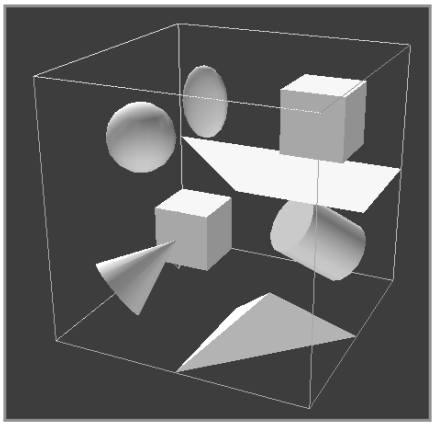
3D Rendering Pipeline

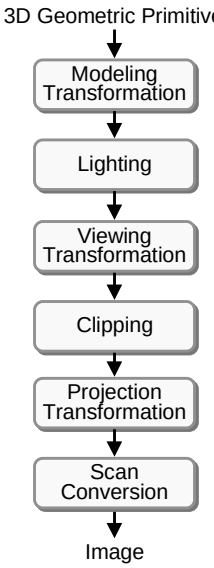
Thomas Funkhouser
Princeton University
C0S 426, Fall 1999

3D Rendering Example

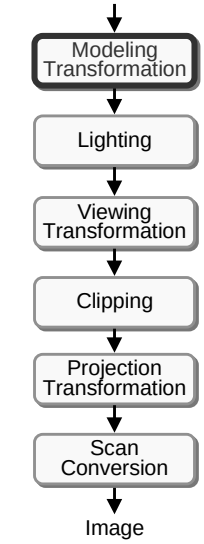


What issues must be addressed by a
3D rendering system?

3D Rendering Pipeline (for direct illumination)

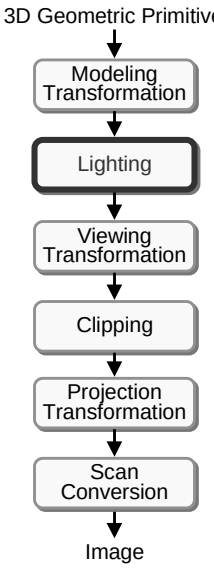


3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

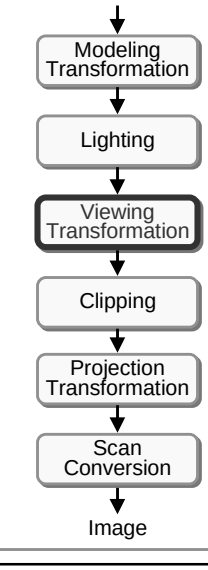
3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

Simulate direct illumination and reflectance

3D Rendering Pipeline (for direct illumination)

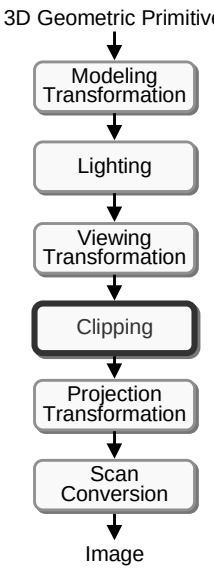


Transform into 3D world coordinate system

Simulate direct illumination and reflectance

Transform into 3D camera coordinate system

3D Rendering Pipeline (for direct illumination)



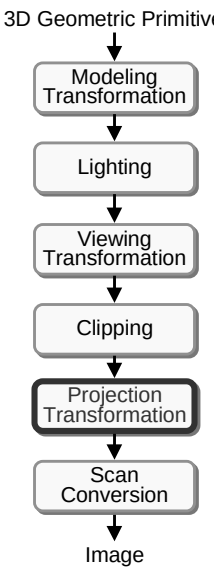
Transform into 3D world coordinate system

