CSC 418/2504: Computer Graphics

Course web site (includes course information sheet):

`http://www.dgp.toronto.edu/~elf`

Instructor: Eugene Fiume

Office:	BA 5266
Phone:	416 978-5472 (not a reliable way)
Email:	elf@dgp.toronto.edu
Hours:	by arrangement, first use bulletin board

Textbooks: Fundamentals of Computer Graphics (recommended)
OpenGL Programming Guide & Reference (optional)

Tutorial lectures: Wednesdays @ 4 (first tutorial this week)

Today's Topics

0. Introduction: What is Computer Graphics?

1. Basics of scan conversion (line drawing)

2. Representing 2D curves

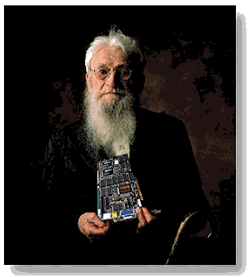
Topic 0.

Introduction:
What Is Computer Graphics?

Computer Graphics: NOT!

"How can I use Corel PhotoPaint™ to turn the sky green?"

Computer Graphics: NOT!



Champion TUROK©: DINOSAUR HUNTER player, "Gramps," recommends Intense 3D Voodoo

Computer Graphics

The science of turning the rules of geometry and physics into (digital) pictures that mean something to people

Computer Graphics

The science of turning the rules of geometry and physics into (digital) pictures that mean something to people

Technology for generation of visual media (images & digital video) with control over style, appearance, realism, motion, …

Key Elements:

- modeling objects & scenes, animation, rendering
- algorithms & data structures
- interface design & programming
- mathematics, physics, optics, psychophysics

CG is Movies

Movies define directions in CG
Set quality standards
Driving medium for CG

Games

Games emphasize the interactivity and AI
Push CG hardware to the limits (for real time performance)

Industrial Design

Costly to build physical prototypes
Often CG models are easier & cheaper alternative

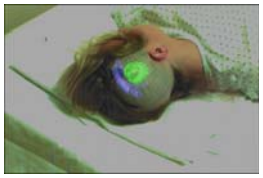
Requires precision modeling
Interactive engineering visualization

Scientific Visualization

Requires handling large datasets
May need device integration

Medical Imaging, Computer-Assisted Therapy

Requires handling large datasets
May need device integration
Real-time interactive modeling & visualization

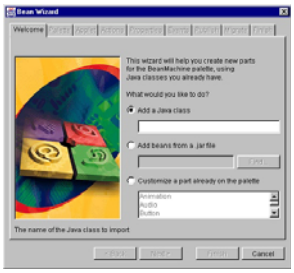


Graphical User Interfaces

Interaction with software & hardware

Emphasis on usability

Typically simpler (need to deal with simple 2D objects)



Computer Graphics: Basic Questions

- Form
  - How do we represent (2D or 3D) objects & environments?
  - How do we build these representations?
- Appearance
  - How do we represent the appearance of objects?
  - How do we simulate the image-forming process?
- Behaviour
  - How do we represent the way objects move?
  - How do we define & control their motion?

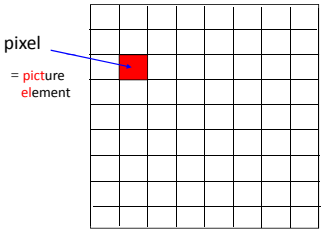
What is an Image?

Image = distribution of light energy E(x,y,λ,t) on 2D "film"

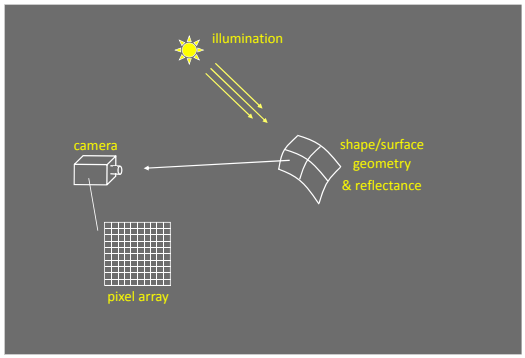
What is an Image?

Image = distribution of light energy E(x,y,λ,t) on 2D "film"

Digital images represented as rectangular arrays of pixels



Form & Appearance in CG



The Graphics Pipeline

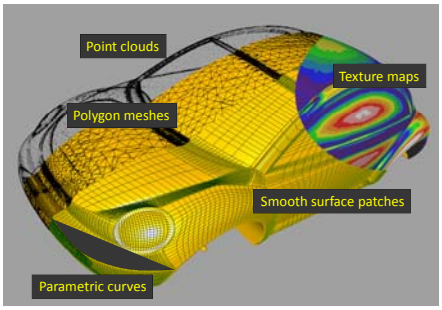
- Geometry: points, curves, & surfaces
- Scene Objects: parts, relations, & pose
- Texture and reflectance (e.g., color, diffusivity, opacity, refractions)
- …

- Key-frame, motion capture, inverse kinematics, dynamics, behaviors, motion planning, …

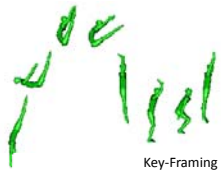
- Visibility
- Simulation of light (e.g., illuminants, emissive surfaces, scattering, transmission, diffraction, …)
- Special effects (e.g., anti-aliasing, motion blur, non-photorealism)

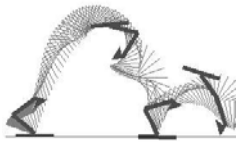
Graphics Pipeline: Modeling

How do we represent an object geometrically on a computer?



Graphics Pipeline: Animation





Graphics Pipeline: Rendering

Input: Scene description, lighting, camera

Output: Image that the camera will observe
Must consider visibility, clipping, scan conversion, projection, textures, …

Putting It All Together…

Course Topics

Principles

Theoretical & practical foundations of CG
(core mathematics, physics, modeling methods)

CG programming (assignments & tutorials)

- Experience with OpenGL (industry-standard CG library)
- Creating CG scenes

Structure of the Course

Part 1: Basic graphics primitives (≈ 3 weeks)

Part 2: Viewing in 3D (≈ 2 weeks)

Part 3: Appearance modeling & rendering (≈ 4 weeks)

Part 4: Interpolation (≈ 2 weeks)

Part 5: Animation (≈ 1 week)

What You Will Take Away …

#1: yes, math IS useful in CS !!

