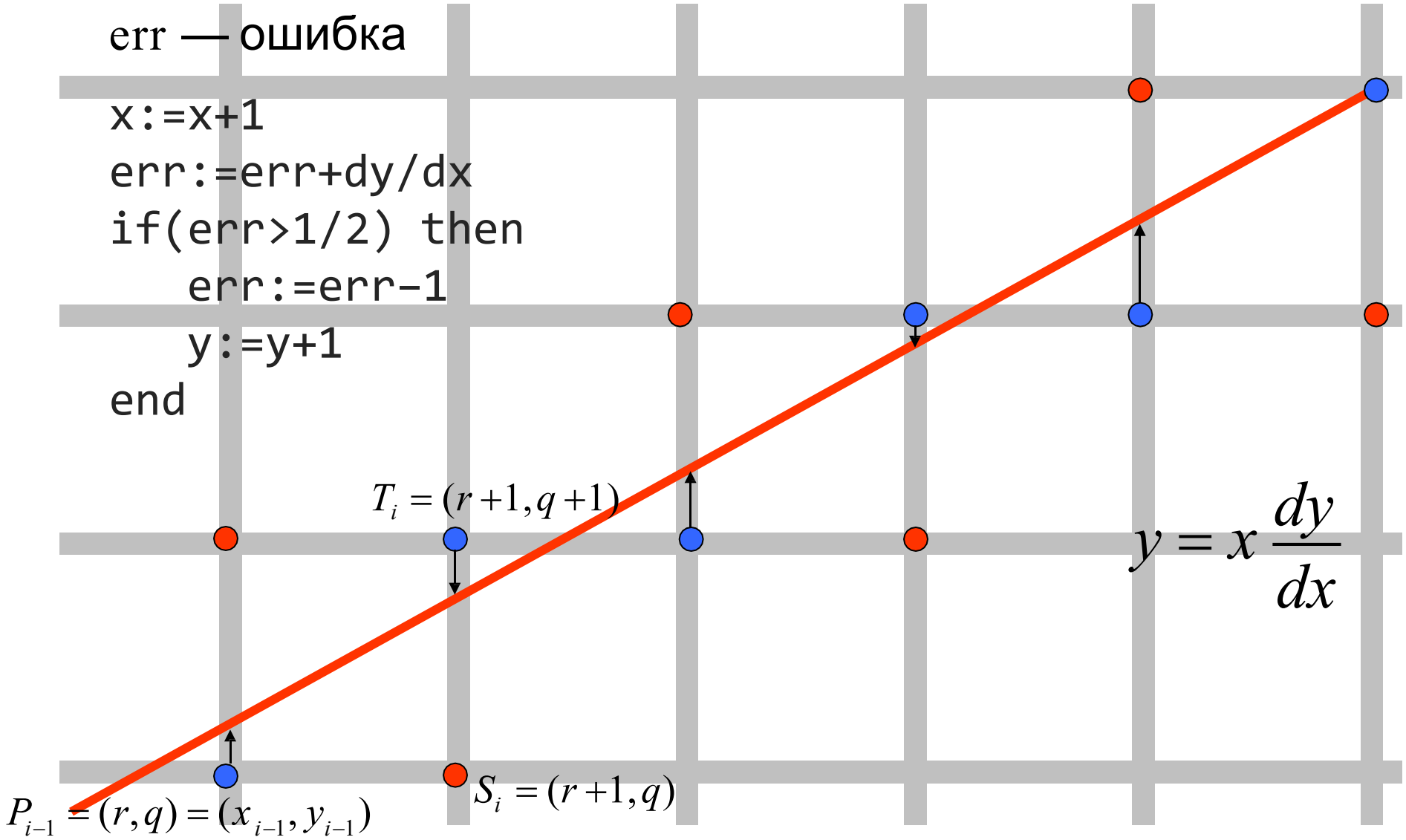
Алгоритм Брезенхэма



Алгоритм Брезенхэма

err — ошибка

```
x:=x+1
err:=err+dy/dx
if(err>1/2) then
    err:=err-1
    y:=y+1
end
```