Simulate direct illumination and reflectance

Transform into 3D camera coordinate system

Clip primitives outside camera's view

3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

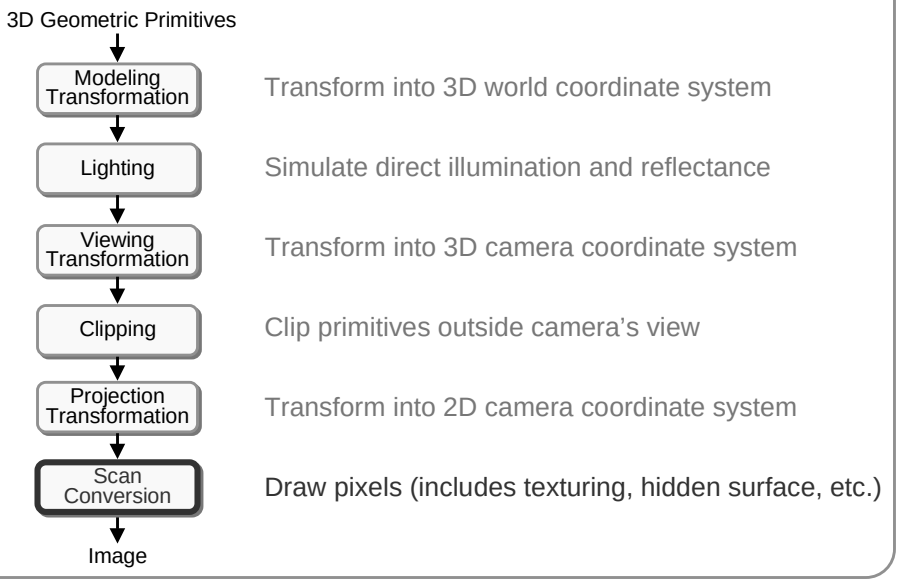
Simulate direct illumination and reflectance

Transform into 3D camera coordinate system

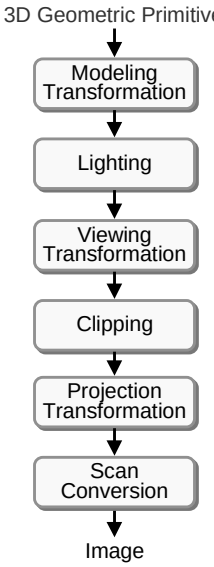
Clip primitives outside camera's view

Transform into 2D camera coordinate system

3D Rendering Pipeline (for direct illumination)



3D Rendering Pipeline (for direct illumination)



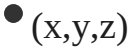
3D Geometric Primitives

- How are these shapes described in a computer?
  - Point
  - Vector
  - Line
  - Ray
  - Triangle
  - Polygon
  - Quadric curve
  - Spline
  - Quadric solid
  - Curved surface
  - etc.

3D Point

- Specifies a location
  - Represented by three coordinates
  - Infinitely small

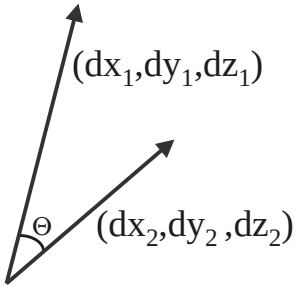
```
typedef struct {
    Coordinate x;
    Coordinate y;
    Coordinate z;
} Point;
```



3D Vector

- Specifies a direction and a magnitude
  - Represented by three coordinates
  - Magnitude ||V|| = sqrt(dxdx + dydy + dzdz)
  - Has no location

```
typedef struct {
    Coordinate dx;
    Coordinate dy;
    Coordinate dz;
} Vector;
```

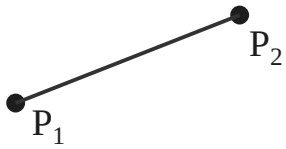


- Dot product of two 3D vectors
  - $V_1 \cdot V_2 = dx_1dx_2 + dy_1dy_2 + dz_1dz_2$
  - $V_1 \cdot V_2 = ||V_1|| \, ||V_2|| \cos(\Theta)$

3D Line Segment

- Specifies a linear combination of two points
  - Parametric representation:
    - » $P = P_1 + t\,(P_2 - P_1), \quad (0 \le t \le 1)$

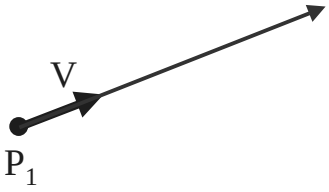
```
typedef struct {
    Point P1;
    Point P2;
} Segment;
```



3D Ray

- Line segment with one endpoint at infinity
  - Parametric representation:
    - » $P = P_1 + t\,V, \quad (0 <= t < \infty)$

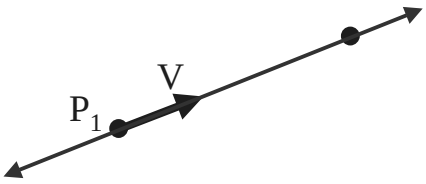
```
typedef struct {
    Point P1;
    Vector V;
} Line;
```