#2: how to turn math & physics into pictures

#3: basics of image synthesis

#4: how to code CG tools

Administrivia

Grading:

- 50%: 4 assignments handed out usually on Mondays (probably equally weighted, but it might change a bit).
- 50%: 1 test in class (15%) + 1 final exam (35%)
- First assignment: on web Wednesday (I hope)
- Check web for schedule, dates, more details & policy on late assignments

Tutorial sessions:

- Math refreshers, OpenGL tutorials, additional topics
- Attendance strongly encouraged since I will not be lecturing on these topics in class

Lecture slides & course notes: on web, after class

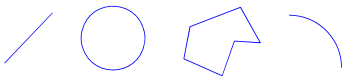
Topic 1.

Basic Raster Operations: Line Drawing

- A simple (but inefficient) line drawing algorithm
- Bresenham's algorithm
- Line anti-aliasing

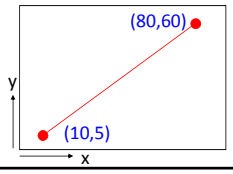
2D Drawing

Common geometric primitives:



When drawing a picture, 2D geometric primitives are specified as if they are drawn on a continuous plane

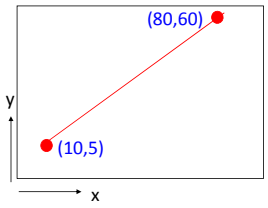
Drawing command:
Draw a line from point (10,5) to point (80,60)



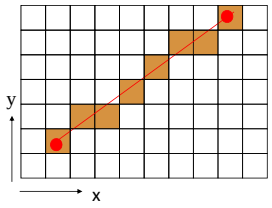
2D Drawing

In reality, computer displays are arrays of pixels, not abstract mathematical continuous planes

Continuous line



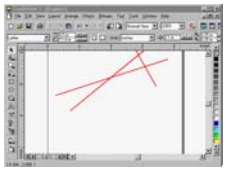
Digital line



In graphics, the conversion from continuous to discrete 2D primitives is called scan conversion or rasterization

Basic Raster Operations (for 2D lines)

1. Scan conversion
   Given a pair of pixels defining the line's endpoints & a color, paint all pixels that lie on the line
2. Clipping
   If one or more endpoints is out of bounds, paint only the line segment that is within bounds
3. Region filling



Line Scan Conversion: Key Objectives

Accuracy

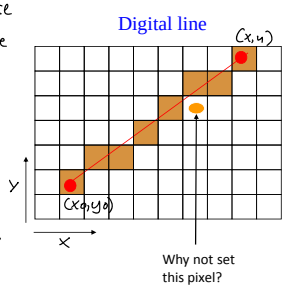
Pixels should approximate line as closely as possible

Speed

Line drawing should be as efficient as possible

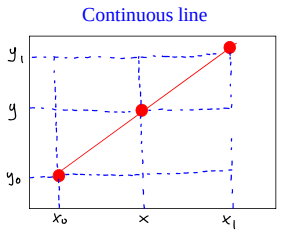
Visual Quality

No discernible "artifacts"



Line Equations

Q: How do we represent the set of all points lying on single line?



Equation of line that passes through points (x_0,y_0) and (x_1,y_1):

$$\frac{y-y_0}{y_1-y_0} = \frac{x-x_0}{x_1-x_0} \iff$$

$$y-y_0 = \frac{x-x_0}{x_1-x_0}(y_1-y_0) \iff$$

$$y = \frac{y_1-y_0}{x_1-x_0}(x-x_0)+y_0 \iff$$

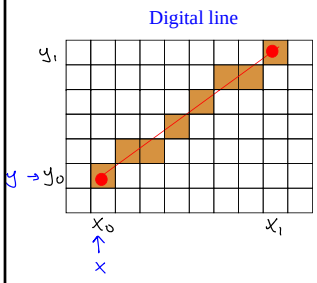
we call this the slope m

$y = mx + b$, $b = y_0 - mx_0$

$y = m(x-x_0)+y_0$

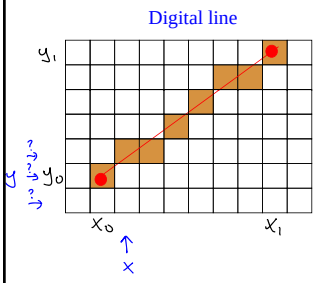
Line Scan Conversion: A Simple Algorithm

Goal: Determine which pixels lie closest to the mathematical line segment $y = mx+b$



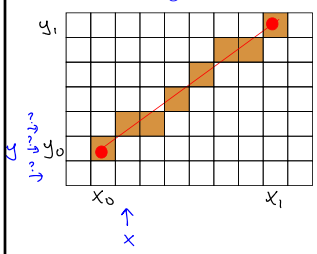
Case A: $0 \le \text{slope} < 1$

- Draw pixel at line start:
  `x=x0; y=y0;`
  `setpixel(x,y,color);`



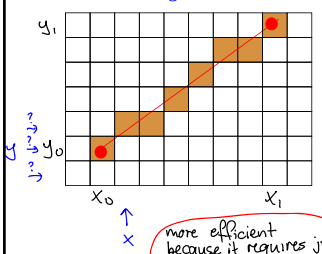
Case A: $0 \le \text{slope} < 1$

- Draw pixel at line start:
  `x=x0; y=y0;`
  `setpixel(x,y,color)`
- Increment x position:
  `x = x+1;`
- Determine y position of pixel in column x that lies closest to line



Case A: $0 \le \text{slope} < 1$

- Draw pixel at line start:
  `x=x0; y=y0;`
  `setpixel(x,y,color)`
- Increment x position:
  `x = x+1;`
- Update y position:
  `y = mx+b;`



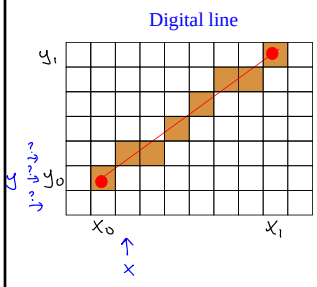
Case A: $0 \le \text{slope} < 1$

- Draw pixel at line start:
  `x=x0; y=y0;`
  `setpixel(x,y,color)`
- Increment x position:
  `x = x+1;`
- Update y position:
  `y = y+m;`

more efficient because it requires just 1 addition per pixel

Simple Line Scan Conversion (0 ≤ slope < 1)

Goal: Determine which pixels lie closest to the mathematical line segment $y = mx+b$



Case A: $0 \le slope < 1$

$m = \frac{y_1 - y_0}{x_1 - x_0}$;

