CS 430/536
Computer Graphics I

Line Drawing

Week 1, Lecture 2

David Breen, William Regli and Maxim Peysakhov
Geometric and Intelligent Computing Laboratory
Department of Computer Science
Drexel University
`http://gicl.cs.drexel.edu`

Outline

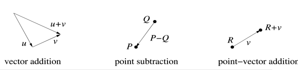
- Math refresher
- Line drawing
- Digital differential analyzer
- Bresenham's algorithm
- XPM file format

Geometric Preliminaries

- **Affine Geometry**
  - Scalars + Points + Vectors and their ops
- **Euclidian Geometry**
  - Affine Geometry lacks angles, distance
  - New op: Inner/Dot product, which gives
    - Length, distance, normalization
    - Angle, Orthogonality, Orthogonal projection
- **Projective Geometry**

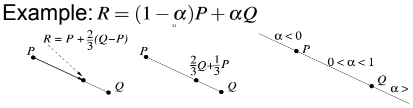
Affine Geometry

- *Affine Operations*:



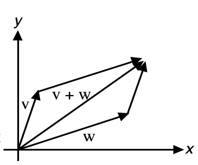
$vector \leftarrow scalar \cdot vector, \quad vector \leftarrow vector / scalar$	scalar-vector multiplication
$vector \leftarrow vector + vector, \quad vector \leftarrow vector - vector$	vector-vector addition
$vector \leftarrow point - point$	point-point difference
$point \leftarrow point + vector, \quad point \leftarrow point - vector$	point-vector addition

- *Affine Combinations*: $\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$
  where $v_1, v_2, \ldots, v_n$ are vectors and $\sum_i \alpha_i = 1$
  Example: $R = (1-\alpha)P + \alpha Q$



Mathematical Preliminaries

- Vector: an *n*-tuple of real numbers
- Vector Operations
  - Vector addition: $u + v = w$
    - Commutative, associative, identity element (0)
  - Scalar multiplication: cv
- Note: Vectors and Points are different
  - Can not add points
  - Can find the vector between two points



Linear Combinations & Dot Products

- A *linear combination* of the vectors
  $v_1, v_2, \ldots v_n$
  is any vector of the form
  $\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$
  where α_i is a real number (i.e. a scalar)

- *Dot Product*:

$$u \cdot v = \sum_{k=1}^{n} u_k v_k$$

a real value $u_1 v_1 + u_2 v_2 + \ldots + u_n v_n$ written as $u \bullet v$

Fun with Dot Products

- *Euclidian Distance* from *(x,y)* to *(0,0)*
  $\sqrt{x^2 + y^2}$ in general: $\sqrt{x_1^{\,2} + x_2^{\,2} + \ldots + x_n^{\,2}}$
  which is just: $\sqrt{\vec{x} \bullet \vec{x}}$
- This is also the length of vector $\underline{v}$:
  $||\underline{v}||$ or $|\underline{v}|$
- *Normalization* of a vector: $\hat{v} = \dfrac{\vec{v}}{|\vec{v}|}$.
- *Orthogonal* vectors: $\vec{u} \cdot \vec{v} = 0$

7

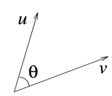
Projections & Angles

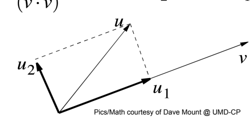
- *Angle between vectors,* θ $\quad \vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}|\cos(\theta)$

$$\theta = \operatorname{ang}(\vec{u}, \vec{v}) = \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}\right) = \cos^{-1}(\hat{u} \cdot \hat{v}).$$

- *Projection of vectors*

$$\vec{u}_1 = \frac{(\vec{u} \cdot \vec{v})}{(\vec{v} \cdot \vec{v})}\vec{v} \qquad \vec{u}_2 = \vec{u} - \vec{u}_1.$$





Pics/Math courtesy of Dave Mount @ UMD-CP

8

Matrices and Matrix Operators

- A *n*-dimensional vector: $\begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix}$
- Matrix Operations:
  - Addition/Subtraction
  - Identity
  - Multiplication
    - Scalar
    - Matrix Multiplication
- Implementation issue: Where does the index start? (0 or 1, it's up to you…)

$$A + B = B + A$$
$$A + (B + C) = (A + B) + C$$
$$(cd)A = c(dA)$$
$$1A = A$$
$$c(A + B) = cA + cB$$
$$(c + d)A = cA + dA$$