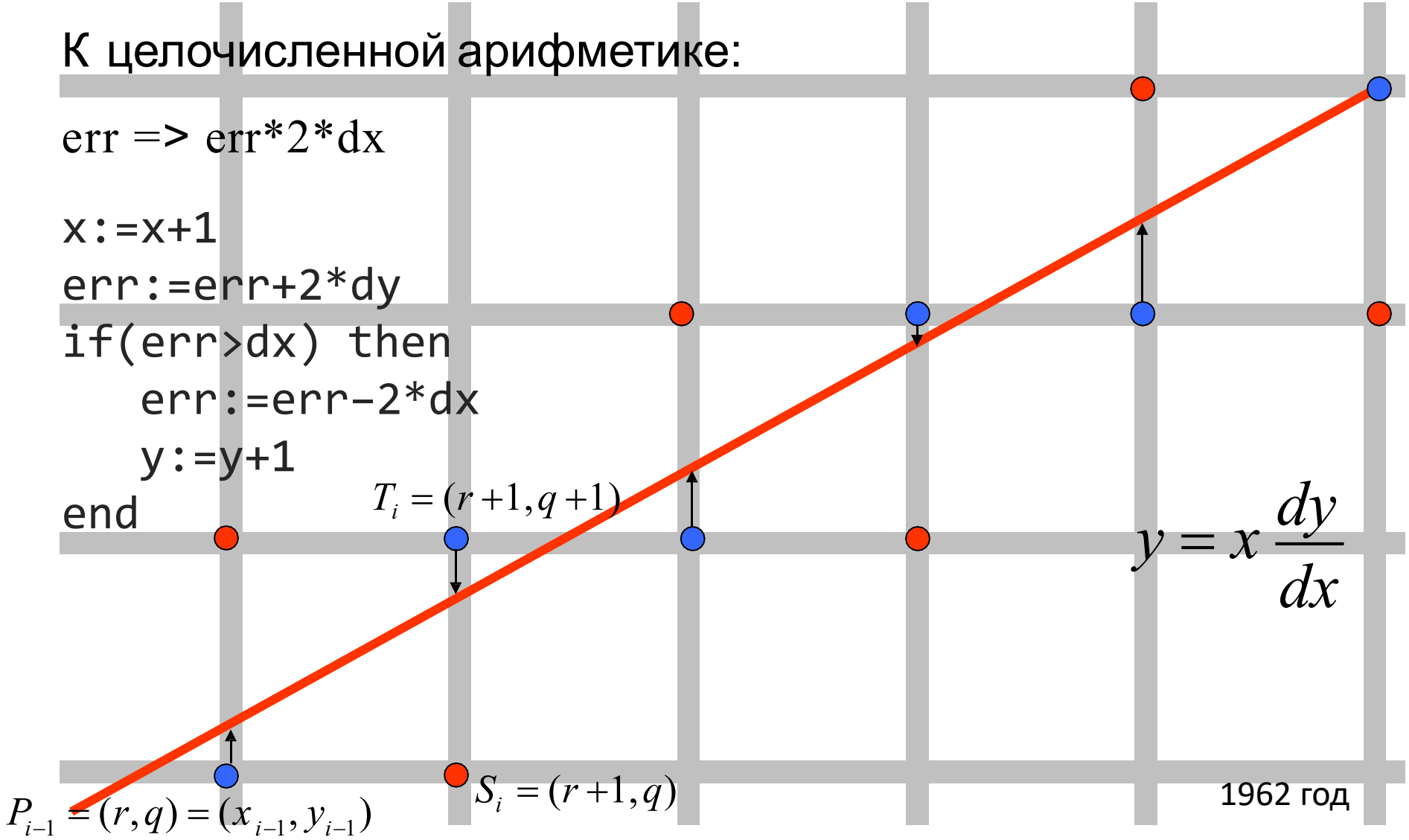


Алгоритм Брезенхэма

К целочисленной арифметике:

err => err*2*dx

```
x:=x+1
err:=err+2*dy
if(err>dx) then
    err:=err-2*dx
    y:=y+1
end
```



1962 год

Алгоритм Брезенхэма

err => err – dx

err в начале равен –dx

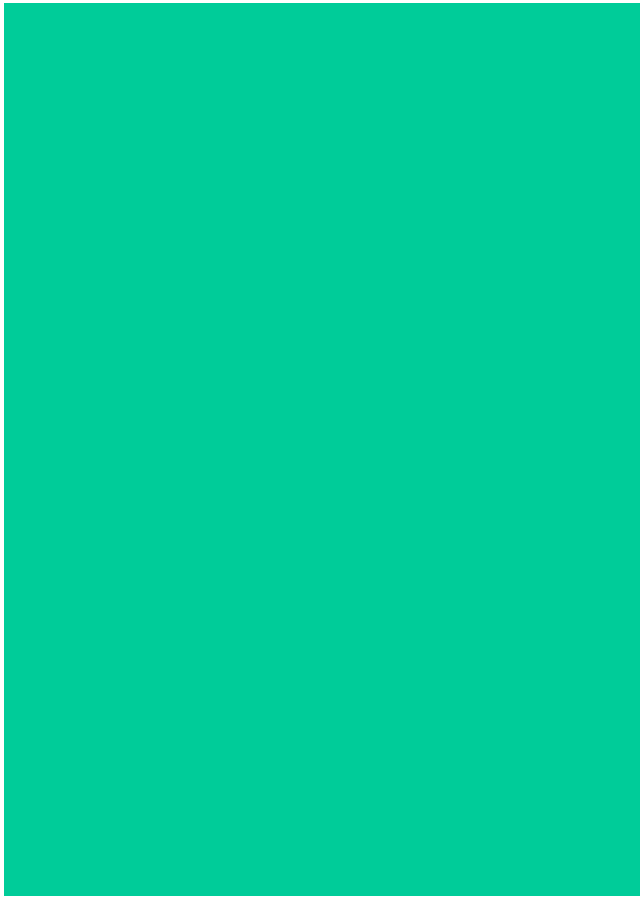
```
x:=x+1
err:=err+2*dy
if(err>0) then
   err:=err-2*d
   x  y:=y+1
end
```

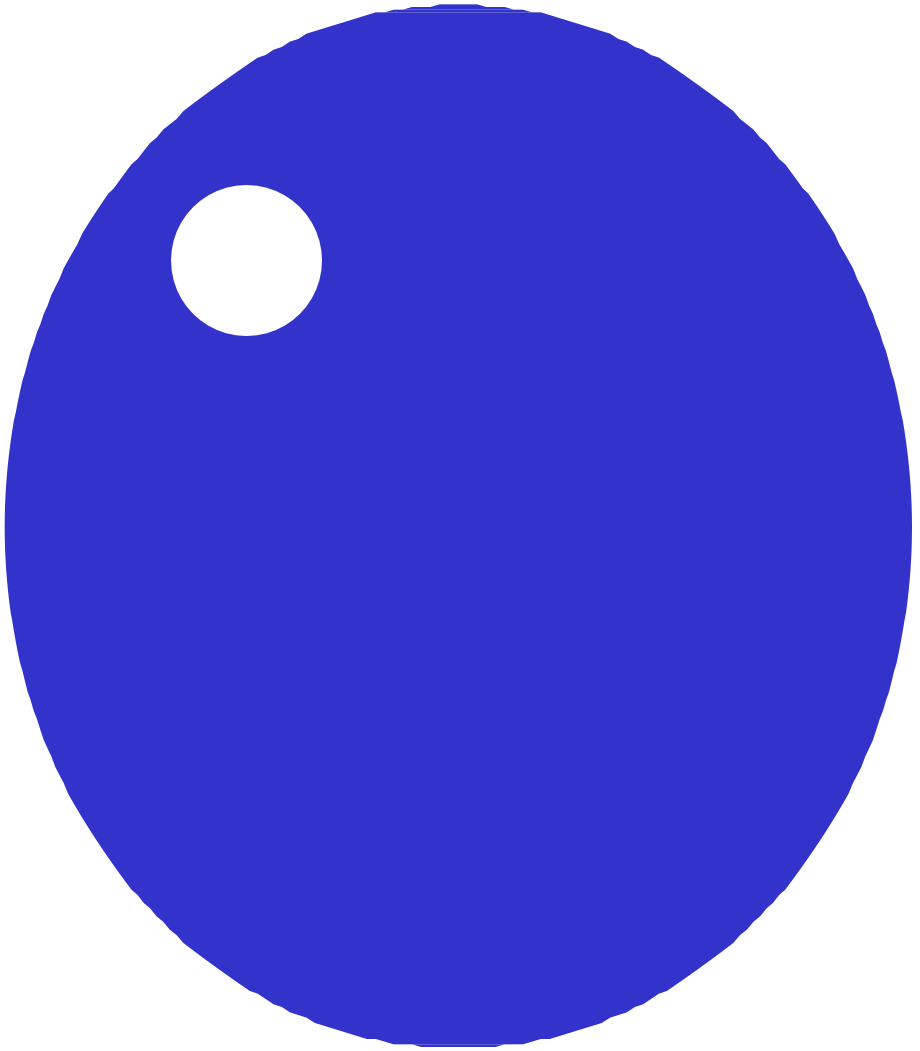
C++

```
int x = x0;
int y = y0;
int err = -dx;
for( int i=0; i<dx; ++i) {
   x++;
   err += 2*dy;
   if (err > 0) {
      err -= 2*dx;
      y++;
   }
   SetPixel(x,y,Color);
}
```