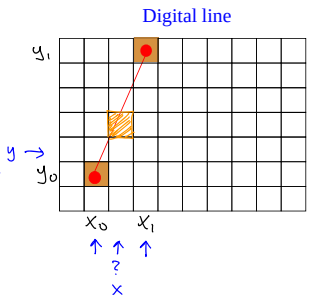
for $(x = x_0; x \le x_1; x{+}{+}, y{+}{=}m)$

setpixel(x, round(y), color);

if m is floating pont, so will y

Simple Line Scan Conversion (slope > 1)

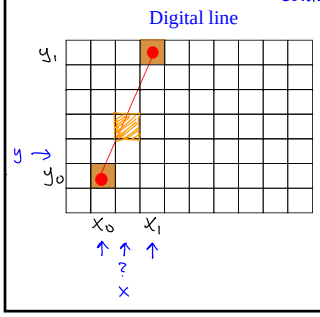
Q: What happens if slope > 1 ?
(i.e. $x_1 - x_0 < y_1 - y_0$)



Simple Line Scan Conversion (slope > 1)

Q: What happens if slope > 1 ?
(i.e. $x_1 - x_0 < y_1 - y_0$)

Ans: Line contains sparse pixels
Solution: swap roles of x and y



Case B: slope > 1

$m' = \frac{x_1 - x_0}{y_1 - y_0}$;

for $(y = y_0; y \le y_1; y{+}{+}, x{+}{=}m')$

setpixel(round(x), y, color);

if m' is floating pont, so will x

The Impact of Floating Point Operations

Two major problems:

- Inefficiency ← Is this true?
  (esp. when drawing millions of lines)
- Accumulation of round-off errors:
  Example: if m is rounded to 0.9 even though it is 0.99, lines of length > 10 will be drawn inaccurately

$m = \frac{y_1 - y_0}{x_1 - x_0}$;

for $(x = x_0; x \le x_1; x{+}{+}, y{+}{=}m)$

setpixel(x, round(y), color)

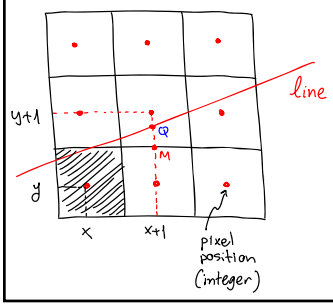
Topic 1.

Basic Raster Operations: Line Drawing

- A simple (but inefficient) line drawing algorithm
- Bresenham's algorithm
- Line anti-aliasing

Bresenham's Algorithm (integral endpoints)

Goal: Updates pixel position w/out using floating point!

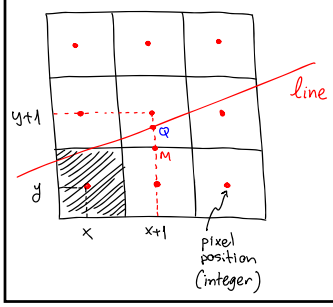


Basic idea. ($0 \le slope < 1$)

- Assume pixel (x,y) has been drawn
- If midpoint M is below Q then
  setpixel(x+1, y+1)
  else
  setpixel(x+1, y)

Bresenham's Algorithm (0 ≤ slope < 1)

Derivation: Implementing the midpoint test using integer ops



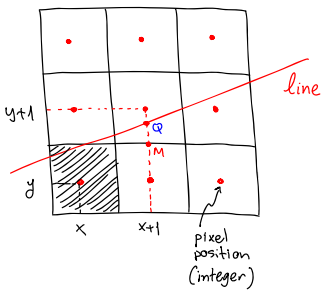
- $y_Q - y_M > 0$

⇕

- If midpoint M is below Q then setpixel$(x+1, y+1)$ else setpixel$(x+1, y)$

Bresenham's Algorithm (0 ≤ slope < 1)

Derivation: Implementing the midpoint test using integer ops



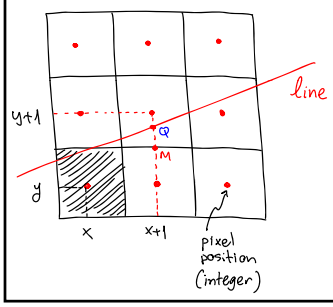
- $y_Q - y_M > 0$

⇕

- If midpoint M is below Q then setpixel$(x+1, y+1)$ else setpixel$(x+1, y)$

Bresenham's Algorithm (0 ≤ slope < 1)

Derivation: Implementing the midpoint test using integer ops



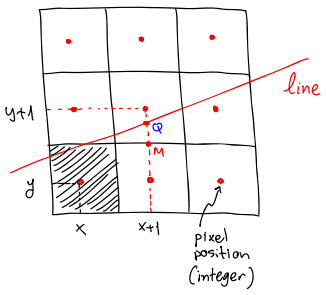
$y_Q - y_M > 0 \Leftrightarrow$

Q is on the line: $m(x+1-x_0)+y_0$, $y + \frac{1}{2}$

$m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{H}{W}$

Bresenham's Algorithm (0 ≤ slope < 1)

Derivation: Implementing the midpoint test using integer ops



$y_Q - y_M > 0 \Leftrightarrow$

Q is on the line: $\frac{H}{W}(x+1-x_0)+y_0$, $y + \frac{1}{2}$

Bresenham's Algorithm (0 ≤ slope < 1)

Derivation: Implementing the midpoint test using integer ops

$\frac{H}{W}(x+1-x_0)+y_0 - y - \frac{1}{2} > 0 \Leftrightarrow$

$y_Q - y_M > 0 \Leftrightarrow$

Q is on the line: $\frac{H}{W}(x+1-x_0)+y_0$, $y + \frac{1}{2}$

Bresenham's Algorithm (0 ≤ slope < 1)

Derivation: Implementing the midpoint test using integer ops

$\frac{H}{W}(x+1-x_0)+y_0 - y - \frac{1}{2} > 0 \Leftrightarrow$

$2H(x+1-x_0) + 2W(y_0-y) - W > 0 \Leftrightarrow$

$2H(x-x_0) - 2W(y-y_0) + 2H - W > 0$

$y_Q - y_M > 0 \Leftrightarrow$

Q is on the line: $\frac{H}{W}(x+1-x_0)+y_0$, $y + \frac{1}{2}$

Bresenham's Algorithm (0 ≤ slope < 1)

Derivation: Implementing the midpoint test using integer ops

$\frac{H}{W}(x+1-x_0)+y_0-y-\frac{1}{2}>0 \Leftrightarrow$

$2H(x+1-x_0)+2W(y_0-y)-W>0 \Leftrightarrow$

$2H(x-x_0)-2W(y-y_0)+2H-W>0$

↑ define a function $f(x,y)$ equal to this expression

Properties of f:

* $f(x,y)$ requires only integer ops to compute
* $f(x,y)+2H-W = f(x+1, y+\frac{1}{2})$

$y_Q - y_M > 0 \Leftrightarrow$

Q is on the line

$y + \frac{1}{2}$

$\frac{H}{W}(x+1-x_0)+y_0$

Bresenham's Algorithm (0 ≤ slope < 1)

Derivation: Implementing the midpoint test using integer ops

$\frac{H}{W}(x+1-x_0)+y_0-y-\frac{1}{2}>0 \Leftrightarrow$

$2H(x+1-x_0)+2W(y_0-y)-W>0 \Leftrightarrow$

$2H(x-x_0)-2W(y-y_0)+2H-W>0$

↑ define a function $f(x,y)$ equal to this expression

Properties of f:

* $f(x,y)$ requires only integer ops to compute
* $f(x,y)+2H-W = f(x+1, y+\frac{1}{2})$

$y_Q - y_M > 0 \Leftrightarrow$

$f(x+1, y+1/2) + 2H - W > 0$

requires only integer ops to compute!

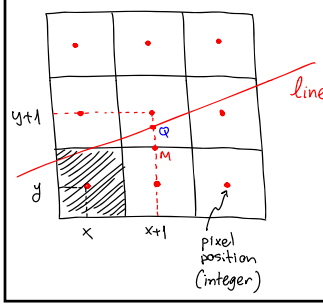
where

$f(x,y) = 2H(x-x_0) - 2W(y-y_0)$

$H = y_1 - y_0 \quad W = x_1 - x_0$

Bresenham's Algorithm (0 ≤ slope < 1)

Geometric interpretation of function f



$y_Q - y_M > 0 \Leftrightarrow$

$f(x+1, y+1/2) + 2H - W > 0 \Leftrightarrow$

$f(x\_m, y\_m) + 2H - W > 0$

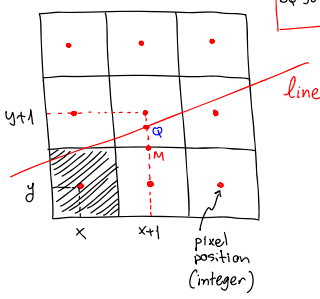
where

$f(x,y) = 2H(x-x_0) - 2W(y-y_0)$

$H = y_1 - y_0 \quad W = x_1 - x_0$

Bresenham's Algorithm (0 ≤ slope < 1)

Geometric interpretation of function f (aka implicit function for the line)



$y_Q - y_0 = \frac{H}{W}(x_Q - x_0) \Leftrightarrow H(x_Q - x_0) - W(y_Q - y_0) = 0$

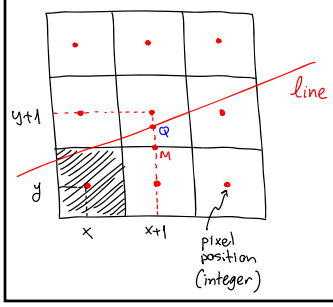
$\Leftrightarrow \frac{1}{2} f(x_Q, y_Q) = 0$

Given the coordinates (x_M, y_M) of a point M,

- $f(x_M, y_M) = 0 \Leftrightarrow$ M is on line
- $f(x_M, y_M) > 0 \Leftrightarrow$ M above line
- $f(x_M, y_M) < 0 \Leftrightarrow$ M below line

Bresenham's Algorithm (0 ≤ slope < 1)

Putting the midpoint test to use:



- Assume pixel (x,y) has been drawn

if $f(x+1, y+1/2) + 2H - W > 0$

$x = x+1 \; ; \; y = y+1$

else

$x = x+1$;

setpixel $(x, y,$ color$)$

Bresenham's Algorithm (0 ≤ slope < 1)

Computing f incrementally (to minimize # integer ops)

From the definition of f:

$f(x,y) = 2H(x-x_0) - 2W(y-y_0)$

it follows that

$f(x+1, y+1) = f(x,y) + 2H - 2W$ ← y changes

$f(x+1, y) = f(x,y) + 2H$ ← y stays the same

- Assume pixel (x,y) has been drawn

if $f(x+1, y+1/2) + 2H - W > 0$

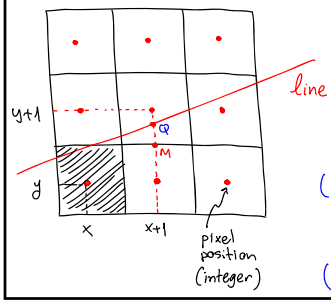
$x = x+1 \; ; \; y = y+1$

else

$x = x+1$;

setpixel $(x, y,$ color$)$

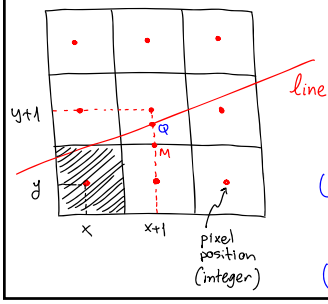
Bresenham's Algorithm (0 ≤ slope < 1)



Complete algorithm

```
y=y0; H=y1-y0; W=x1-x0;
f=0;
for (x=x0; x≤x1; x++)
    setpixel(x,y)
    if (f+2H-W)>0
(y changes)     y=y+1;
                f=f+2H-2W;
    else
(y stays the same)  f=f+2H;
```

Bresenham's Algorithm (0 ≤ slope < 1)



Complete algorithm (eliminates addition/subtraction in midpoint test)

```
y=y0; H=y1-y0; W=x1-x0;
f=2H-W;
for (x=x0; x≤x1; x++)
    setpixel(x,y)
    if  f>0
(y changes)     y=y+1;
                f=f+2H-2W;
    else
(y stays the same)  f=f+2H;
```

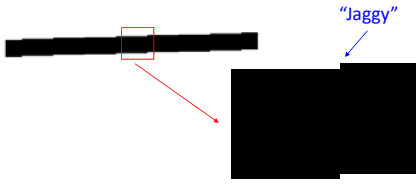
Topic 1.

Basic Raster Operations: Line Drawing

- A simple (but inefficient) line drawing algorithm
- Bresenham's algorithm
- Line anti-aliasing

Jaggies

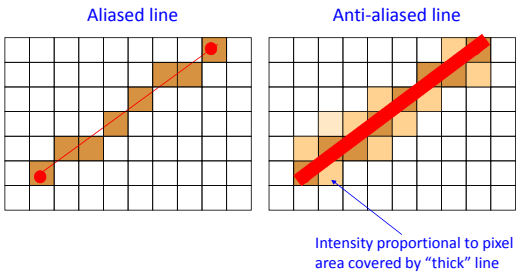
An unwanted artifact of the line drawing algorithms already discussed is that the lines generated have a "jaggy" appearance



- Jaggies are an instance of a phenomenon called aliasing
- Removal of these artifacts is achieved with the help of anti-aliasing techniques

Anti-Aliasing

How can we make a digital line appear less jaggy?



