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**

Mathematical Preliminaries

- Vector: an n-tuple of real numbers
- Vector Operations
  - Vector addition: \(v + w = w\)
  - 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
  \[\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n\]
  where \(\alpha_i\) is a real number (i.e. a scalar)
- Dot Product:
  \[u \cdot v = \sum_{k=1}^{n} u_k v_k\]
  a real value \(\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n\) written as \(u \cdot v\)
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:
  \[ \sqrt{\sum x_i^2} \]
- This is also the length of vector \(v\):
  \[ ||v|| \]
- Normalization of a vector: \[ \hat{v} = \frac{v}{||v||} \]
- **Orthogonal vectors:** \[ \hat{u} \cdot \hat{v} = 0 \]

Projections & Angles

- **Angle** between vectors, \(\theta\):
  \[ \hat{u} \cdot \hat{v} = ||u|| \cdot ||v|| \cdot \cos(\theta) \]
  \[ \theta = \text{ang}(\hat{u}, \hat{v}) = \cos^{-1} \left( \frac{\hat{u} \cdot \hat{v}}{||u|| \cdot ||v||} \right) = \cos^{-1}(\hat{u} \cdot \hat{v}) \]
- **Projection** of vectors:
  \[ \hat{u}_1 = \left( \hat{u} \cdot \hat{v} \right) \frac{\hat{v}}{||v||^2} \]
  \[ \hat{u}_2 = \hat{u} - \hat{u}_1 \]

Matrices and Matrix Operators

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

Matrices and Matrix Operators

- **Matrix Multiplication**:
  \[ [C] = [A][B] \]
  - Sum over rows & columns
  - Recall: matrix multiplication is not commutative
- **Identity Matrix**:
  \[ \begin{bmatrix} 1_{n \times n} \end{bmatrix} \]
  \(1s\) on diagonal
  \(0s\) everywhere else

Matrix Determinants

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

Cross Product

- Given two non-parallel vectors, \(A\) and \(B\):
  \(A \times B\) calculates third vector \(C\) that is orthogonal to \(A\) and \(B\)
  \[ A \times B = (a_yb_z - a_zb_y, a_zb_x - a_xb_z, a_xb_y - a_yb_x) \]
  \[ A \times B = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \end{bmatrix} \]
Matrix Transpose & Inverse

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

Scan-Conversion Algorithms

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

Line Drawing

- **Drawing a Line**
  - \( y = mx + B \)
  - \( m = \Delta y / \Delta x \)
  - Start at leftmost \( x \) and increment by 1
    \[ \Delta x = 1 \]
  - \( y_i = \text{Round}(mx_i + B) \)
  - 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

Digital Differential Analyzer (DDA)

- If \(|\text{slope}|\) is less than 1
  - \( \Delta x = 1 \)
  - \( \Delta y = \frac{\Delta y}{\Delta x} \)
  - Check for vertical line
    - \( m = \infty \)
  - Compute corresponding \( \Delta y \ (\Delta x) = m (1/m) \)
  - \( x_{i+1} = x_i + \Delta x \)
  - \( y_{i+1} = y_i + \Delta y \)
  - Round \((x,y)\) for pixel location
  - Issue: Would like to avoid floating point operations

Generalizing DDA

- If \(|\text{slope}|\) is less than or equal to 1
  - Ending point should be right of starting point
- If \(|\text{slope}|\) is greater than 1
  - Ending point should be above starting point
- Vertical line is a special case
  \[ \Delta x = 0 \]
Bresenham’s Algorithm

- 1965 @ IBM
- Basic Idea:
  - Only integer arithmetic
  - Incremental
- Consider the implicit equation for a line:
  \[ f(x, y) = ax + by + c = 0 \]

The Algorithm

```c
void bresenham(int Point1, int Point2) {
    int dx, dy, D, x = point1.x, y = point1.y;
    dy = abs(dy); // line width and height
    D = 2 * dy - dx; // initial decision value
    for (x = point1.x; x < point2.x; x++) {
        writePixel(x, y);
        if (D <= 0) D += 2 * dy;
        else D += 2 * dy - 2 * dx;
        y++;
    }
}
```

Assumptions: \( q_x < r_x \)
0 ≤ slope ≤ 1

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......

