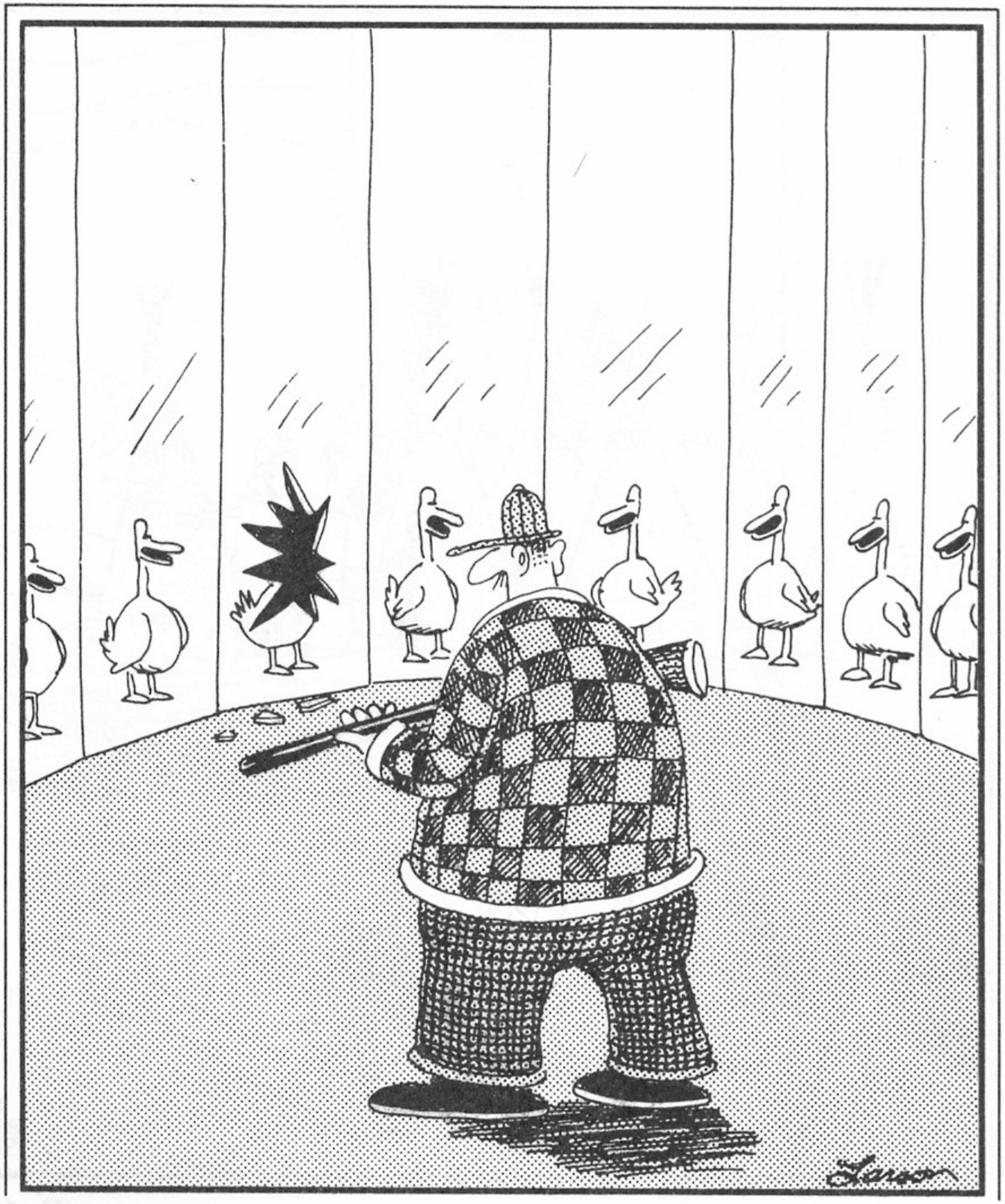
Transformations



"Ah, yes, Mr. Frischberg, I thought you'd come... but which of us is the *real* duck, Mr. Frischberg, and not just an illusion?"

Last Time?

- Ray representation
- Generating rays from eye point / camera
  - orthographic camera
  - perspective camera
- Find intersection point & surface normal
- Primitives:
  - spheres, planes, polygons, triangles, boxes

Assignment 0 – main issues

- Respect specifications!

- Don't put too much in the main function
- Use object-oriented design
  - Especially since you will have to build on this code
- Perform good memory management
  - Use new and delete

- Avoid unnecessary temporary variables
- Use enough precision for random numbers
- Sample a distribution using cumulative probability

Outline

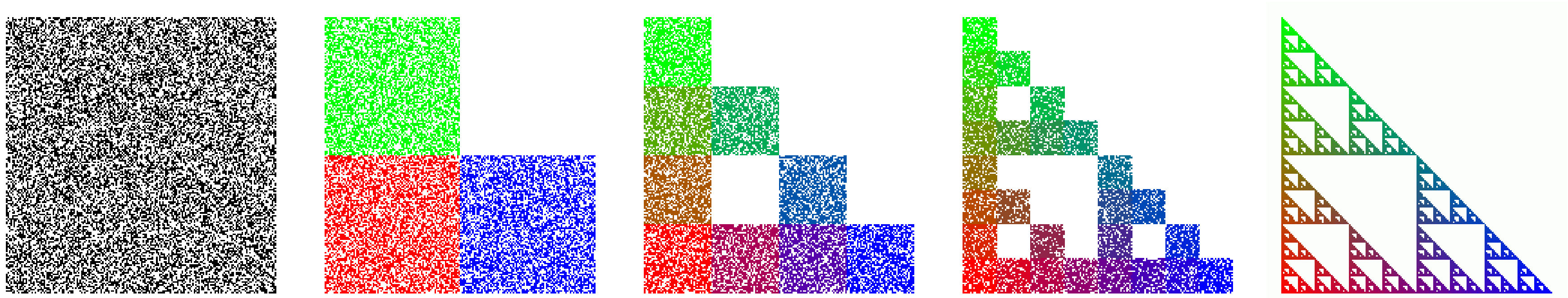
- Intro to Transformations
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modeling
- Adding Transformations to our Ray Tracer

What is a Transformation?

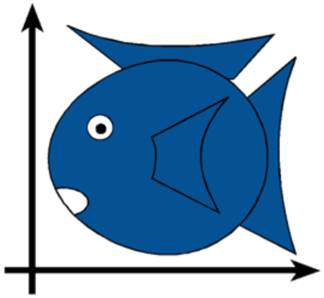
- Maps points (x, y) in one coordinate system to points (x', y') in another coordinate system

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

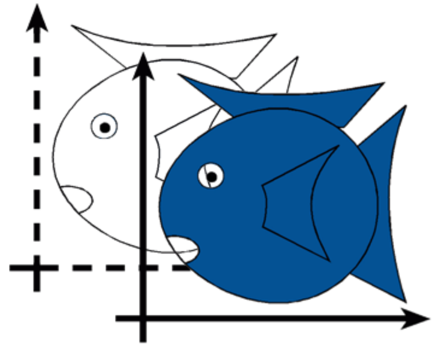
- For example, IFS:



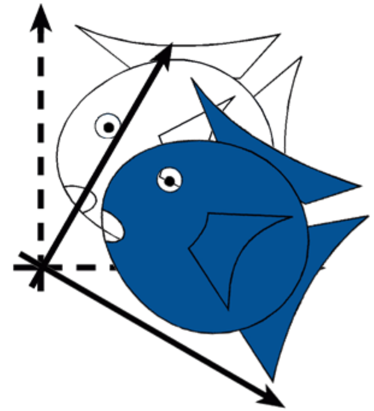
Simple Transformations



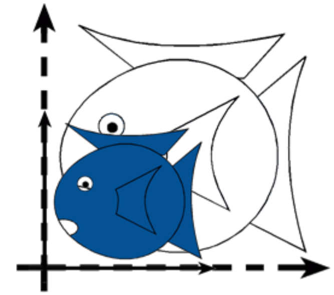
Identity



Translation



Rotation



Isotropic (Uniform) Scaling

- Can be combined
- Are these operations invertible?

  Yes, except scale = 0

Transformations are used:

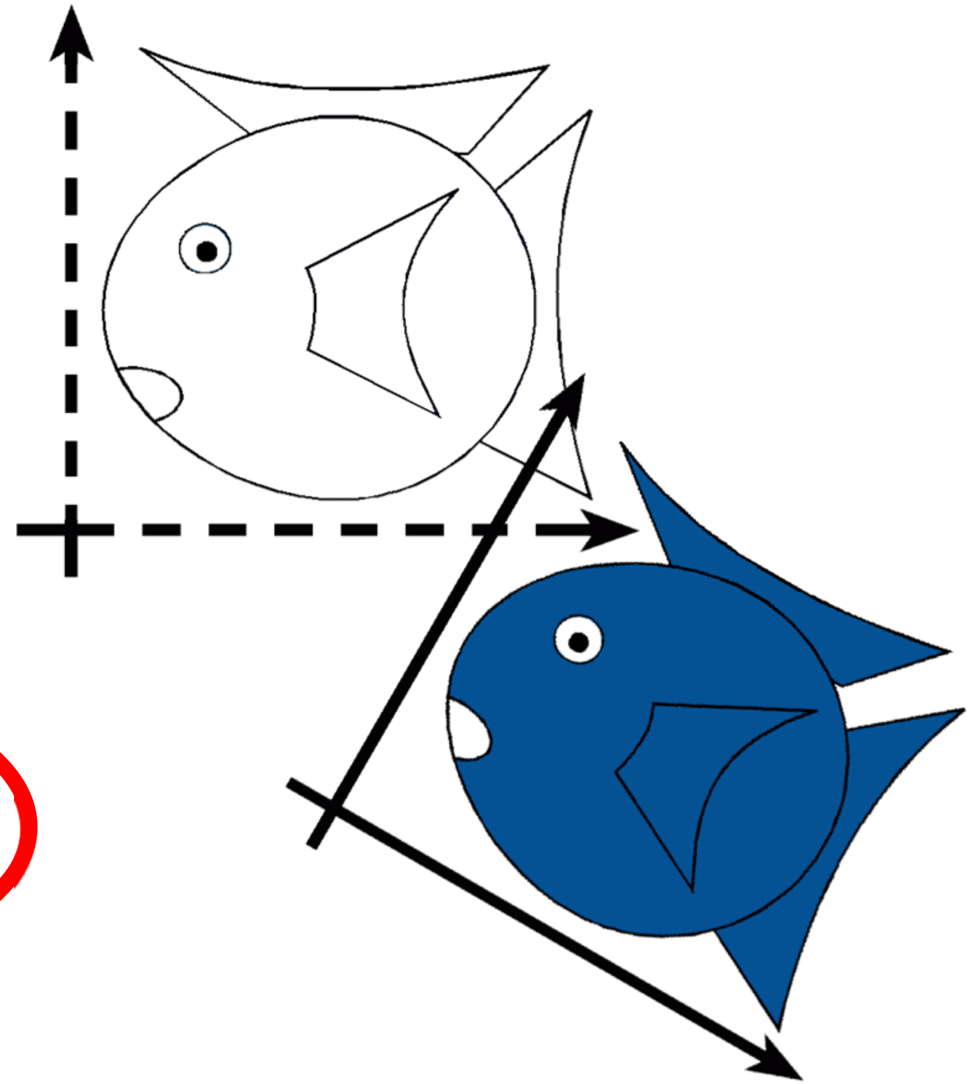
- Position objects in a scene (modeling)
- Change the shape of objects
- Create multiple copies of objects
- Projection for virtual cameras
- Animations

