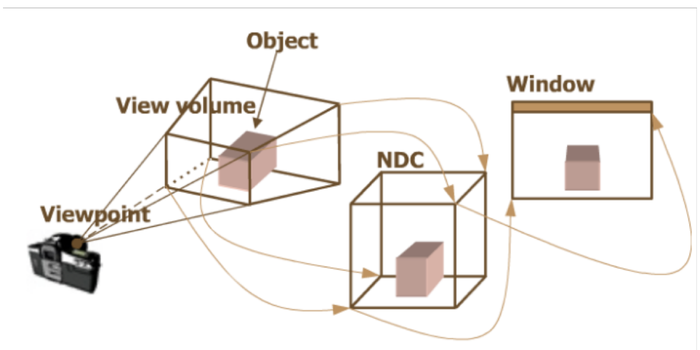
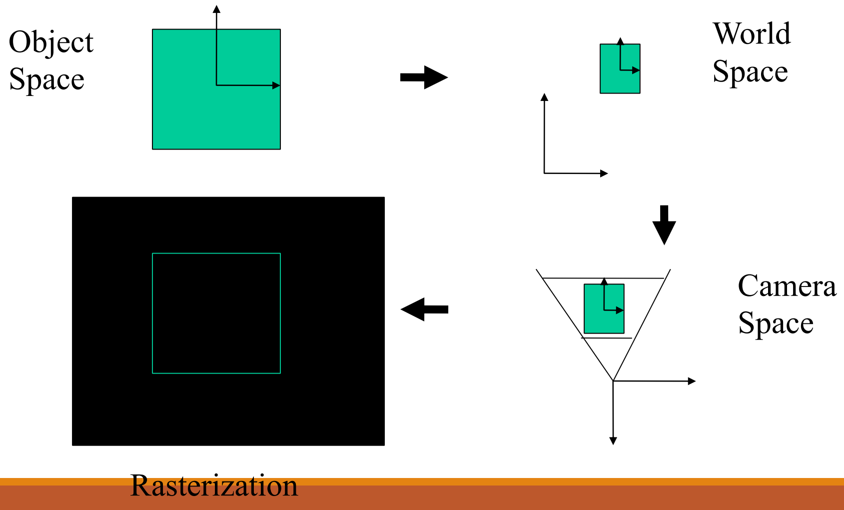
Lecture 6

Rasterizing lines, polygons

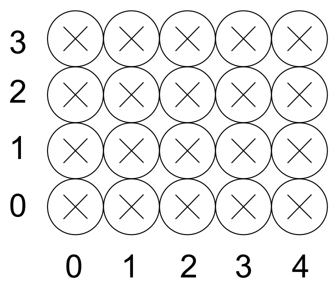


Intuitively



Rasterization

Array of pixels



Rasterization (scan conversion)

Final step in pipeline: rasterization

From screen coordinates (float) to pixels (int)

Writing pixels into frame buffer

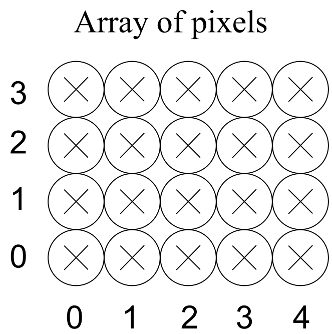
Separate buffers:

– depth (z-buffer),

– display (frame buffer),

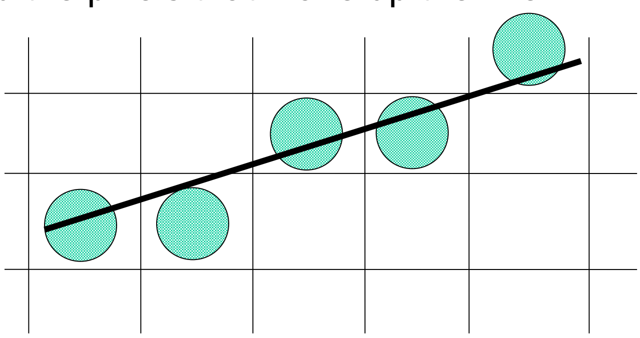
– shadows (stencil buffer)

