Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

CS 560

Computer Graphics

Professor Richard Eckert

Lecture # 11

February 26, 2001

Area Fill Algorithms

- Boundary/Flood Fill
- Scanline Polygon Fill
- Scanline Boundary Fill
- Pattern Fill

- Transformations

The Scanline Boundary Fill Algorithm

Select a Seed Point (x,y)

Push (x,y) onto Stack

While Stack is not empty:

Pop Stack (retrieve x,y)

Fill current run y (iterate on x until borders are hit)

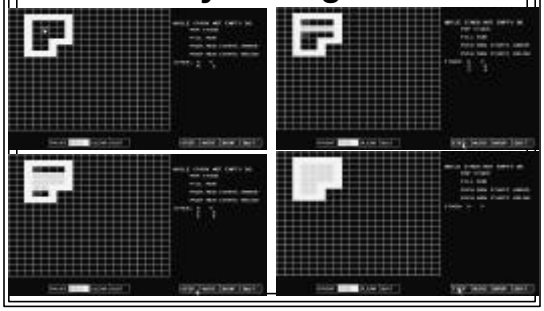
Push left-most unfilled, nonborder pixel above

-->new "above" seed

Push left-most unfilled, nonborder pixel below

-->new "below" seed

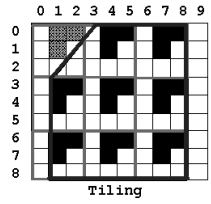
Dino Demo of Scanline Boundary Fill Algorithm



Pattern Filling

- Represent fill pattern with a Pattern Matrix
- Replicate it across the area until covered by non-overlapping copies of the matrix
  - Called Tiling

Pattern Filling--Pattern Matrix



Tiling

Pattern Matrix

H=3 \ W=3	0	1	2
0	0	1	1
1	0	1	0
2	0	0	0

Pattern

x	0	1	2	3	4	5	6	7	8
x posn	0	1	2	0	1	2	0	1	2
y	0	1	2	3	4	5	6	7	8
y posn	0	1	2	0	1	2	0	1	2

In general, posn in matrix:
xpos = x%W, ypos = y%H

Using the Pattern Matrix

- Modify fill algorithm
- As (x,y) pixel in area is examined:

```
if(pat_mat[x%W][y%H] == 1)
    SetPixel(x,y);
```

A More Efficient Way

Store pat_matrix as a 1-D array of bytes or words

y%H --> byte or word in pat_matrix

Shift a mask (e.g., 10000000) by x%W

--> position of bit in the byte or word in pat_matrix

"AND" byte or word with shifted mask

if result != 0, Set the pixel

Color Patterns

- Pattern Matrix contains color values
- So read color value of pixel directly from Pattern Matrix:

SetPixel(x, y, pat_mat[x%W][y%H])

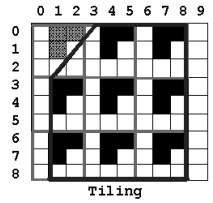
Moving the Filled Polygon

- As done above, pattern doesn't move with polygon
- Need to "anchor" pattern to polygon
- Fix a polygon vertex as "pattern reference point", e.g., (x0,y0)

```
If (pat_matrix[(x-x0)%W][(y-y0)%H]==1)
    SetPixel(x,y)
```

- Now pattern moves with polygon

Pattern Filling--Pattern Matrix



Tiling

Pattern Matrix W=3

H=3	0	1	2
0	0	1	1
1	0	1	0
2	0	0	0

Pattern

	0	1	2
0		■	
1		■	
2			

x	0	1	2	3	4	5	6	7	8
x posn	0	1	2	0	1	2	0	1	2
y	0	1	2	3	4	5	6	7	8
y posn	0	1	2	0	1	2	0	1	2

In general, posn in matrix:
xpos = x%W, ypos = y%H

Geometric Transformations

- Moving objects relative to a stationary coordinate system
- Common transformations:
  - Translation
  - Rotation
  - Scaling
- Implemented using vectors and matrices

Quick Review of Matrix Algebra

- Matrix--a rectangular array of numbers
- a_{ij}: element at row i and column j
- Dimension: m x n

  m = number of rows

  n = number of columns

A Matrix

An m x n matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

Degenerate case: m = 1 (a row vector)

$$V = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \end{bmatrix} \quad \text{or:}$$

$$V = \begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_n \end{bmatrix}$$

Vectors and Scalars

Degenerate Case (n=1) a column vector--

$$V = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \vdots \\ a_{m1} \end{bmatrix} \quad \text{or:} \quad V = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_m \end{bmatrix}$$

Point in space (x,y) or (x,y,z)--
Use vectors:

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \text{ or } P = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

2D 3D

Transpose of a Matrix A^T

$a^T_{ij} = a_{ji}$ The transpose of a row vector is a column vector.

