Rasterization光栅化

光栅展示pixel

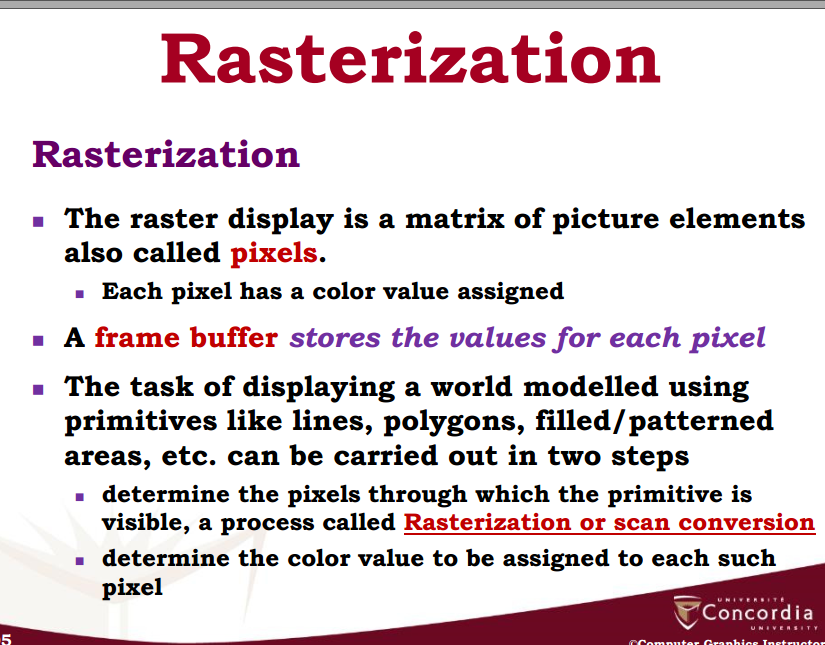
每一个pixel都有一个color值

Frame buffer存储着每个pixel的值

展现我们们所生成成的primitive例如线条三角一共两个步骤

1.决定那个pixel是可见的，这个过程叫做rasterization

2.决定每个Pixel的颜色



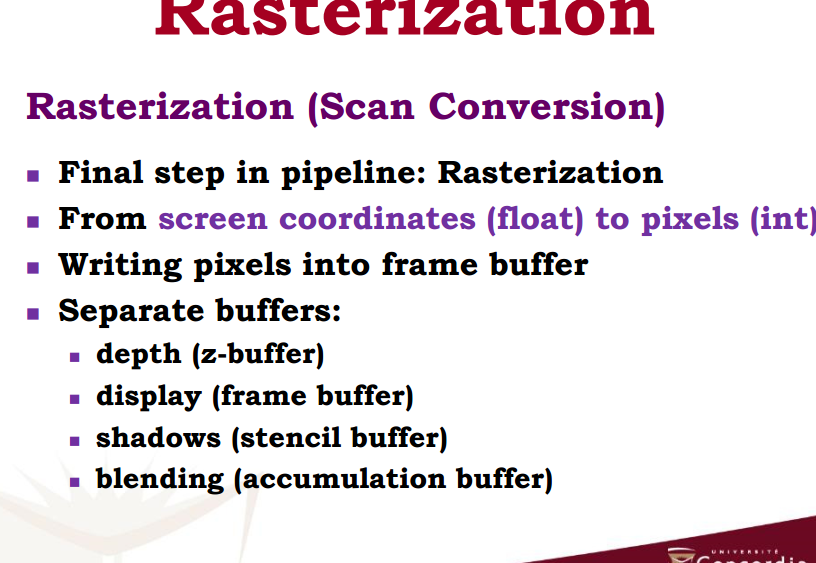
Rasterization

Pipeline的最后一步