– blending (accumulation buffer)



Rasterizing Lines

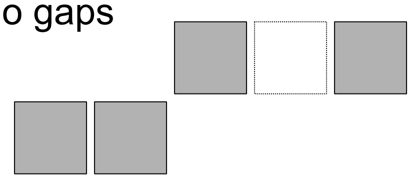
Given two endpoints, $(x_0, y_0), (x_1, y_1)$ find the pixels that make up the line.



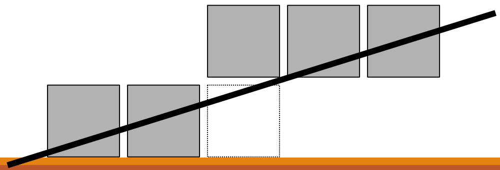
Rasterizing Lines

Requirements

1. No gaps



1. Minimize error (distance to line)



Rasterizing Lines

Equation of a Line:

$$y = mx + h = f(x)$$

Taylor Expansion: $f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$

$$f(x + \Delta x) = f(x) + f'(x)\Delta x$$

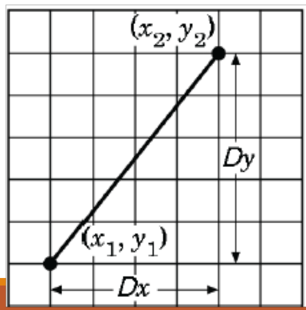
So if we have an x,y on the line,
we can find the next point incrementally.

Rasterizing Lines

$$y = mx + h \qquad \text{where} \qquad m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

if Δx = 1 pixel,

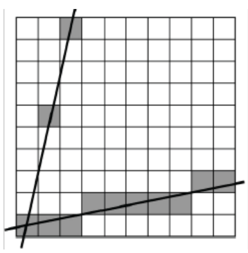
we have Δy = m,



Rasterizing Lines

Case1: **–1 < m < 1**, x0 < x1

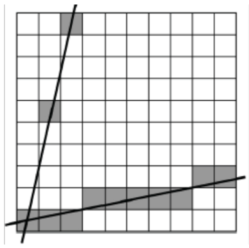
```
Line(int x0, int y0, int x1, int y1)
  float dx = x1 – x0;
  float dy = y1 – y0;
  float m  = dy/dx;
  float x = x0, y= y0;

  for(x = x0; x <= x1; x++)
    writePixel(x,round(y));
    y = y+m;
```



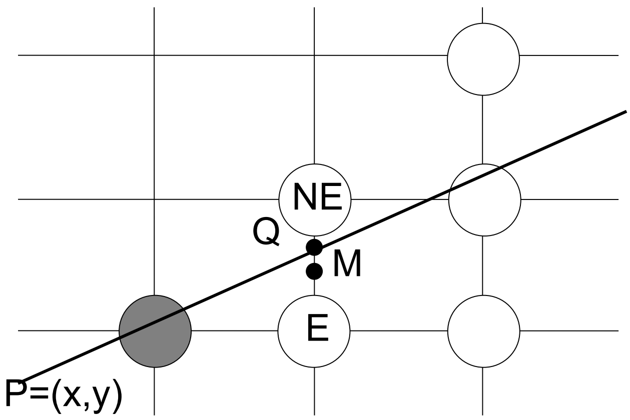
Rasterizing Lines

Problems with previous algorithm

1. round takes time
2. requires floating point addition
3. missing pixels with steep slopes:
   - slope restriction needed