3D Line

- Line segment with both endpoints at infinity
  - Parametric representation:
    - » $P = P_1 + t\,V, \quad (-\infty < t < \infty)$

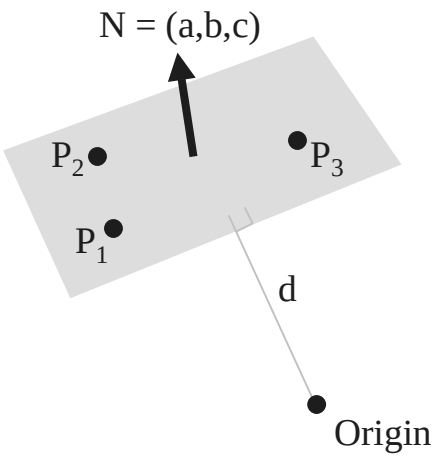
```
typedef struct {
    Point P1;
    Vector V;
} Line;
```



3D Plane

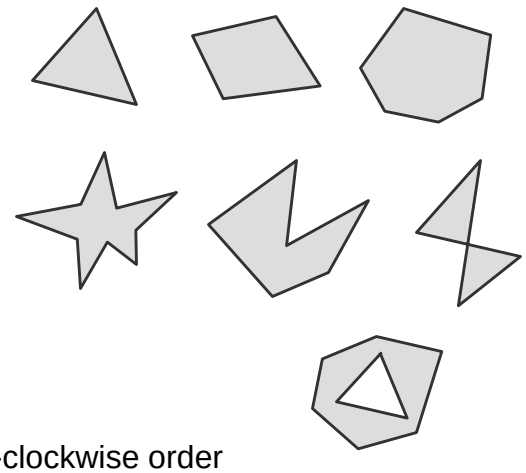
- Specifies a linear combination of three points
  - Implicit representation:
    - » P·N + d = 0, or
    - » ax + by + cz + d = 0

```
typedef struct {
    Vector N;
    Distance d;
} Line;
```



3D Polygon

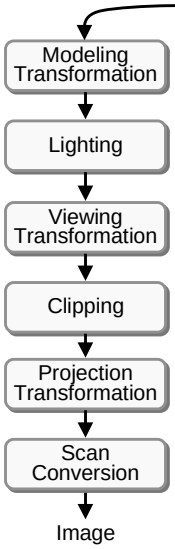
- Area “inside” a sequence of coplanar points
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting
  - Holes



```
typedef struct {
    Point *points;
    int npoints;
} Polygon;
```

Points are in counter-clockwise order

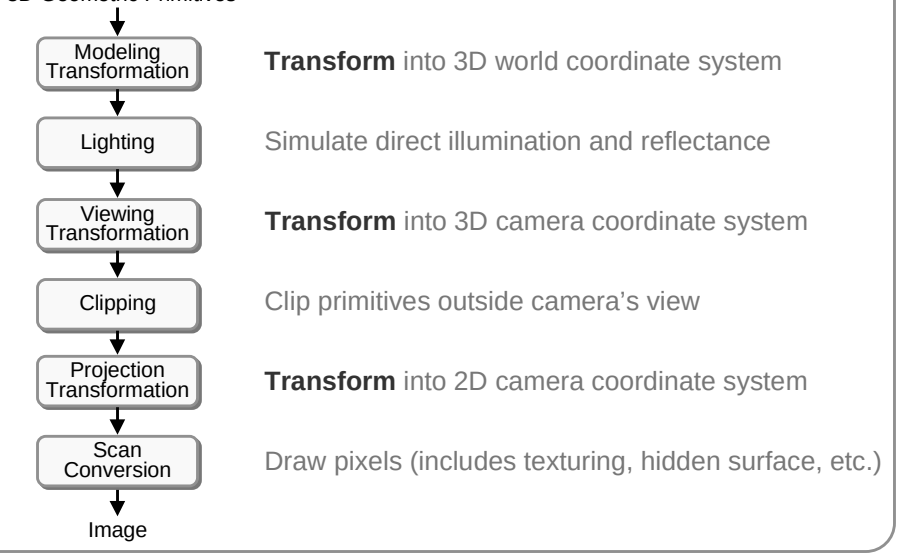
OpenGL Rendering



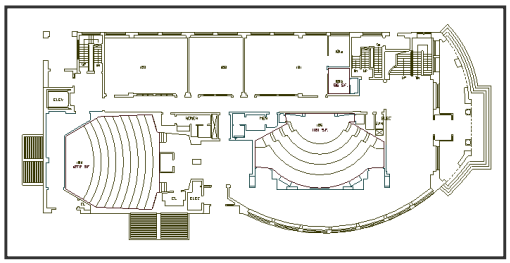

```
glBegin(GL_POLYGON);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(0.0, 0.0, 0.0);
glEnd();
```

