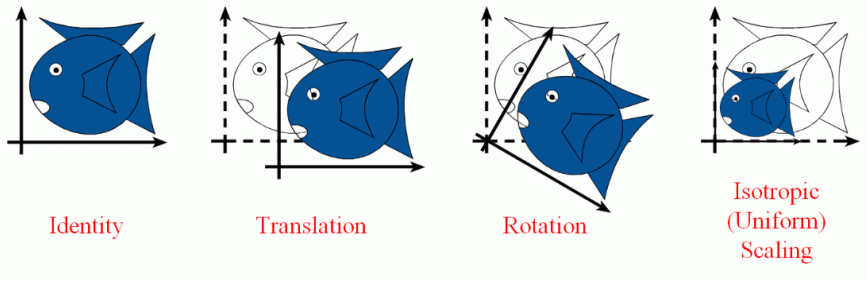
# Transformations in Ray Tracing

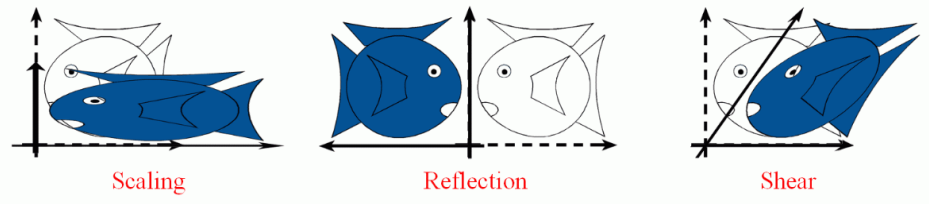
# Linear Algebra Review Session

- Tonight!
- Room 2-139
- 7:30 – 9 PM

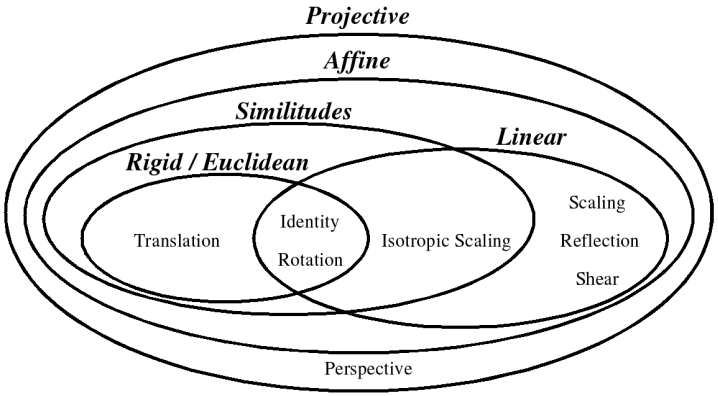
# Last Time:

- Simple Transformations




- Classes of Transformations
- Representation
  - homogeneous coordinates
- Composition
  - not commutative



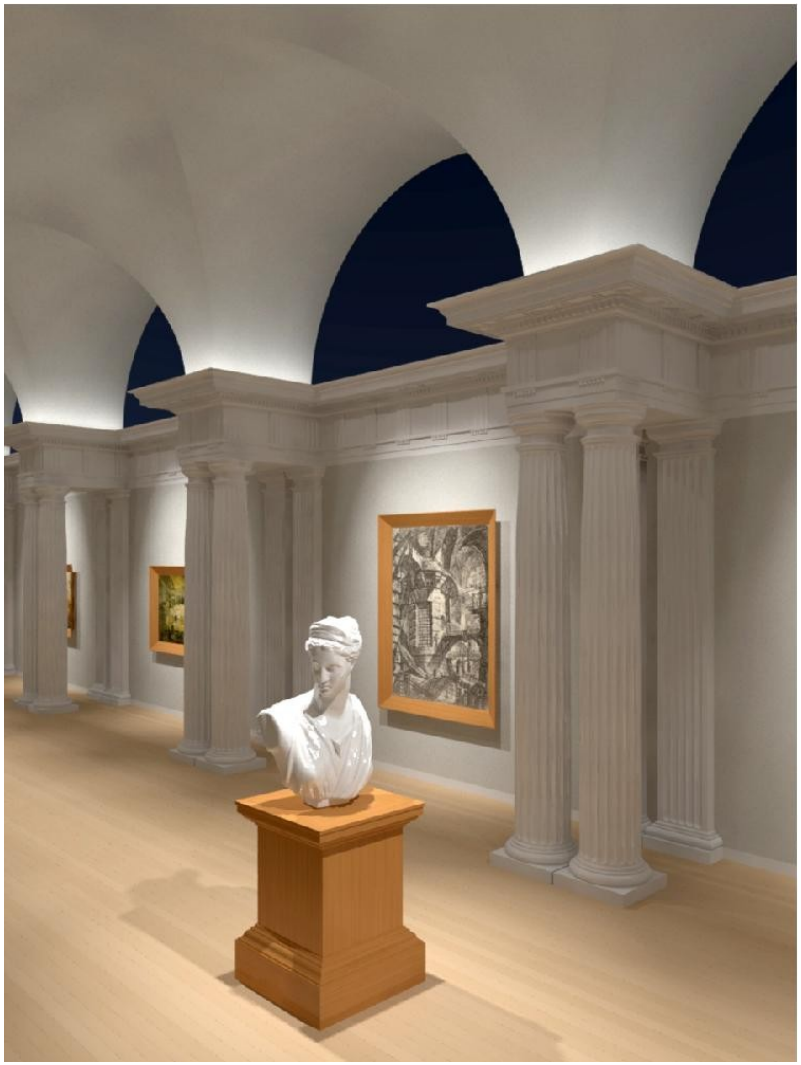

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

# Today

- Motivations
- Transformations in Modeling
- Adding Transformations to our Ray Tracer
- Constructive Solid Geometry (CSG)
- Assignment 2

# Modeling

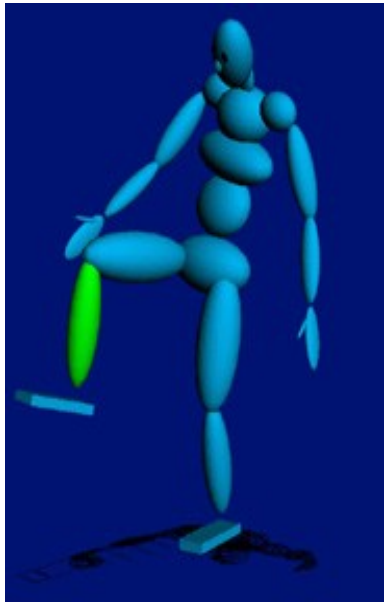
- Create / acquire objects
- Placing objects
- Placing lights
- Describe materials
- Choose camera position and camera parameters
- Specify animation
- ....

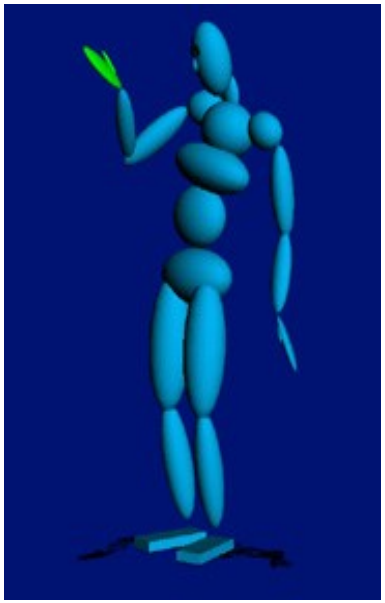


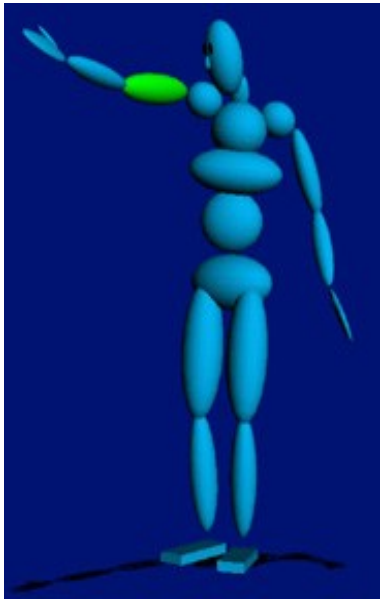
Stephen Duck

# Transformations in Modeling

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



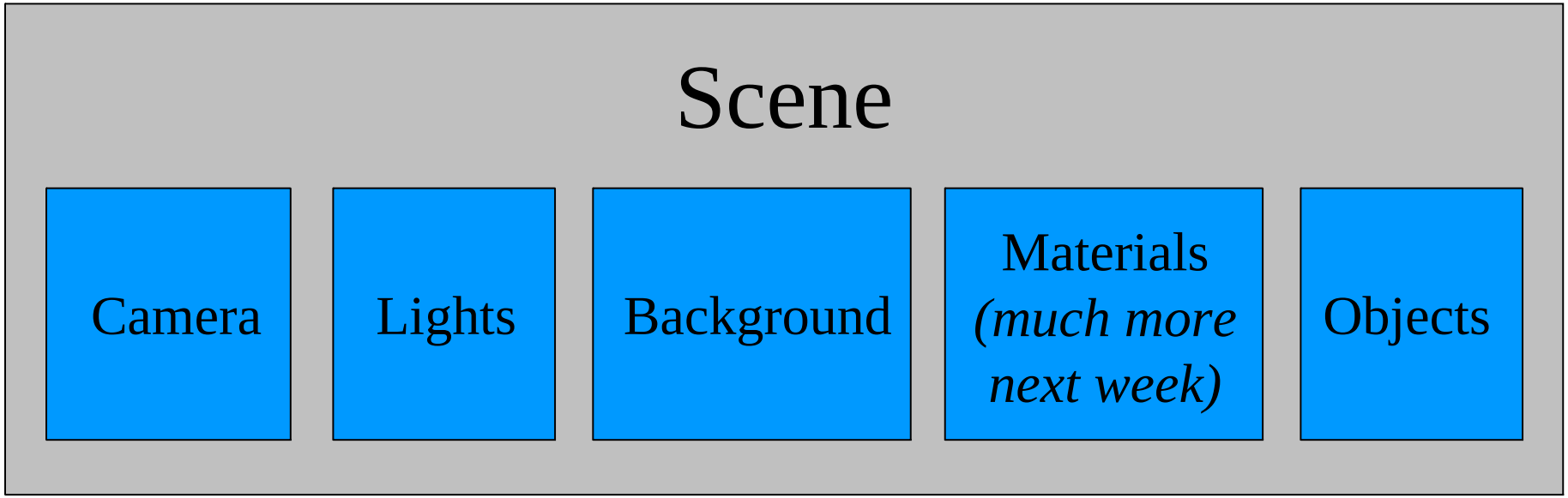




# Today

- Motivations
- Transformations in Modeling
  - Scene description
  - Class Hierarchy
  - Transformations in the Hierarchy
- Adding Transformations to our Ray Tracer
- Constructive Solid Geometry (CSG)
- Assignment 2

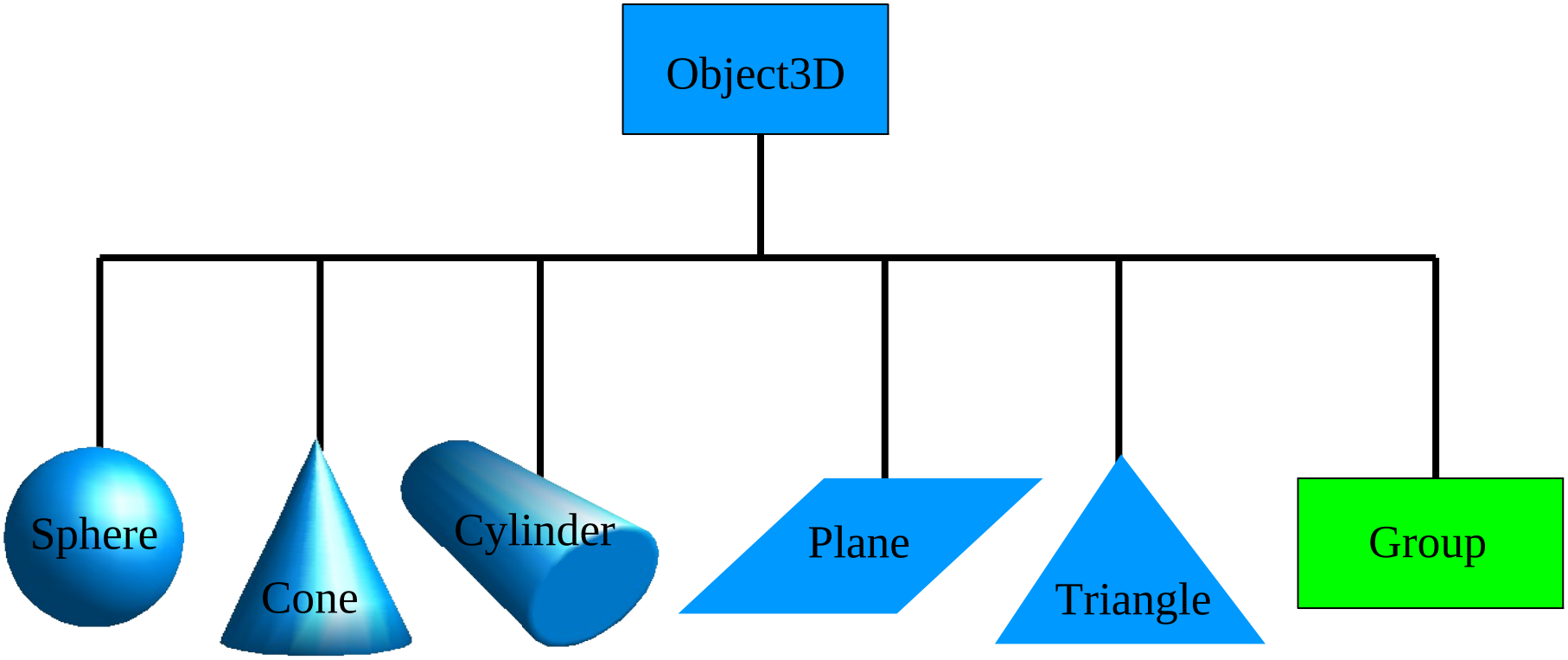
# Scene Description



# Simple Scene Description File

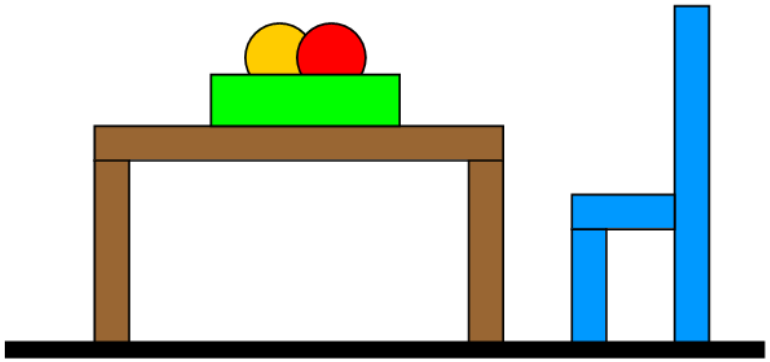
```
OrthographicCamera {
    center 0 0 10
    direction 0 0 -1
    up 0 1 0
    size 5  }

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> }
```