Midpoint Algorithm



If Q <= M, choose E. If Q > M, choose NE

Implicit Form of a Line

Implicit form

$$ax + by + c = 0$$

Explicit form

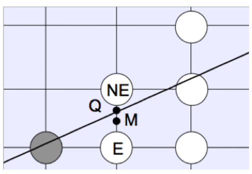
$$y = \frac{dy}{dx} x + B$$

$$dy\ x - dx\ y + B\ dx = 0$$

$$a = dy \quad b = -dx \quad c = B\ dx$$

Decision Function

Assume 0 < m < 1, x0 < x1



Line equation: $ax + by + c = 0$

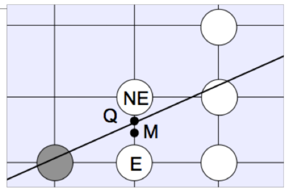
$$d = F(x,y) = a\ x + b\ y + c$$

mid point: $d = F(x+1, y+\frac{1}{2}) = a\,(x+1) + b\,(y+\frac{1}{2}) + c$

If d positive: M is below the line
If d negative: M is above the line
Zero on the line

Choose NE if d > 0
Choose E if d <= 0

Incrementing d



If choosing E:

→next midpoint:

$$d_{new} = F(x+2, y+\tfrac{1}{2}) = a(x+2) + b(y+\tfrac{1}{2}) + c$$

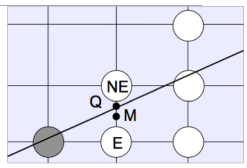
But:

$$d_{old} = F(x+1, y+\tfrac{1}{2}) = a(x+1) + b(y+\tfrac{1}{2}) + c$$

So:

$$d_{inc} = d_{new} - d_{old} = a = \Delta E$$

Incrementing d



If choosing NE:

→next midpoint:

$$d_{new} = F(x+2, y+\tfrac{3}{2}) = a(x+2) + b(y+\tfrac{3}{2}) + c$$

But:

$$d_{old} = F(x+1, y+\tfrac{1}{2}) = a(x+1) + b(y+\tfrac{1}{2}) + c$$

So:

$$d_{inc} = d_{new} - d_{old} = a + b = \Delta NE$$

Initializing d

$$d = F(x_0+1, y_0+\frac{1}{2}) = a\,(x_0+1) + b\,(y_0+\frac{1}{2}) + c$$

$$= a\,x_0 + b\,y_0 + c + a + b\frac{1}{2}$$

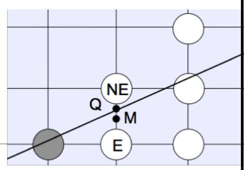
$$= a + b\frac{1}{2}$$

dy x – dx y + B dx = 0
→d = dy – ½ dx
→d = 2dy-dx

Multiply everything by 2 to remove fractions
(doesn't change the sign)

See: Bresenham Algorithm

Midpoint Algorithm



0 < m < 1, x0 < x1 only!!

```
Line(int x0, int y0, int x1, int y1)
  int dx = x1 – x0, dy = y1 – y0;
  int d = 2*dy-dx;
  int delE = 2*dy;
  int delNE = 2*(dy-dx);

  int x = x0, y = y0;
  setPixel(x,y);

  while(x < x1)
    if(d<=0)
      d += delE; x = x+1;
    else
      d += delNE; x = x+1; y = y+1;
    setPixel(x,y);
```

dy x – dx y + B dx = 0
a = dy
b = -dx

$$d = a + b\frac{1}{2} = dy - \frac{1}{2}dx$$

$$\Delta E = a = dy$$

$$\Delta NE = a + b = dy - dx$$

Bresenham's Algorithm

Highly efficient

Easy to implement

Widely used

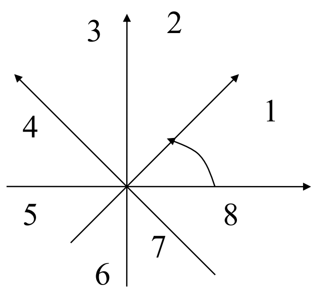
Limitations?