Пиксельные области (ПО)

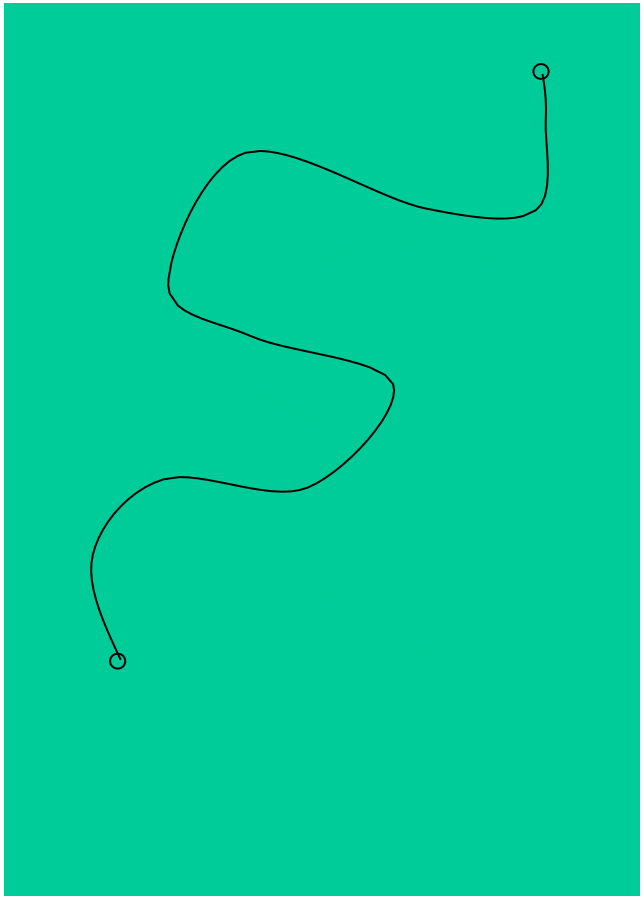
Область – это …

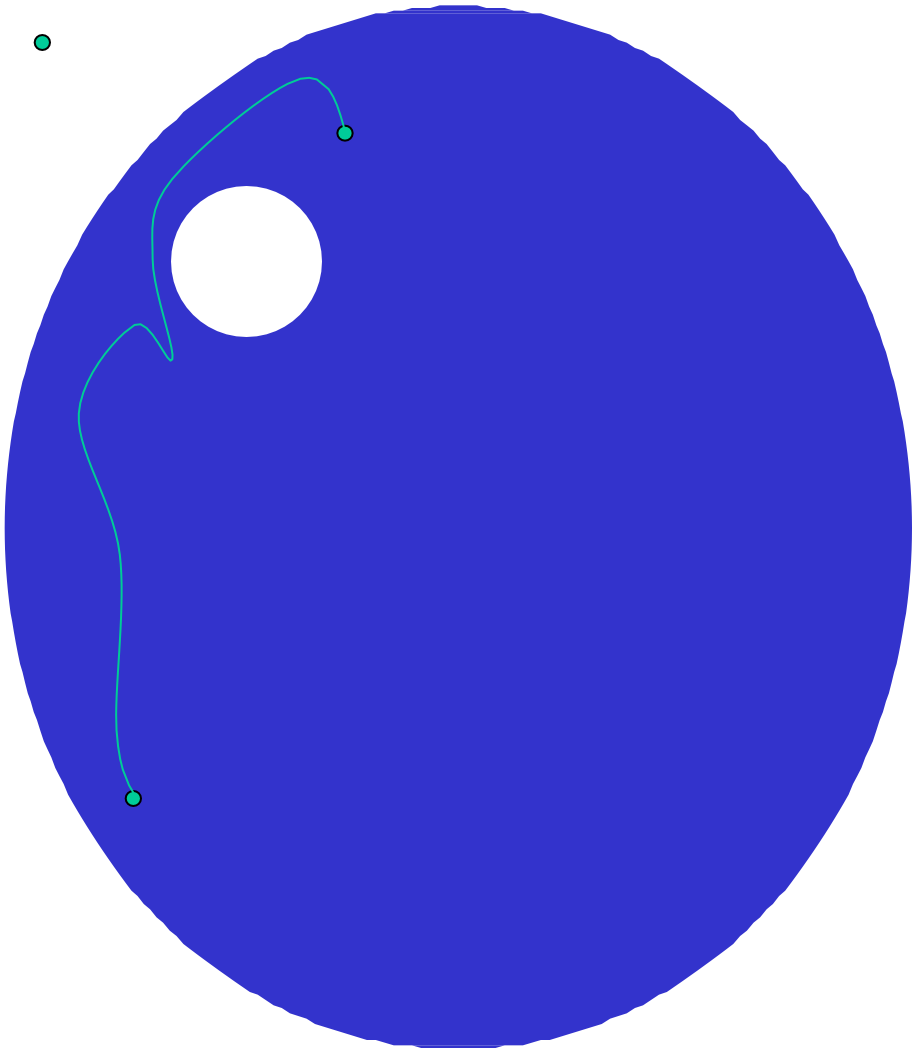




Пиксельные области (ПО)

Область – это …

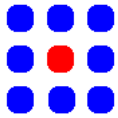




Пиксельные области (ПО)

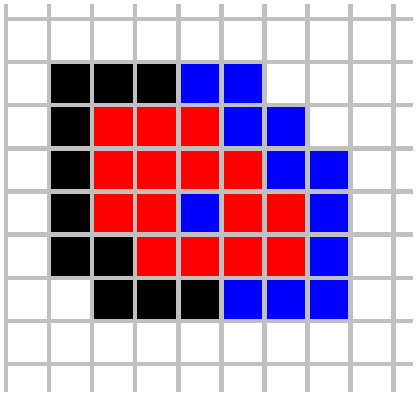
Каждый пиксель определяет значение – цвет RGB.
Как правило, слово "область" подразумевает связность.
Понятие границы (или, что то же самое – связности) пиксельной области неоднозначно:

4-связность – путь от любого пикселя до любого другого пикселя может состоять из 4 элементарных движений



8-связность – путь от любого пикселя до любого другого пикселя может состоять из 8 элементарных движений

Определение области

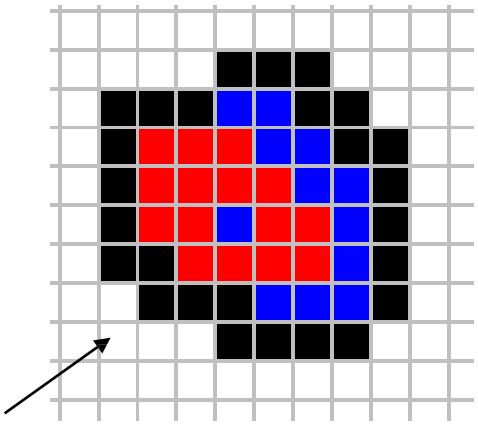


Внутренне определенные ПО. Все внутренние пиксели области имеют одно и то же значение old_value. Граница области может быть многосвязной (напр., дырки). На таких областях работают *внутренне заполняющие* алгоритмы:

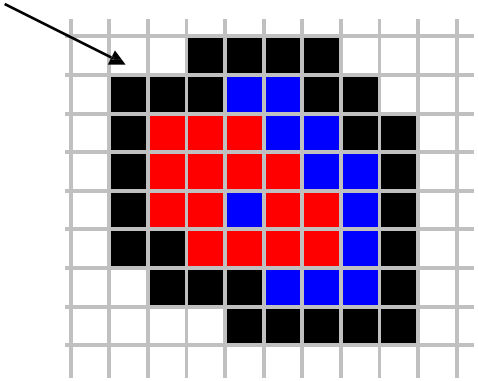
```
if (pixel == old_value)
    pixel = new_value
```

Может быть и более сложный предикат принадлежности к пиксельной области

Определение области



Связность?



Гранично определенные ПО. Пикселы границы имеют значения boundary_value. *Гранично заполняющие* алгоритмы каждому пикселю области присваивают одно и то же значение new_value. Заметим, что многие реализации требуют, чтобы ни один пиксель границы области не имел значения new_value.

Для 8-связных областей необходимо задавать 4-связную границу.

Затравка Seed

- Важная особенность алгоритмов заполнения ПО – им надо задать какой-либо внутренний пиксел области. В этом, например, заключается определенная трудность заполнения областей с криволинейными (параметрические, алгебраические кривые) границами. Хотя с применением интерактивного режима эти задачи решаются очень просто. Сначала рисуем (растеризуем) кривую нужной связности. Затем указываем мышью на любой внутренний пиксель для заливки области.
- И в заключение отметим, что ПО очень важны, поскольку многие редакторы рисунков могут решать различные задачи для отдельных областей, а не для рисунка целиком.

Заполнение шаблоном

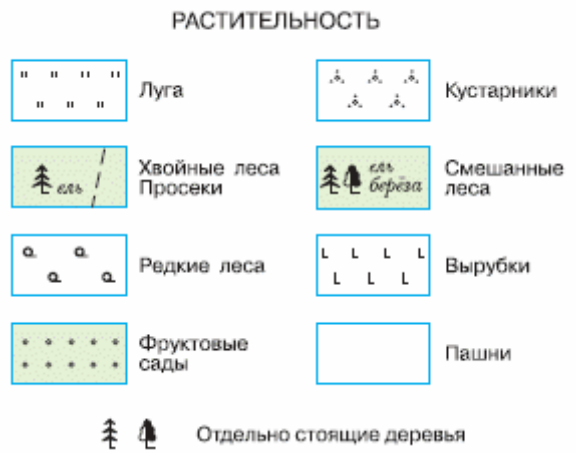
Задача заполнения пиксельной области другим рисунком.

Рассмотрим виртуальную растровую плоскость **P**, точки которой **(x,y)** могут иметь целочисленные координаты от $-\infty$ до $+\infty$. Ограниченная часть ее около начала координат – это наш растр:

R[n × m] = {(x,y), x=0…(n-1), y=0…(m-1)}

В нем выделена пиксельная область **O**. Будем считать, что у нас есть характеристическая функция этой области – **Fo(x,y)**. Таким образом мы можем задавать «независимые от растра» фигуры, которые будут иметь всегда гладкие границы даже при многократном увеличении.

Шаблон –растр обычно меньше – **Pt[s,t]**. точку привязки шаблона – **(x0,y0)** Вся плоскость **P** покрывается шаблоном как кафельной плиткой.

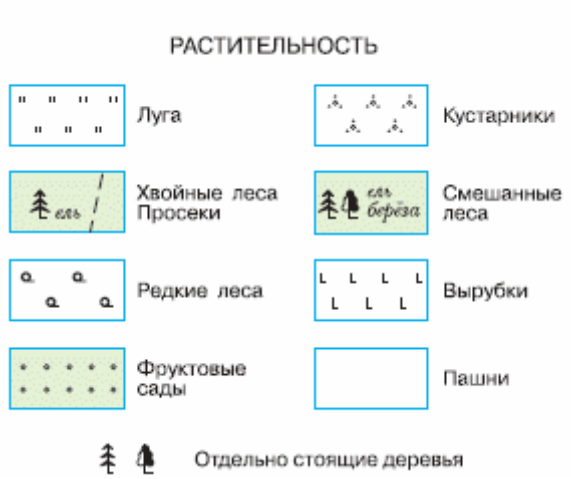


Алгоритм заполнения шаблоном

Вцелом данный метод является простейшим способом текстурирования и лежит в основе всех остальных текстурных покрытий.

Идея алгоритма может быть выражена следующим образом (для несимметричного шаблона):

```
for (j = 0; j < m; j++) // построчное сканирование растра
  for (i = 0; i < n; i++) {
    if (Fo(i, j) == 1) {
      // определить координаты "внутри своей копии плитки"
      is = (i – x0) % s;
      jt = (j – y0) % t;
      // переписать значение из шаблона
      R[i][j] = Pt[is][jt];
    }
  }
```



Алгоритм заполнения шаблоном

Вцелом данный метод является простейшим способом текстурирования и лежит в основе всех остальных текстурных покрытий.