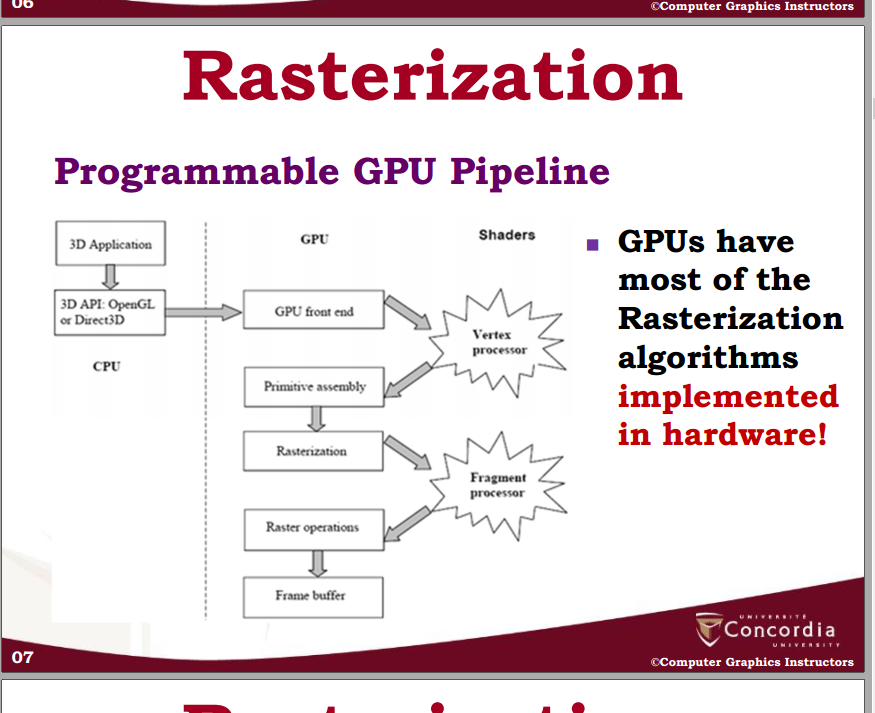
把屏幕坐标（float）转化成对应Pixel（int）

再把pixel写入frame buffer

有不同的buffer

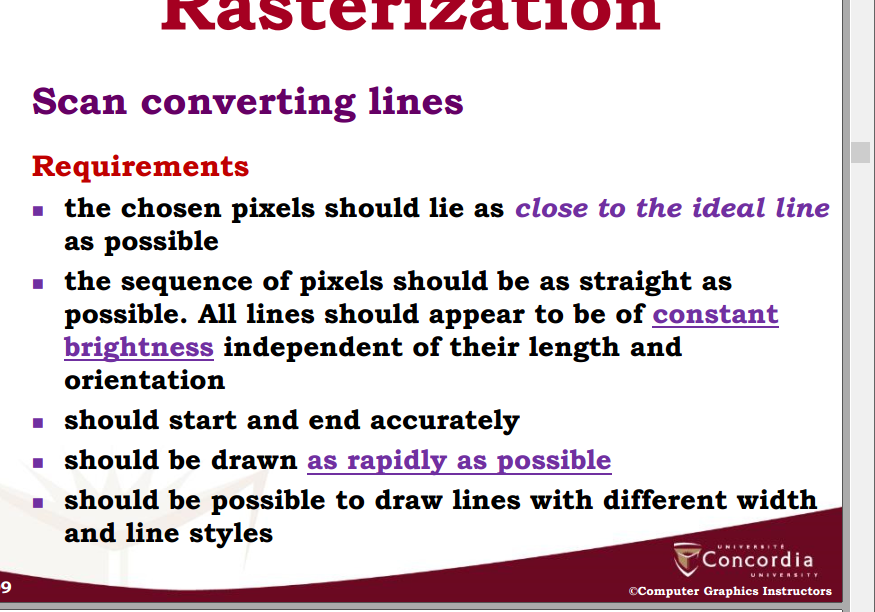


Zbuffer是death ,display是frame buffer， 影子是stencil buffer模板缓冲，



光栅化算法都装在硬件中

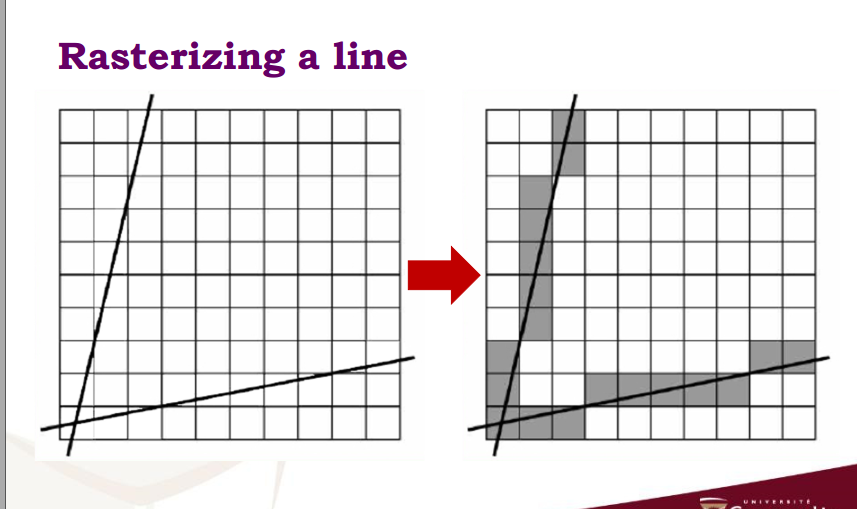
画线



选好的pixel要尽量接近理想线，一串sequence必须要近可能的直，所有线亮度应该一样