Need different cases to handle m > 1

- The midpoint line algorithm assumes that the slope (m) is between 0 and 1

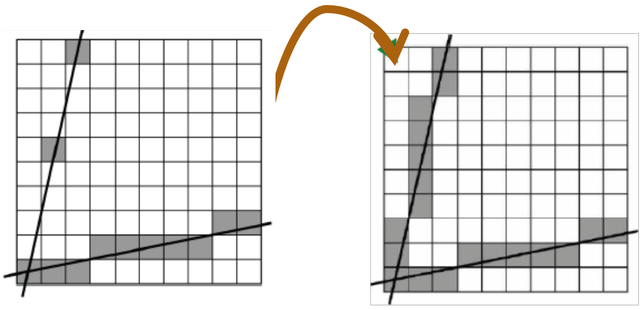
This implies that this algorithm only applies to lines in region 1

Extending to other regions left as a programming assignment



Midpoint Algorithm

Region2



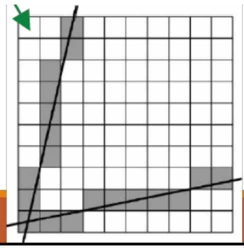
Anti-aliasing

Aliasing

Artifacts created during scan conversion

Inevitable (going from continuous to discrete)

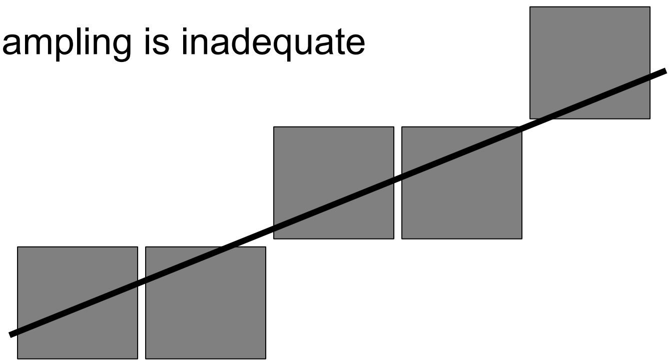
Aliasing
- we sample a continues image at grid points
- Jagged edges