Идея алгоритма может быть выражена следующим образом (для несимметричного шаблона):

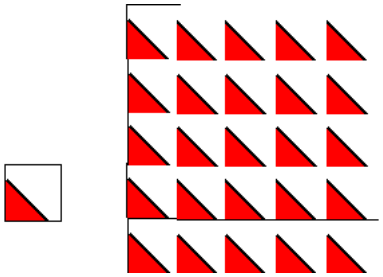
```
for (j = 0; j < m; j++) // построчное сканирование растра
  for (i = 0; i < n; i++) {
    if (Fo(i, j) == 1) {
      // определить координаты "внутри своей копии плитки"
      is = (i – x0) & (s-1);
      jt = (j – y0) & (t-1);
      // переписать значение из шаблона
      R[i][j] = Pt[is][jt];
    }
  }
```

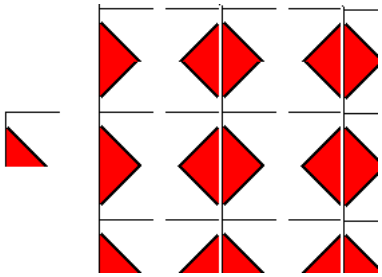
если:

$s = 2^N$

$t = 2^M$

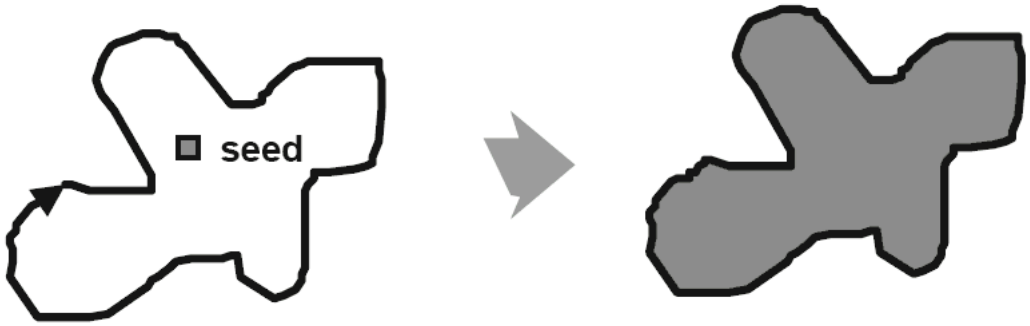
Для симметричных





Такие возможности предлагает, например, OpenGL. Использование симметрии позволяет экономить память под шаблоны. Очевидно, что можно предложить покрытие, когда по горизонтали идет просто повторение, а по вертикали зеркальное отражение, и наоборот. Другими словами, за счет симметрии мы для случая шаблон размером $2N \times 2N$ представили растром размером $N \times N$

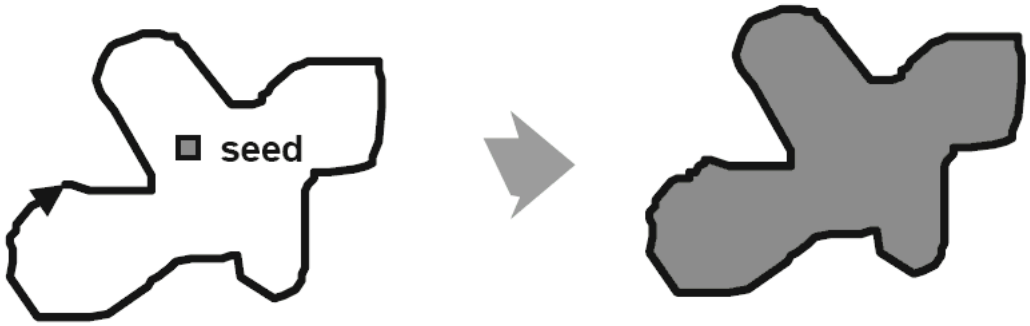
Заливка с затравкой (seed)




```
// Recursive Flood Fill
void FloodFill4(x,y,Oldval, Newval) {
  if(Pixel(x,y) == Oldval) {
    SetPixel(x,y,Newval);
    FloodFill4(x,y+1,Oldval,Newval);
    FloodFill4(x,y-1,Oldval,Newval);
    FloodFill4(x+1,y,Oldval,Newval);
    FloodFill4(x-1,y,Oldval,Newval);
  }
}
```

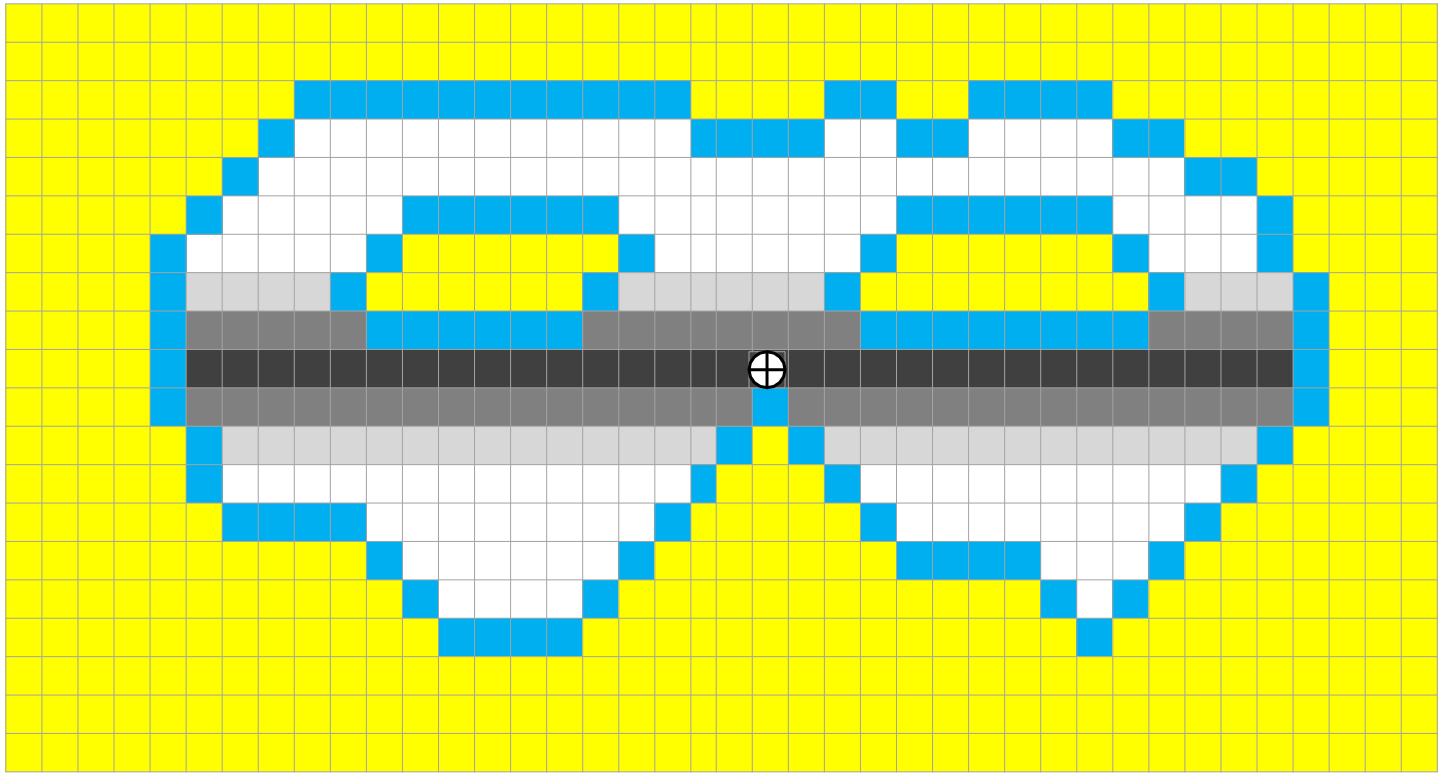
Для 4-х связной области. А как для 8?