应该正确落在start 与end点上

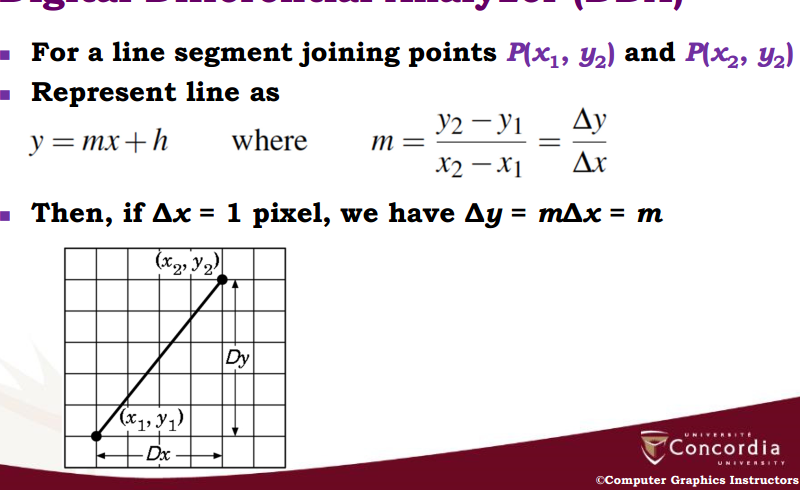
画的越快越好



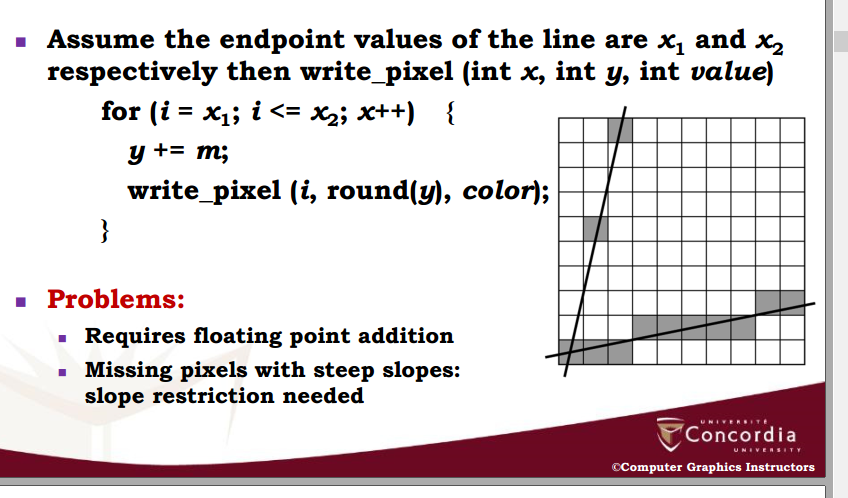
过程:

Digital differential analyzer,

通过两个点我们能得到一个数学上的线



然后求出斜率，然后如果deltax等于1，那么y的差距就是m



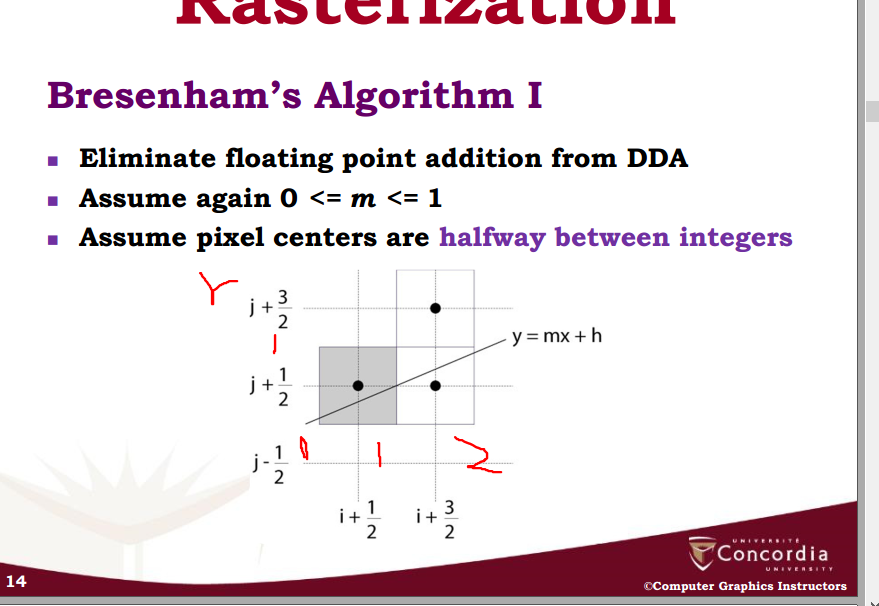
假设我们已知x1x2，那么每次就只要在y上加上对应的m也就是每次往右一格来回画就行

