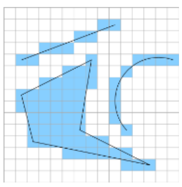
CS5620

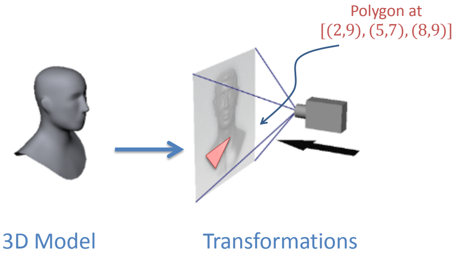
Intro to Computer Graphics

Rasterization

Rasterization

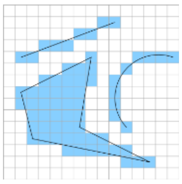


Reminder – Graphics Pipeline



Raster Display

- The screen is a discrete grid of elements called ***pixels***
- Shapes drawn by setting some pixels "on"

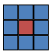


Rasterization

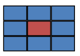
- How do we draw geometric primitives?
  - Convert from geometric definition to pixels
  - *rasterization* = selecting the pixels
- Will be done frequently
  - must be fast:
    - use integer arithmetic
    - use addition instead of multiplication

Terminology

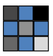
- Pixel: Picture element
  - Smallest accessible element in picture
  - Usually rectangular or circular



- Aspect Ratio: Ratio between physical dimensions of pixel (not necessarily 1 !!)

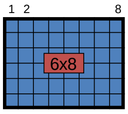


- Dynamic Range: Ratio between minimal (not zero!) and maximal light intensity emitted by displayed pixel. Measured in bits.

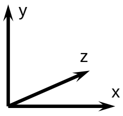


Terminology

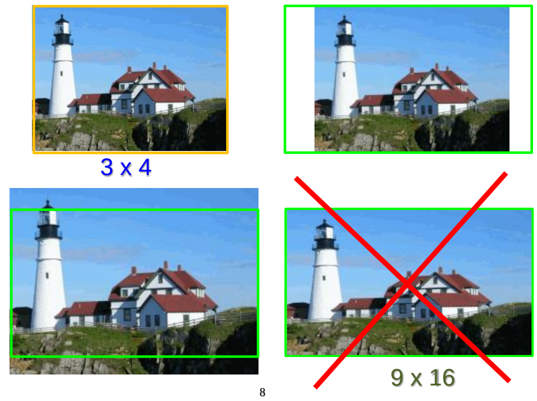
- Resolution: number of distinguishable rows and columns on device. Measured in
  - Absolute values (1K x 1K)
  - Relative values (300 dots per inch)
- Screen Space: discrete 2D Cartesian coordinate system of screen pixels



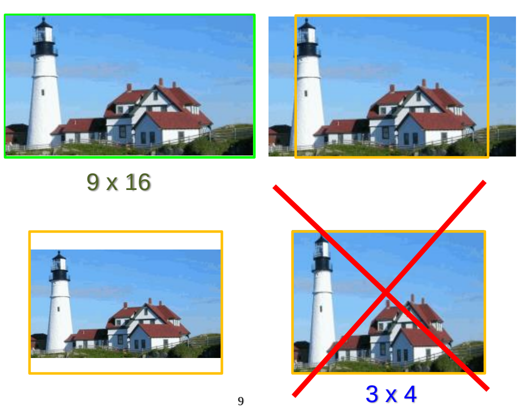
- Object Space: 3D Cartesian coordinate system of the universe where the objects (to be displayed) are embedded



JACOBS TECHNION-CORNELL INSTITUTE AT CORNELL TECH



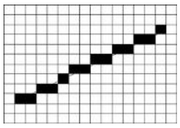
8



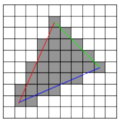
9

Today

- Drawing lines



- Filling polygons



10

Naïve Algorithm for Lines

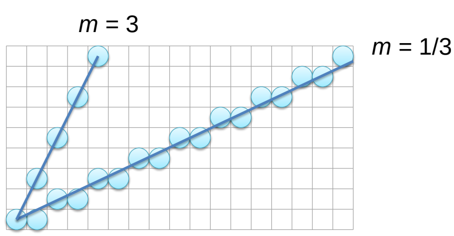
- Line definition: $ax + by + c = 0$
- Also expressed as: $y(x) = mx + g$
  - m = slope
  - $g = y(0)$

```
for x=xmin to xmax
  y = m*x+g
  light pixel (x,y)
```