OpenGL executes steps of 2D and/or 3D rendering pipeline

Transformations



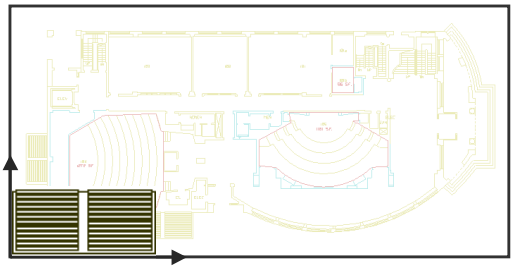
Basic 2D Transformations



- Translation:
  - x’ = x + tx
  - y’ = y + ty
- Scale:
  - x’ = x * sx
  - y’ = y * sy
- Shear:
  - x’ = x + hx*y
  - y’ = y + hy*x
- Rotation:
  - x’ = x*cosΘ - y*sinΘ
  - y’ = x*sinΘ + y*cosΘ

Transformations can be combined (with simple algebra)

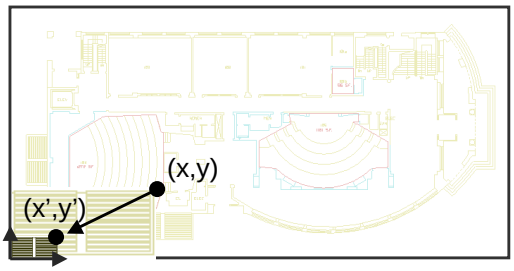
Basic 2D Transformations



- Translation:
  - x’ = x + tx
  - y’ = y + ty
- Scale:
  - x’ = x * sx
  - y’ = y * sy
- Shear:
  - x’ = x + hx*y
  - y’ = y + hy*x
- Rotation:
  - x’ = x*cosΘ - y*sinΘ
  - y’ = x*sinΘ + y*cosΘ

Basic 2D Transformations

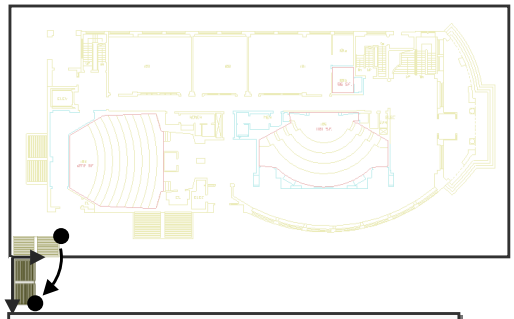
- Translation:
  - x’ = x + tx
  - y’ = y + ty
- Scale:
  - x’ = x * sx
  - y’ = y * sy
- Shear:
  - x’ = x + hx*y
  - y’ = y + hy*x
- Rotation:
  - x’ = x*cosΘ - y*sinΘ
  - y’ = x*sinΘ + y*cosΘ



x’ = x*sx
y’ = y*sy

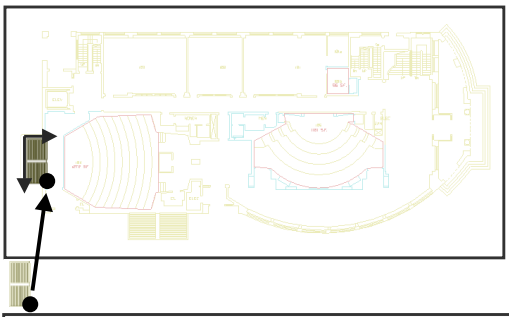
Basic 2D Transformations

- Translation:
  - x’ = x + tx
  - y’ = y + ty
- Scale:
  - x’ = x * sx
  - y’ = y * sy
- Shear:
  - x’ = x + hx*y
  - y’ = y + hy*x
- Rotation:
  - x’ = x*cosΘ - y*sinΘ
  - y’ = x*sinΘ + y*cosΘ