导致的问题，如果斜率过大，例如那根竖着的线会少很多点，需要加上小数

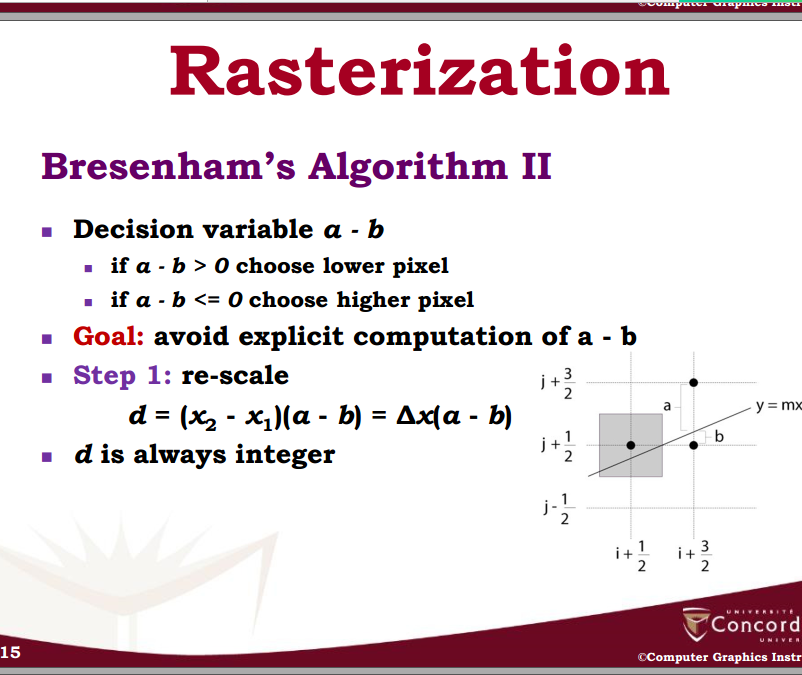
我们使用bresenham算法

可以消除小数加减

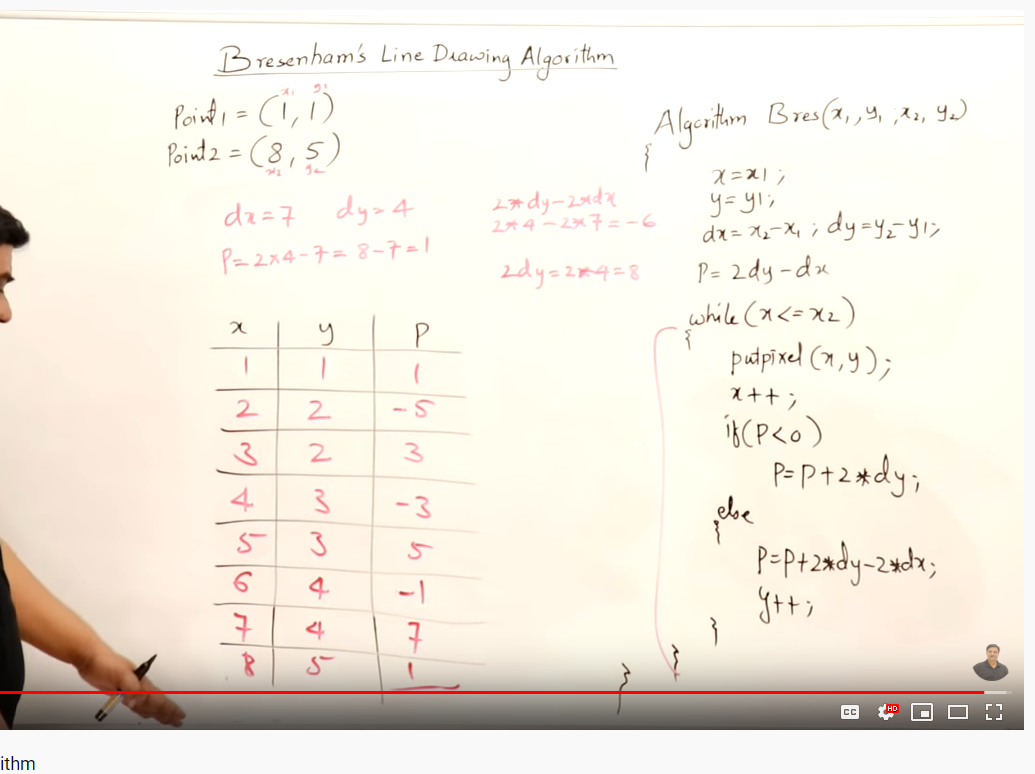
假设m小于1

，

取每个像素的中间格，



如果上面一个像素中间格到这条线的距离大于下面一条距离，那么就选下面一格



已知两个点，求X差值Y差值

求P，，等于两倍Y差值-一倍X差值

然后就是循环

每次X+1,如果当前P<0那么P家两倍Dy

如果P大于0，加两倍DY减两倍DX，Y+1

11 85

Dx=7 dy=4

P=2dy-dx=1

Step1. 画点，x++,p>0,所以下一步Y+1，P=1+2\*4-2\*7=-5

Step2 画22 x++ p<0所以p+8=3 所以Y不进1

STEP3 画32 X++ p>0 P=3+8-14=-3所以Y进1…..