9

Matrix Multiplication

- [C] = [A][B]
- Sum over rows & columns
- Recall: matrix multiplication is *not* commutative
- *Identity Matrix*:
  1s on diagonal
  0s everywhere else

$$c_{ij} = \sum_{s=1}^{m} a_{is} b_{sj}$$

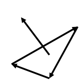
$$\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

10

Matrix Determinants

- A single real number
- Computed recursively $\quad \det(A) = \sum_{j=1}^{n} A_{i,j}(-1)^{i+j} M_{i,j}$
- Example: $\det\begin{bmatrix} a & c \\ b & d \end{bmatrix} = ad - bc$
- Uses:
  - Find vector ortho to two other vectors
  - Determine the plane of a polygon



11

Cross Product

- Given two non-parallel vectors, A and B
- A x B calculates third vector C that is orthogonal to A and B
- A x B = ($a_y b_z$ - $a_z b_y$, $a_z b_x$ - $a_x b_z$, $a_x b_y$ - $a_y b_x$)

$$A \times B = \begin{vmatrix} \vec{x} & \vec{y} & \vec{z} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

12

Matrix Transpose & Inverse

- *Matrix Transpose*: Swap rows and cols: $A = \begin{bmatrix} 2 \\ 8 \end{bmatrix}$ $A^T = \begin{bmatrix} 2 & 8 \end{bmatrix}$
- Facts about the transpose:
  $(A^T)^T = A$
  $(A+B)^T = A^T + B^T$
  $(cA)^T = c(A^T)$
  $(AB)^T = B^T A^T$
- *Matrix Inverse*: Given A, find B such that
  $AB = BA = I$ $\quad B \rightarrow A^{-1}$
  (only defined for square matrices)

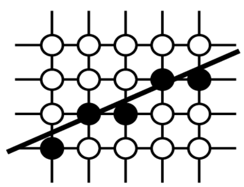
13

Line Drawing

14

Scan-Conversion Algorithms

- Scan-Conversion: Computing pixel coordinates for *ideal* line on 2D raster grid
- Pixels best visualized as circles/dots
  - Why? Monitor hardware



15

1994 Foley/VanDam/Finer/Huges/Phillips ICG

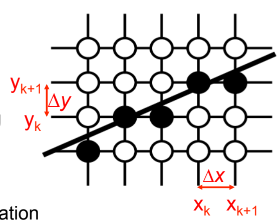
Drawing a Line

- $y = mx + B$
- $m = \Delta y / \Delta x$
- Start at leftmost x and increment by 1
  $\rightarrow \Delta x = 1$
- $y_i = \text{Round}(mx_i + B)$
- This is expensive and inefficient
- Since $\Delta x = 1$, $y_{i+1} = y_i + \Delta y = y_i + m$
  - No more multiplication!
- This leads to an incremental algorithm

16

Digital Differential Analyzer (DDA)

- If |slope| is less then 1
  - $\Delta x = 1$
  - else $\Delta y = 1$
- Check for vertical line
  - $m = \infty$
- Compute corresponding Δy (Δx) = m (1/m)
- $x_{k+1} = x_k + \Delta x$
- $y_{k+1} = y_k + \Delta y$
- Round (x,y) for pixel location
- Issue: Would like to avoid floating point operations



17

1994 Foley/VanDam/Finer/Huges/Phillips ICG

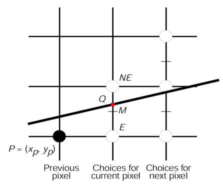
Generalizing DDA

- If |slope| is less than or equal to 1
  - Ending point should be right of starting point
- If |slope| is greater than 1
  - Ending point should be above starting point
- Vertical line is a special case
  $\Delta x = 0$

18

Bresenham's Algorithm

- 1965 @ IBM
- Basic Idea:
  - Only integer arithmetic
  - Incremental



- Consider the *implicit* equation for a line:

$$f(x,y) = ax + by + c = 0$$

1994 Foley/VanDam/Finer/Huges/Phillips ICG

The Algorithm

```
void bresenham(IntPoint q, IntPoint r) {
    int dx, dy, D, x, y;
    dx = r.x - q.x;                 // line width and height
    dy = r.y - q.y;
    D = 2*dy - dx;                  // initial decision value
    y = q.y;                        // start at (q.x,q.y)
    for (x = q.x; x <= r.x; x++) {
        writePixel(x, y);
        if (D <= 0) D += 2*dy;      // below midpoint - go to E
        else {                      // above midpoint - go to NE
            D += 2*(dy - dx); y++;
        }
    }
}
```