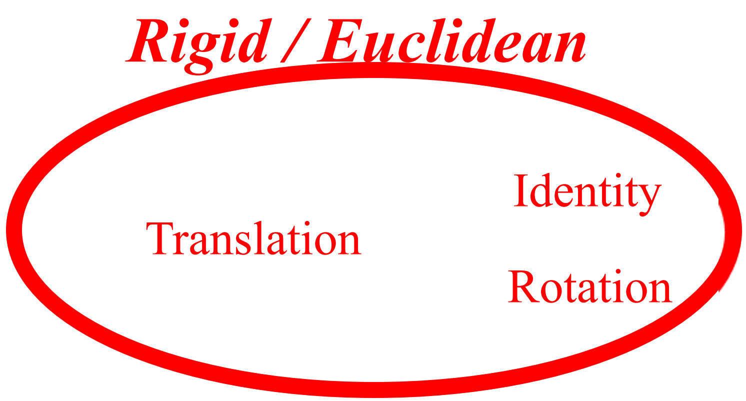
Outline

- Intro to Transformations
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modeling
- Adding Transformations to our Ray Tracer

Rigid-Body / Euclidean Transforms

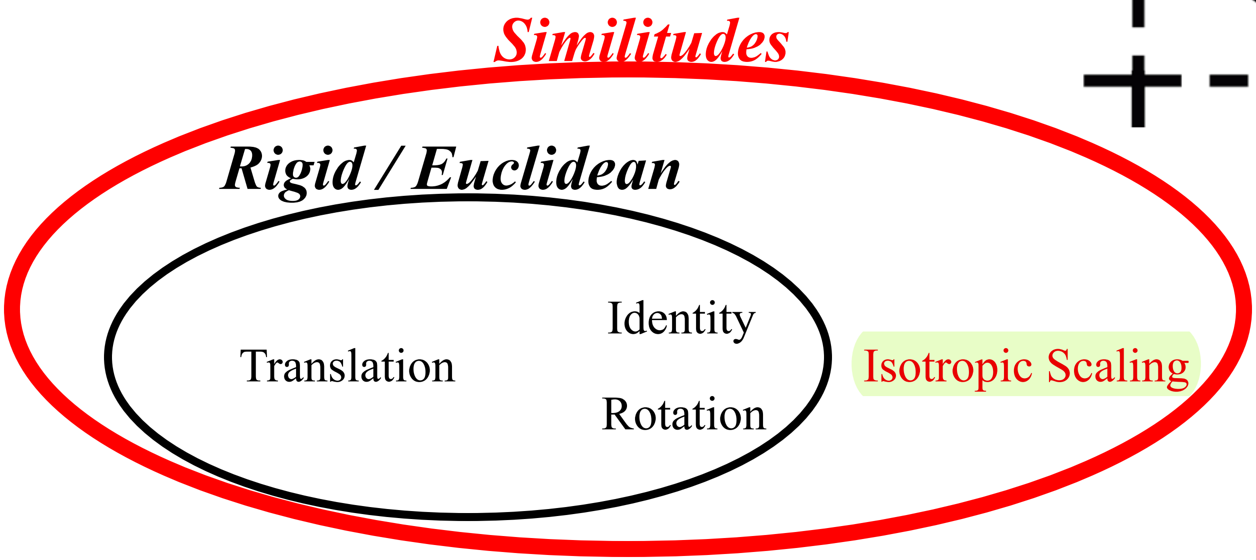
- Preserves distances
- Preserves angles