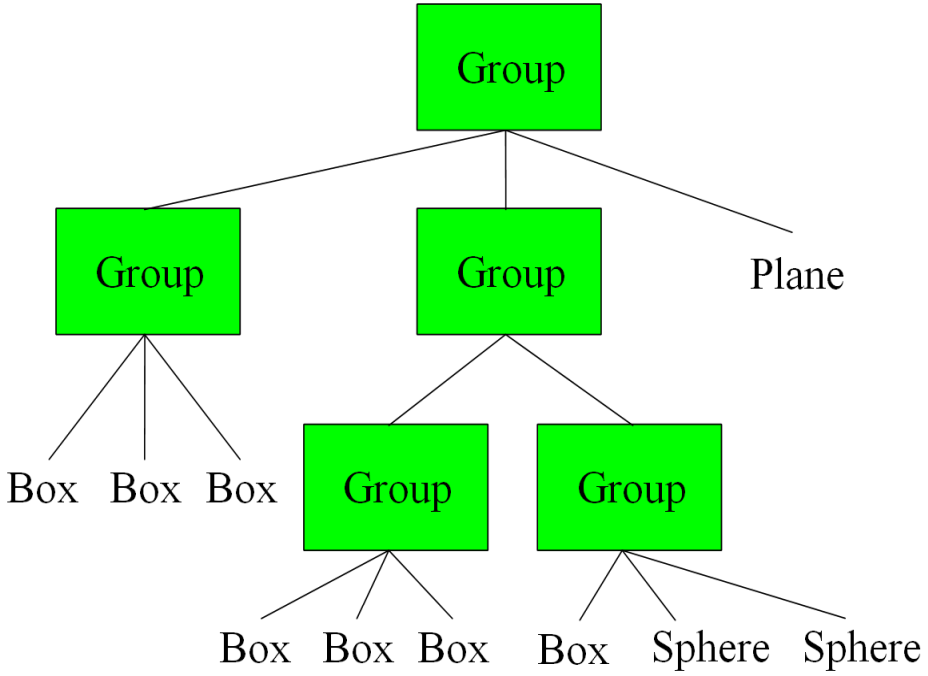
# Class Hierarchy



# Why is a Group an Object3D?

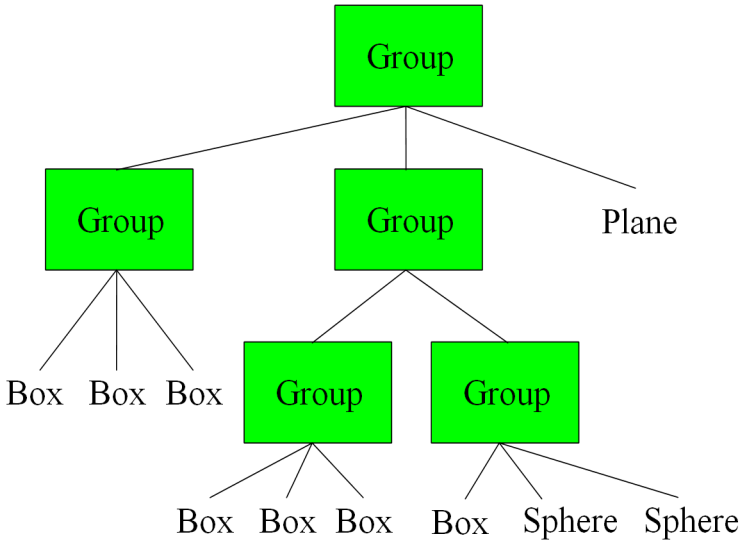
- Logical organization of scene

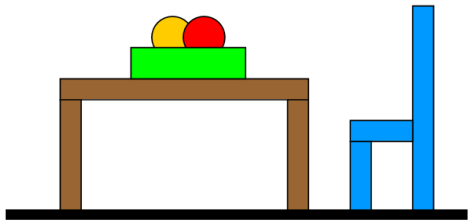




# Simple Example with Groups

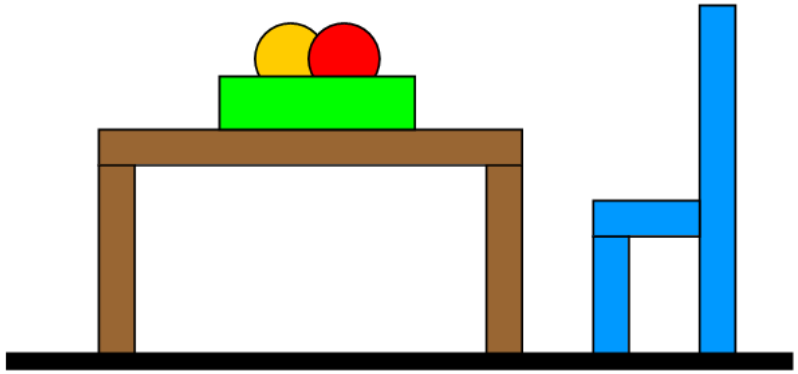
```
Group {
     numObjects 3
     Group {
          numObjects 3
          Box { <BOX PARAMS> }
          Box { <BOX PARAMS> }
          Box { <BOX PARAMS> } }
     Group {
          numObjects 2
          Group {
               Box { <BOX PARAMS> }
               Box { <BOX PARAMS> }
             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



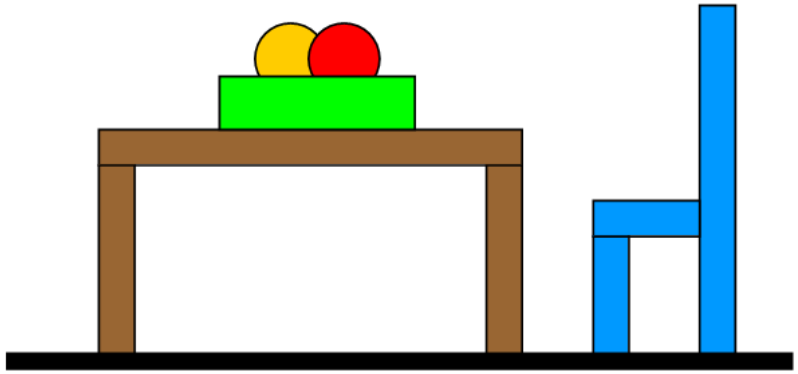
# Adding Materials

```
Group {
     numObjects 3
     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> } }
```

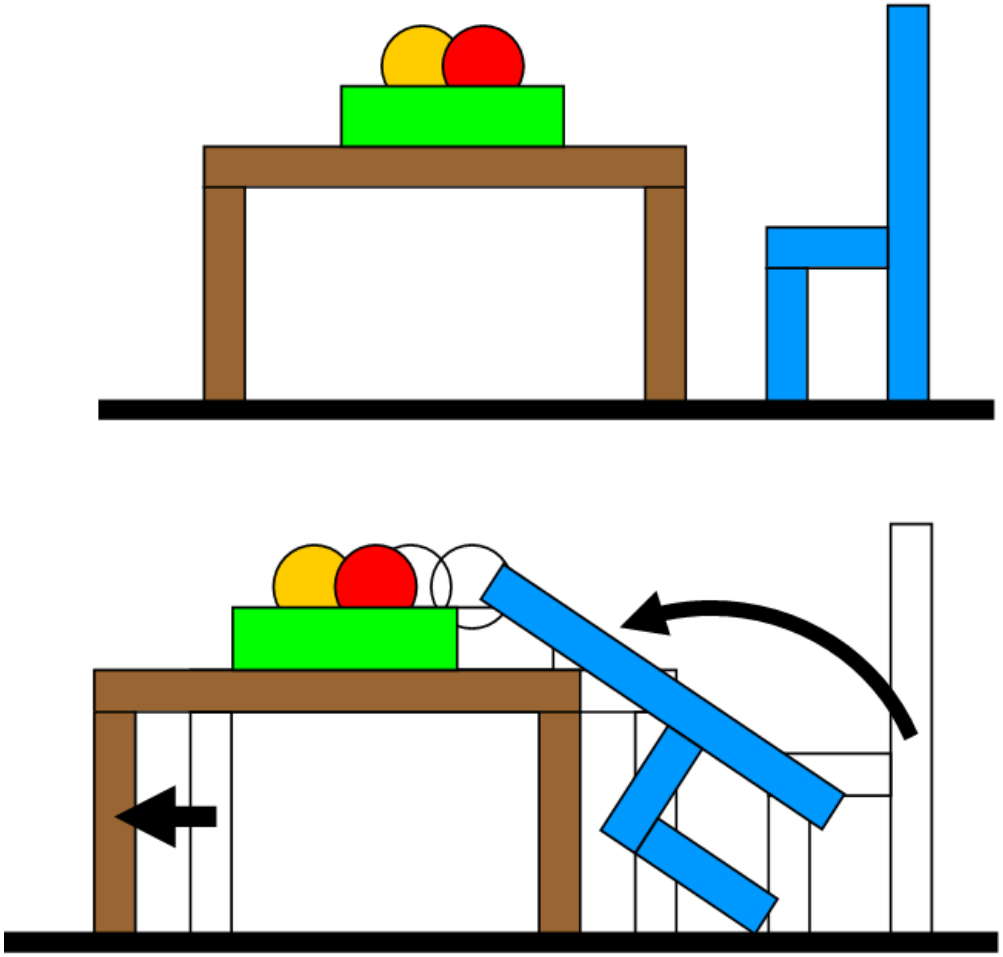


# Adding Materials

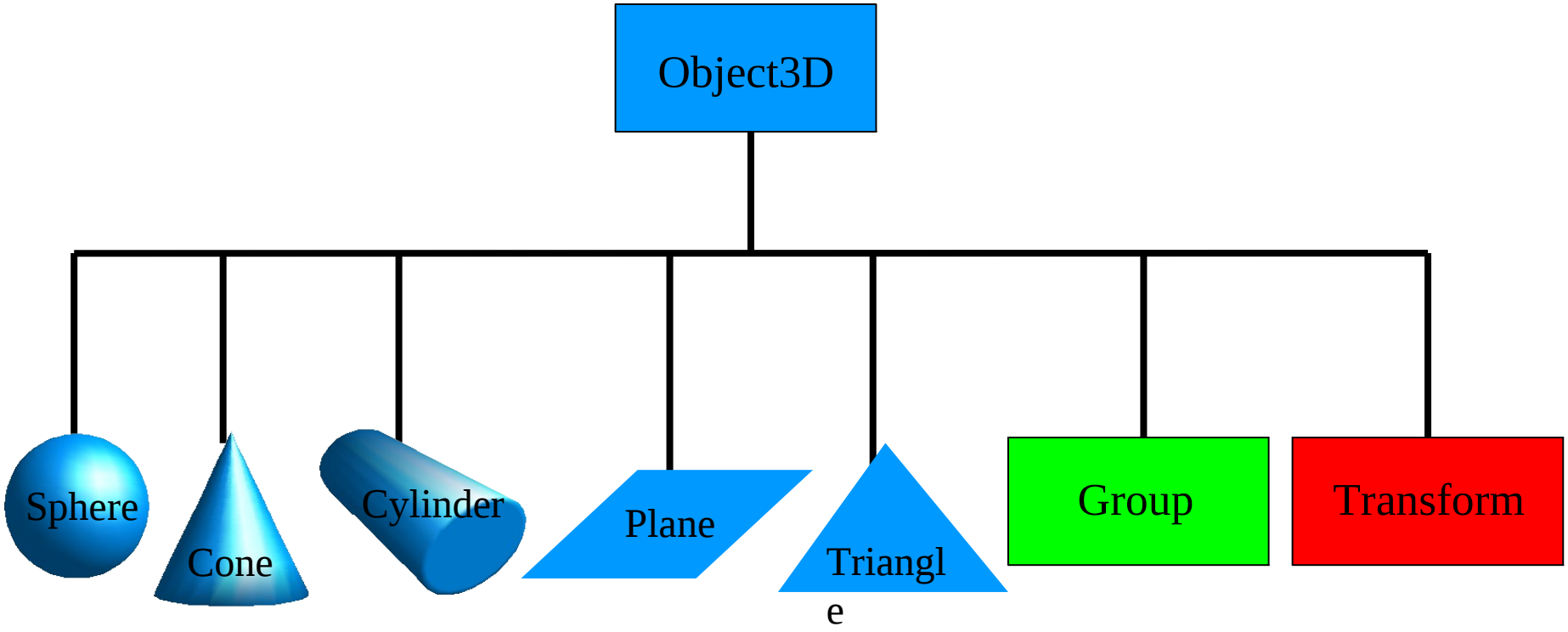
```
Group {
    numObjects 3
    Material { <BROWN> }
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> } }
    Group {
        numObjects 2
        Material { <BLUE> }
        Group {
            Box { <BOX PARAMS> }
            Box { <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