Assumptions: $q_x < r_x$
$0 \le$ slope ≤ 1

Pre-computed: $2d_y \quad 2(d_y - d_x)$

Pics/Math courtesy of Dave Mount @ UMD-CP

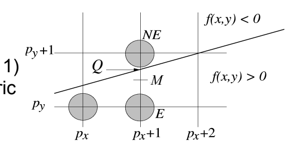
Bresenham's Algorithm

Given:
implicit line equation: $f(x,y) = ax + by + c = 0$
Let: $d_x = r_x - q_x,\ d_y = r_y - q_y$
where r and q are points on the line and d_x, d_y are positive
$a = d_y,\ b = -d_x,\ c = -(q_x r_y - r_x q_y)$
Then:
Observe that all of these are integers
and: $f(x,y) < 0$ for points above the line
$f(x,y) > 0$ for points below the line
Now…..

Pics/Math courtesy of Dave Mount @ UMD-CP

Bresenham's Algorithm

- Suppose we just finished (p_x, p_y)
  - (assume $0 \le$ slope ≤ 1) other cases symmetric
- Which pixel next?
  - *E* or *NE*



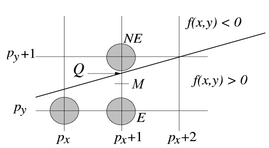
East $(E = (p_x + 1, p_y))$
NorthEast $(NE = (p_x + 1, p_y + 1))$

Pics/Math courtesy of Dave Mount @ UMD-CP

Bresenham's Algorithm

Assume:
- Q = exact y value at $x = p_x + 1$
- y midway between *E* and *NE*: $M = p_y + 1/2$

Observe:
If $Q < M$, then pick *E*
Else pick *NE*
If $Q = M$,
it doesn't matter

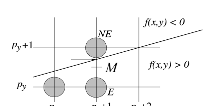


Pics/Math courtesy of Dave Mount @ UMD-CP

Bresenham's Algorithm

- Create "modified" implicit function (2x)
  $f(x,y) = 2ax + 2by + 2c = 0$
- Create a *decision variable D* to select, where *D* is the value of *f* at the midpoint:

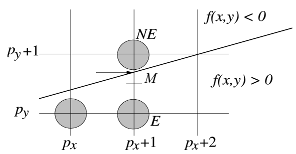
$$\begin{aligned} D &= f(p_x + 1, p_y + (1/2)) \\ &= 2a(p_x + 1) + 2b\left(p_y + \frac{1}{2}\right) + 2c \\ &= 2ap_x + 2bp_y + (2a + b + 2c) \end{aligned}$$



Pics/Math courtesy of Dave Mount @ UMD-CP

Bresenham’s Algorithm

- If $D > 0$ then M is below the line $f(x,y)$
  - *NE* is the closest pixel
- If $D \le 0$ then M is above the line $f(x,y)$
  - *E* is the closest pixel



25

Bresenham’s Algorithm

- If $D > 0$ then M is below the line $f(x,y)$
  - *NE* is the closest pixel
- If $D \le 0$ then M is above the line $f(x,y)$
  - *E* is the closest pixel

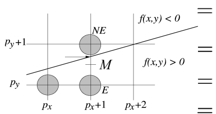
- Note: because we multiplied by 2x, *D* is now an integer---which is very good news
- How do we make this incremental??

26

Case I: When *E* is next

- What increment for computing a new *D*?
- Next midpoint is: $(p_x+2, p_y+(1/2))$

$$\begin{aligned} D_{new} &= f(p_x+2, p_y+(1/2)) \\ &= 2a(p_x+2)+2b\left(p_y+\frac{1}{2}\right)+2c \\ &= 2ap_x+2bp_y+(4a+b+2c) \\ &= 2ap_x+2bp_y+(2a+b+2c)+2a \\ &= D+2a = D+2d_y \end{aligned}$$



- Hence, increment by: $2d_y$

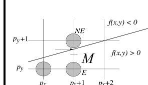
27

Pics/Math courtesy of Dave Mount @ UMD-CP

Case II: When *NE* is next

- What increment for computing a new *D*?
- Next midpoint is: $(p_x+2, p_y+1+(1/2))$

$$\begin{aligned} D_{new} &= f(p_x+2, p_y+1+(1/2)) \\ &= 2a(p_x+2)+2b\left(p_y+\frac{3}{2}\right)+2c \\ &= 2ap_x+2bp_y+(4a+3b+2c) \\ &= 2ap_x+2bp_y+(2a+b+2c)+(2a+2b) \\ &= D+2(a+b) = D+2(d_y-d_x) \end{aligned}$$



- Hence, increment by: $2(d_y-d_x)$

28

Pics/Math courtesy of Dave Mount @ UMD-CP

How to get an initial value for *D*?