58 911

Dx=4 dy=3

2dy-dx=6-4=2

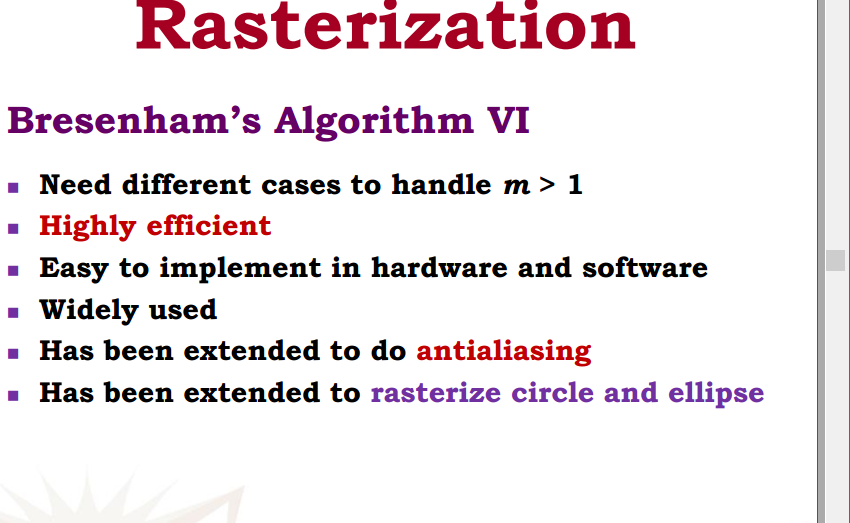
5 8 2

6 9 0 p>0所以+6-8=0还是y+1

7 10 -2

8 10 4

9 11 2



Bresenham算法效率高，但是对于m>1需要另外写

容易Implement

可以用来抗锯齿

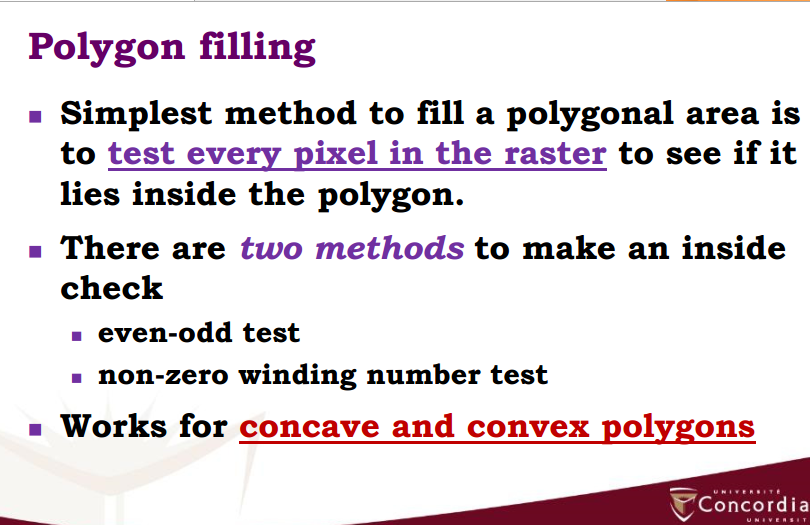
可以用来给圆与椭圆光栅化

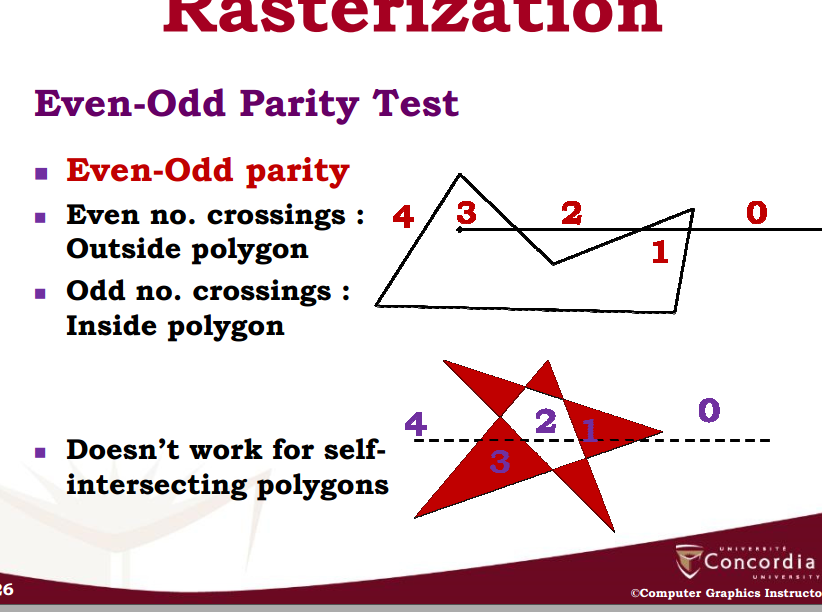