Degenerate Case: m=n=1, a scalar

$S = a_{11}$

Matrix Operations-- Multiplication by a Scalar

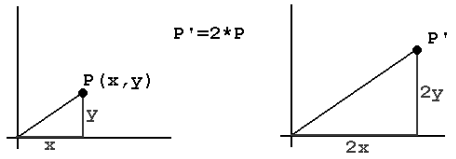
C = k*A

$c_{ij} = k * a_{ij}$, $1 <= i <= m$, $1 <= j <= n$

- Example: multiplying position vector by a constant:
  - Multiplies each component by the constant
  - Gives a scaled position vector (k times as long)

Example of Multiplying a Position Vector by a Scalar

Multiplying Position Vector by a Scalar--
Scales the Position Vector



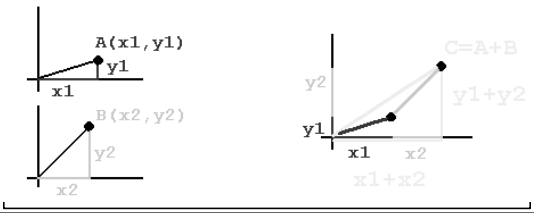
Adding two Matrices

- Must have the same dimension
- C = A + B

  $c_{ij} = a_{ij} + b_{ij}$, $1 <= i <= m$, $1 <= j <= n$
- Example: adding two position vectors
  - Add the components
  - Gives a vector equal to the net displacement

Adding two Position Vectors: Result is the Net Displacement

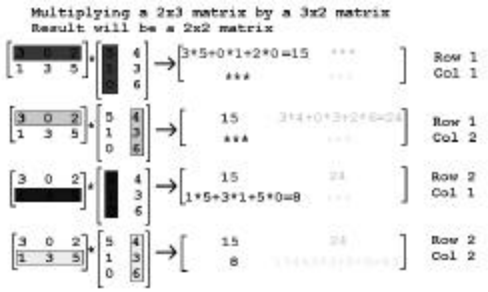
Adding Two Position Vectors



Multiplying Two Matrices

- m x n = (m x p) * p x n)
- C = A * B
- $c_{ij} = \Sigma\ a_{ik} * b_{kj}$, 1<=k<=p
- In other words:
  - To get element in row i, column j
    - Multiply each element in row i by each corresponding element in column j
    - Add the partial products

Matrix Multiplication An Example



Multipy a Vector by a Matrix

- V1 = A*V
- If V is a m-dimensional column vector, A must be an m x m matrix
- $v1_i = \Sigma\ a_{ik} * v_k$, 1<=k<=m
  - So to get element i of product vector:
    - Multiply each row i matrix element by each corresponding element of vector
    - Add the partial products

An Example

Multiplying a 2-D Vector by a Matrix

$$V = \begin{bmatrix} 3 & 0 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} 5 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 3*5+0*2=15 \\ *** \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} 5 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 15 \\ 1*5+4*2=13 \end{bmatrix}$$

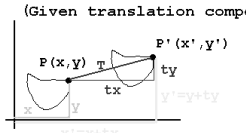
$$V = \begin{bmatrix} 15 \\ 13 \end{bmatrix}$$

Geometrical Transformations

- Alter or move objects on screen
- Affine Transformations:
  - Preserve straight lines
- Transform points in the object
  - Translation:
    - A Vector Sum
  - Rotation and Scaling:
    - Matrix Multiplies

Translation: Moving Objects

TRANSLATIONS IN 2-D
(Given translation components, tx, ty)



$$P = \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Component rule:
x' = x + tx
y' = y + ty

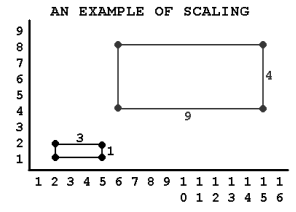
So: $P' = \begin{bmatrix} x+tx \\ y+ty \end{bmatrix}$

General rule:

P' = P + T

where $T = \begin{bmatrix} tx \\ ty \end{bmatrix}$

Scaling: Sizing Objects



SCALING FACTORS:
sx = 3, sy = 4

P1 = (2,1) ⟶ (6,4)
P2 = (5,1) ⟶ (15,4)
P3 = (5,2) ⟶ (15,8)
P4 = (2,2) ⟶ (6,8)

Resulting figure is
3 times as wide,
4 times as high

Component Rule: x' = sx*x
y' = sy*y

Want a general rule for vectors
Adding won't work

Try P' = S*P But what is S?