- Suppose we start at: (q_x, q_y)
- Initial midpoint is: $(q_x+1, q_y+1/2)$

Then:

$$\begin{aligned} D_{init} &= f(q_x+1, q_y+1/2) \\ &= 2a(q_x+1)+2b\left(q_y+\frac{1}{2}\right)+2c \\ &= (2aq_x+2bq_y+2c)+(2a+b) \\ &= 0+2a+b \\ &= 2d_y-d_x \end{aligned}$$

29

Pics/Math courtesy of Dave Mount @ UMD-CP

The Algorithm

```
void bresenham(IntPoint q, IntPoint r) {
    int dx, dy, D, x, y;
    dx = r.x - q.x;                 // line width and height
    dy = r.y - q.y;
    D = 2*dy - dx;                  // initial decision value
    y = q.y;                        // start at (q.x,q.y)
    for (x = q.x; x <= r.x; x++) {
        writePixel(x, y);
        if (D <= 0) D += 2*dy;      // below midpoint - go to E
        else {                      // above midpoint - go to NE
            D += 2*(dy - dx); y++;
        }
    }
}
```

Assumptions: $q_x < r_x$
0 ≤ slope ≤ 1

Pre-computed: $2d_y$ $2(d_y-d_x)$

30

Pics/Math courtesy of Dave Mount @ UMD-CP

Generalize Algorithm

- If $q_x > r_x$, swap points
- If slope > 1, always increment y, conditionally increment x
- If -1 <= slope < 0, always increment x, conditionally decrement y
- If slope < -1, always decrement y, conditionally increment x
- Rework D increments

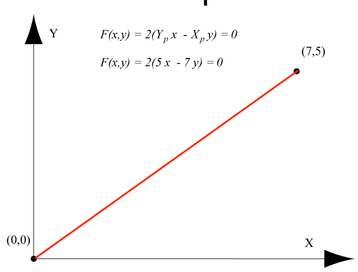
31

Generalize Algorithm

- Reflect line into first case
- Calculate pixels
- Reflect pixels back into original orientation