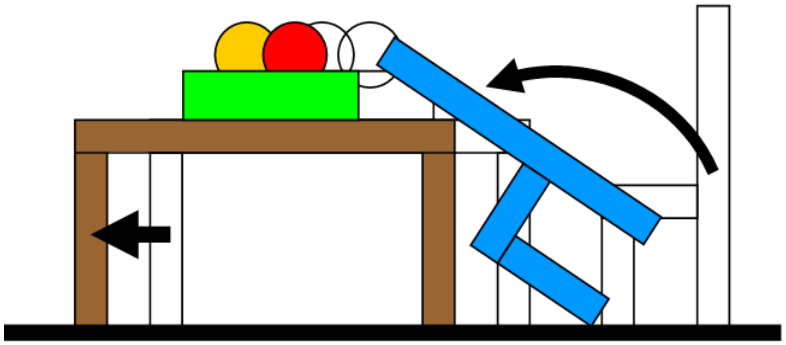


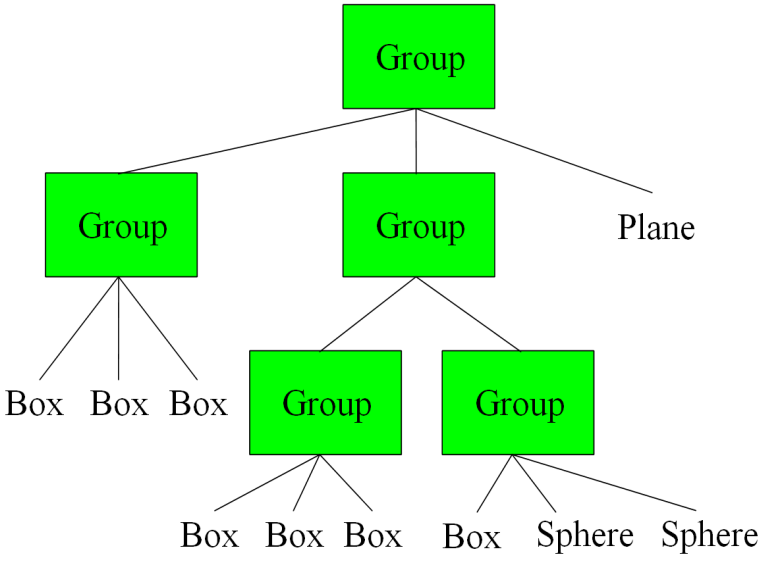
# Class Hierarchy with Transformations

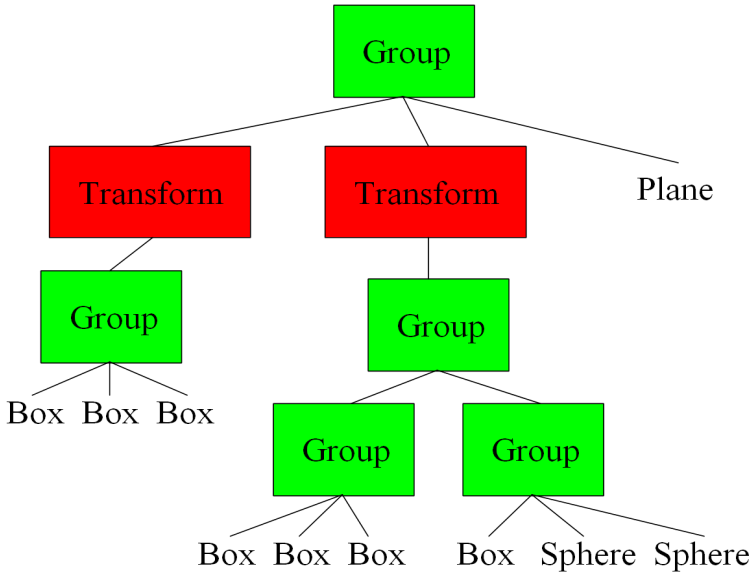


# Why is a Transform an Object3D?

- To position the logical groupings of objects within the scene

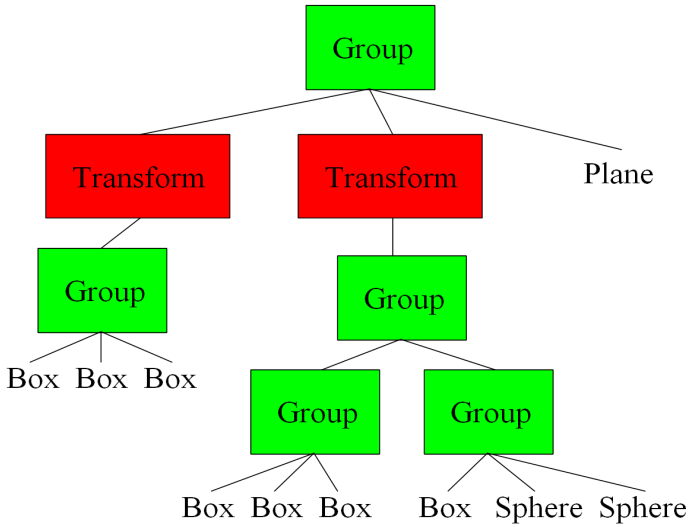




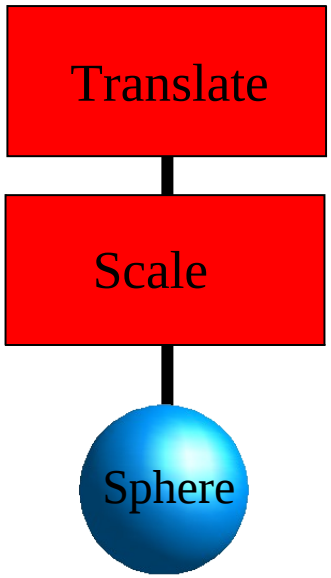
# Simple Example with Transforms

```
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> } }
```

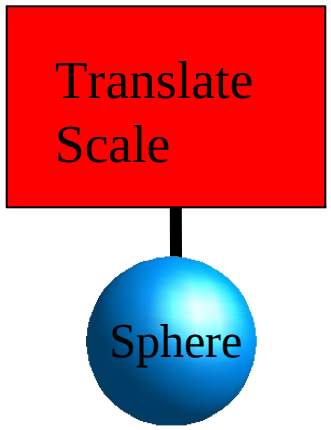


# Nested Transforms

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




same as




```
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 } }
```

# Questions?

# Today

- Motivations
- Transformations in Modeling
- Adding Transformations to our Ray Tracer
  - Transforming the Ray
  - Handling the depth, *t*
  - Transforming the Normal
- Constructive Solid Geometry (CSG)
- Assignment 2

# Incorporating Transforms

1. Make each primitive handle any applied transformations

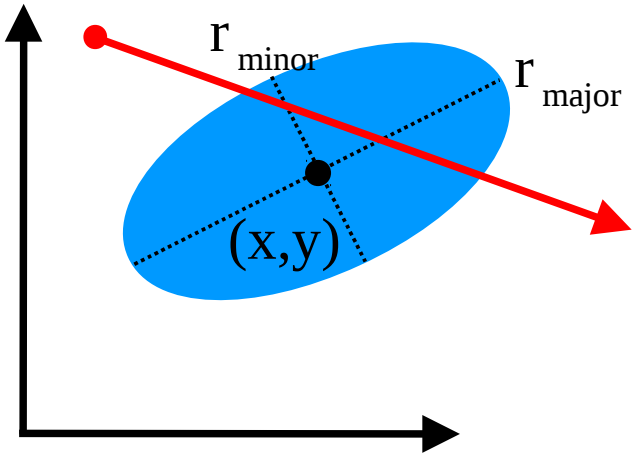
```
Sphere {
    center 1 0.5 0
    radius 2
}
```

2. Transform the Rays

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

# Primitives handle Transforms

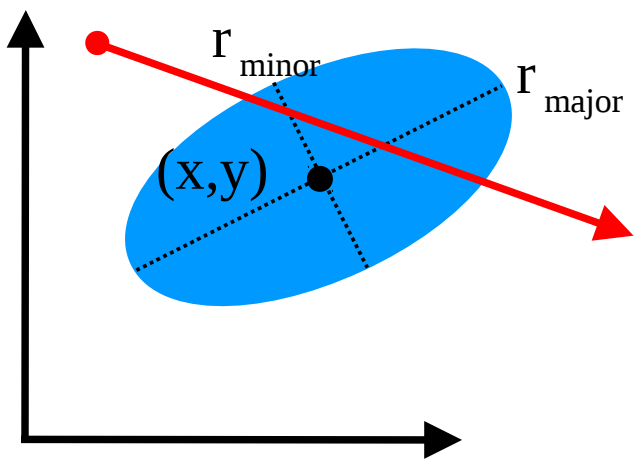
```
Sphere {
     center 3 2 0
     z_rotation 30
     r_major 2
     r_minor 1
}
```



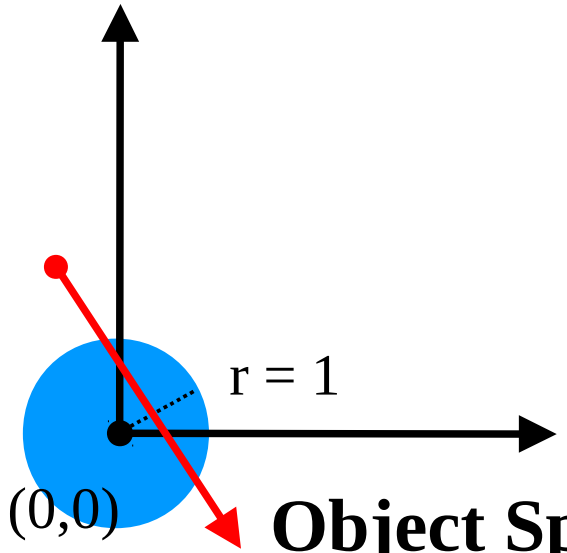

- Complicated for many primitives

# Transform the Ray

- Move the ray from *World Space* to *Object Space*




**World Space**




**Object Space**

$$p_{WS} = \mathbf{M}\ p_{OS}$$

$$p_{OS} = \mathbf{M}^{-1}\ p_{WS}$$

# Transform Ray

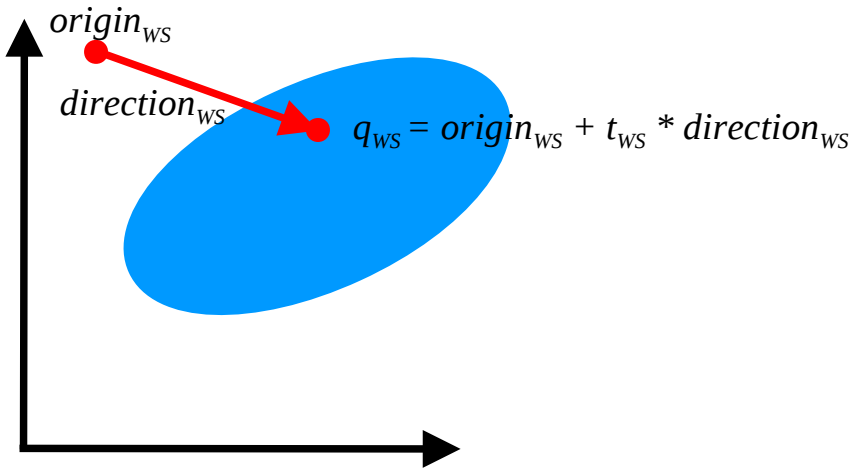
- New origin:

  $origin_{OS} = \mathbf{M}^{-1}\, origin_{WS}$

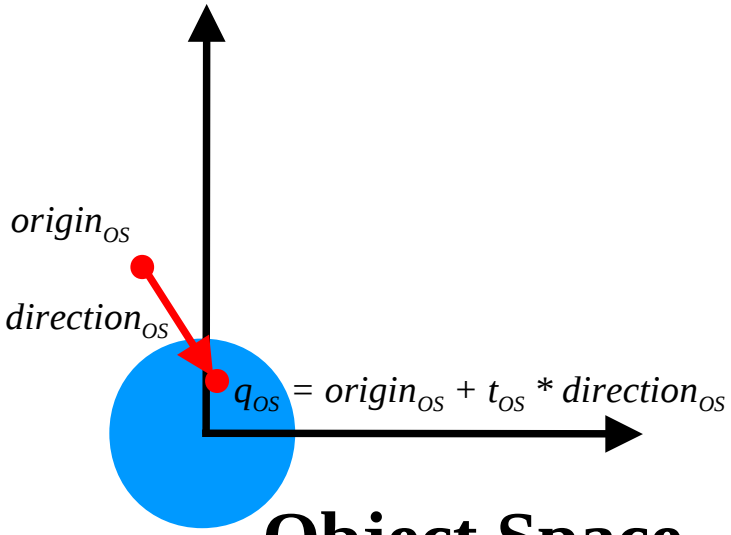
- New direction:

  $direction_{OS} = \mathbf{M}^{-1}\,(origin_{WS} + 1 * direction_{WS}) \;\; - \;\; \mathbf{M}^{-1}\, origin_{WS}$

  $direction_{OS} = \mathbf{M}^{-1}\;\; direction_{WS}$




**World Space**




**Object Space**

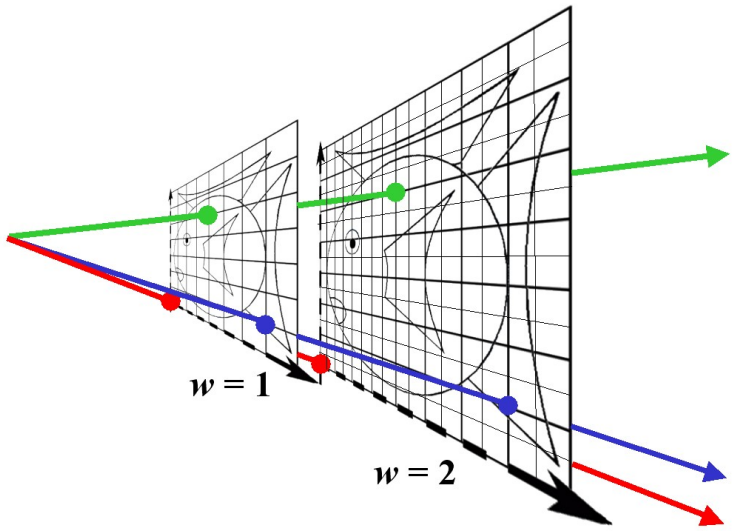
# Transforming Points & Directions

- Transform point

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax+by+cz+d \\ ex+fy+gz+h \\ ix+jy+kz+l \\ 1 \end{pmatrix}$$

- Transform direction

$$\begin{pmatrix} x' \\ y' \\ z' \\ 0 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} ax+by+cz \\ ex+fy+gz \\ ix+jy+kz \\ 0 \end{pmatrix}$$




Homogeneous Coordinates: *(x,y,z,w)*

*W = 0* is a point at infinity (direction)

# What to do about the depth, *t*

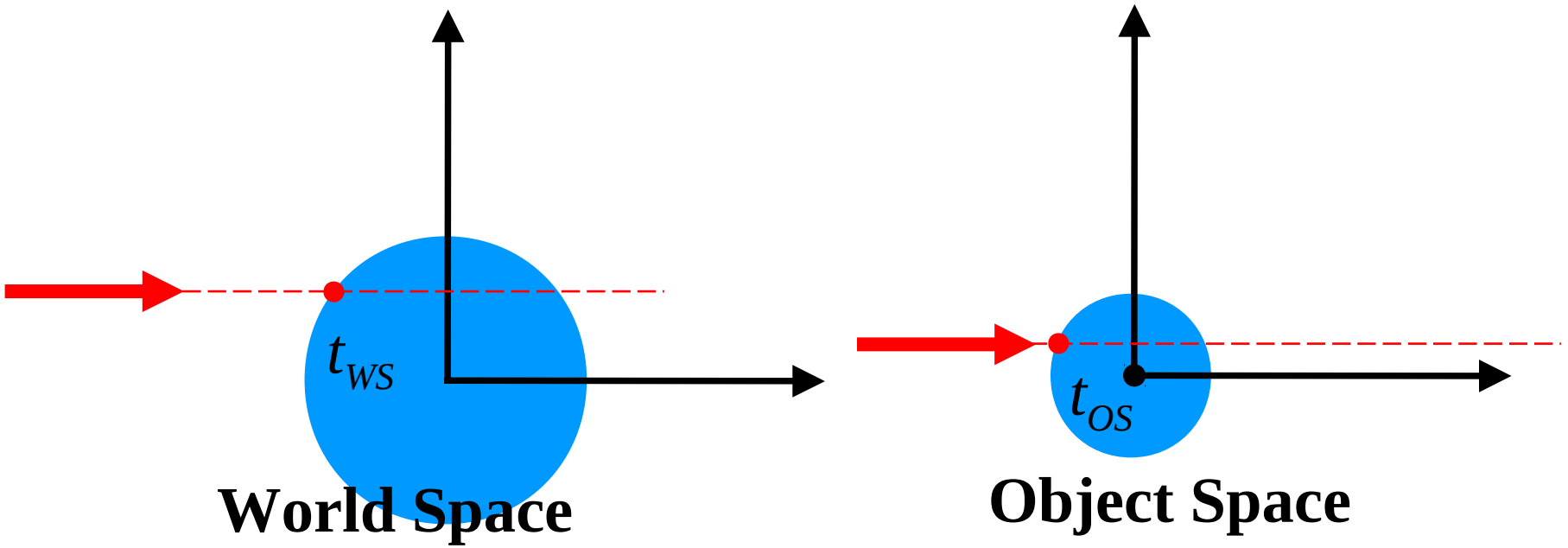
If **M** includes scaling, $direction_{OS}$ will NOT be normalized

1. Normalize the direction

2. Don't normalize the direction

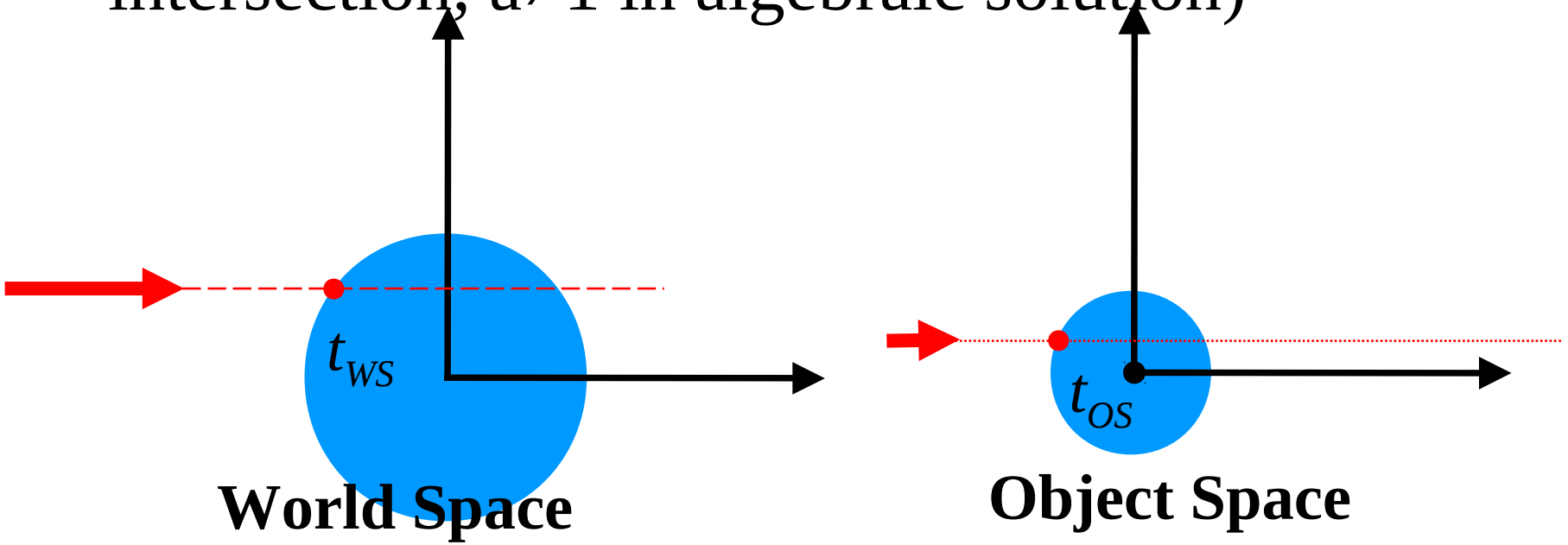
# 1. Normalize direction

- $t_{OS} \neq t_{WS}$
  and must be rescaled after intersection



# 2. Don't normalize direction

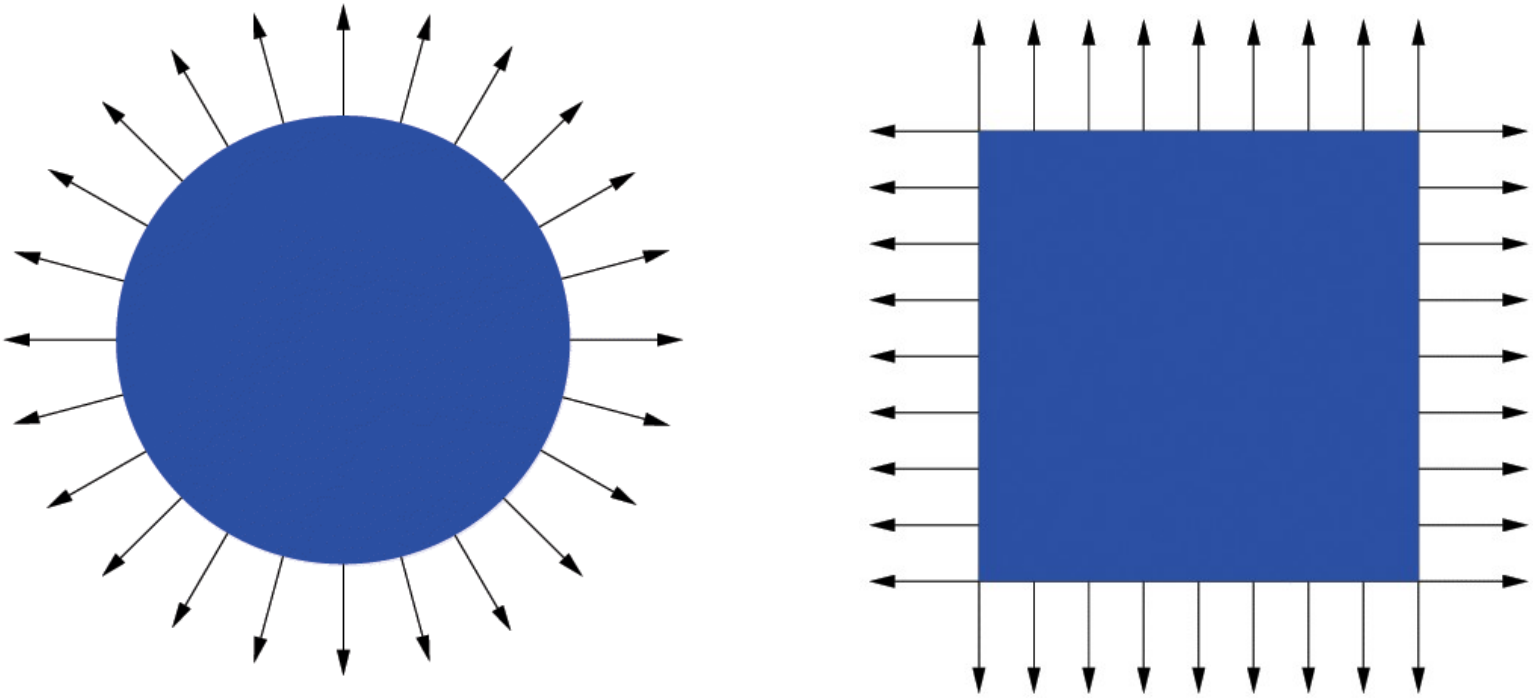
- $t_{OS} = t_{WS}$
- Don't rely on $t_{OS}$ being true distance during intersection routines (e.g. geometric ray-sphere intersection, a≠1 in algebraic solution)



# Questions?

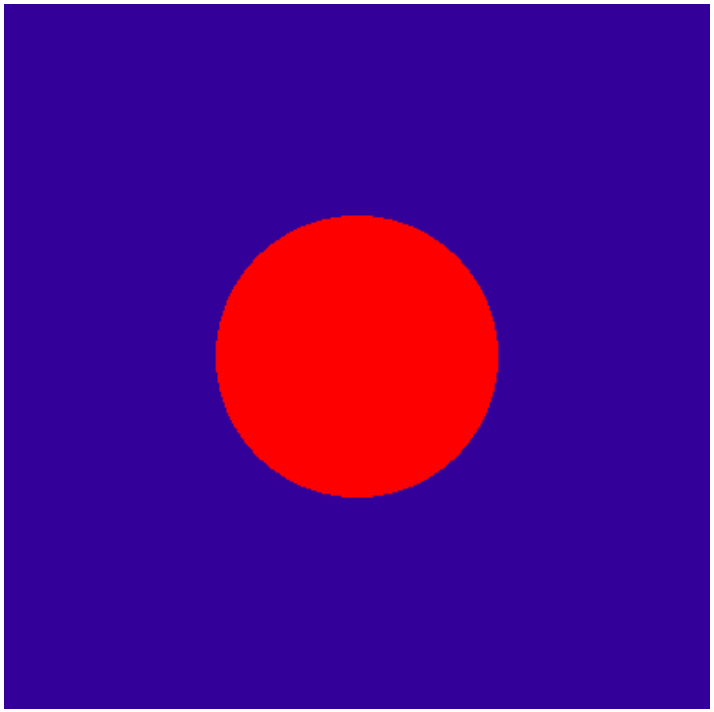
# New component of the Hit class

- Surface Normal: unit vector that is locally perpendicular to the surface

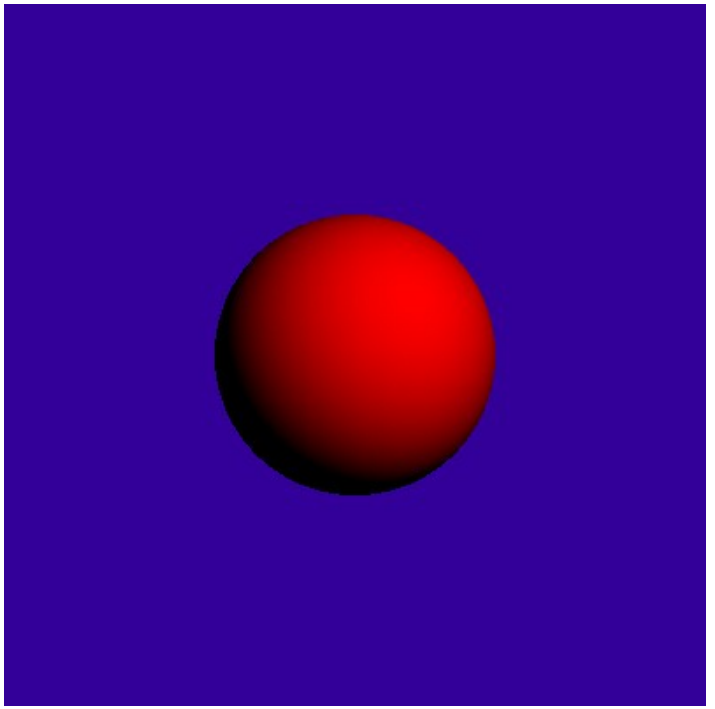


# Why is the Normal important?

- It's used for shading — makes things look 3D!