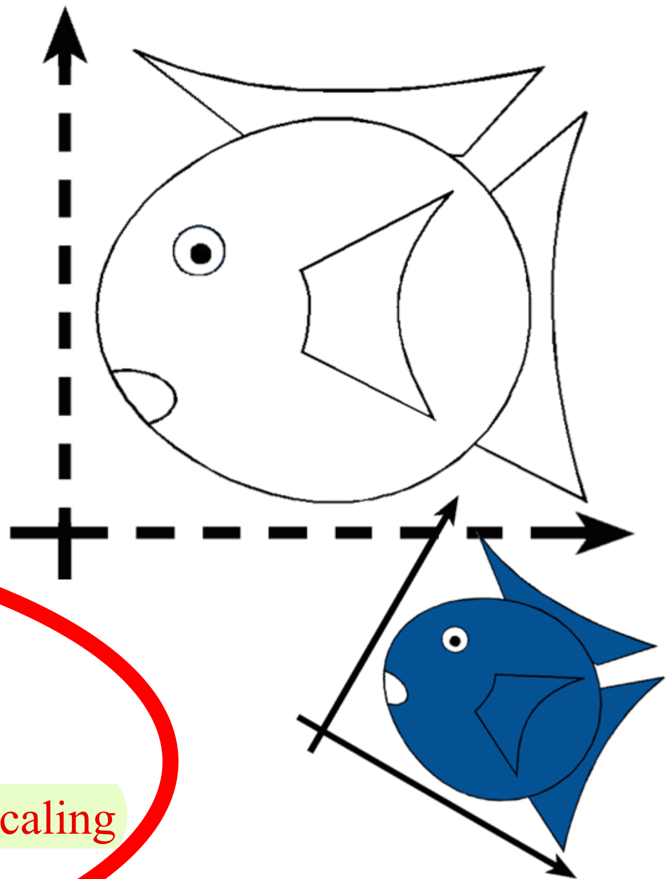




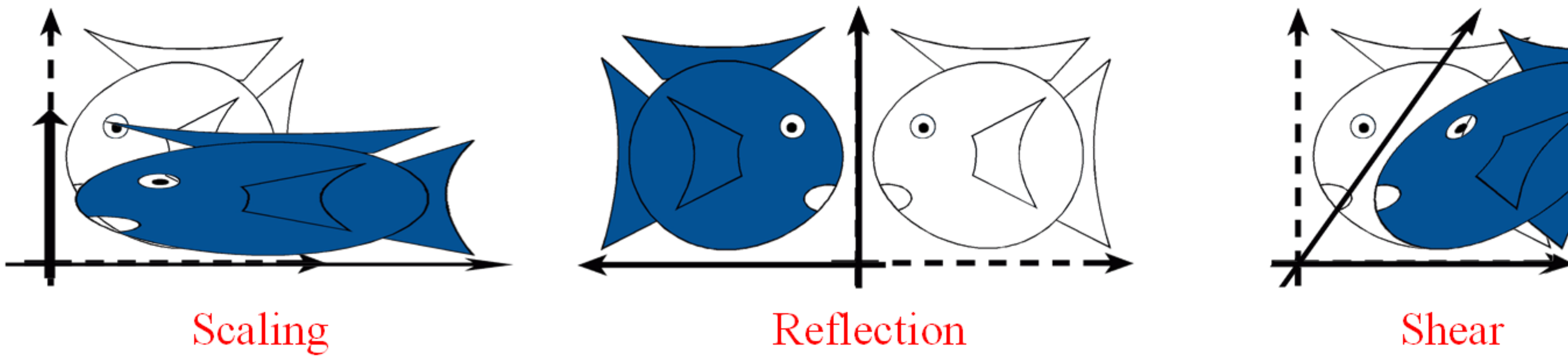
Similitudes / Similarity Transforms

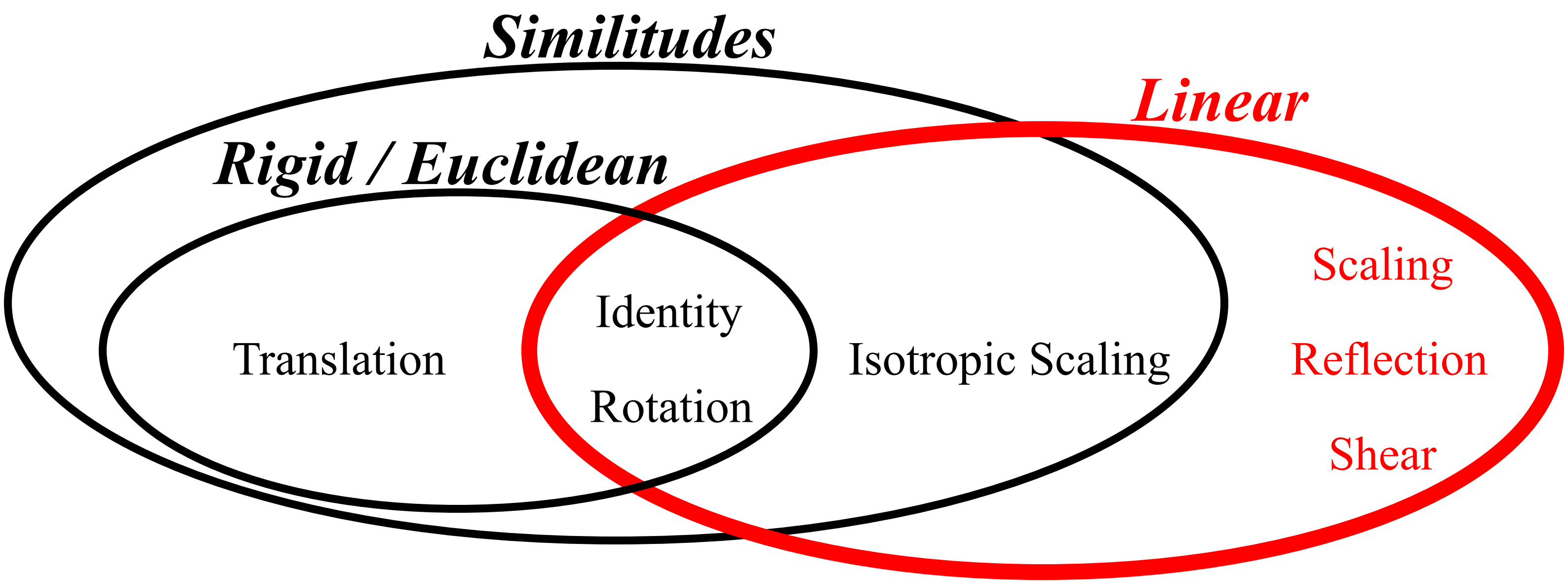
- Preserves angles





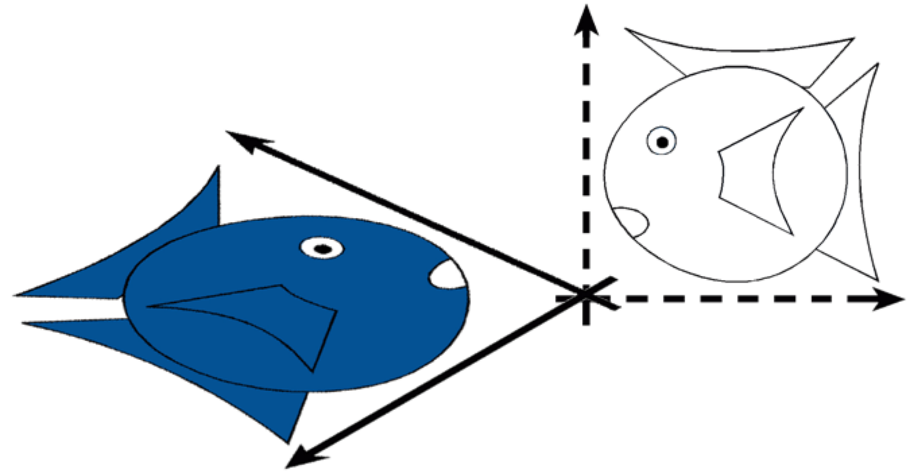
Linear Transformations

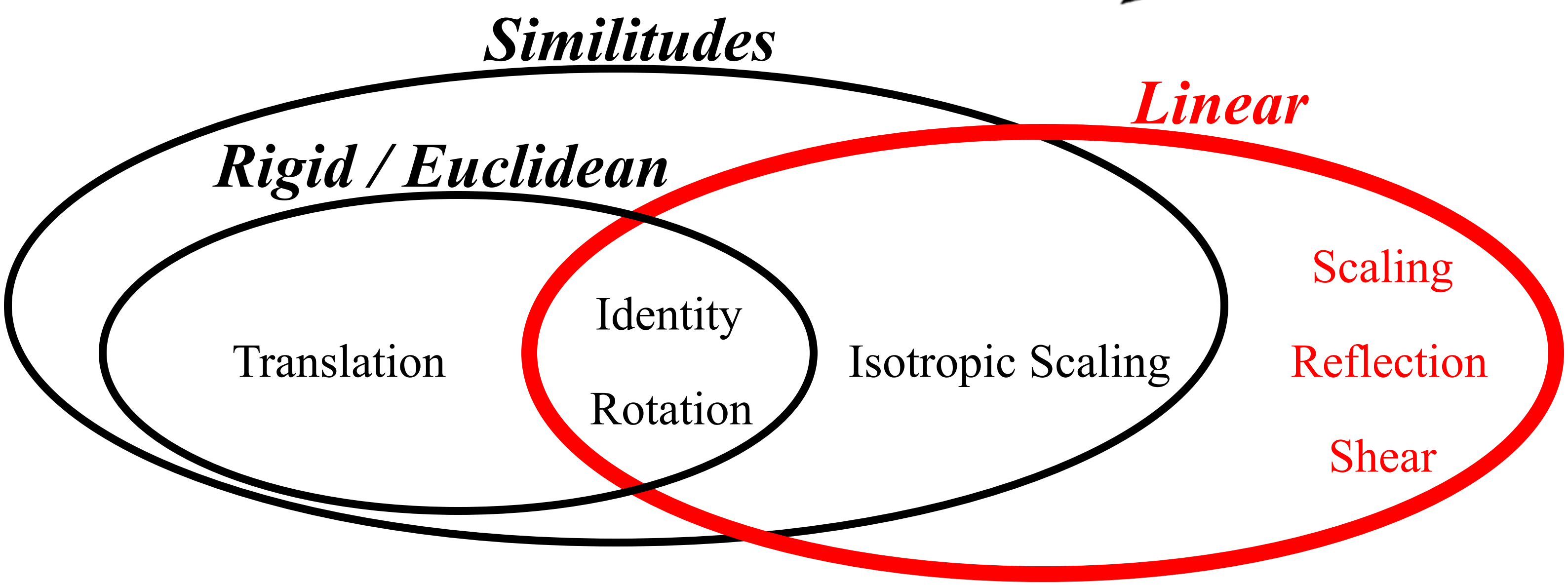




Linear Transformations

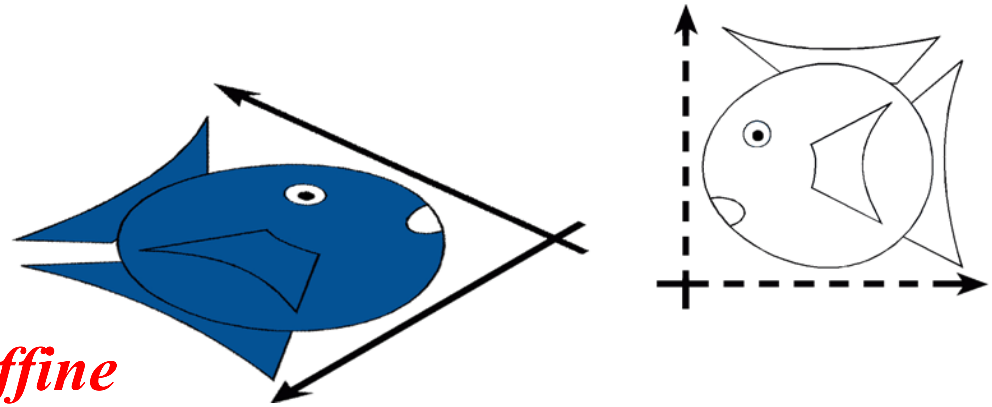
- $L(p + q) = L(p) + L(q)$
- $L(ap) = a\,L(p)$