Заливка с затравкой (seed)



- **Span filling algorithm**
- Без рекурсии, более эффективен.
- Span – непрерывный горизонтальный «отрезок» из пикселей, которые принадлежат области (имеют значение OV) и ограниченный с обоих концов пикселями, которые не принадлежат ей (имеют другие значения).
- Исходные данные: Пиксельная область ПО с границей. Все пиксели области имеют значение OV. Затравка seed – это пиксель с координатами (Xs,Ys), принадлежащий области.
- Задача: заменить значение OV на значение NV у всех пикселей области

Span



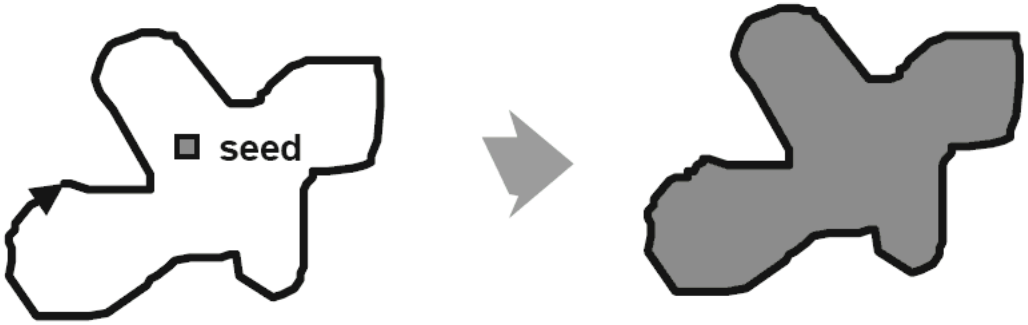
Внешняя ПО (заливать не надо)

Граница ПО

Примеры спанов

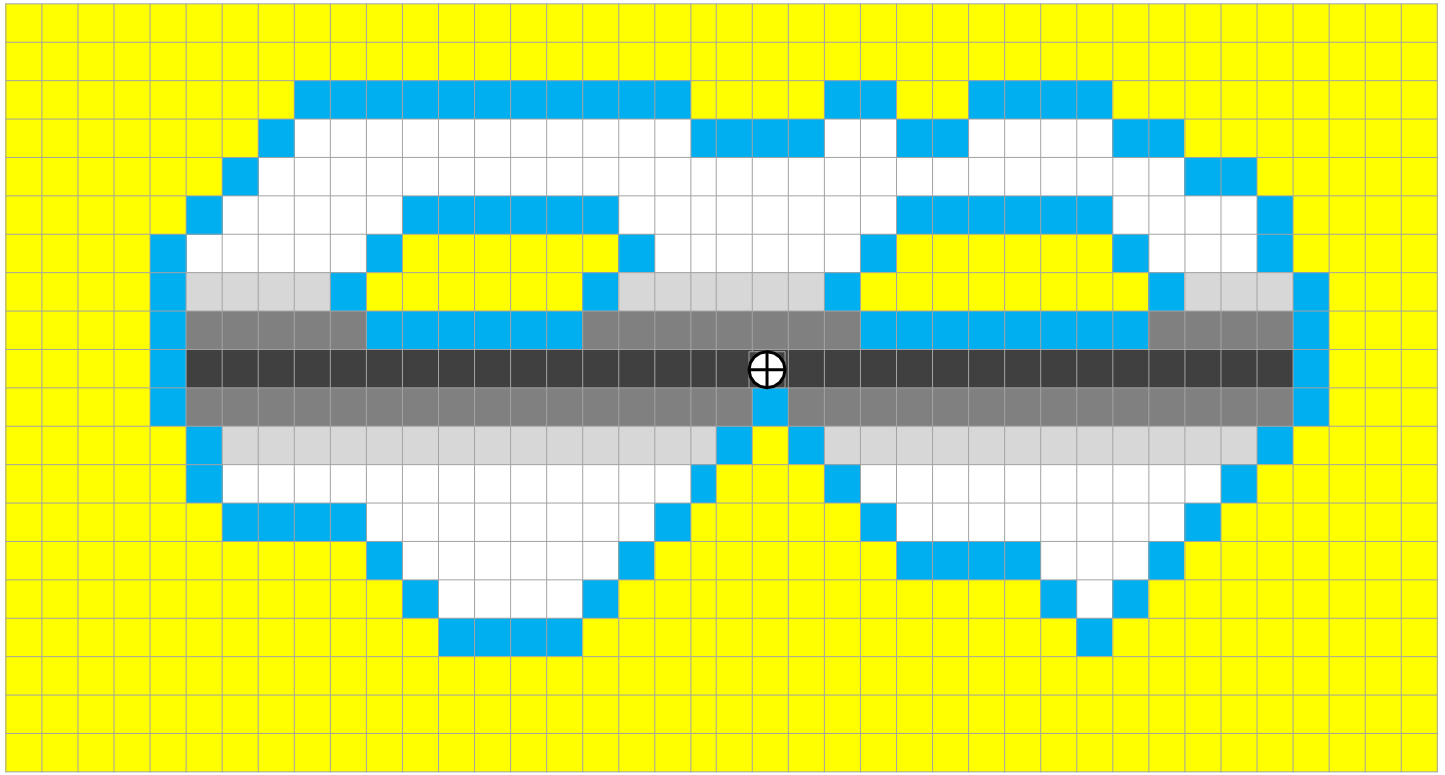
Затравка

Span filling algorithm



- Начало. Определить span, содержащий seed. Поместить его в стек спанов (сначала стек пуст).
- Пока стек не пуст
  - Берем из стека спан C;
  - Заполняем пиксели C значением NV;
  - Поднимаемся вверх на 1 линию растра. Находим все спаны связанные с C (согласно связности 4 или 8) и помещаем их в стек.
  - Опускаемся вниз на 1 линию растра. Находим все спаны связанные с C (согласно связности 4 или 8) и помещаем их в стек.