怎么给polygon填充

最简单的方法是test每个raster内的像素，看看在不在polygon内

两种test法

哪怕对凹凸多边形都有效



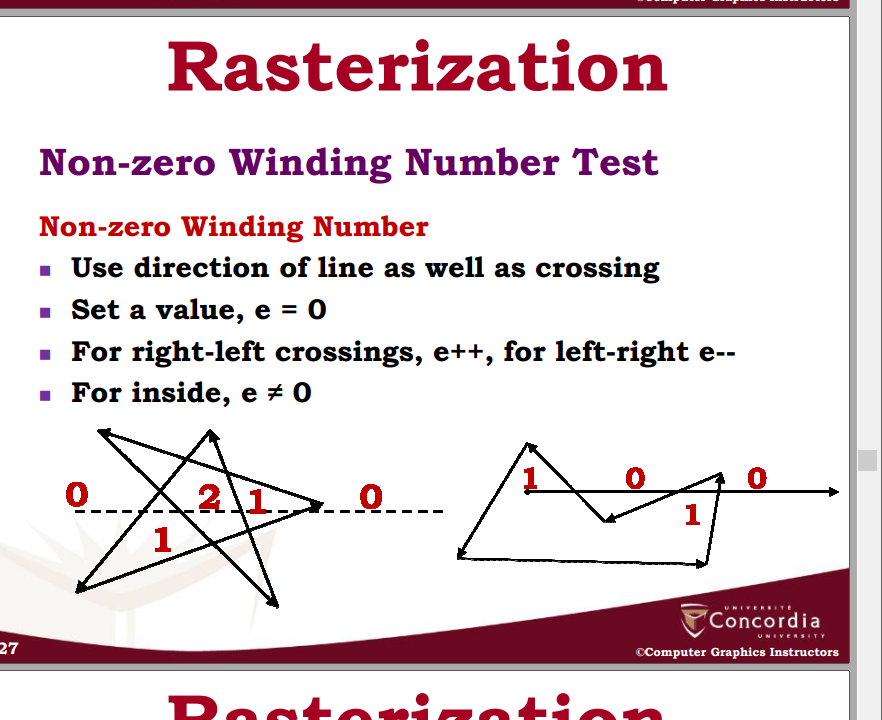


Even-odd穿越法

Polygon里面的奇数外面的偶数

但是对那种self-intersecting polygons无效

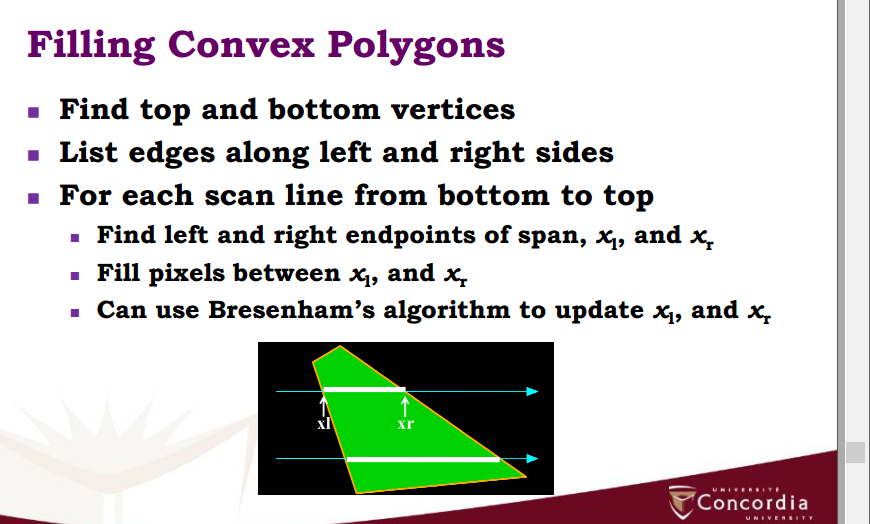
Non-zero winding法