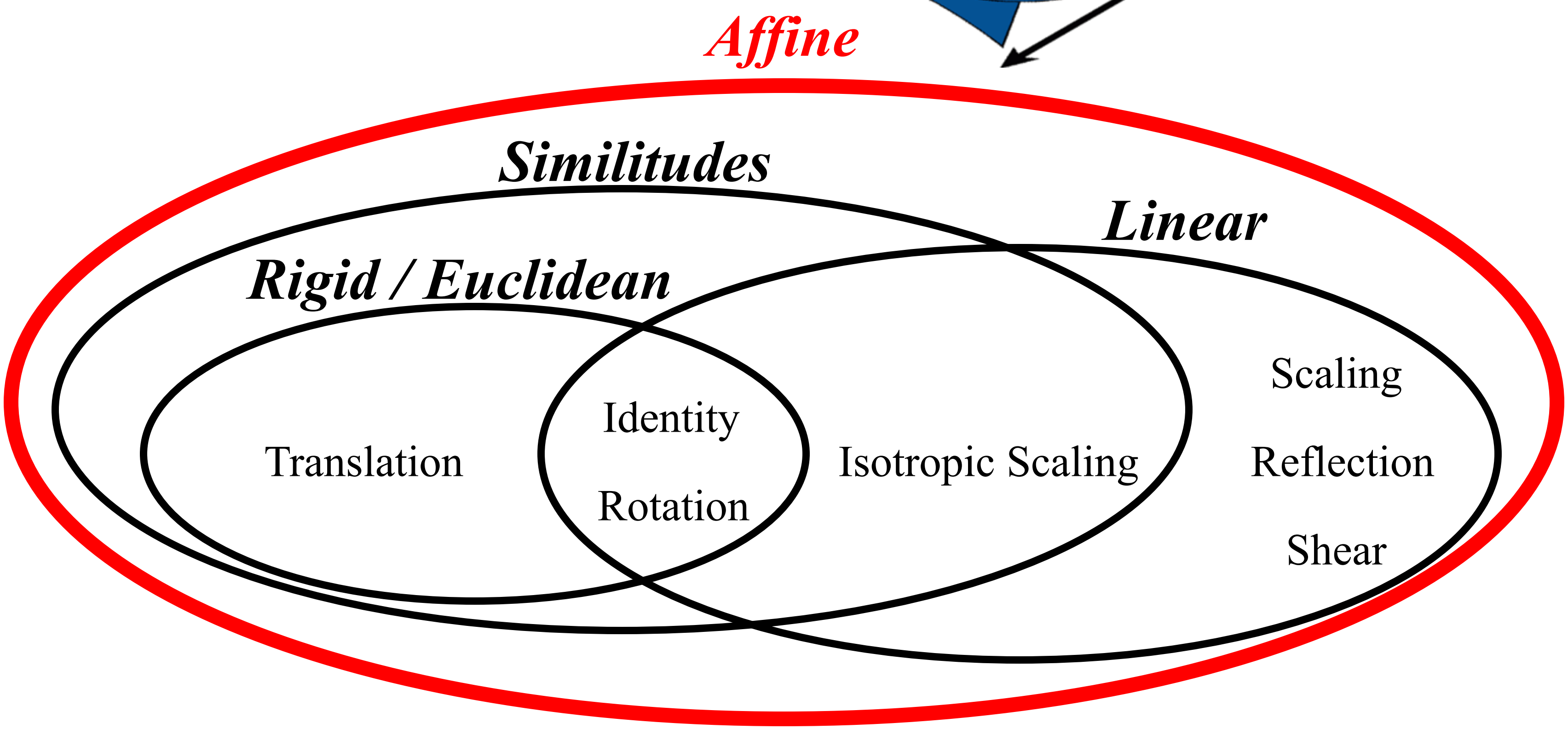




Affine Transformations

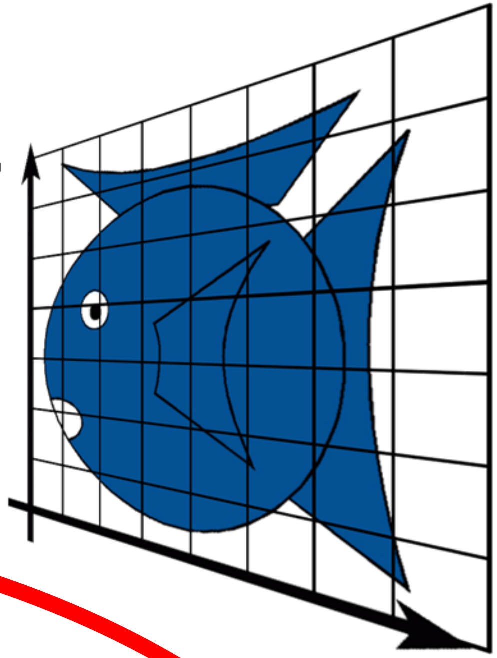
- preserves parallel lines

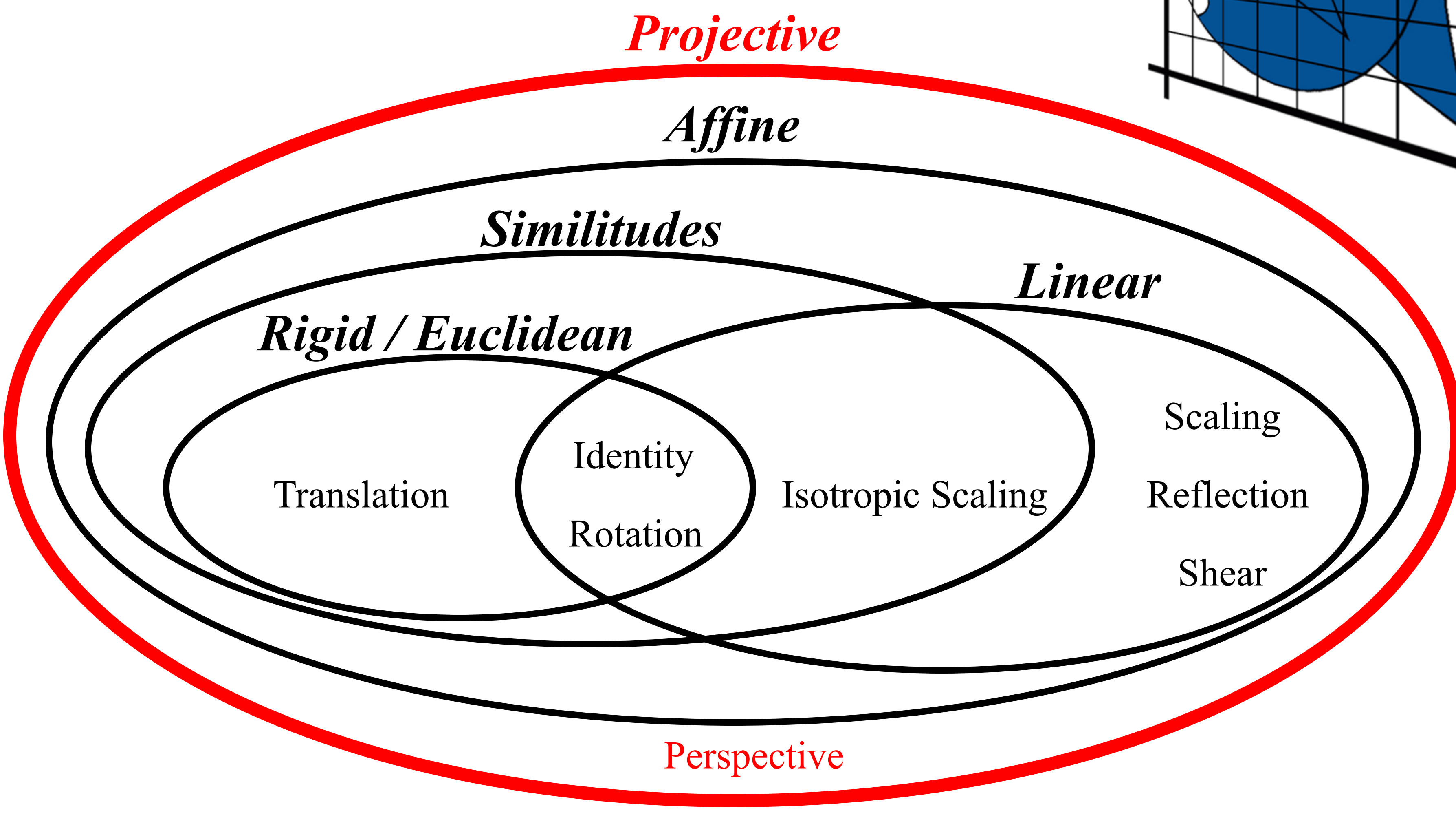




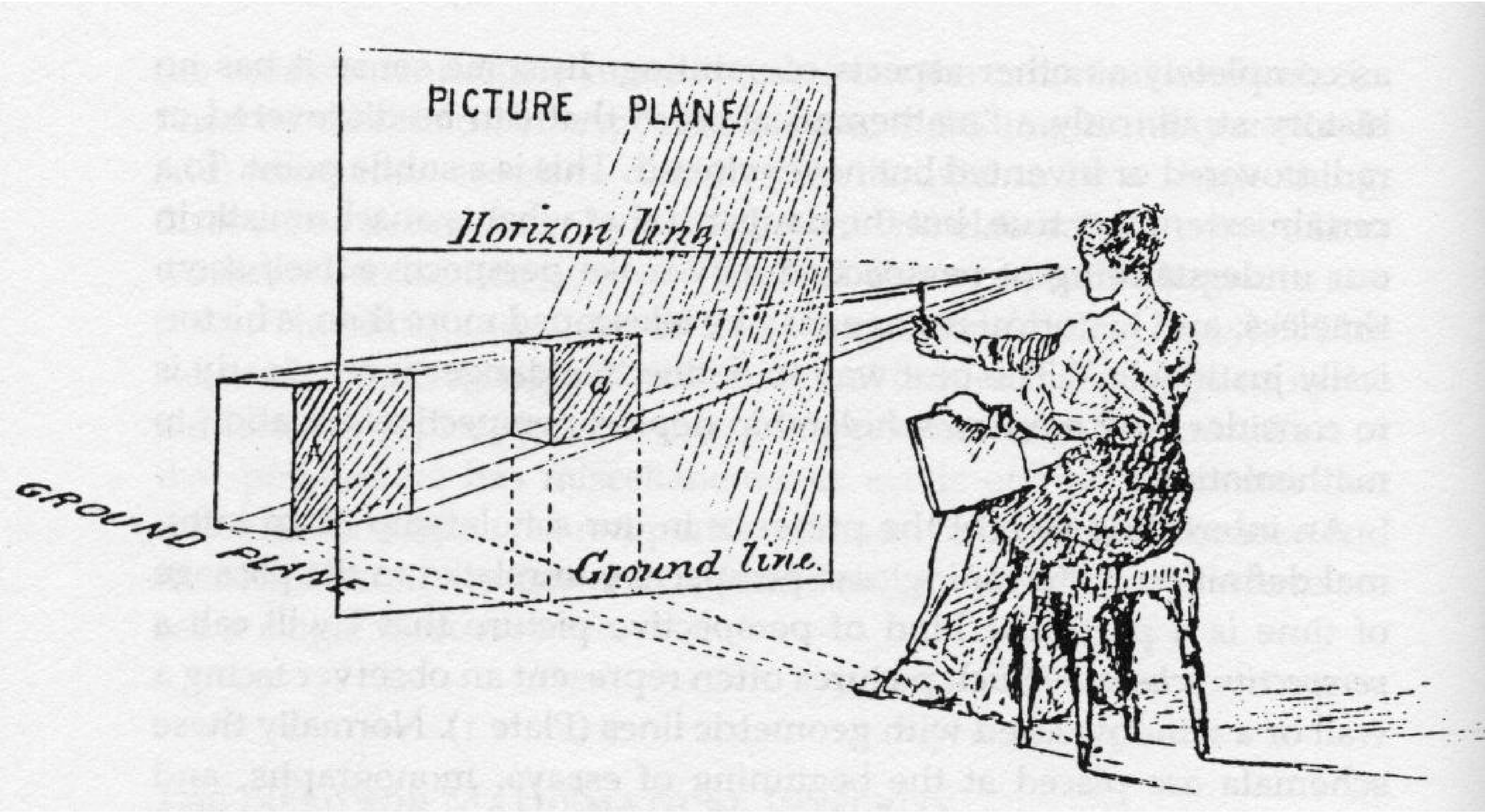
Projective Transformations

- preserves lines





Perspective Projection



General (free-form) transformation

- Does not preserve lines
- Not as pervasive, computationally more involved
- Won't be treated in this course

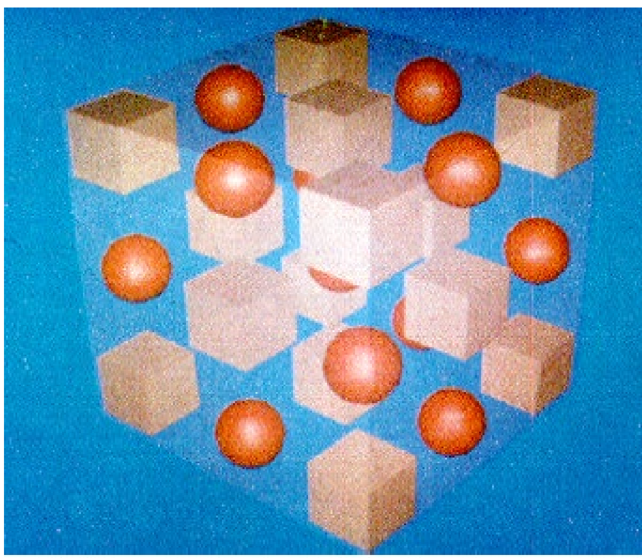


Fig 1. Undeformed Plastic

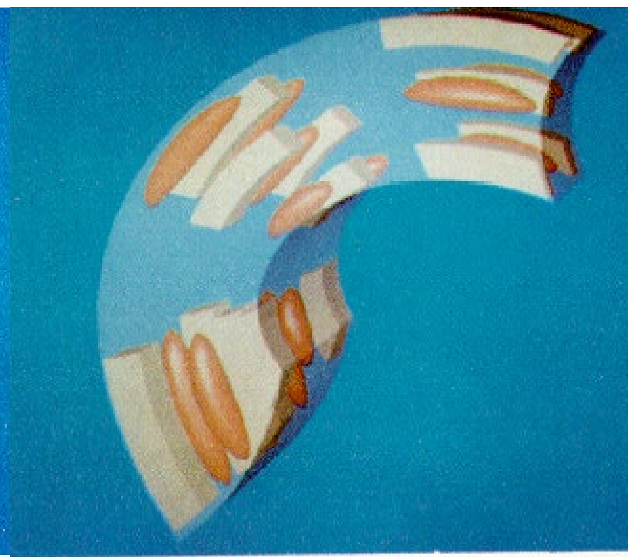


Fig 2. Deformed Plastic

From Sederberg and Parry, Siggraph 1986

Outline

- Intro to Transformations
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modeling
- Adding Transformations to our Ray Tracer

How are Transforms Represented?

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

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

$$p' = M p + t$$

Homogeneous Coordinates

- Add an extra dimension
  - in 2D, we use 3 x 3 matrices
  - In 3D, we use 4 x 4 matrices
- Each point has an extra value, *w*