Span



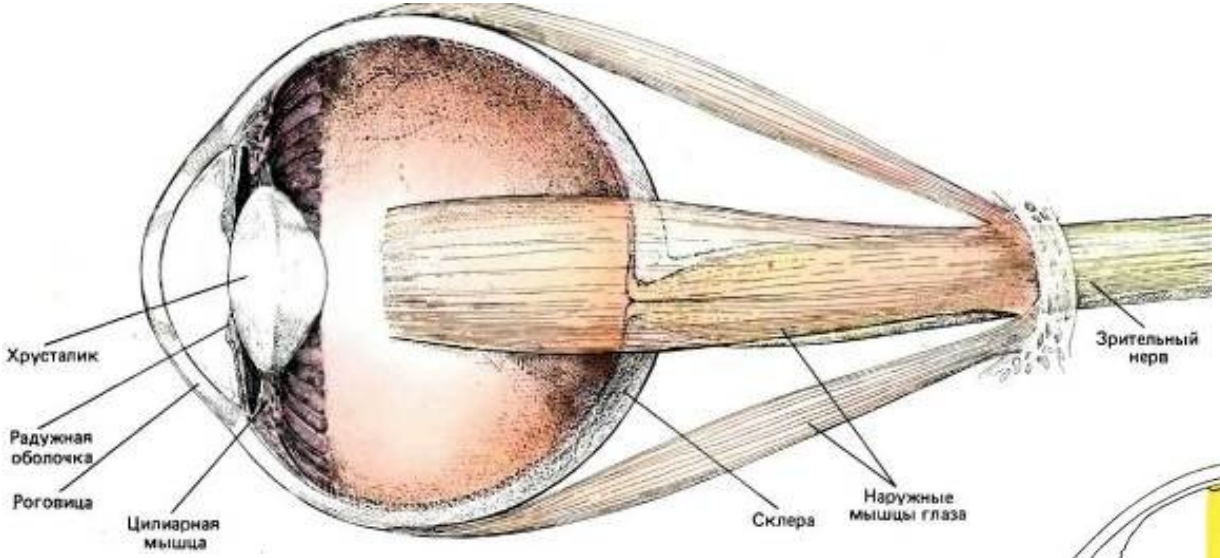
Внешняя ПО (заливать не надо)