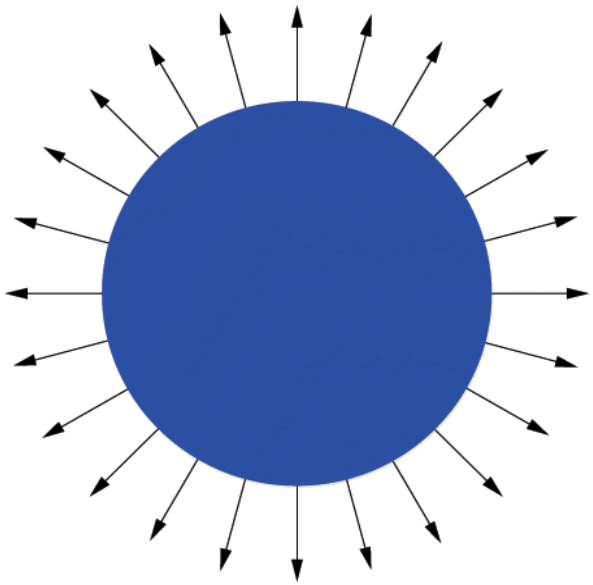


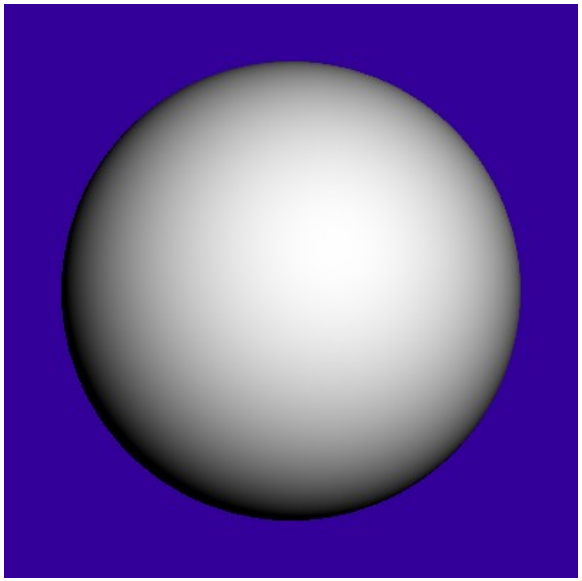
object color only
(Assignment 1)

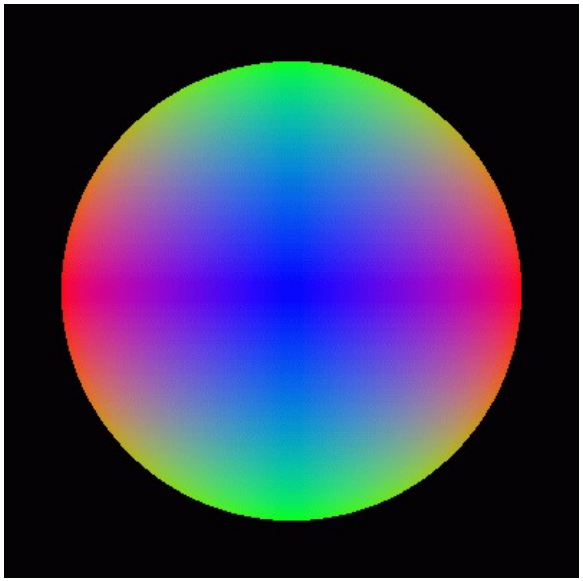


Diffuse Shading
(Assignment 2)

# Visualization of Surface Normal





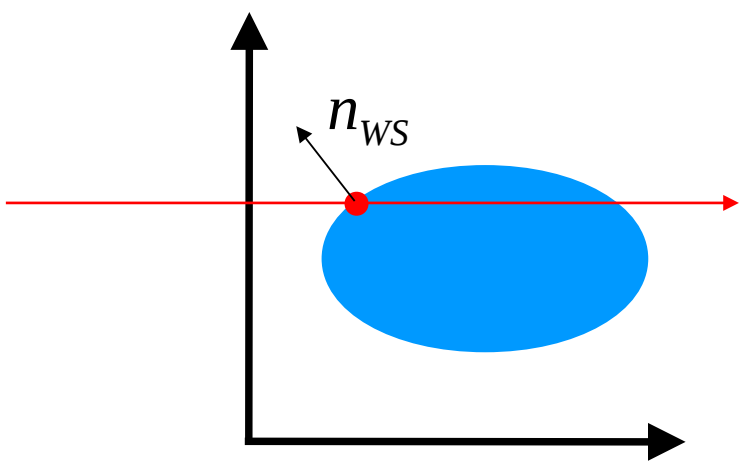


$\pm x$ = Red

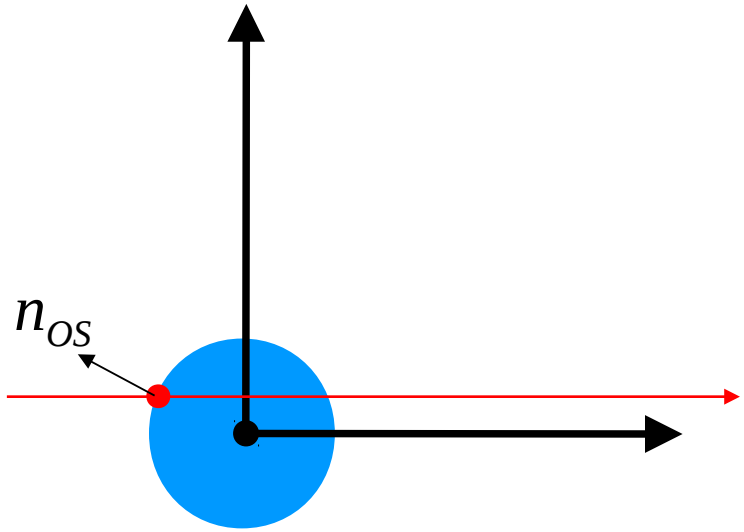
$\pm y$ = Green

$\pm z$ = Blue

# How do we transform normals?



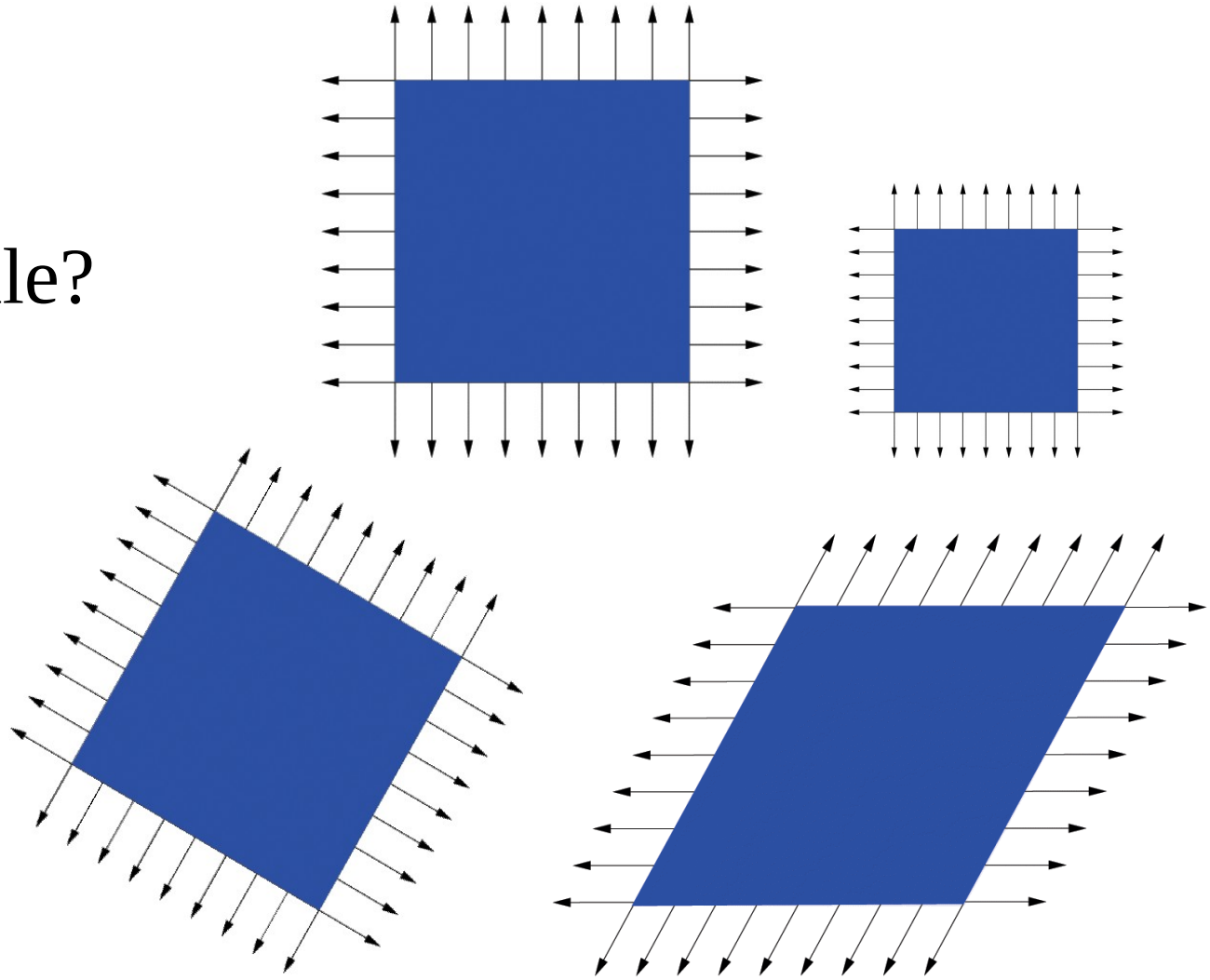

**World Space**




**Object Space**

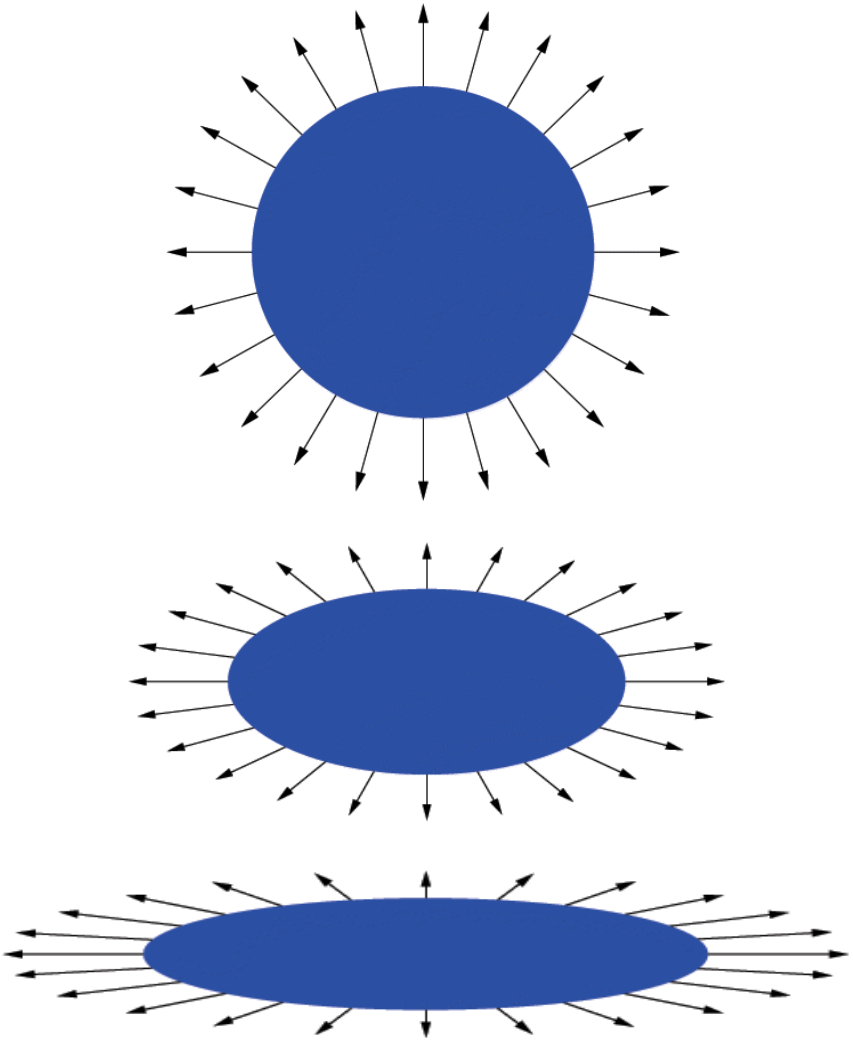
# Transform the Normal like the Ray?