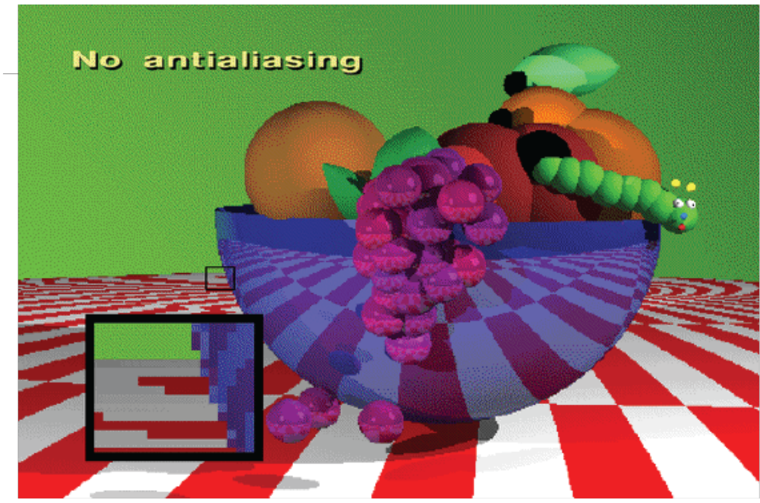


Anti-aliasing Lines

Lines appear jaggy

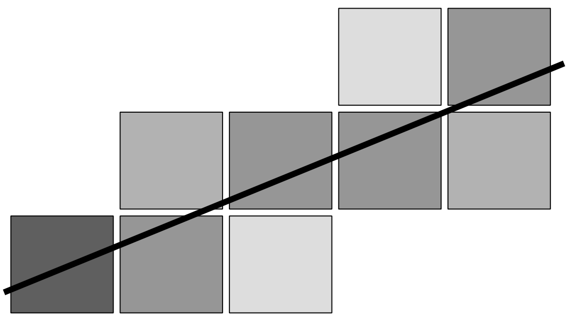
Sampling is inadequate

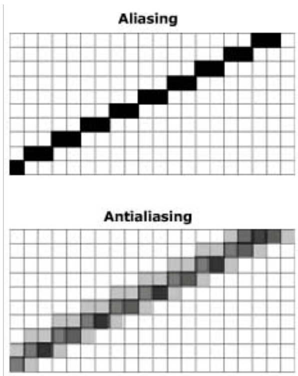




Anti-aliasing Lines

Trade intensity resolution for spatial resolution

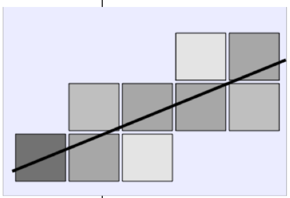




Anti-aliasing Lines

Assume 0 < m < 1, x0 < x1

```
Line(int x0, int y0, int x1, int y1)
  float dx = x1 - x0;
  float dy = y1 - y0;
  float m  = dy/dx;
  float x = x0, y= y0;

  for(x = x0; x <= x1; x++)
    int yi = floor(y); float f = y - yi;
    setPixel(x,yi,  1-f);
    setPixel(x,yi+1, f);
    y = y+m;
```



Putting it all together!!

Take your representation (points) and transform it from Object Space to World Space

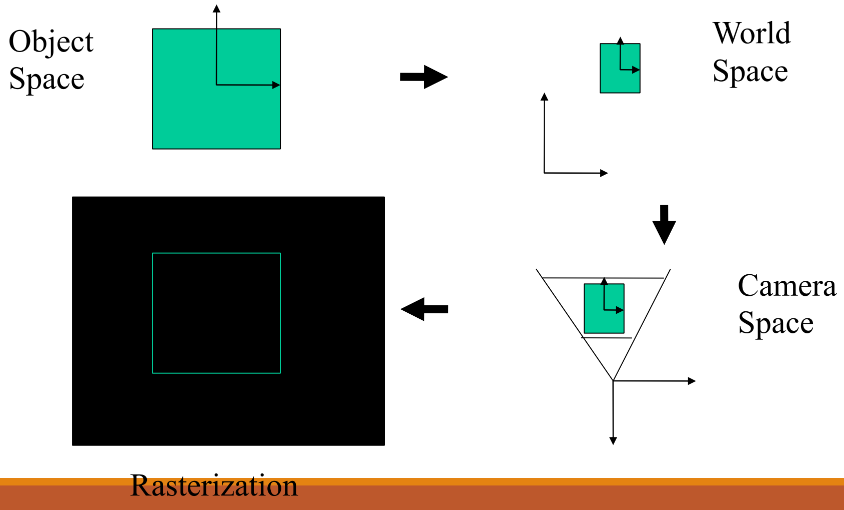
Take your World Space point and transform it to Camera Space

Perform the remapping and projection onto the image plane in Normalized Device Coordinates

Perform this set of transformations on each point of the polygonal object

"Connect the dots" through line rasterization

Intuitively

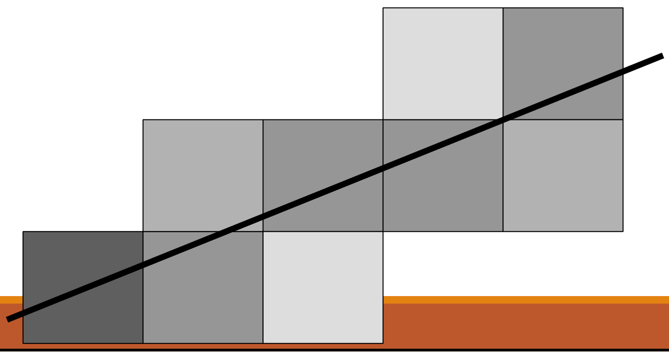


Lab05

Visualize your result!

demo: (due 4/12 midnight)

- Upload your code only (not your entire project)



Next: Rasterizing Polygons

Given a set of vertices and edges,
find the pixels that fill the polygon.