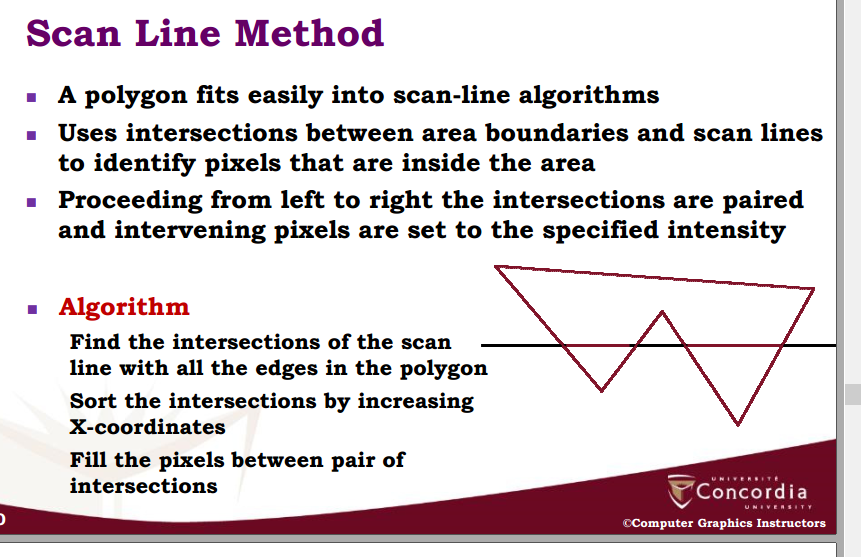
让多边形的Line带有方向

从右边往左穿e+1，（从上往下的是左边往右），否则-1

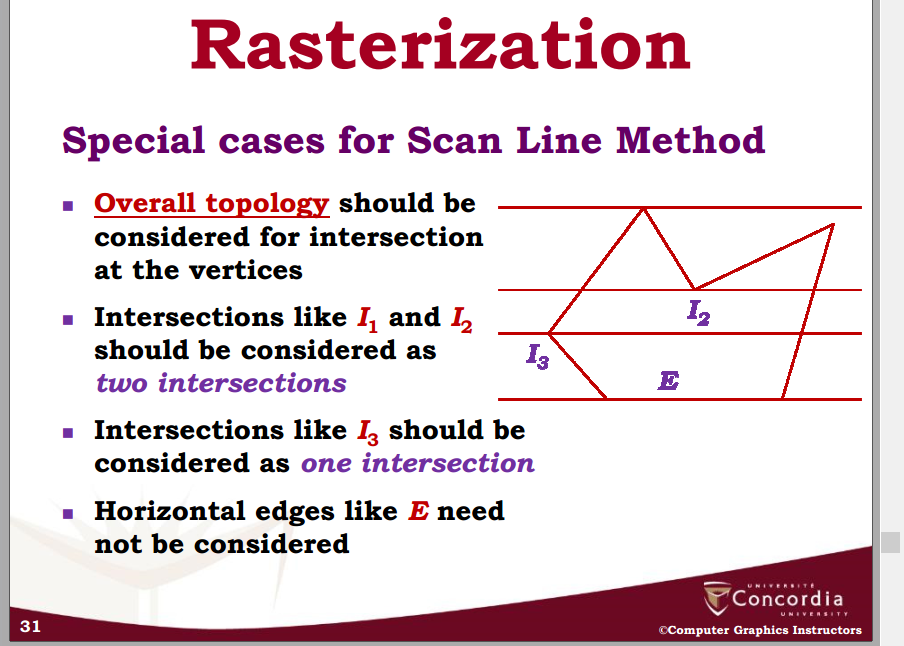
只要不等于0就是inside



Filling凸面polygon，从左到右画一条线，然后把所有交点按左到右排好，两两成对，然后填色



这种简单的同样



复杂的的就不一样了

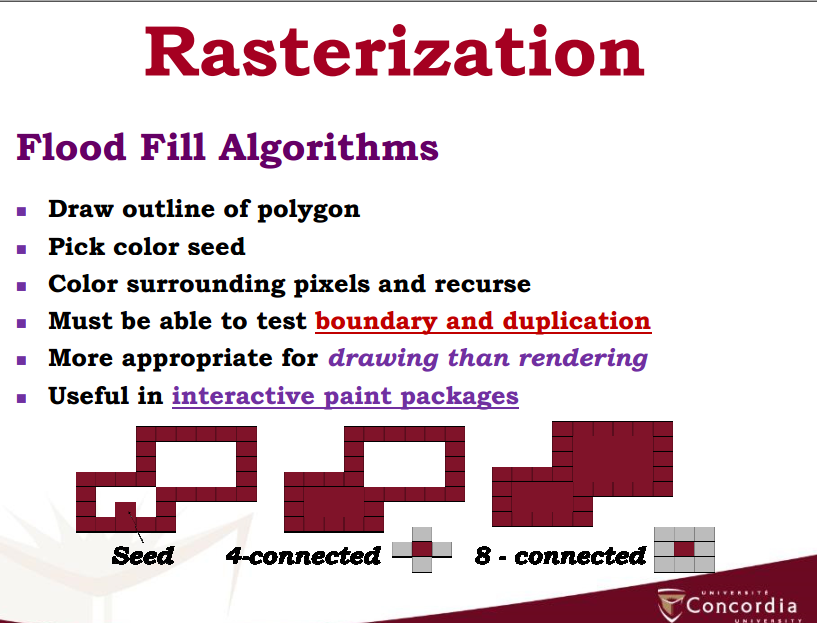