11

Slope Dependency

- Only works with -1 ≤ m ≤ 1:



Extend by symmetry for m > 1

12

JACOBS TECHNION-CORNELL INSTITUTE AT CORNELL TECH

Problems

- 2 floating-point operations per pixel
- Improvement:

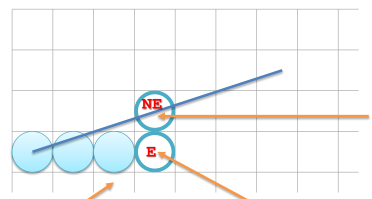
```
y = m*xmin + g
for x=xmin to xmax
   y += m
   light pixel (x,y)
end
```

- Still 1 floating-point operation per pixel
- Compute in floats, pixels in integers

13

Bresenham Algorithm: Idea

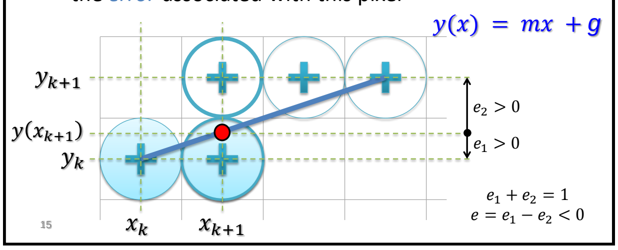
- At each step, choice between 2 pixels ($0 \leq m \leq 1$)



Bresenham Algorithm

- Need a criterion to choose
- Distance between line and center of pixel:

  the *error* associated with this pixel

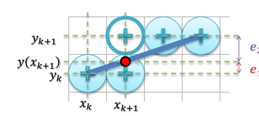


15

Bresenham Algorithm

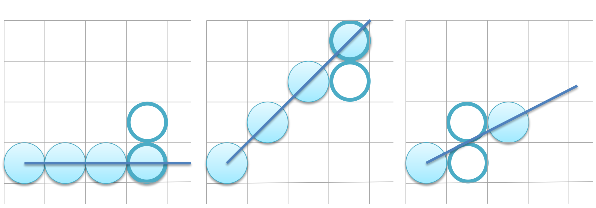
- Choose by sign of $e = e_1 - e_2$

```
if e < 0
  go East (E)
  update e
else
  go North East (NE)
  update e
```



16

Special Cases



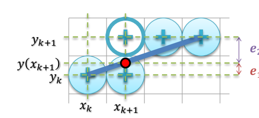
$e_1 = 0$ $e_2 = 1$ $e = -1 < 0$ E	$e_1 = 1$ $e_2 = 0$ $e = 1 > 0$ NE	$e_1 = 1/2$ $e_2 = 1/2$ $e = 0$ E or NE

17

Bresenham Algorithm

Choose between **E** and **NE** by sign of $e = e_1 - e_2$

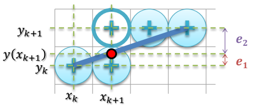
```
y = ymin
for x = xmin to xmax
   if e < 0
     // E
     update e   // e1 = e1 + m, e2 = e2 - m
   else
     y++ // NE
     update e   // e1 = e1 + m - 1, e2 = e2 - (m - 1)
   light pixel(x,y)
```



18

Bresenham Algorithm

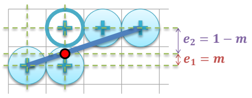
```
y = ymin
for x = xmin to xmax
   if e < 0
     // E
     e += 2m
   else
     y++   // NE
     e += 2m-2
   light pixel(x,y)
```



19

Bresenham Algorithm

```
// Initialize e
// e1 = m, e2 = 1 − m
e = 2m-1
y = ymin
for x = xmin to xmax
   if e < 0
     // E
     e += 2m
   else
     y++ // NE
     e += 2m-2
   light pixel(x,y)
```



20

Bresenham Algorithm
in integers

```
e = 2m-1
y = ymin
for x = xmin to xmax
   if e < 0
     // E
     e += 2m
   else
     y++ // NE
     e += 2m-2
   light pixel(x,y)
```

$$m = \frac{\Delta y}{\Delta x} = \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

define new variable

$$d = e\,\Delta x$$

21

Bresenham Algorithm
In integers

```
d = 2Δy − Δx
y = ymin
for x = xmin to xmax
   if d < 0
     // E
     d += 2Δy
   else
     y++ //NE
     d += 2Δy-2Δx
   light pixel(x,y)
```

$$m = \frac{\Delta y}{\Delta x} = \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

$$d = e\,\Delta x$$

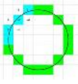
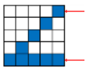
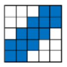
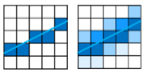
22