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

$$p' = \color{red}{M}\, p$$

Translation in homogenous coordinates

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

Affine formulation

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

$$p' = M p + t$$

Homogeneous formulation

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

$$p' = M p$$

Homogeneous Coordinates

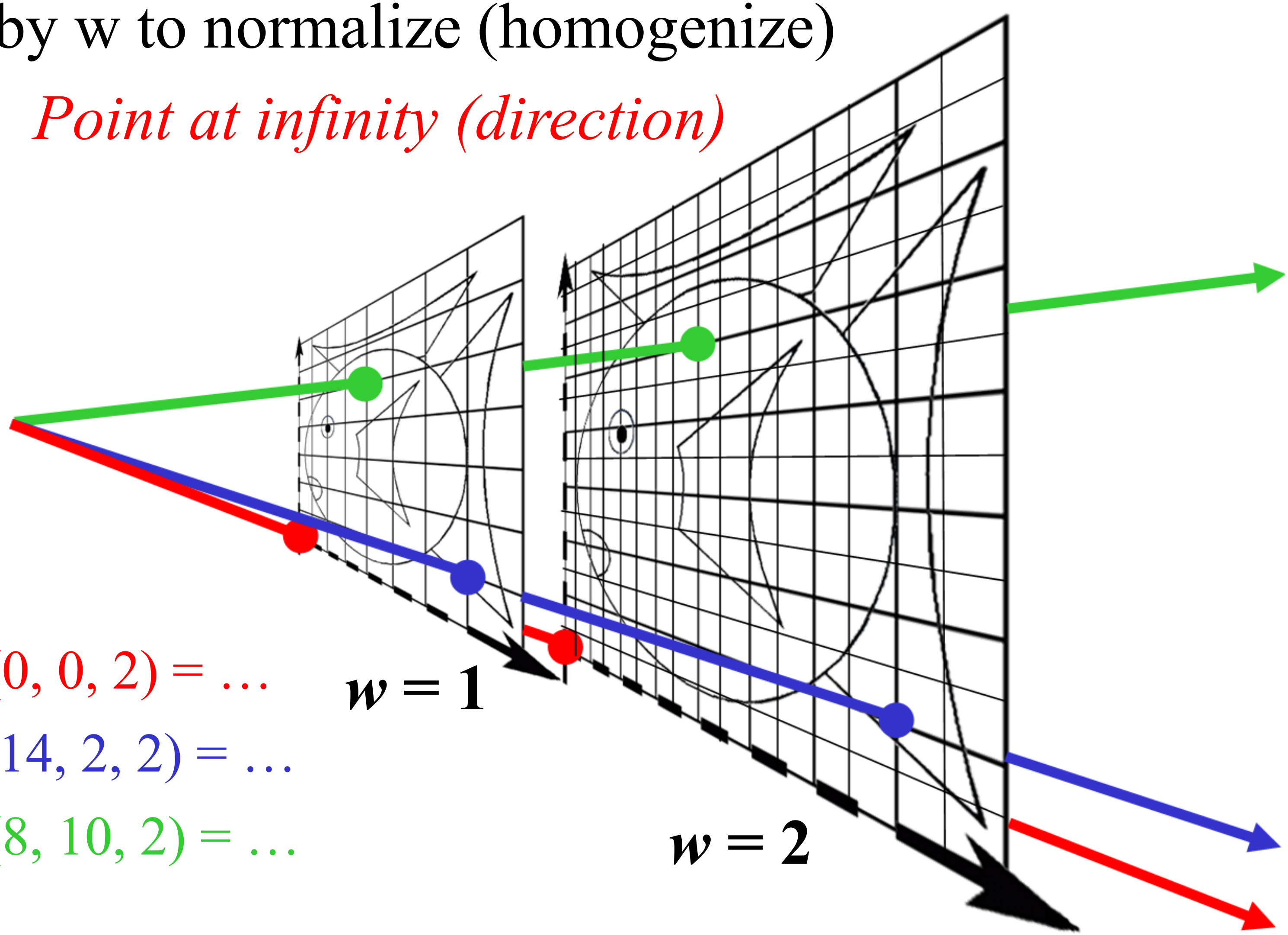
- Most of the time w = 1, and we can ignore it

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- If we multiply a homogeneous coordinate by an *affine matrix*, w is unchanged

Homogeneous Visualization

- Divide by w to normalize (homogenize)
- W = 0? *Point at infinity (direction)*



$(0, 0, 1) = (0, 0, 2) = \ldots$

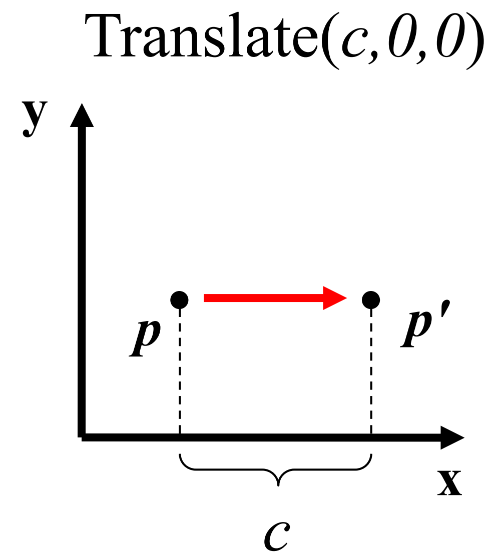
$(7, 1, 1) = (14, 2, 2) = \ldots$

$(4, 5, 1) = (8, 10, 2) = \ldots$

Translate (t_x, t_y, t_z)

- Why bother with the extra dimension?

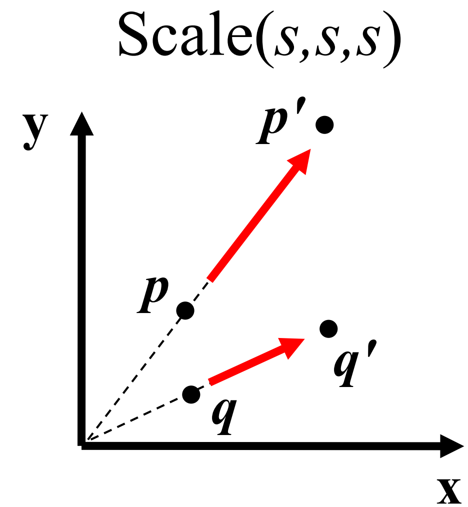
  Because now translations can be encoded in the matrix!



$$
\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}
$$

Scale (s_x, s_y, s_z)

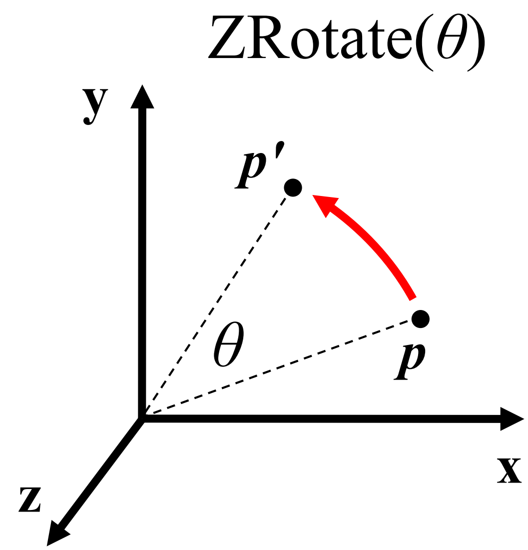
- Isotropic (uniform) scaling: $s_x = s_y = s_z$



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotation

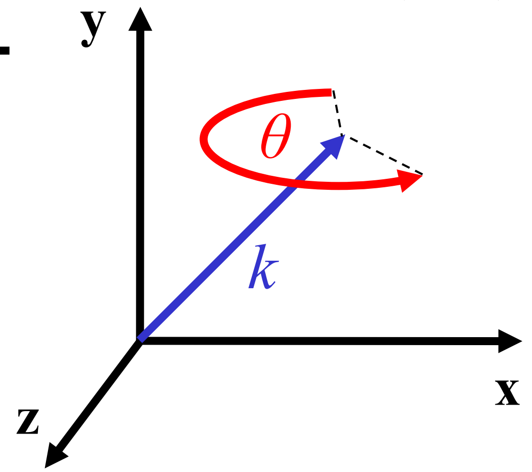
- About z axis



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotation

Rotate(k, θ)



- About (k_x, k_y, k_z), a unit vector on an arbitrary axis (Rodrigues Formula)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} k_x k_x(1-c)+c & k_z k_x(1-c)-k_z s & k_x k_z(1-c)+k_y s & 0 \\ k_y k_x(1-c)+k_z s & k_z k_x(1-c)+c & k_y k_z(1-c)-k_x s & 0 \\ k_z k_x(1-c)-k_y s & k_z k_x(1-c)-k_x s & k_z k_z(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

where $c = \cos\theta$ & $s = \sin\theta$

Storage

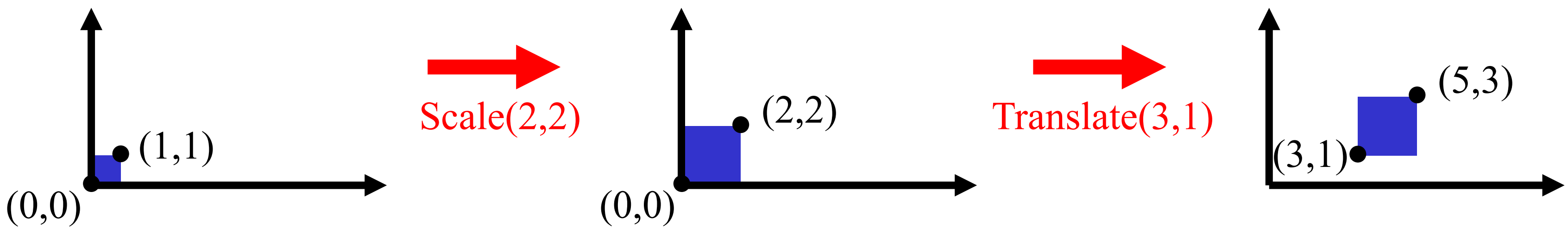
- Often, *w* is not stored (always 1)
- Needs careful handling of direction vs. point
  - Mathematically, the simplest is to encode directions with w=0
  - In terms of storage, using a 3-component array for both direction and points is more efficient
  - Which requires to have special operation routines for points vs. directions

Outline

- Intro to Transformations
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modeling
- Adding Transformations to our Ray Tracer

How are transforms combined?

Scale then Translate



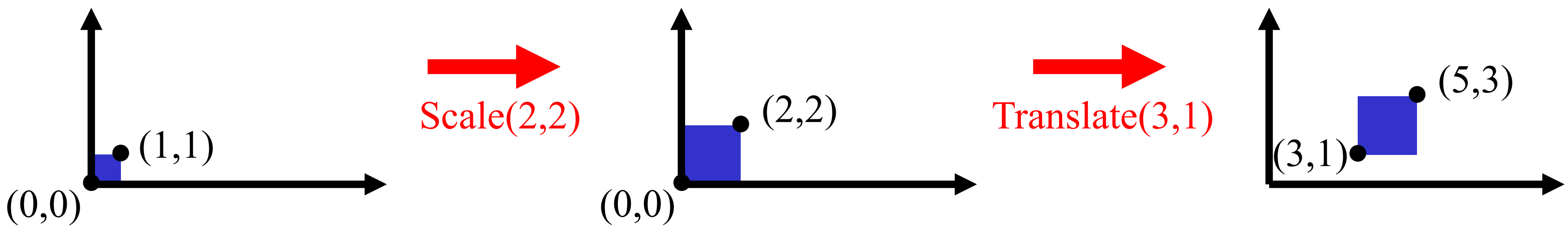
Use matrix multiplication: p' = T (S p) = TS p

$$TS = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

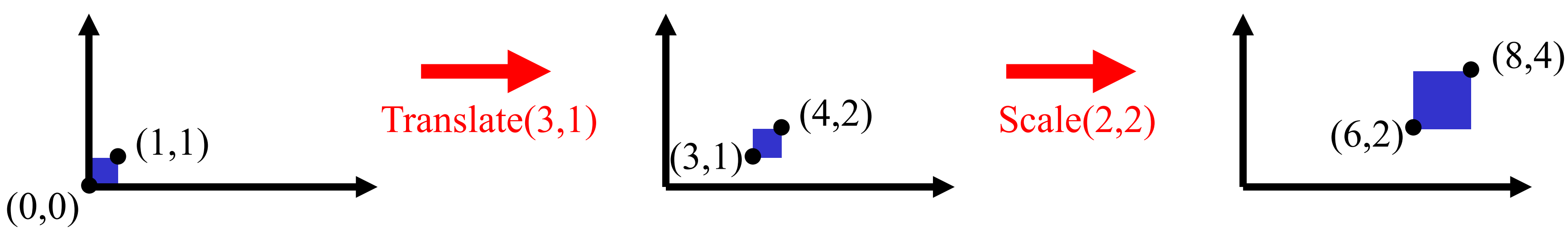
Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