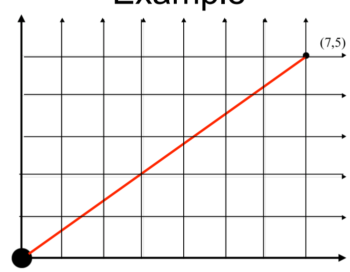
32

Bresenham's Algorithm: Example



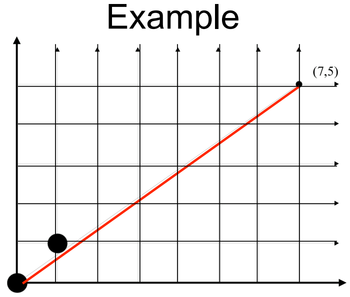
33

Bresenham's Algorithm: Example



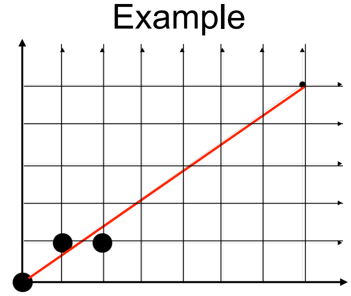
34

Bresenham's Algorithm: Example



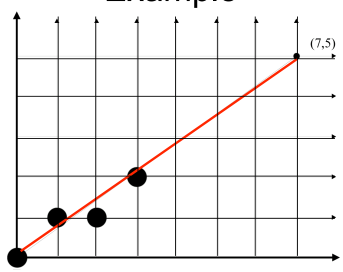
35

Bresenham's Algorithm: Example



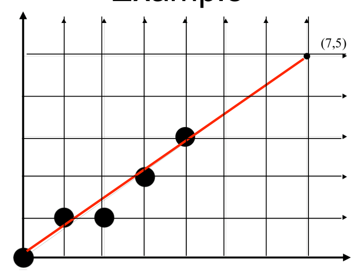
36

Bresenham's Algorithm: Example



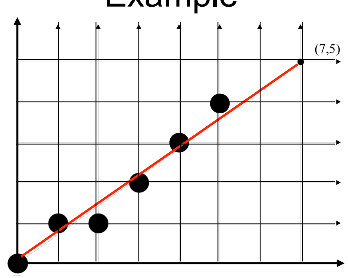
37

Bresenham's Algorithm: Example



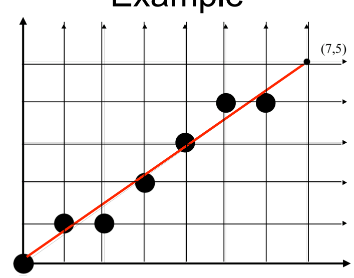
38

Bresenham's Algorithm: Example



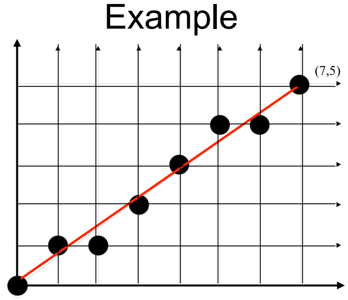
39

Bresenham's Algorithm: Example



40

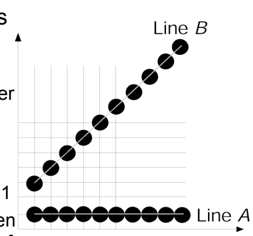
Bresenham's Algorithm: Example



41

Some issues with Bresenham's Algorithms

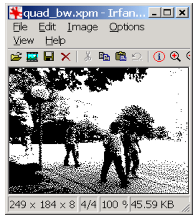
- Pixel 'density' varies based on slope
  - straight lines look darker, more pixels per unit length
  - Endpoint order
  - Line from P1 to P2 should match P2 to P1
  - Always choose *E* when hitting *M*, regardless of direction

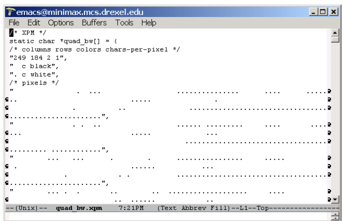


42

XPM Format

- Encoded pixels
- C code





- ASCII Text file
- Viewable on Unix w/ **`display`**
- On Windows with IrfanVIew
- Translate w/ ***`convert`***

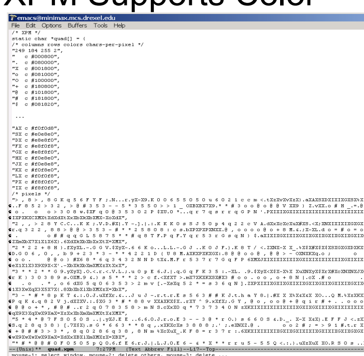
44

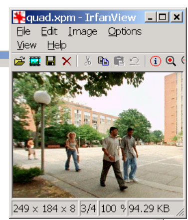
XPM Basics

- **X P**ixel**M**ap (XPM)
- Native file format in X Windows
- Color cursor and icon bitmaps
- Files are actually C source code
- Read by compiler instead of viewer
- Successor of **X BitM**ap (XBM) B-W format

45

XPM Supports Color





XPM: Defining Grayscales and Colors

- Each pixel specified by an ASCII char
- *key* describes the context this color should be used within. You can always use "*c*" for "color".
- Colors can be specified:
  - color name
  - "#" followed by the RGB code in hexadecimal
- RGB – 24 bits (2 characters '`0`' - '`f`') for each color.

47

XPM: Specifying Color

Color Name	RGB	Color
black	**`# 00 00 00`**	
white	**`# ff ff ff`**	
	`# 80 80 80`	
red	**`# ff 00 00`**	
green	**`# 00 ff 00`**	
blue	**`# 00 00 ff`**	

48

XPM Example

- Array of C strings
- The XPM format assumes the origin (0,0) is in the upper left-hand corner.
- First string is "width height ncolors cpp"
- Then you have "ncolors" strings associating characters with colors.
- And last you have "height" strings of "width" * "chars_per_pixel" characters

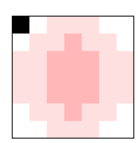
```
/* XPM */
static char *sco100[] = {

/* width height num_colors chars_per_pixel */
"7 7 4 1",

/* colors */
"- c #ffffff",
"# c #ffe0e0",
"a c #ffb7b7",
"X c #010101",

/* pixels */
"X-###--",
"-##a##-",
"##aaa##",
"##aaa##",
"##aaa##",
"-##a##-",
"--###--"

};
```



49

Programming assignment 1

- Input PostScript-like file
- Output B/W XPM
- Primary I/O formats for the course
- Create data structure to hold points and lines in memory (*the world model*)
- Implement 2D translation, rotation and scaling of the world model
- Implement line drawing and clipping
- January 20th
- Get started now!

51

Questions?

Go to Assignment 1

52