Main idea: Rather than just drawing in 0's and 1's, use "in-between" values in neighborhood of the mathematical line

Anti-Aliasing: Example

Topic 2.

2D Curve Representations

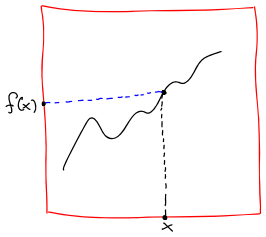
- Explicit representation
- Parametric representation
- Tangent & normal vectors
- Implicit representation

Explicit Curve Representations: Definition

Curve represented by a function f such that

$$y = f(x)$$

i.e. given x (abscissa) $f()$ gives us y (the ordinate)

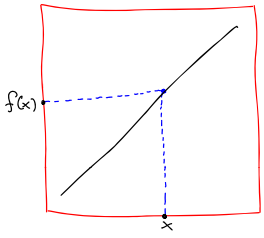


Example: Explicit Representation of a 2D Line

Curve represented by a function f such that

$$y = mx + b$$

i.e. given x (abscissa) $f()$ gives us y (the ordinate)

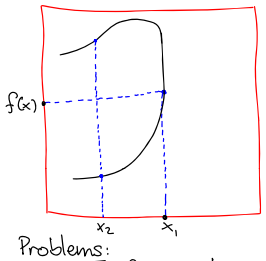


Explicit Curve Representations: Limitations

Curve represented by a function f such that

$$y = f(x)$$

i.e. given x (abscissa) $f()$ gives us y (the ordinate)



Problems:
* What if curve becomes vertical?
* What if curve contains pts with same x coord?

Topic 2.

2D Curve Representations

- Explicit representation
- Parametric representation
- Tangent & normal vectors
- Implicit representation

Parametric Curve Representation: Definition

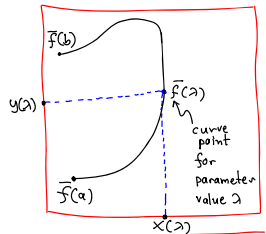
Curve represented by
- two functions $x(), y()$
- an interval (a,b)

such that every point

$$\bar{f}(\lambda) = (x(\lambda), y(\lambda))$$

belongs on the curve for $\lambda \in (a,b)$

* The functions $x()$ and $y()$ are called the coordinate functions of the curve



Formally, the curve is a vector-valued function

$$\bar{f}: (a,b) \subseteq \mathbb{R} \to \mathbb{R}^2, \quad \lambda \mapsto \bar{f}(\lambda)$$

Parametric Representation: Closed Curves

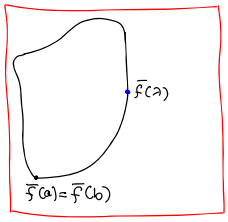
Curve represented by
- two functions $x(\lambda), y(\lambda)$
- an interval (a,b)

such that every point

$$\bar{f}(\lambda) = (x(\lambda), y(\lambda))$$

belongs on the curve for $\lambda \in (a,b)$

A curve is closed if $\bar{f}(a) = \bar{f}(b)$



Formally, the curve is a vector-valued function

$\bar{f}: (a,b) \subseteq \mathbb{R} \rightarrow \mathbb{R}^2$

$\lambda \longmapsto \bar{f}(\lambda)$

Parametric Representation: Digital Curves

Example: A general, digital 2D curve that is N pixels long

$\bar{f}: (1, N) \rightarrow \mathbb{R}^2$

(value of parameter at 1st pixel; value of parameter at end)

$\lambda \longrightarrow (x(\lambda), y(\lambda))$

(λ^{th} point along the curve)

Data structures:
- x[]: N-element matrix holding x-coordinates of curve pts
- y[]: holds y-coordinates
- λ: the array index

Parametric Representation: Smooth Curves

Curve represented by
- two functions $x(\lambda), y(\lambda)$
- an interval (a,b)

such that every point

$$\bar{f}(\lambda) = (x(\lambda), y(\lambda))$$

belongs on the curve for $\lambda \in (a,b)$

Simple geometric objects (lines, circles, ellipses, etc) can be represented much more compactly using analytic expressions for $x(\lambda)$ and $y(\lambda)$

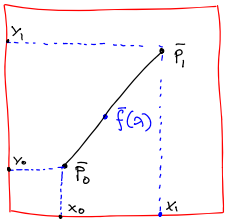
A curve is smooth if $x(\lambda), y(\lambda)$ have continuous derivatives

Parametric Representation of a Line Segment

Line segment from point $\bar{P}_0$ to point $\bar{P}_1$:

$$\bar{f}(\lambda) = \bar{P}_0 + \lambda(\bar{P}_1 - \bar{P}_0)$$

with $0 \le \lambda \le 1$



To get the coordinate functions, expand:

$$\bar{f}(\lambda) = (x_0, y_0) + \lambda(x_1 - x_0, y_1 - y_0)$$

Generalizations:
- If $0 \le \lambda < \infty$ $\longrightarrow$ Ray from $\bar{P}_0$ in direction of $\bar{P}_1$
- If $-\infty < \lambda < \infty$ $\longrightarrow$ Line through $\bar{P}_0$ and $\bar{P}_1$

Parametric Representation of a Circle

- Circle centered at (0,0) with unit radius:

$$\bar{f}(\vartheta) = (\cos\vartheta, \sin\vartheta)$$

with $0 \le \vartheta \le 2\pi$

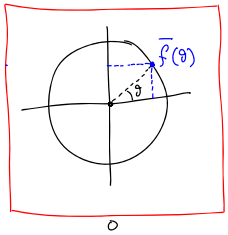
- With radius r:

$$\bar{f}(\vartheta) = (r\cos\vartheta, r\sin\vartheta)$$

- With a parameter $0 \le \lambda \le 1$:

$$\bar{f}(\lambda) = (r\cos(2\pi\lambda), r\sin(2\pi\lambda))$$

$\Rightarrow$ easy to generate points along circle by sampling λ values

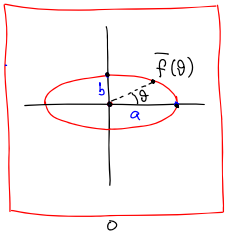


Parametric Representation of an Ellipse

- Ellipse centered at (0,0) with major & minor axes equal to a and b

$$\bar{f}(\vartheta) = (a\cos\vartheta, b\sin\vartheta)$$

with $0 \le \vartheta \le 2\pi$

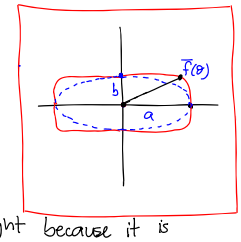


Super-Ellipses: Definition

- Super-ellipse at (0,0) with major & minor axes equal to a and b

$$\left(a(\cos\vartheta)^{\frac{2}{n}},\ b(\sin\vartheta)^{\frac{2}{n}} \right)$$

the parameter $n \in \mathbb{R}$ is called "squareness" and is always positive



- Above expression not quite right because it is undefined for some n (e.g. when n=4, cosϑ<0)
  Solution: do not exponentiate the sign