x’ = (x*sx)*cosΘ – (y*sy)*sinΘ
y’ = (x*sx)*sinΘ + (y*sy)*cosΘ

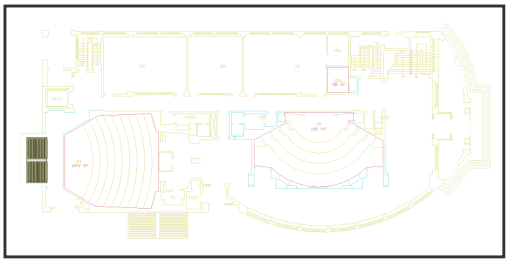
Basic 2D Transformations



- Translation:
  - x’ = x + tx
  - y’ = y + ty
- Scale:
  - x’ = x * sx
  - y’ = y * sy
- Shear:
  - x’ = x + hx*y
  - y’ = y + hy*x
- Rotation:
  - x’ = x*cosΘ - y*sinΘ
  - y’ = x*sinΘ + y*cosΘ

x’ = ((x*sx)*cosΘ – (y*sy)*sinΘ) + tx
y’ = ((x*sx)*sinΘ + (y*sy)*cosΘ) + ty

Basic 2D Transformations



- Translation:
  - x’ = x + tx
  - y’ = y + ty
- Scale:
  - x’ = x * sx
  - y’ = y * sy
- Shear:
  - x’ = x + hx*y
  - y’ = y + hy*x
- Rotation:
  - x’ = x*cosΘ - y*sinΘ
  - y’ = x*sinΘ + y*cosΘ

x’ = ((x*sx)*cosΘ – (y*sy)*sinΘ) + tx
y’ = ((x*sx)*sinΘ + (y*sy)*cosΘ) + ty

Matrix Representation

- We can represent a 2D transformation by a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- Multiplying a matrix by a column vector corresponds to applying the transformation to a point

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned}$$

Matrix Representation

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Matrices are a convenient and efficient way to represent a sequence of transformations

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Identity?

$$x' = x \qquad y' = y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Scale around (0,0)?

$$x' = sx * x \qquad y' = sy * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

$$x' = \cos\Theta * x - \sin\Theta * y \qquad y' = \sin\Theta * x + \cos\Theta * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Shear?

$$x' = x + shx * y \qquad y' = shy * x + y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & shx \\ shy & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Mirror over Y axis?

$$x' = -x$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Mirror over (0,0)?

$$x' = -x$$
$$y' = -y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

$$x' = x + tx$$
$$y' = y + ty$$

NO!

Only linear 2D transformations can be represented with a 2x2 matrix

Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Properties of linear transformations:
  - Satisfies: $T(s_1\mathbf{p}_1 + s_2\mathbf{p}_2) = s_1T(\mathbf{p}_1) + s_2T(\mathbf{p}_2)$
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

2D Translation

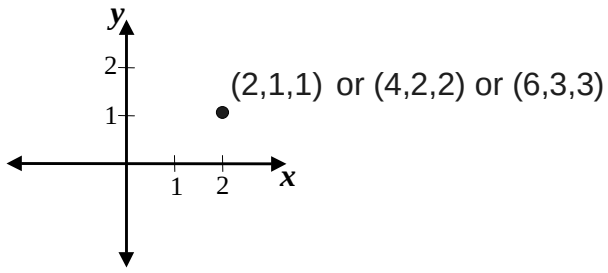
- 2D translation can be represented with a 3x3 matrix

$$x' = x + tx$$
$$y' = y + ty$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous Coordinates

- Add a 3rd coordinate to every 2D point
  - (x, y, w) represents a point at location (x/w, y/w)
  - (x, y, 0) represents a point at infinity
  - (0, 0, 0) is not allowed



Convenient coordinate system to
represent many useful transformations

Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

Affine Transformations

- Affine transformations are combinations of …
  - Linear transformations, and
  - Translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of affine transformations:
  - Origin does not map to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

Projective Transformations

- Projective transformations …

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

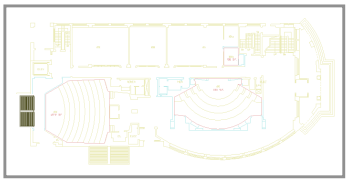
- Properties of projective transformations:
  - Origin does not map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved (but “cross-ratios” are)
  - Closed under composition