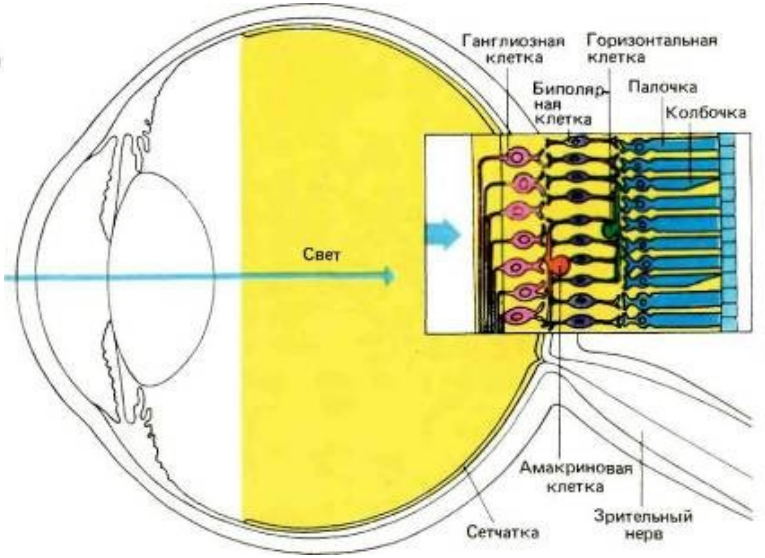
Граница ПО

Примеры спанов

Затравка

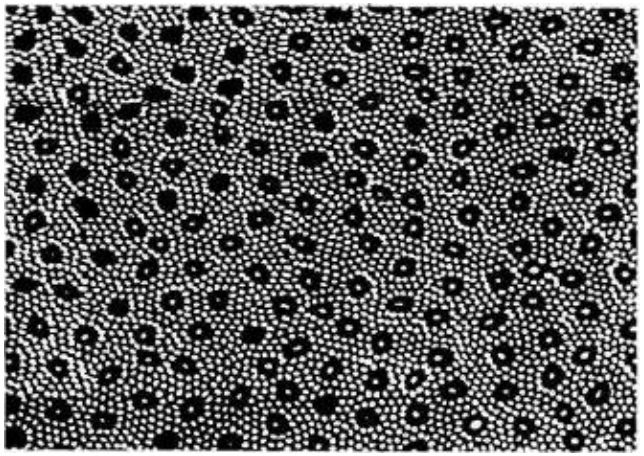
Строение глаза



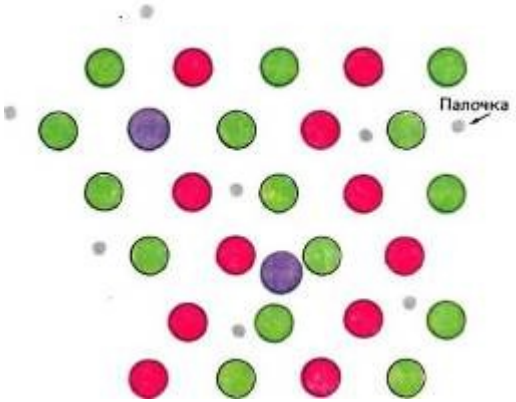


Угловое разрешение

0,5 угловых минуты в центре



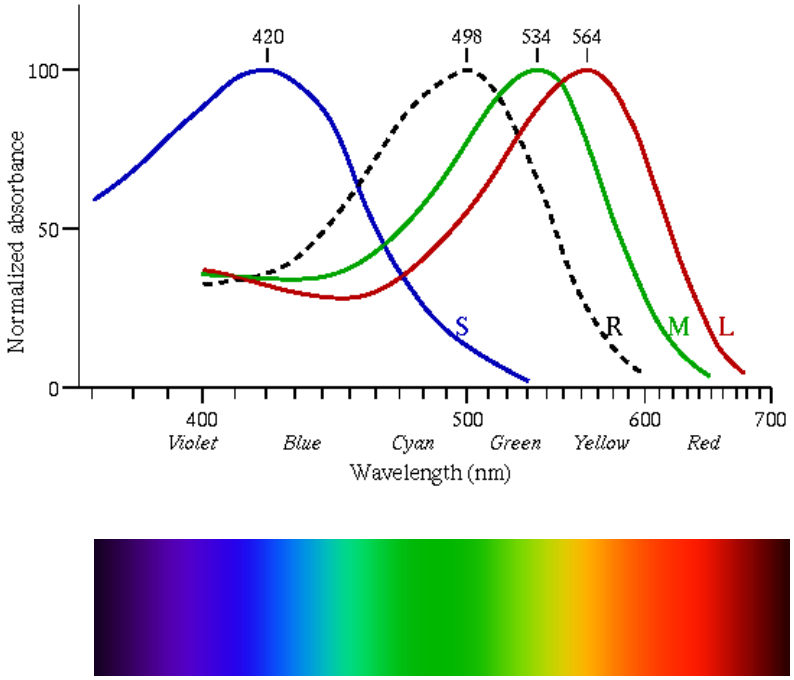
Белые точки – пигмент
Темные пятна – колбочки



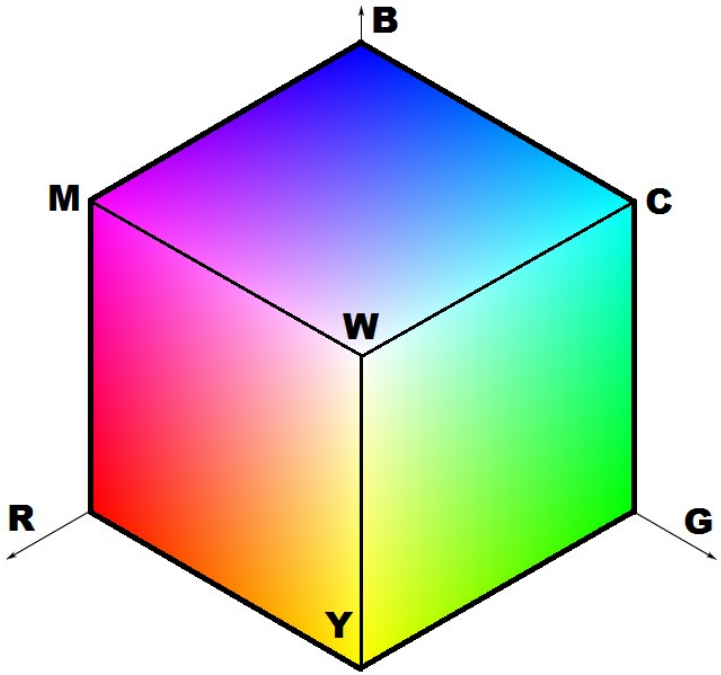
Угловое разрешение по яркости
в разы выше разрешения по цвету

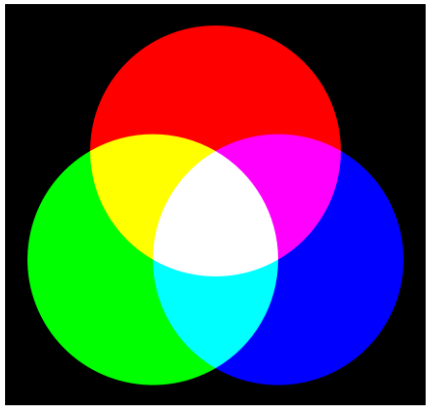
Цветовые модели

- Аддитивная
  RGB
- Субтрактивная
  CMY/CMYK
- “Человеческая”
  HSV/HLS
- Телевизионная
  YCbCr/YUV
- “Научная”
  L*a*b

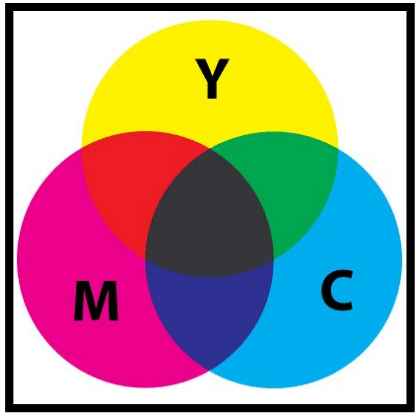


Цветовые модели

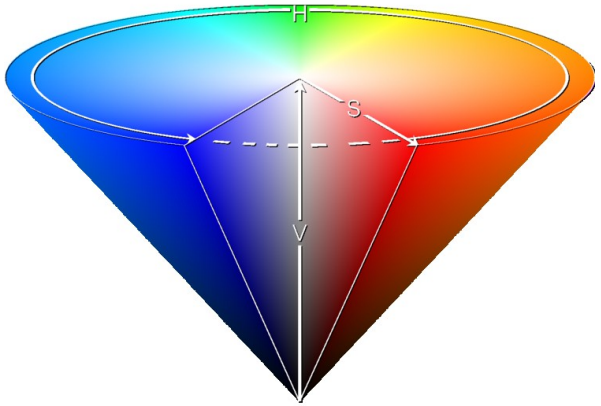




RGB



CMY



HSV

Цветовые модели

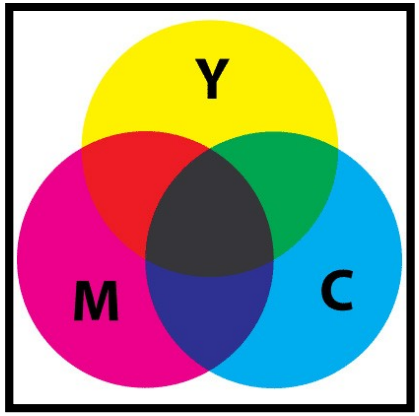
$C,M,Y \rightarrow C',M',Y',K$

$K(blacK) = \min(C,M,Y)$

$C' = C - K$

$M' = M - K$

$Y' = Y - K$



CMYK

Черная краска – дешевая и «точная»

Смесь трех красок в равной пропорции дает грязно-серый, а не черный цвет