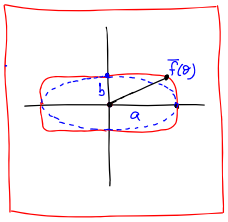
$$\mathrm{sgn}(x) = \begin{cases} 1 & x>0 \\ 0 & x=0 \\ -1 & x<0 \end{cases}$$

$$\bar{f}(\vartheta) = \left(a\cdot \mathrm{sgn}(\cos\vartheta)|\cos\vartheta|^{\frac{2}{n}},\ b\,\mathrm{sgn}(\sin\vartheta)|\sin\vartheta|^{\frac{2}{n}} \right)$$

Super-Ellipses: The "Squareness" Parameter

Effect of changing n:

$n=2 \longrightarrow$ ellipse
$n>2 \longrightarrow$ increasingly "squared" ellipse



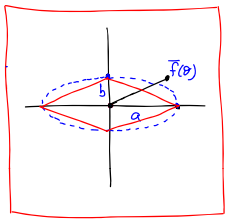
$$\mathrm{sgn}(x) = \begin{cases} 1 & x>0 \\ 0 & x=0 \\ -1 & x<0 \end{cases}$$

$$\bar{f}(\vartheta) = \left(a\cdot \mathrm{sgn}(\cos\vartheta)|\cos\vartheta|^{\frac{2}{n}},\ b\,\mathrm{sgn}(\sin\vartheta)|\sin\vartheta|^{\frac{2}{n}} \right)$$

Super-Ellipses: The "Squareness" Parameter

Effect of changing n:

$n=2 \longrightarrow$ ellipse
$n>2 \longrightarrow$ increasingly "squared" ellipse
$n=1 \longrightarrow$ diamond



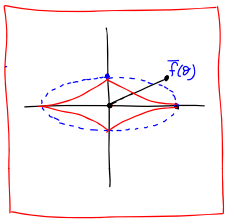
$$\mathrm{sgn}(x) = \begin{cases} 1 & x>0 \\ 0 & x=0 \\ -1 & x<0 \end{cases}$$

$$\bar{f}(\vartheta) = \left(a\cdot \mathrm{sgn}(\cos\vartheta)|\cos\vartheta|^{\frac{2}{n}},\ b\,\mathrm{sgn}(\sin\vartheta)|\sin\vartheta|^{\frac{2}{n}} \right)$$

Super-Ellipses: The "Squareness" Parameter

Effect of changing n:

$n=2 \longrightarrow$ ellipse
$n>2 \longrightarrow$ increasingly "squared" ellipse
$n=1 \longrightarrow$ diamond
$n<1 \longrightarrow$ "inward-bent" diamond
$n \to \infty \longrightarrow$?
$n \to 0 \longrightarrow$?



$$\mathrm{sgn}(x) = \begin{cases} 1 & x>0 \\ 0 & x=0 \\ -1 & x<0 \end{cases}$$

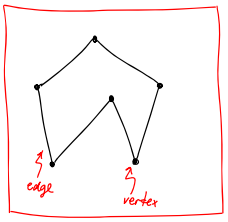
$$\bar{f}(\vartheta) = \left(a\cdot \mathrm{sgn}(\cos\vartheta)|\cos\vartheta|^{\frac{2}{n}},\ b\,\mathrm{sgn}(\sin\vartheta)|\sin\vartheta|^{\frac{2}{n}} \right)$$

Parametric Representation of a Polygon

Polygon:

A continuous, piecewise-linear, closed planar curve

- Simple polygon: non-self-intersecting
- Regular polygon: simple, equilateral, equiangular
- n-gon: regular polygon with n sides



To find its vertices, sample n equispaced points on a circle:

$$(x_i, y_i) = r\left(\cos\frac{2\pi}{n}i,\ \sin\frac{2\pi}{n}i\right)$$

- To translate: add (x_t, y_t) to each pt
- To rotate: add $\Delta\vartheta$ to each $\frac{2\pi}{n}i$

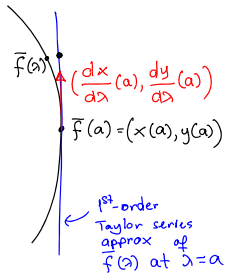
often used to approx a circle (with n=25–40) but better to extend Bresenham to circle-drawing

Topic 2.

2D Curve Representations

- Explicit representation
- Parametric representation
- Tangent & normal vectors
- Implicit representation

The Tangent Vector



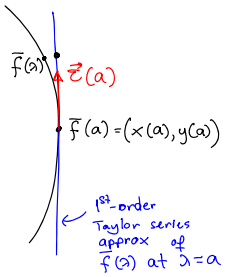
1st order Taylor series approx of $\vec{f}(\lambda)$ near $\lambda = a$

$$\vec{f}(\lambda) = (x(\lambda), y(\lambda))$$
$$\simeq \left(x(a) + (\lambda - a)\frac{dx}{d\lambda}(a),\ y(a) + (\lambda - a)\frac{dy}{d\lambda}(a)\right)$$
(point)
$$= (x(a), y(a)) + (\lambda - a)\left(\frac{dx}{d\lambda}(a), \frac{dy}{d\lambda}(a)\right)$$
(tangent vector)

Definition (Tangent vector $\vec{t}(\lambda)$)

$$\vec{t}(\lambda) = \frac{d\vec{f}}{d\lambda}(\lambda) = \left(\frac{dx}{d\lambda}(\lambda), \frac{dy}{d\lambda}(\lambda)\right)$$

Tangent Directions: Key Property



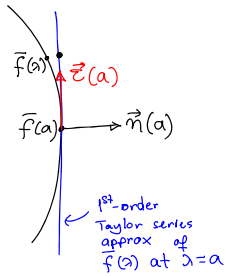
- We can parameterize a curve in many different ways (i.e. using various functions $\vec{f}$).
- BUT: regardless of the parameterization, the direction of the tangent remains unchanged

Key property of the tangent!