Scale then Translate: $p' = T(S\,p) = TS\,p$



Translate then Scale: $p' = S(T\,p) = ST\,p$



Non-commutative Composition

Scale then Translate: p' = T (S p) = TS p

$$TS = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Translate then Scale: p' = S (T p) = ST p

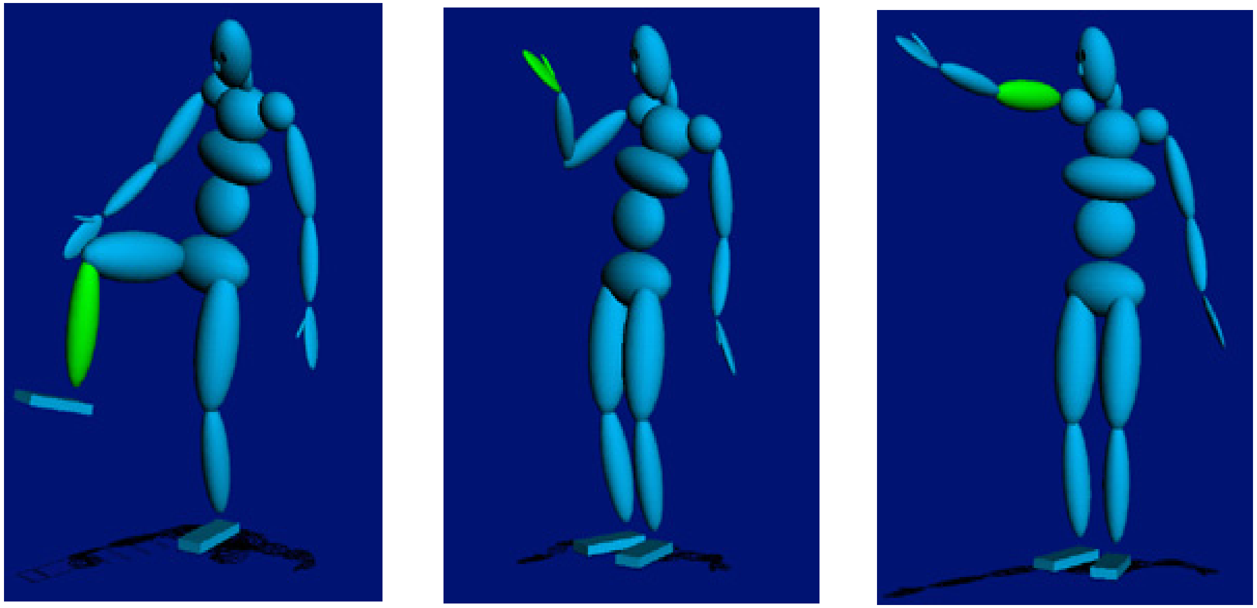
$$ST = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

Today

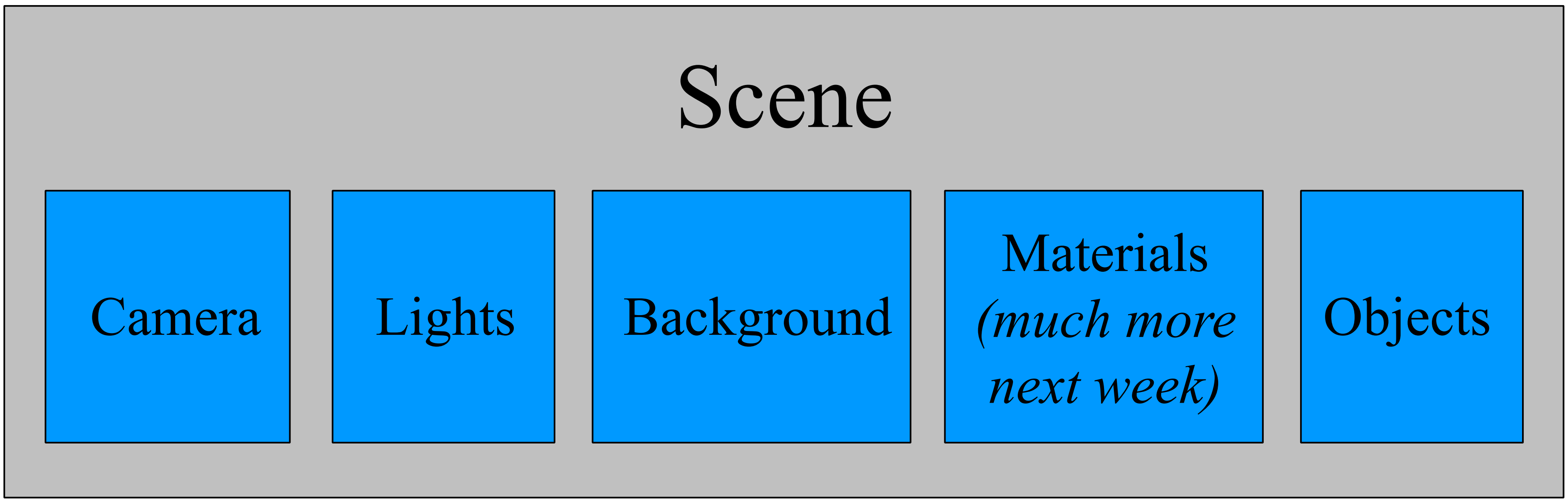
- Intro to Transformations
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modeling
- Adding Transformations to our Ray Tracer

Transformations in Modeling

- Position objects in a scene
- Change the shape of objects
- Create multiple copies of objects
- Projection for virtual cameras
- Animations



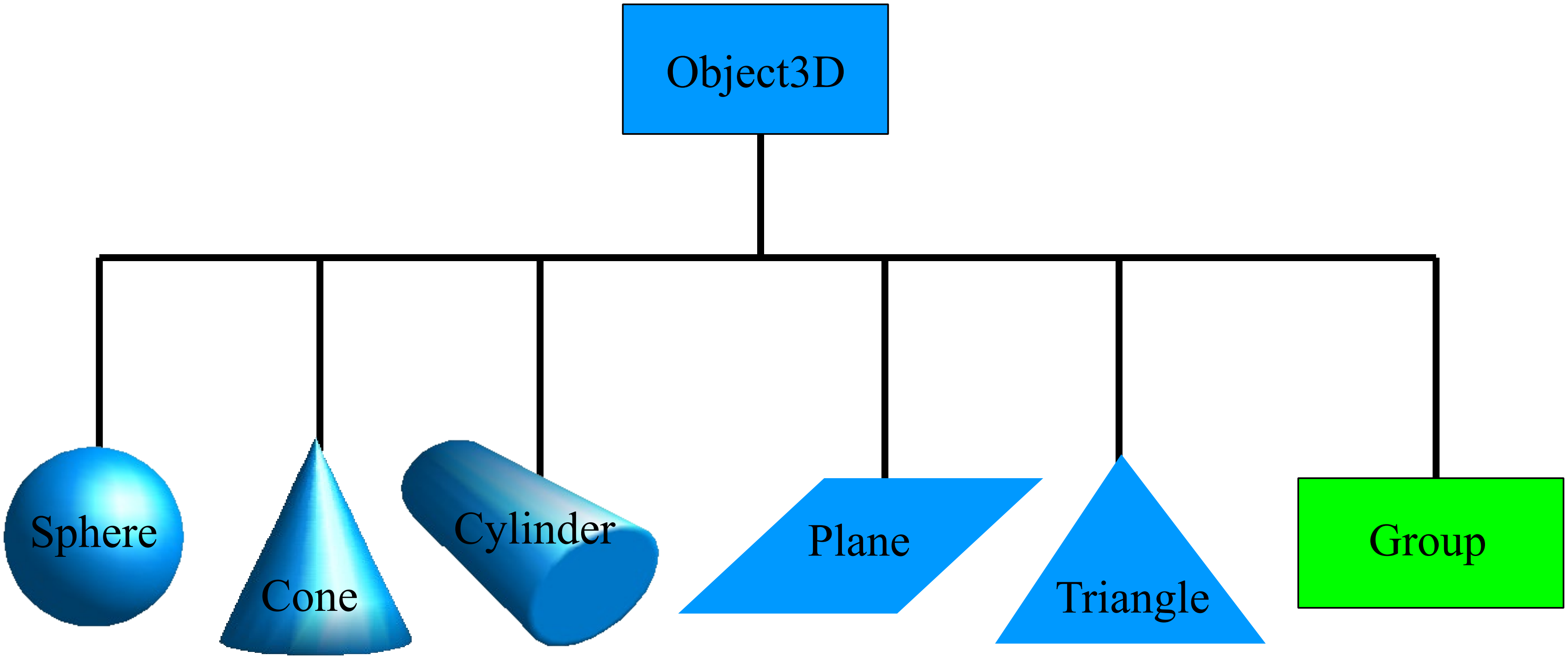
Scene Description



Simple Scene Description File

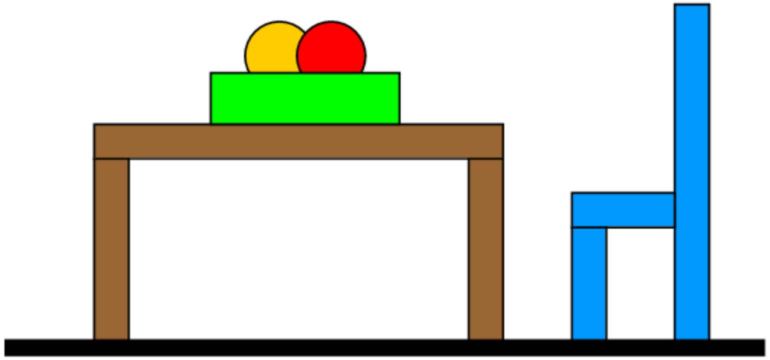
```
OrthographicCamera {
    center 0 0 10
    direction 0 0 -1
    up 0 1 0
    size 5  }

Lights {
    numLights 1
    DirectionalLight {
        direction -0.5 -0.5 -1
        color 1 1 1 } }

Background { color 0.2 0 0.6 }

Materials {
    numMaterials <n>
 <MATERIALS> }

Group {
    numObjects <n>
    <OBJECTS> }
```

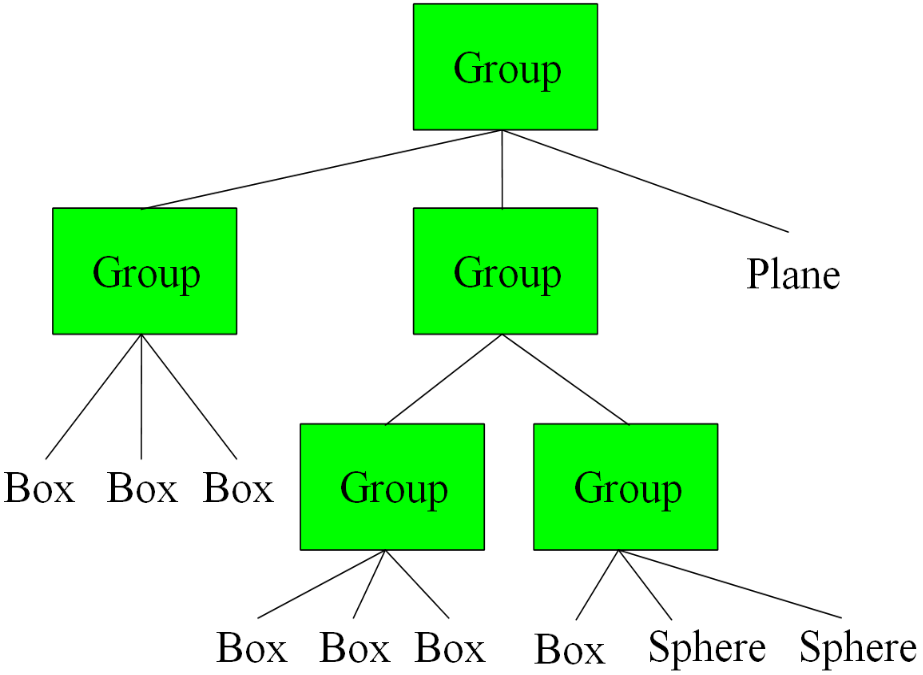
Class Hierarchy



Why is a Group an Object3D?