- translation?
- rotation?
- isotropic scale?
- scale?
- reflection?
- shear?
- perspective?

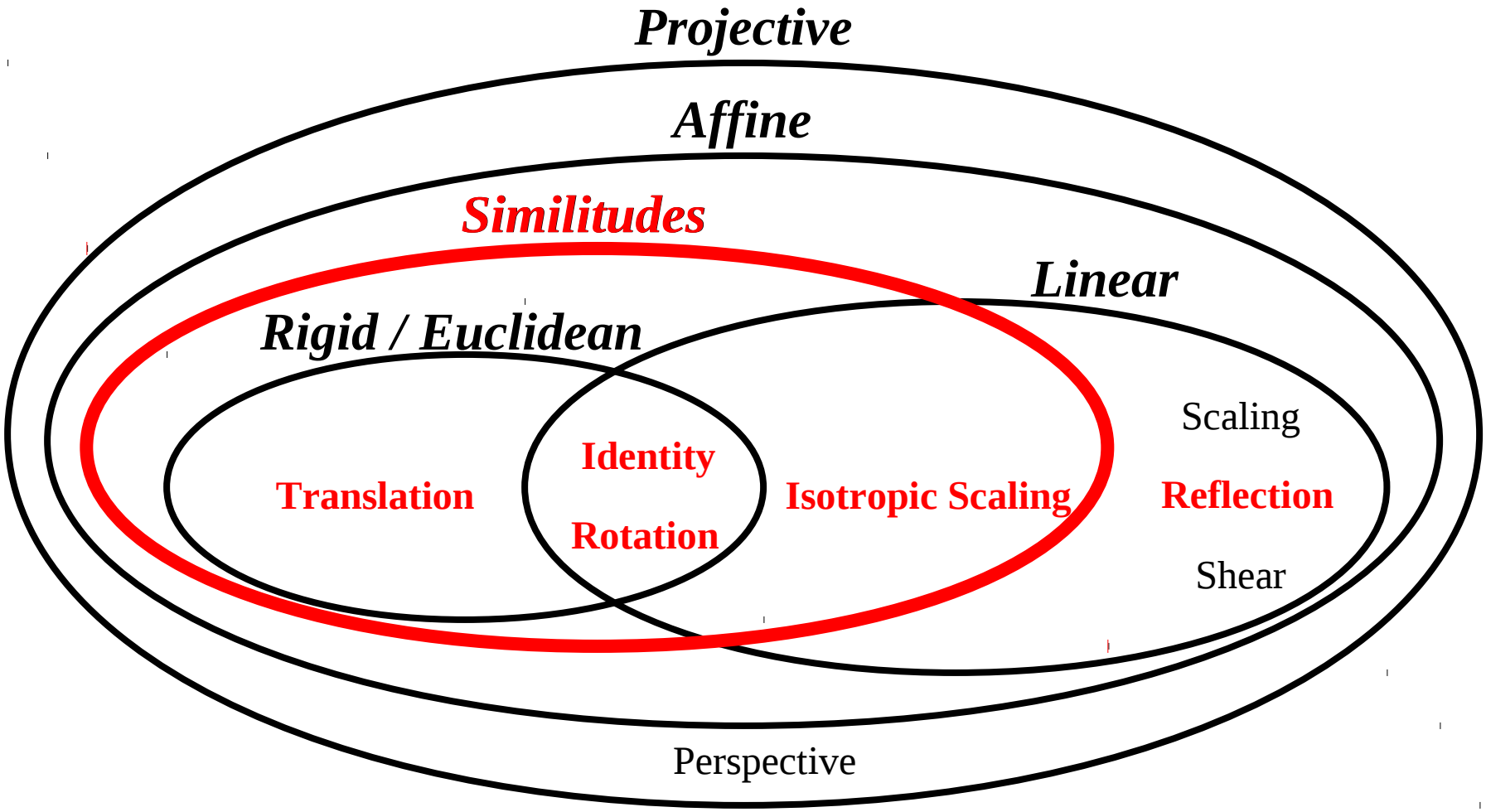


# Transform the Normal like the Ray?

- translation?
- rotation?
- isotropic scale?
- scale?
- reflection?
- shear?
- perspective?



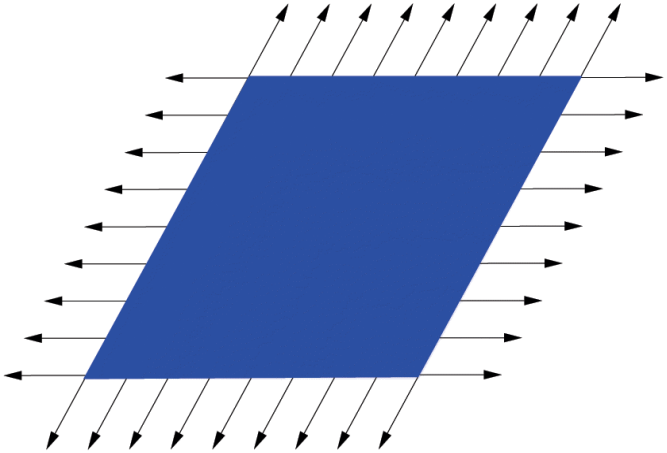
# What class of transforms?

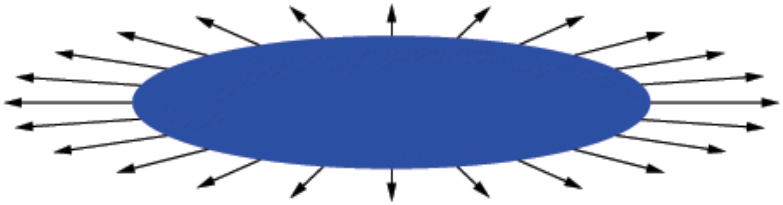



***a.k.a. Orthogonal Transforms***

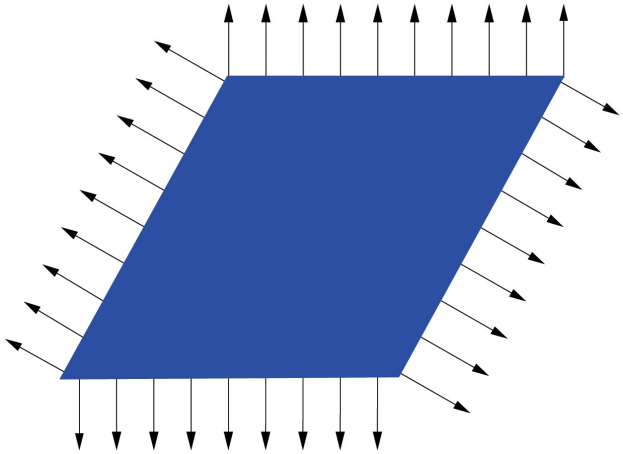
# Transformation for shear and scale

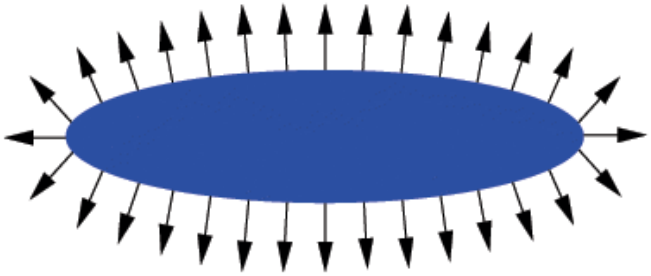
Incorrect Normal Transformation



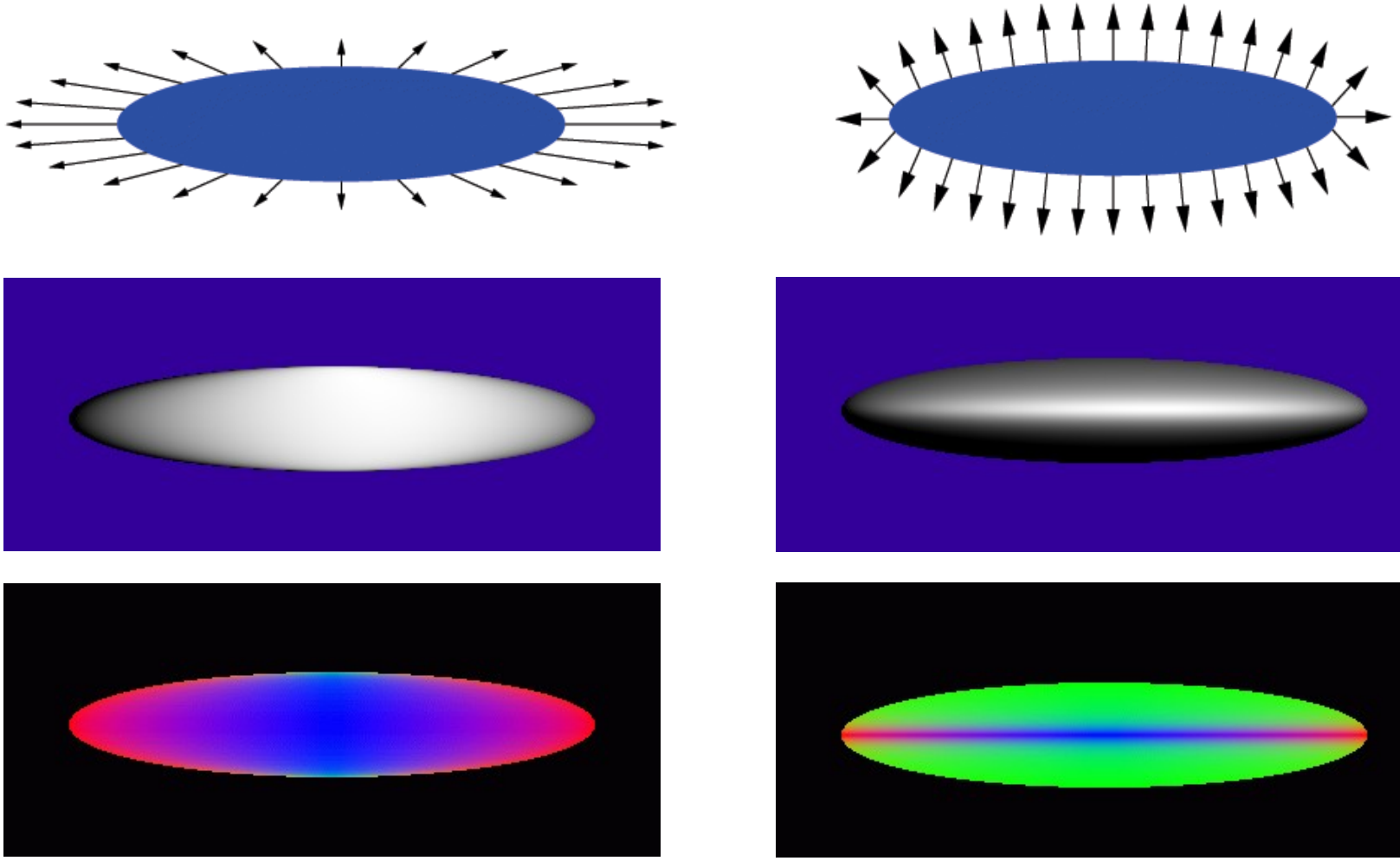


Correct Normal Transformation





# More Normal Visualizations

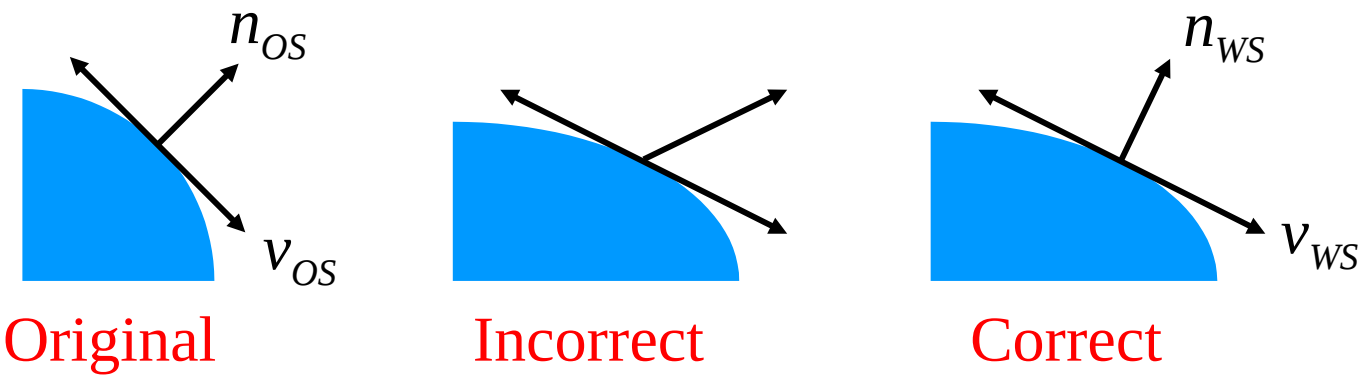


Incorrect Normal Transformation

Correct Normal Transformation

# So how do we do it right?

- Think about transforming the *tangent plane* to the normal, not the normal *vector*




Pick any vector $v_{OS}$ in the tangent plane, how is it transformed by matrix **M**?