Definition (Tangent vector $\vec{t}(\lambda)$)

$$\vec{t}(\lambda) = \frac{d\vec{f}}{d\lambda}(\lambda) = \left(\frac{dx}{d\lambda}(\lambda), \frac{dy}{d\lambda}(\lambda)\right)$$

The Normal Vector



Definition (Normal vector)

$\vec{n}(\lambda)$ = vector perpendicular to $\vec{t}(\lambda)$

$$= \left(-\frac{dy}{d\lambda}(\lambda), \frac{dx}{d\lambda}(\lambda)\right)$$

* By definition, $\vec{n}(\lambda) \cdot \vec{t}(\lambda) = 0$
* We often refer to the unit tangent & unit normal, computed by dividing $\vec{t}(\lambda), \vec{n}(\lambda)$ by their length

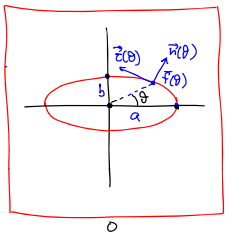
Definition (Tangent vector $\vec{t}(\lambda)$)

$$\vec{t}(\lambda) = \left(\frac{dx}{d\lambda}(\lambda), \frac{dy}{d\lambda}(\lambda)\right)$$

Tangent & Normal Vectors: Example

- Ellipse centered at (0,0) with major & minor axes equal to a and b

$$\vec{f}(\theta) = (a\cos\theta, b\sin\theta)$$
with $0 \le \theta \le 2\pi$



- Tangent at $\vec{f}(\theta)$:

$$\vec{t}(\theta) = \frac{d\vec{f}}{d\theta}(\theta) = (-a\sin\theta, b\cos\theta)$$

In-Class Lecture ended here

Topic 2.

2D Curve Representations

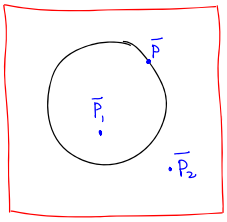
- Explicit representation
- Parametric representation
- Tangent & normal vectors
- Implicit representation

Implicit Curve Representation: Definition

A function $f(x,y)$ that is zero if and only if (x,y) is on the curve

$$f(\bar{p}) = 0$$

called the implicit equation of the curve



$f(\bar{p}) = 0$

$f(\bar{p}_1) \neq 0$

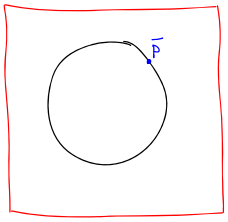
$f(\bar{p}_2) \neq 0$

Implicit Curve Representation: Definition

A function $f(x,y)$ that is zero if and only if (x,y) is on the curve

$$f(\bar{p}) = 0$$

called the implicit equation of the curve



Circle with radius r centered at $(0,0)$:

$f(x,y) = x^2 + y^2 - r^2$

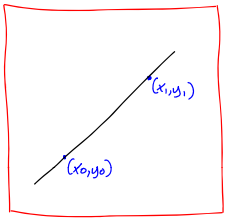
(because x,y must satisfy $x^2+y^2=r^2$)

Implicit Curve Representation: Definition

A function $f(x,y)$ that is zero if and only if (x,y) is on the curve

$$f(\bar{p}) = 0$$

called the implicit equation of the curve

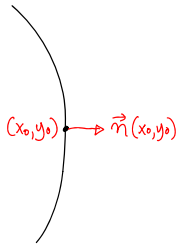


Line through (x_0,y_0) and (x_1,y_1)

$f(x,y) = (y-y_0)(x_1-x_0) - (y_1-y_0)(x-x_0)$

(because x,y must satisfy $\frac{y-y_0}{y_1-y_0} = \frac{x-x_0}{x_1-x_0}$)

Normal Vectors from the Implicit Equation



If $f(x,y)$ is the implicit eq of a curve and (x_0,y_0) is a point on it, then

$$\vec{n}(x_0,y_0) = \nabla f(x_0,y_0)$$

where $\nabla f(x_0,y_0)$ is the gradient of f at point (x_0,y_0), i.e.

$$\nabla f(x_0,y_0) = \left(\frac{\partial f}{\partial x}(x_0), \frac{\partial f}{\partial y}(y_0) \right)$$

Normal Vectors from the Implicit Equation

Derivation.

- Let $(x(\lambda), y(\lambda))$ be the parametric rep
- Then

$f(x(\lambda), y(\lambda)) = 0 \ \forall \lambda \Leftrightarrow$

$\frac{d}{d\lambda} f(x(\lambda), y(\lambda)) = 0 \Leftrightarrow$

$\frac{\partial f}{\partial x} \cdot \frac{dx}{d\lambda} + \frac{\partial f}{\partial y} \cdot \frac{dy}{d\lambda} = 0 \Leftrightarrow$

$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \cdot \left(\frac{dx}{d\lambda}, \frac{dy}{d\lambda} \right) \Leftrightarrow$ ($\vec{r}(\lambda)$)

$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \perp$ tangent

Q.E.D.

If $f(x,y)$ is the implicit eq of a curve and (x_0,y_0) is a point on it, then

$$\vec{n}(x_0,y_0) = \nabla f(x_0,y_0)$$

where $\nabla f(x_0,y_0)$ is the gradient of f at point (x_0,y_0), i.e.

$$\nabla f(x_0,y_0) = \left(\frac{\partial f}{\partial x}(x_0), \frac{\partial f}{\partial y}(y_0) \right)$$

Lecture 1 ends here

Lecture 1 extras

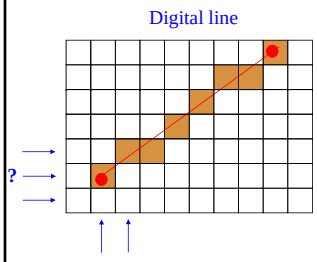
Topic 1.