I2 I1这样的intersection应该被认作是两个点

I3这种被认作是一个点

效率很高每个pixel只visit一次

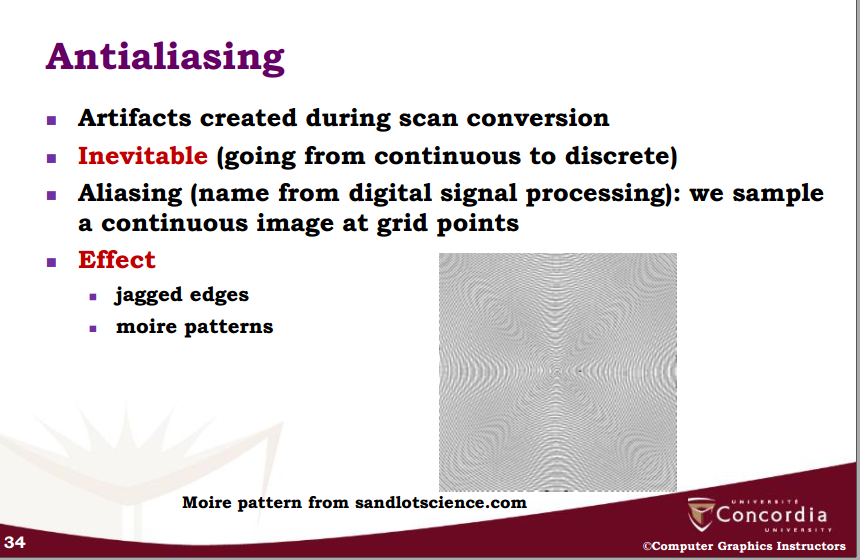
Flood fill algorithm

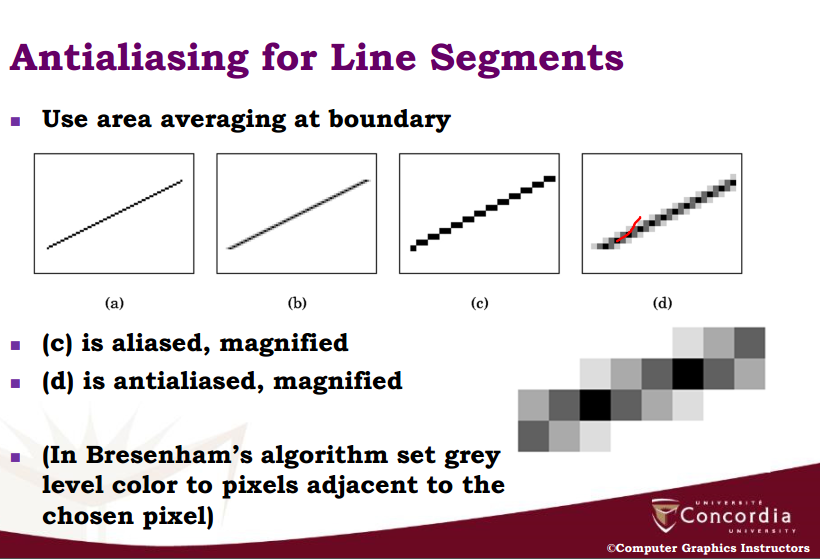
选取颜色然后涂范围内的颜色



Antialiasing抗锯齿

锯齿在scan conversion也就是光栅化的时候必然出现



线条抗锯齿，用灰度来表示

通常用3X3然后看在里面还是外面，如果9个颜色都一样了，就停止