Scaling, continued

P' = S*P

P, P' are 2D vectors, so S must be 2x2 matrix

Component equations:

x' = sx*x, y' = sy*y

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s11 & s12 \\ s21 & s22 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{or} \quad \begin{aligned} x' &= s11*x + s12*y \\ y' &= s21*x + s22*y \end{aligned}$$

So: s11=sx, s12=0, s21=0, s22=sy

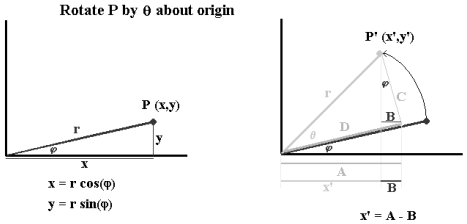
Therefore: $S = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix}$ (The scaling matrix)

Rotation about Origin

- Rotate point P by θ about origin
- Rotated point is P'
- Want to get P' from P and θ
- P' = R*P
- R is the rotation matrix
- Look at components:

Rotation: X Component

Rotate P by θ about origin



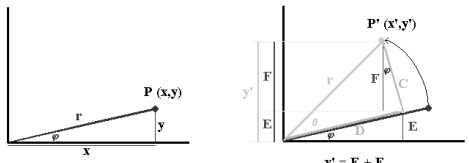
x = r cos(φ)
y = r sin(φ)

So:
x' = r cos(θ) cos(φ) - r sin(θ) sin(φ)
x' = x cos(θ) - y sin(θ)

x' = A - B
A = D cos(φ)
B = C sin(φ)
D = r cos(θ)
C = r sin(θ)

Rotation: Y Component

Rotate P by θ about origin



x = r cos(φ)
y = r sin(φ)

y' = E + F
E = D sin(φ)
F = C cos(φ)
D = r cos(θ)
C = r sin(θ)

So:
y' = r cos(θ) sin(φ) + r sin(θ) cos(φ)
y' = y cos(θ) + x sin(θ)

Rotation: Result

P' = R*P

R must be a 2x2 matrix

Component equations:

x' = x cos(θ) - y sin(θ)

y' = x sin(θ) + y cos(θ)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} r11 & r12 \\ r21 & r22 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{or} \quad \begin{aligned} x' &= r11*x + r12*y \\ y' &= r21*x + r22*y \end{aligned}$$

So: r11=cos(θ), r12=-sin(θ), r21=sin(θ), r22=cos(θ)

Therefore: $R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$

The Rotation Matrix

Transforming Objects

- For example, lines
  1. Transform every point & plot (too slow)
  2. Transform endpoints, draw the line
     - Since these transformations are affine, result is the transformed line

Composite Transformations

- Successive transformations
- e.g., scale then rotate an n-point object:
  1. Scale points: P' = S*P (n matrix multiplies)
  2. Rotate pts: P'' = R*P' (n matrix multiplies)

  But:

  P'' = R*(SP), & matrix mult. is associative

  P'' = (R*S)*P = M_{comp}*P

  So Compute M_{comp} = R*S (1 matrix mult.)

  P'' = M_{comp}*P (n matrix multiplies)

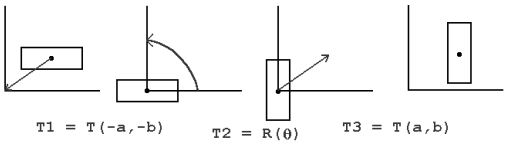
  n+1 multiplies vs. 2*n multiplies

Composite Transformations

Another example: Rotate in place

center at (a,b)

1. Translate to origin: T(-a.-b)
2. Rotate: R(θ)
3. Translate back: T(a,b)



Rotation in place:

1. P' = P + T1
2. P'' = R*P' = R*(P+T1)
3. P''' = P''+T3 = R*(P+T1) + T3

Can't be put into single matrix mult. form:

i.e., P''' != T_{comp} * P

But we want to be able to do so!!

Problem is: translation--vector adds rotation/scaling--matrix multiplies