Bresenham Algorithm

```
Line ( x1, y1, x2, y2 )
begin
      int  Δx, Δy, d, Δe, Δne
      int  x, y, d ;
      x ⇐ x1 ;          y ⇐ y1 ;
      Δx ⇐ x2 − x1 ;    Δy ⇐ y2 − y1 ;
      d ⇐ 2∗Δy − Δx ;
      Δe ⇐ 2∗Δy ;       Δne ⇐ 2∗(Δy − Δx) ;
      PlotPixel ( x, y ) ;
      while ( x < x2 ) do
            if ( d < 0 ) then
            begin  // E
                  d ⇐ d + Δe ;
                  x ⇐ x + 1 ;
            end ;
            else begin  // NE
                  d ⇐ d + Δne ;
                  x ⇐ x + 1 ;
                  y ⇐ y + 1 ;
            end ;
            PlotPixel ( x, y ) ;
      end ;
end ;
```

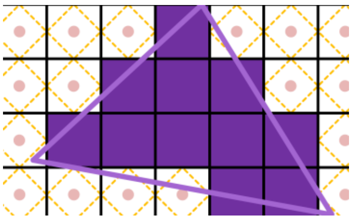
23

Generalizations

- Circles 
- Other algebraic curves
- Line intensity 
- Line thickness 
- Anti-aliasing 

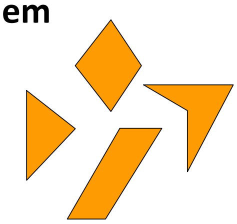
24

Polygon Fill



The Problem

- Problem:
  - Given a closed simple 2D polygon, fill its interior with specified color on graphics display

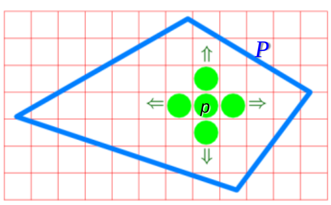


- Solutions:
  - Flood fill
  - Scan conversion

26

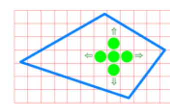
Flood Fill Algorithm

- Let P be a polygon whose boundary is already drawn
- Let C be the color to fill the polygon
- Let $p = (x,y) \in P$ be a point inside P

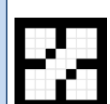


27

Flood Fill



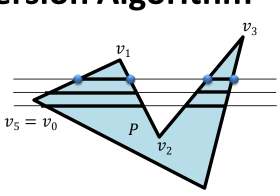
FloodFill(Polygon P, int x, int y, Color C)
if not (**OnBoundary**(x, y, P) or **Colored**(x, y, C))
begin
  PlotPixel(x, y, C);
  FloodFill$(P, x+1, y, C)$;
  FloodFill$(P, x, y+1, C)$;
  FloodFill$(P, x, y-1, C)$;
  FloodFill$(P, x-1, y, C)$;
end ;



28

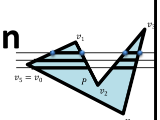
Basic Scan Conversion Algorithm

- Let P be a polygon with n vertices v_0 to v_{n-1} $(v_n = v_0)$
- Let C be the color
- Each intersection of ***straight line*** with ***boundary*** moves in/out the polygon
- Detect (and set) pixels inside the polygon boundary



29

Basic Scan Conversion

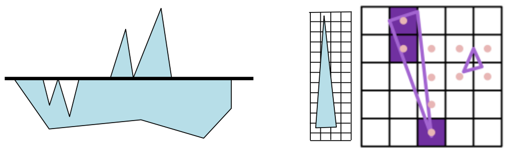


ScanConvert (Polygon P, Color C)
for $y := 0$ to ScreenYMax do
  $I \Leftarrow$ Points of intersections of edges of P with line $Y = y$;
  Sort I in increasing X order and
  Fill with color C alternating segments ;
end

30

JACOBS TECHNION-CORNELL INSTITUTE AT CORNELL TECH

Special Cases



31

Comparison

Flood Fill	Scan Conversion
Very simple	More complex
Discrete algorithm in screen space	Discrete algorithm in object and/or screen space
Requires *GetPixelVal* system call	Device independent
Requires a seed point	No seed point required
Requires very large stack	Requires small stack
Common in paint packages	Used in image rendering
Unsuitable for line-based Z-buffer	Suitable for line-based Z-buffer

32