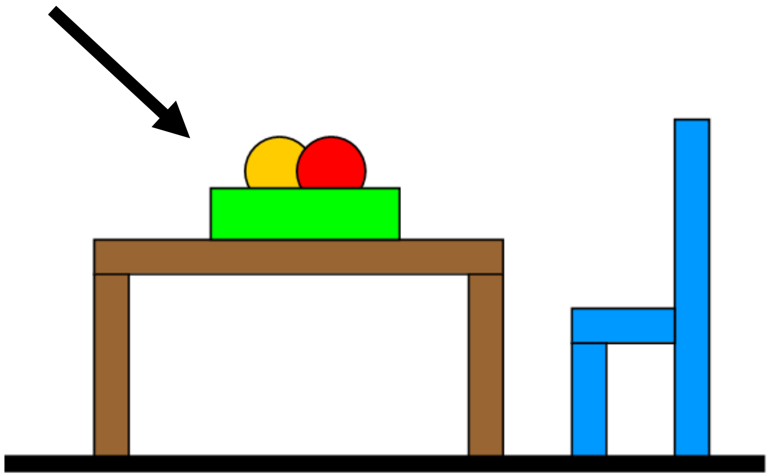
- Logical organization of scene

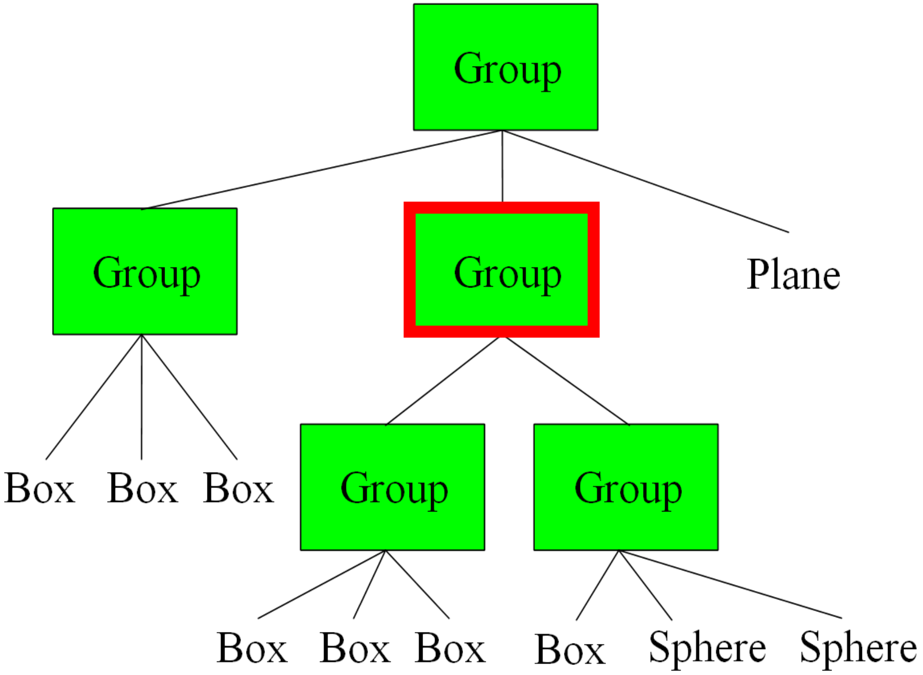




Ray-group intersection

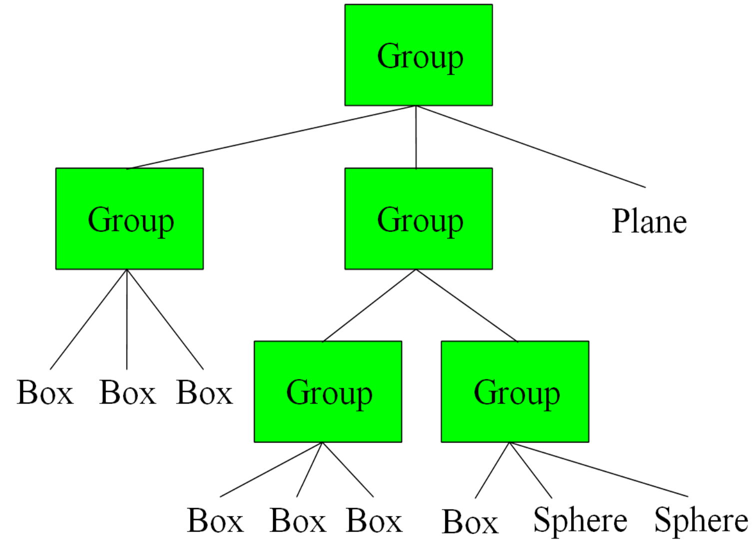
- Recursive on all sub-objects

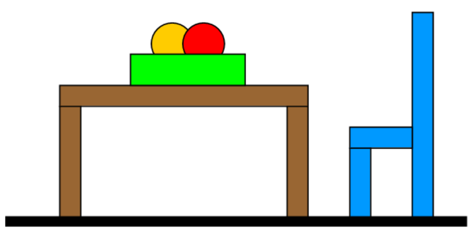




Simple Example with Groups

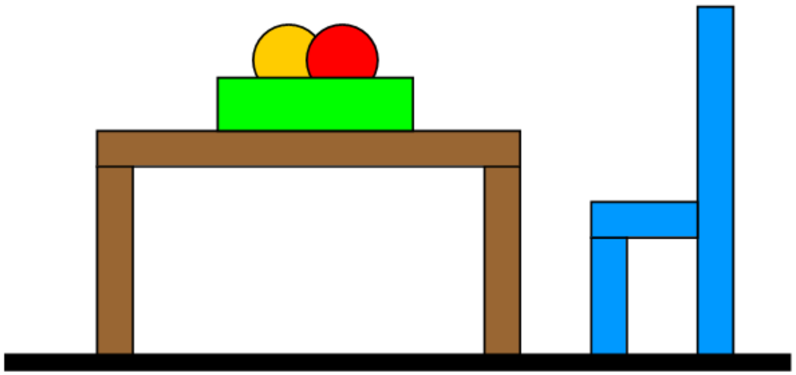
```
Group {
    numObjects 3
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> } }
    Group {
        numObjects 2
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } }
        Group {
            Box { <BOX PARAMS> }
            Sphere { <SPHERE PARAMS> }
            Sphere { <SPHERE PARAMS> } } }
    Plane { <PLANE PARAMS> } }
```