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

Bresenham’s Algorithm

Suppose we just finished \((p_x, p_y)\)
- (assume 0 ≤ slope ≤ 1)
other cases symmetric
- Which pixel next?
  - \( E \) or \( NE \)
  - East \((E = (p_x + 1, p_y))\)
  - NorthEast \((NE = (p_x + 1, p_y + 1))\)

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:
\[ D = f(p_x + 1, p_y) + f(p_x + 1/2, p_y) + 2(c) \]
Bresenham’s Algorithm

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

![Diagram of line and pixels](image)

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

Case I: When $E$ is next

- What increment for computing a new $D$?
- Next midpoint is: $(p_x + 2, p_y + (1/2))$
  
  $D_{new} = f(p_x + 2, p_y + (1/2))$
  
  $= 2a(p_x + 2) + 2b\left(p_y + \frac{1}{2}\right) + 2c$
  
  $= 2ap_x + 2bp_y + (4a + b + 2c) + 2a$
  
  $= D + 2a = D + 2d_x$
  
  Hence, increment by: $2d_x$

Case II: When $N E$ is next

- What increment for computing a new $D$?
- Next midpoint is: $(p_x + 2, p_y + 1 + (1/2))$
  
  $D_{new} = f(p_x + 2, p_y + 1 + (1/2))$
  
  $= 2a(p_x + 2) + 2b\left(p_y + \frac{3}{2}\right) + 2c$
  
  $= 2ap_x + 2bp_y + (4a + 3b + 2c) + 2a$
  
  $= D + 2(a + b) = D + 2(d_y - d_x)$
  
  Hence, increment by: $2(d_y - d_x)$

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

The Algorithm

```c
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++) { // write pixels(x, y);
        if (x < 0) d += 2*dy; // below midpoint - go to E
        else if (x > 0) d -= 2*dy; // above midpoint - go to NE
        if (d < 0) y++; // below midpoint - go to NE
        else y--;
    }
}
```

Assumptions: $q_x \geq r_x$

$0 \leq \text{slope} \leq 1$

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

- If \( q_x > r_x \), swap points
- If slope > 1, always increment y, conditionally increment x
- If \(-1 \leq \text{slope} < 0\), always increment x, conditionally decrement y
- If slope < -1, always decrement y, conditionally increment x
- Rework D increments

Generalize Algorithm

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

Bresenham’s Algorithm: Example

Bresenham’s Algorithm: Example

Bresenham’s Algorithm: Example

Bresenham’s Algorithm: Example
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
XPM Format
• Encoded pixels
• C code
• ASCII Text file
• Viewable on Unix w/ `display`
• On Windows w/ `IrfanView`
• Translate w/ `convert`

XPM Basics
• X PixelMap (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 BitMap (XBM) B-W format

XPM Supports Color
XPM: Defining Grayscale
• 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.

XPM: Specifying Color

<table>
<thead>
<tr>
<th>Color Name</th>
<th>RGB</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td># 00 00 00</td>
<td><img src="image" alt="black" /></td>
</tr>
<tr>
<td>white</td>
<td># ff ff ff</td>
<td><img src="image" alt="white" /></td>
</tr>
<tr>
<td>red</td>
<td># ff 00 00</td>
<td><img src="image" alt="red" /></td>
</tr>
<tr>
<td>green</td>
<td># 00 ff 00</td>
<td><img src="image" alt="green" /></td>
</tr>
<tr>
<td>blue</td>
<td># 00 00 ff</td>
<td><img src="image" alt="blue" /></td>
</tr>
</tbody>
</table>

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
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
• Due October 5th
• Get started now!

Questions?

Go to Assignment 1