$$v_{WS} = \mathbf{M}\ v_{OS}$$

# Transform tangent vector $v$

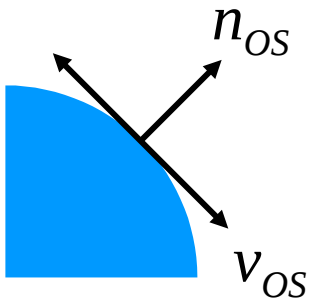
$v$ is perpendicular to normal $n$:

$$n_{OS}^{\mathbf{T}}\, v_{OS} = 0$$

$$n_{OS}^{\mathbf{T}}\, (\mathbf{M}^{-1}\, \mathbf{M})\, v_{OS} = 0$$

$$(n_{OS}^{\mathbf{T}}\, \mathbf{M}^{-1})\, (\mathbf{M}\, v_{OS}) = 0$$

$$(n_{OS}^{\mathbf{T}}\, \mathbf{M}^{-1})\, v_{WS} = 0$$

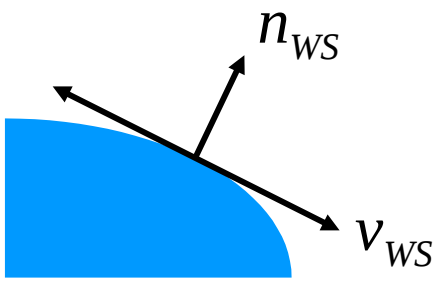



$v_{WS}$ is perpendicular to normal $n_{WS}$:

$$n_{WS}^{\mathbf{T}} = n_{OS}\, (\mathbf{M}^{-1})$$

$$n_{WS} = (\mathbf{M}^{-1})^{\mathbf{T}}\, n_{OS}$$

$$n_{WS}^{\mathbf{T}}\, v_{WS} = 0$$



# Comment

- So the correct way to transform normals is:

$$n_{WS} = (\mathbf{M}^{-1})^{\mathrm{T}}\, n_{OS}$$

- But why did $n_{WS} = \mathbf{M}\, n_{OS}$ work for similitudes?
- Because for similitude / similarity transforms,

$$(\mathbf{M}^{-1})^{\mathrm{T}} = \lambda\, \mathbf{M}$$

- e.g. for orthonormal basis:

$$\mathbf{M} = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{pmatrix} \qquad \mathbf{M}^{-1} = \begin{pmatrix} x_u & x_v & x_n \\ y_u & y_v & y_n \\ z_u & z_v & z_n \end{pmatrix}$$

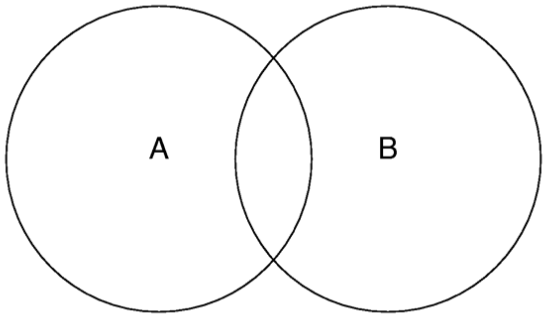
# Questions?

# Today

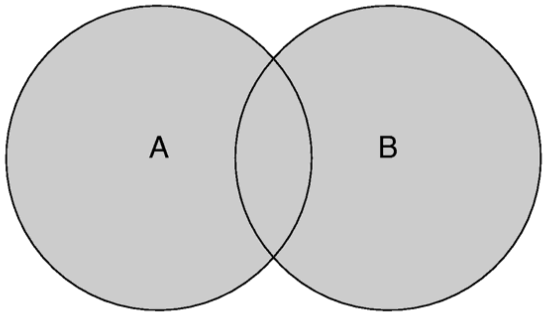
- Motivations
- Transformations in Modeling
- World Space vs Object Space
- Adding Transformations to our Ray Tracer
- Constructive Solid Geometry (CSG)
- Assignment 2

# Constructive Solid Geometry (CSG)

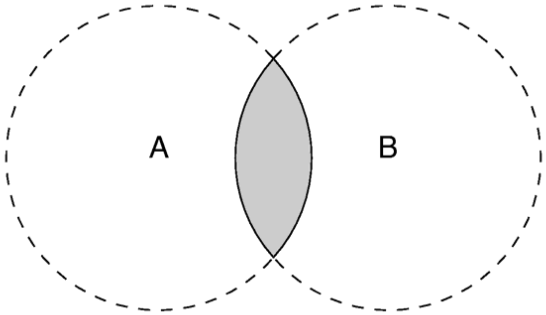
Given overlapping shapes A and B:



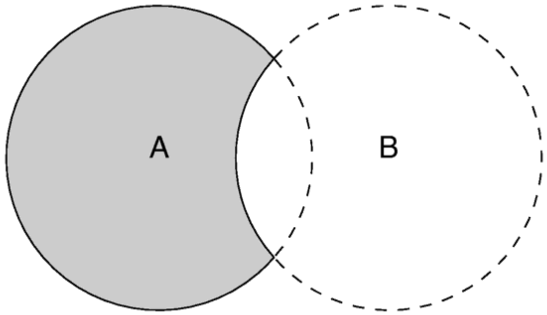

Union



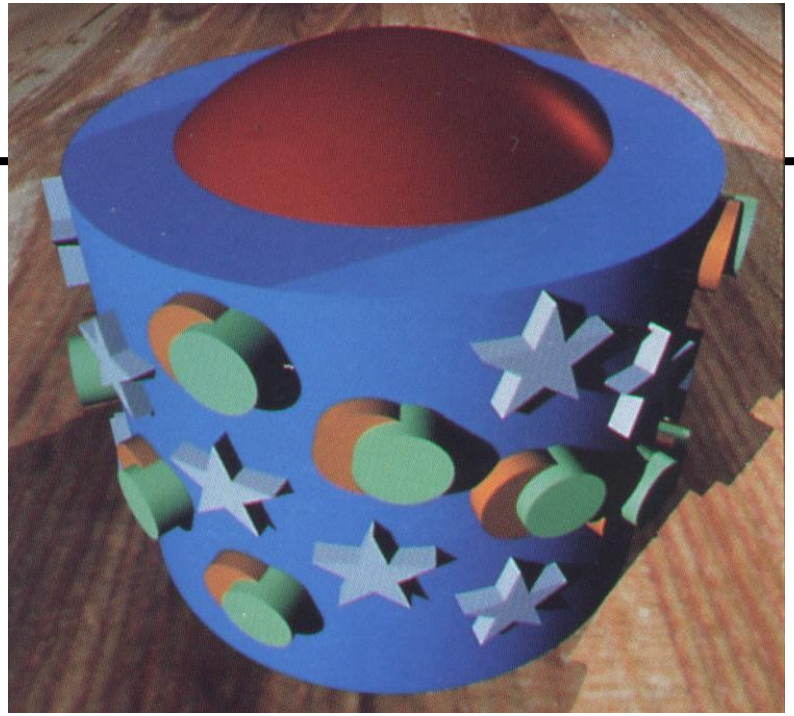

Intersection

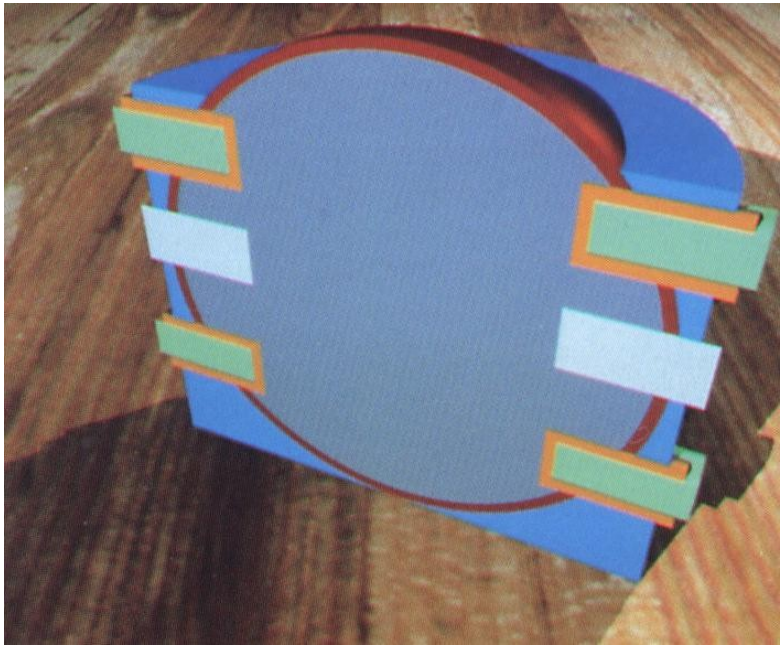



Subtraction

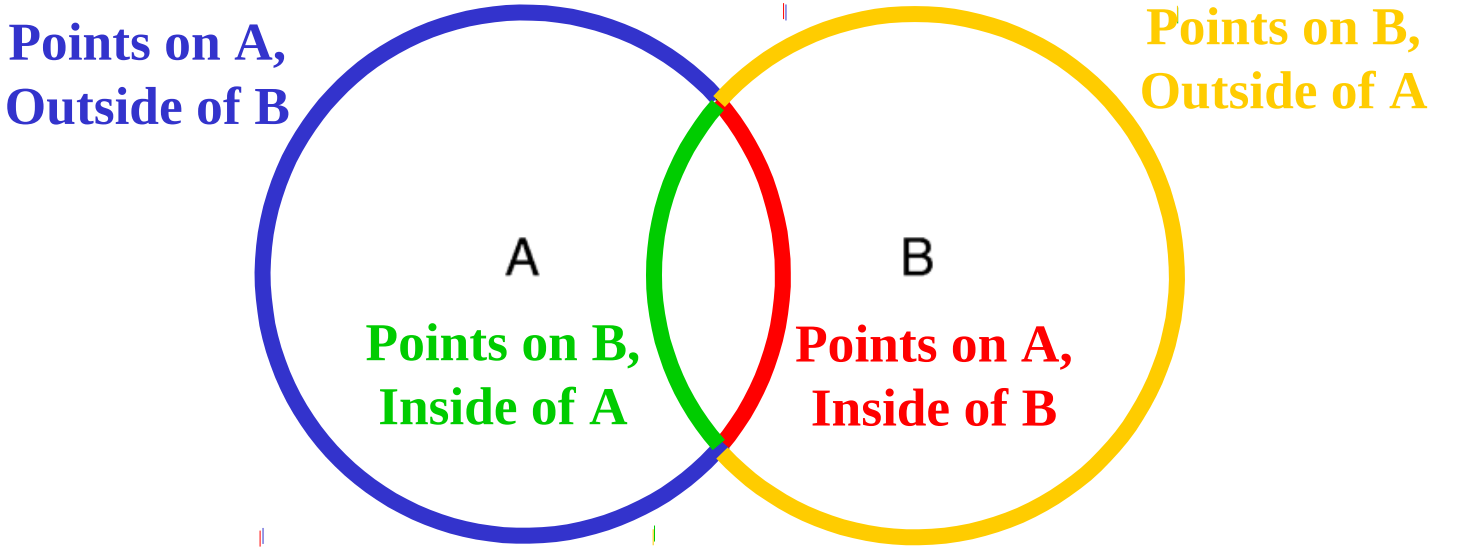


# For example:

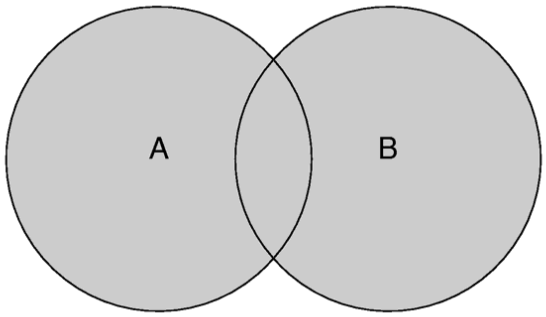




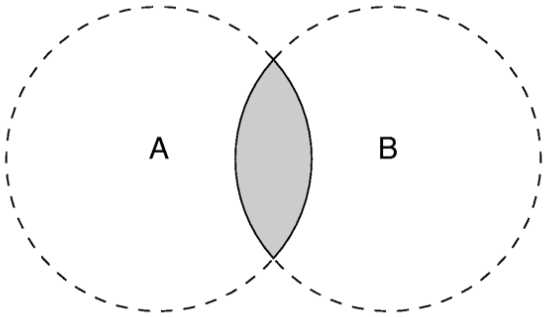
# How can we implement CSG?