Basic Raster Operations: Line Drawing

•Hidden slides

Drawing Circles

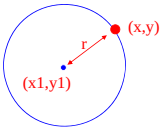
The basic principle behind Bresenham's algorithm can be applied to the problem of drawing circles, ellipses, conic sections, etc



1. Draw pixel at a circle point
2. Increment x pixel position
3. Determine the y position of the pixel lying closest to the circle

Circle Equation

A circle is completely defined by its center & radius

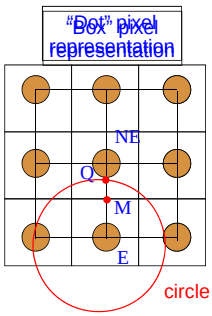


The equation of the circle is given by

$$(x-x1)^2 + (y-y1)^2 = r^2$$

Circle Drawing: The Midpoint Algorithm

Given: the center pixel (x1, y1) and radius of a circle



Divide circle into octants
Basic idea (top-right octant)

- Test if M is inside or outside the circle
- If M inside, draw NE
- If M outside, draw E
- How do we check whether a point is inside or outside a circle?
- Ans: Distance from center < r
- Perform test using integer ops

Today's Topics

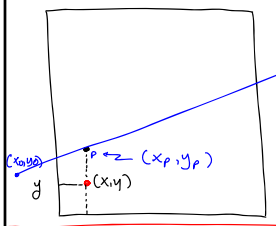
Drawing digital lines & other curves

- DDA line drawing algorithm
- Bresenham's algorithm
- Midpoint algorithm for drawing circles
- **Anti-aliasing techniques**

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le \text{slope} < 1$)



Is (x,y) below P ?

Then $g(x_P,y_P)=0$

satisfied by any point P on the line

Define (Implicit Line function)

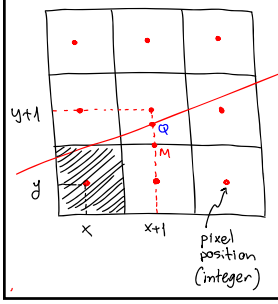
$g(x,y) = H(x-x_0) - W(y-y_0)$

$H(x_P-x_0) - W(y_P-y_0) = 0$

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le \text{slope} < 1$)



Is $M = (x+1, y+\frac{1}{2})$ below Q ?

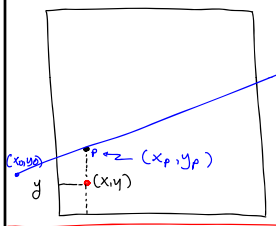
If $2f(x,y) + 2H - W > 0$ then M below Q else M above Q

$f(x,y) = H(x-x_0) - W(y-y_0)$

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le \text{slope} < 1$)



Is (x,y) below P ?

$y_P = \frac{(y_1-y_0)}{(x_1-x_0)}(x_P-x_0) + y_0$ ($\frac{(y_1-y_0)}{(x_1-x_0)} = \frac{H}{W}$)

$\Leftrightarrow y_P = \frac{H}{W}(x_P-x_0) + y_0$

$\Leftrightarrow W y_P = H(x_P-x_0) + W y_0$

$\Leftrightarrow H(x_P-x_0) - W(y_P-y_0) = 0$

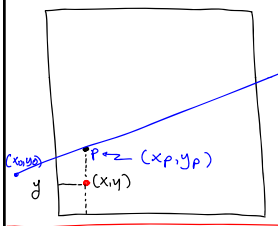
Define

$g(x,y) = H(x-x_0) - W(y-y_0)$

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le \text{slope} < 1$)



Is (x,y) below P ?

Then $g(x_P,y_P)=0$

satisfied by any point P on the line

If $g(x,y) > 0$ then (x,y) is below P

If $g(x,y) < 0$ then (x,y) is above P

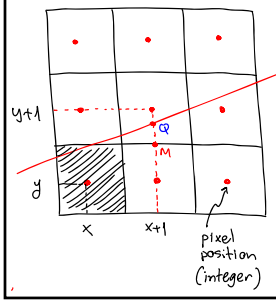
Define (Implicit Line function)

$g(x,y) = H(x-x_0) - W(y-y_0)$

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le \text{slope} < 1$)



Is $M = (x+1, y+\frac{1}{2})$ below Q ?

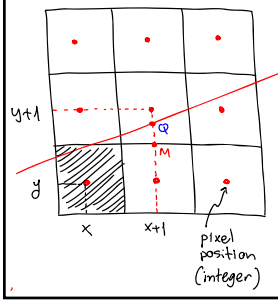
$g(x,y) = H(x-x_0) - W(y-y_0)$

If $g(x+1, y+\frac{1}{2}) > 0$ then M is below Q

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le \text{slope} < 1$)



Is $M = (x+1, y+\frac{1}{2})$ below Q ?

$g(x,y) = H(x-x_0) - W(y-y_0)$

$g(x+1, y+\frac{1}{2}) > 0 \Leftrightarrow$

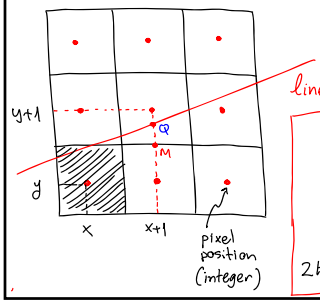
$H(x+1-x_0) - W(y+\frac{1}{2}-y_0) > 0 \Leftrightarrow$

$2H(x-x_0) + 2H - 2W(y-y_0) - W > 0$

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le slope < 1$)



Is $M = (x+1, y+\frac{1}{2})$ below Q ?

$g(x,y) = H(x-x_0) - W(y-y_0)$

$2g(x,y) + 2H - W > 0$

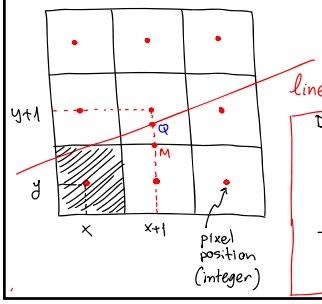
$\Updownarrow$

$2H(x-x_0) + 2H - 2W(y-y_0) - W > 0$

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le slope < 1$)



Is $M = (x+1, y+\frac{1}{2})$ below Q ?

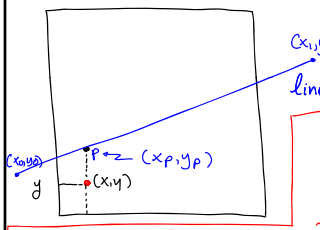
Define $f(x,y) = 2H(x-x_0) - 2W(y-y_0)$

If $f(x,y) + 2H - W > 0$ then M is below Q

Bresenham's Algorithm

Goal: Updates pixel position w/out using floating point

Basic Idea. ($0 \le slope < 1$)



Is (x,y) below P ?

Then $g(x_P, y_P) = 0$

satisfied by any point P on the line

If $g(x,y) > 0$ then (x,y) is below P

If $g(x,y) < 0$ then (x,y) is above P

Define (Implicit Line function)

$g(x,y) = H(x-x_0) - W(y-y_0)$