Matrix Composition

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

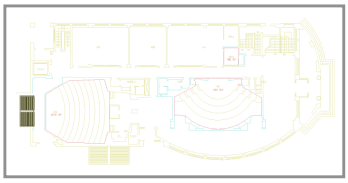
p' = T(tx,ty) R(Θ) S(sx,sy) **p**



Matrix Composition

- Matrices are a convenient and efficient way to represent a sequence of transformations
  - General purpose representation
  - Hardware matrix multiply
  - Efficiency with premultiplication
    - » Matrix multiplication is associative

p' = (T * (R * (S***p**)))
p' = (T*R*S) * **p**

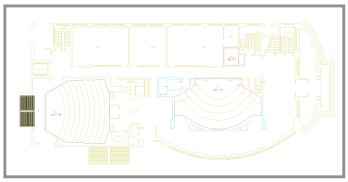


Matrix Composition

- Be aware: order of transformations matters
  - » Matrix multiplication is not commutative

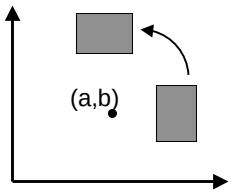
$$\mathbf{p}' = T * R * S * \mathbf{p}$$

"Global" "Local"

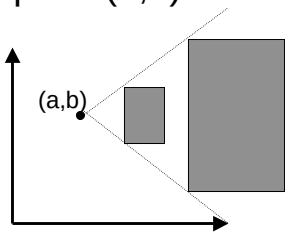


Matrix Composition

- Rotate by Θ around arbitrary point (a,b)
  - M=T(-a,-b) * R(Θ) * T(a,b)



- Scale by sx,sy around arbitrary point (a,b)
  - M=T(-a,-b) * S(sx,sy) * T(a,b)



3D Transformations

- Same idea as 2D transformations
  - Homogeneous coordinates: (x,y,z,w)
  - 4x4 transformation matrices

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Basic 3D Transformations

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Identity

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Mirror over X axis

Basic 3D Transformations

Rotate around Z axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 & 0 \\ \sin\Theta & \cos\Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotate around Y axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} \cos\Theta & 0 & -\sin\Theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\Theta & 0 & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

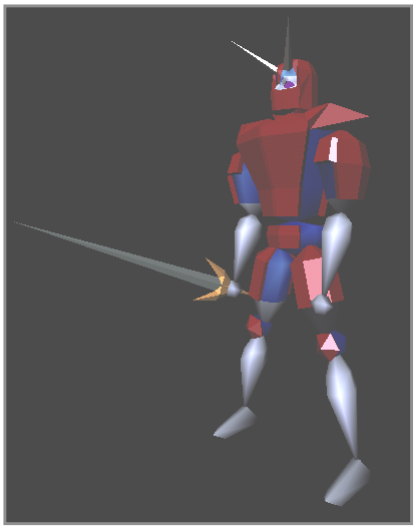
Rotate around X axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta & -\sin\Theta & 0 \\ 0 & \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

OpenGL Matrix Stacks

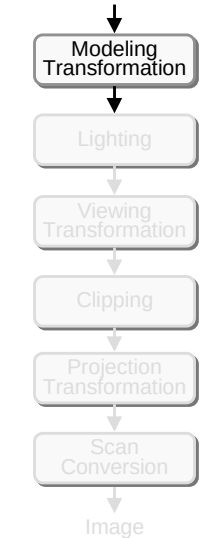
- OpenGL stores stacks of 4x4 matrices
  - GL_MODELVIEW
  - GL_PROJECTION
  - GL_TEXTURE
- OpenGL calls to push, pop, multiply top of stack
  - glLoadMatrix
  - glMultMatrix
  - glPushMatrix
  - glPopMatrix
- All vertices are multiplied by top of stack

3D Transformation Example



Mike Marr
COS 426, 1995

Summary



3D Rendering implemented
as a pipelined sequence of steps

Summary

- Representations of transformations
  - 4x4 Matrices
  - Homogeneous coordinates
- Types of transformations
  - Linear transformations
  - Affine transformations
  - Projective transformations
  - Others (deformations)
- Composition of transformations
  - Transformation hierarchies
  - Order matters