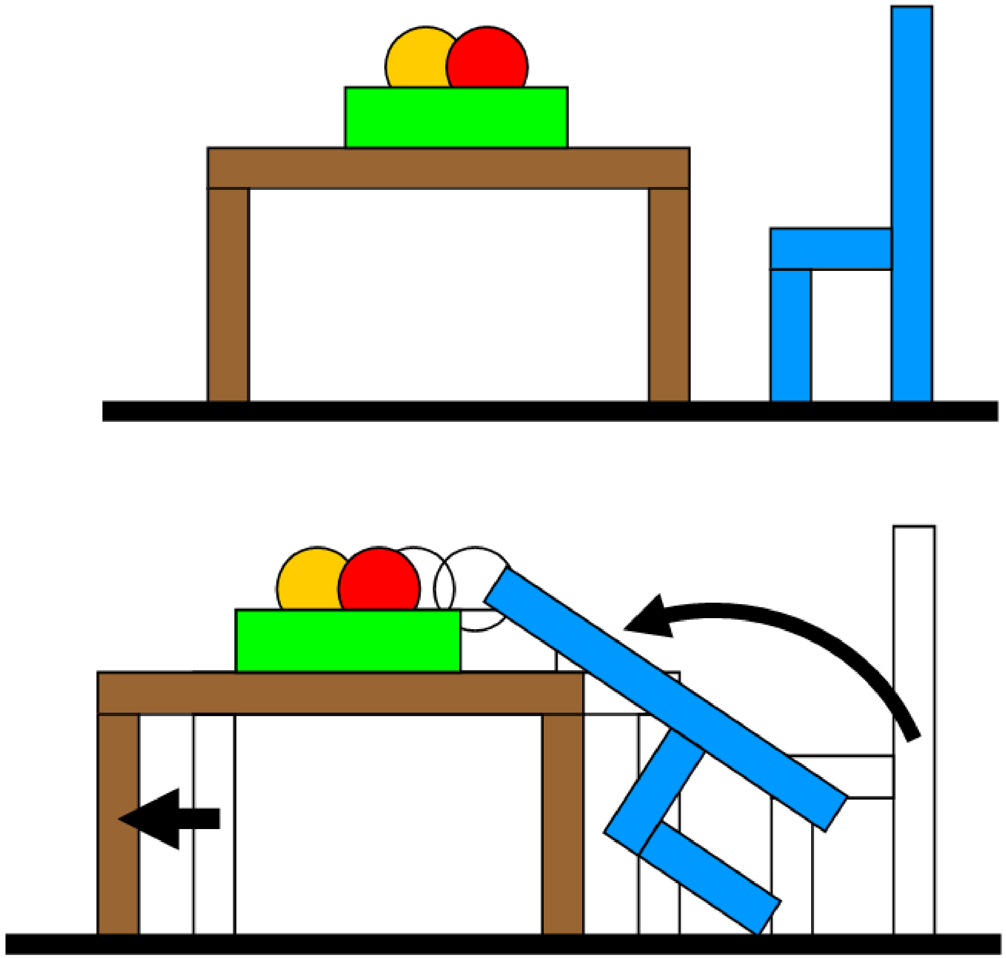


Adding Materials

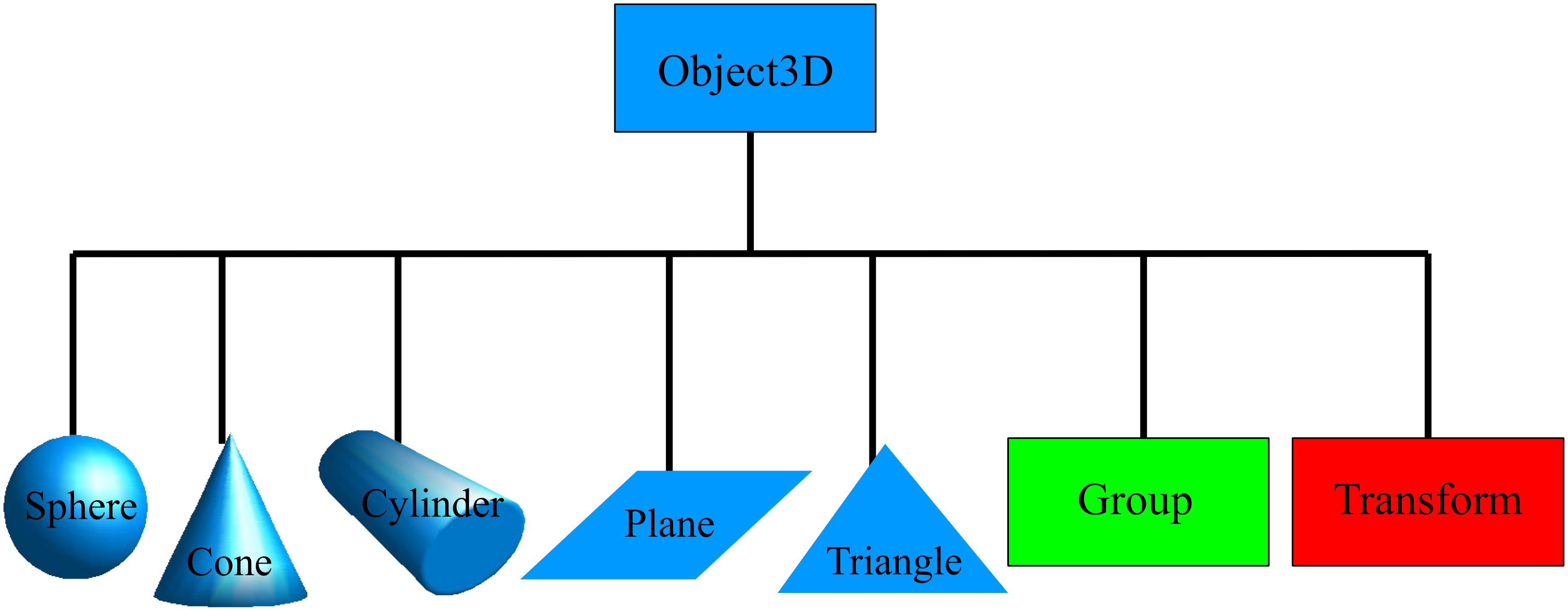
```
Group {
    numObjects 3
    Material { <BLUE> }
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> } }
    Group {
        numObjects 2
        Material { <BROWN> }
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } }
        Group {
            Material { <GREEN> }
            Box { <BOX PARAMS> }
            Material { <RED> }
            Sphere { <SPHERE PARAMS> }
            Material { <ORANGE> }
            Sphere { <SPHERE PARAMS> } } }
            Material { <BLACK> }
    Plane { <PLANE PARAMS> } }
```



Adding Transformations

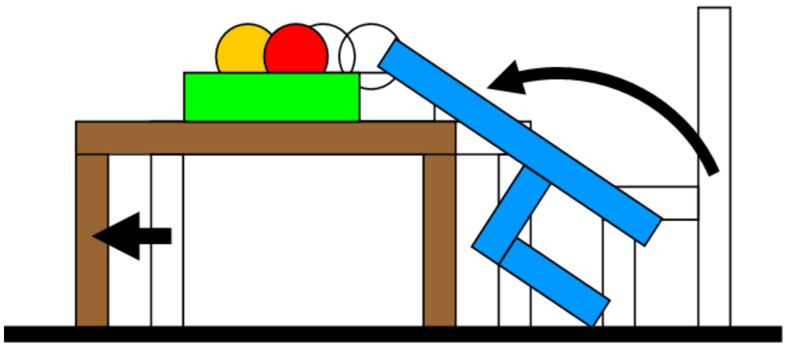


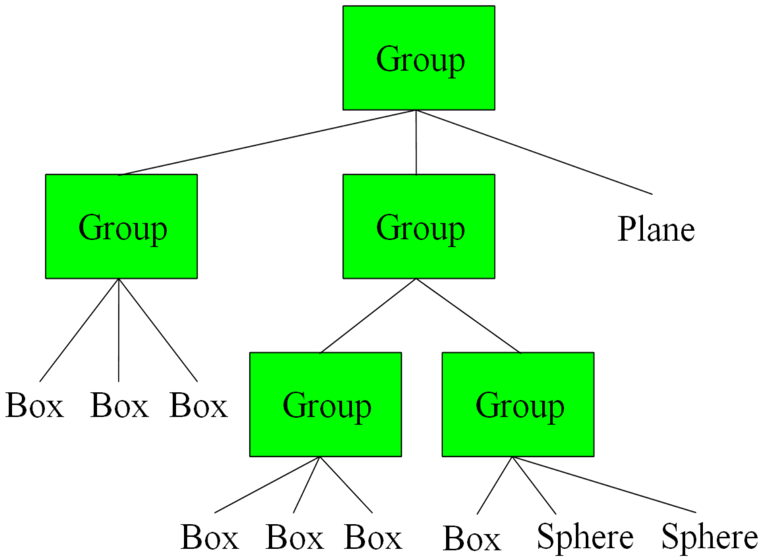
Class Hierarchy with Transformations

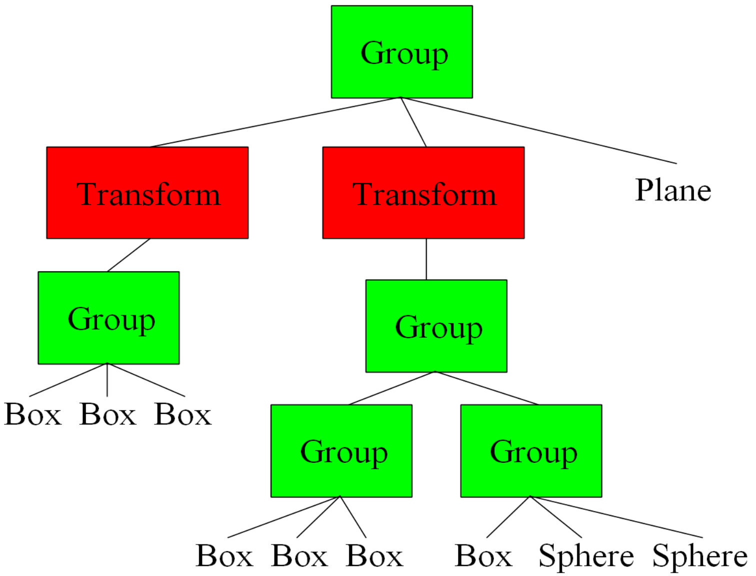


Why is a Transform an Object3D?

- To position the logical groupings of objects within the scene

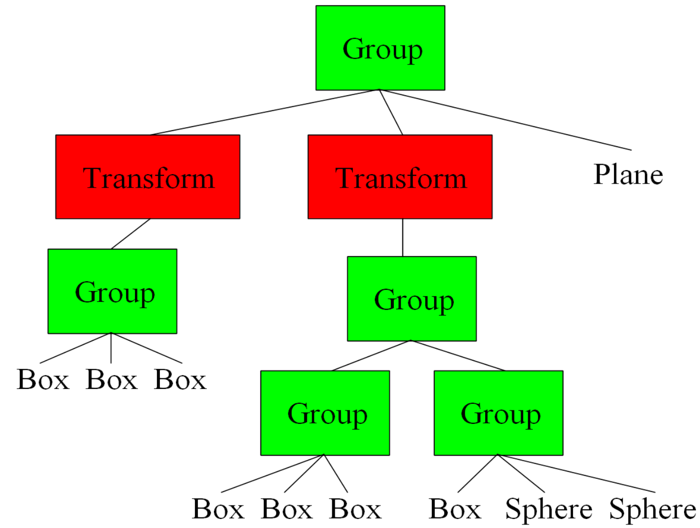






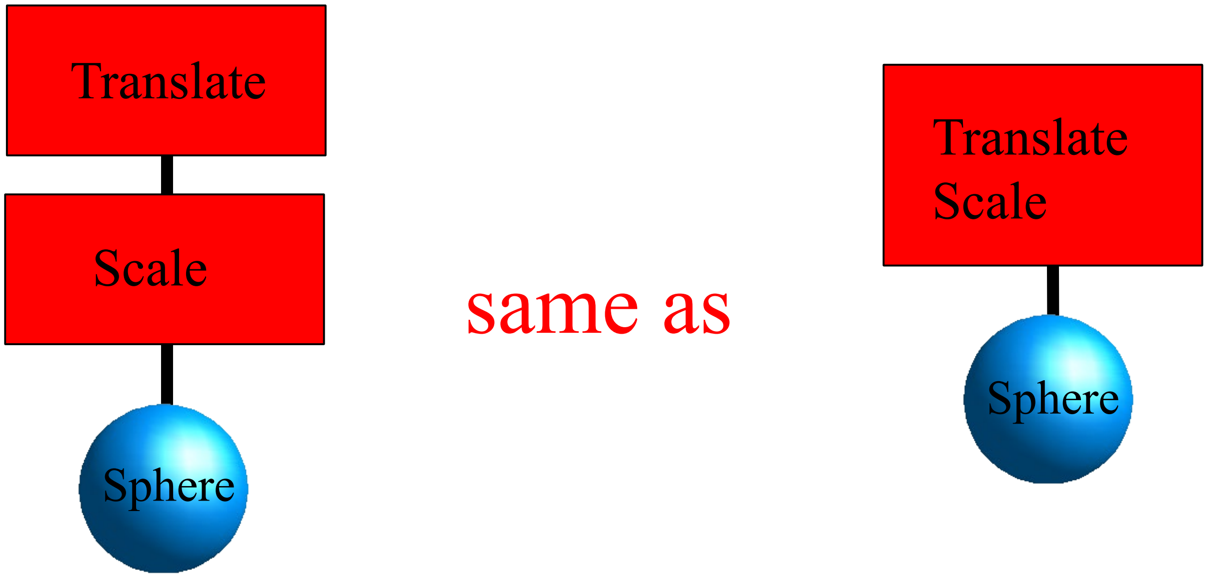
Simple Example with Transforms

```
Group {
     numObjects 3
     Transform {
          ZRotate { 45 }
          Group {
               numObjects 3
               Box { <BOX PARAMS> }
               Box { <BOX PARAMS> }
               Box { <BOX PARAMS> } } }
     Transform {
          Translate { -2 0 0 }
          Group {
               numObjects 2
               Group {
                    Box { <BOX PARAMS> }
                    Box { <BOX PARAMS> }
                    Box { <BOX PARAMS> } }
               Group {
                    Box { <BOX PARAMS> }
                    Sphere { <SPHERE PARAMS> }
                    Sphere { <SPHERE PARAMS> } } } }
     Plane { <PLANE PARAMS> } }
```



Nested Transforms

$$p' = T(S\,p) = TS\,p$$




```
Transform {
    Translate { 1 0.5 0 }
    Transform {
        Scale { 2 2 2 }
        Sphere {
            center 0 0 0
            radius 1 } } }
```

```
Transform {
    Translate { 1 0.5 0 }
    Scale { 2 2 2 }
    Sphere {
        center 0 0 0
        radius 1 } }
```