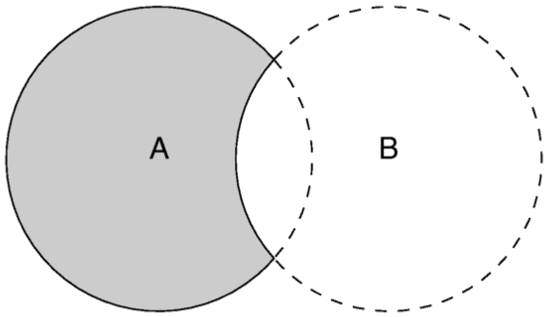

Union



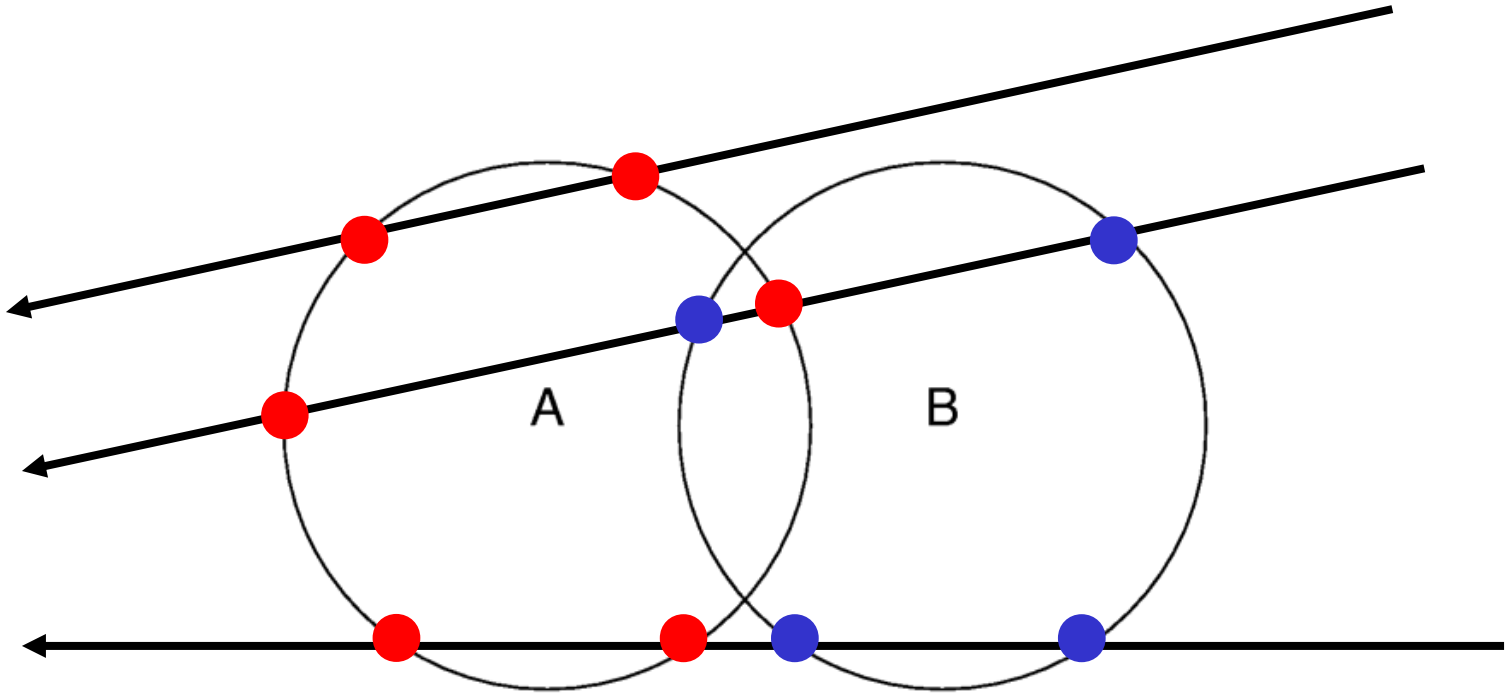

Intersection



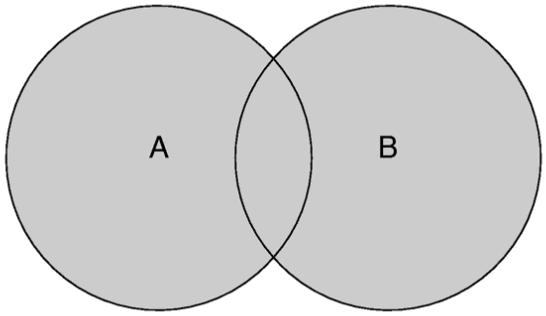

Subtraction

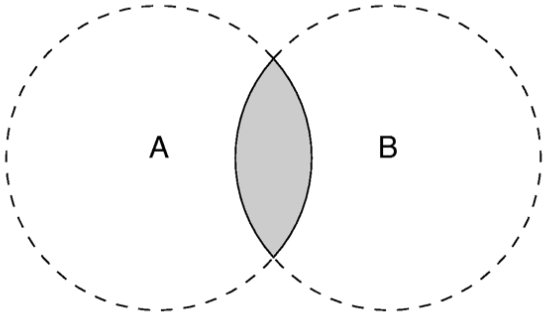


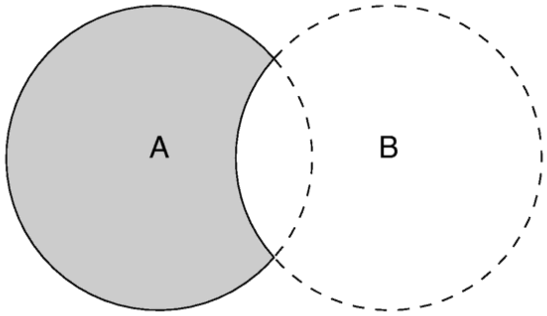
# Collect all the intersections




Union Intersection Subtraction

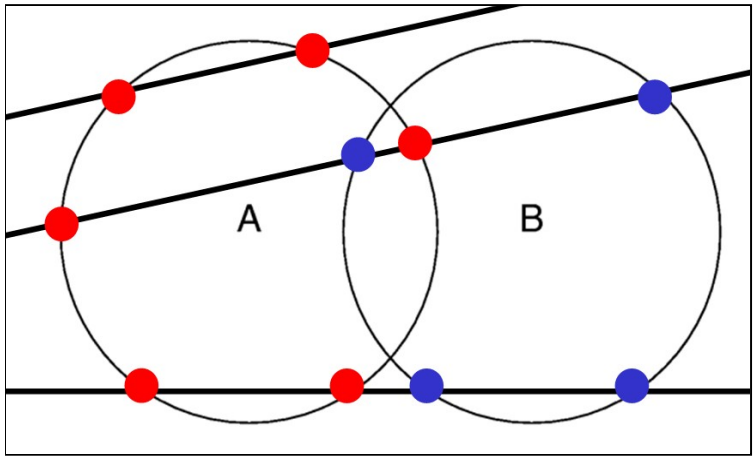


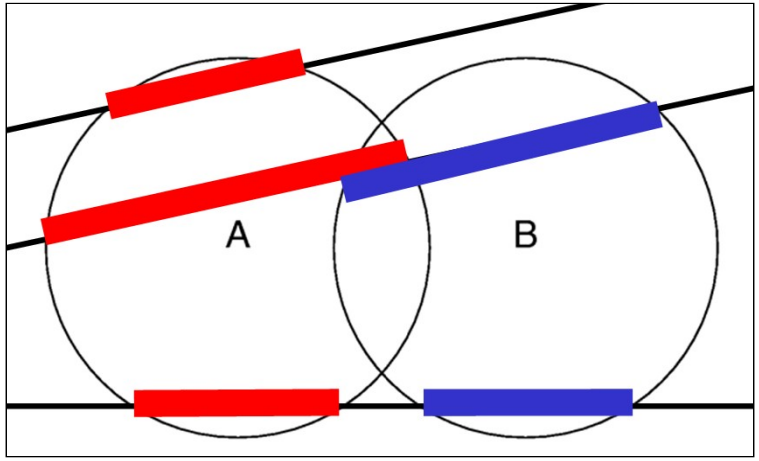



# Implementing CSG

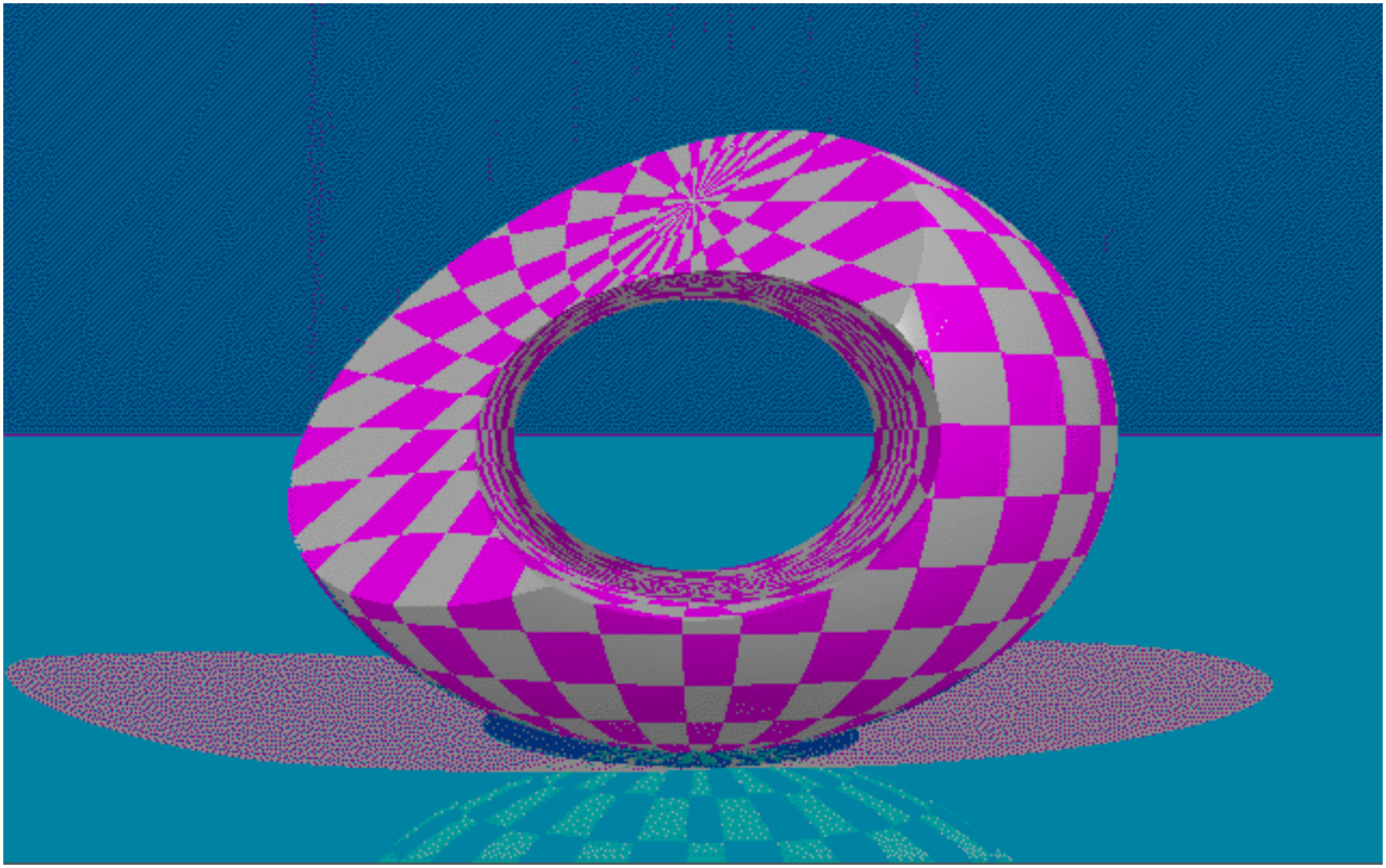
1. Test "inside" intersections:
   - Find intersections with A, test if they are inside/outside B
   - Find intersections with B, test if they are inside/outside A




2. Overlapping intervals:
   - Find the intervals of "inside" along the ray for A and B
   - Compute union/intersection/subtraction of the intervals



# "Fredo's First CSG Raytraced Image"



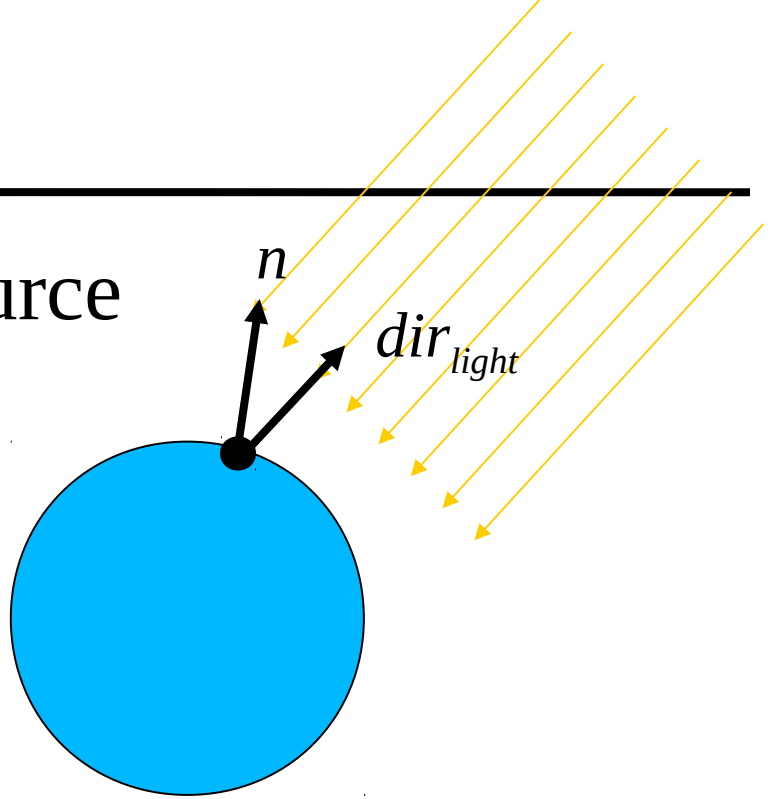
# Questions?

# Today

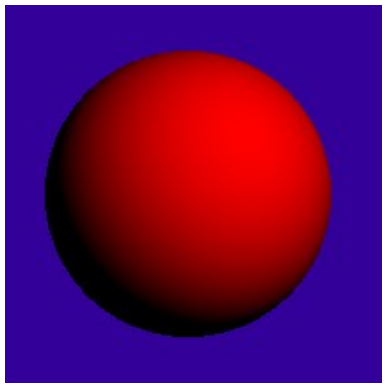
- Motivations
- Transformations in Modeling
- World Space vs Object Space
- Adding Transformations to our Ray Tracer
- Constructive Solid Geometry (CSG)
- Assignment 2
  - Due Wednesday Sept 24th, 11:59pm

# Simple Shading

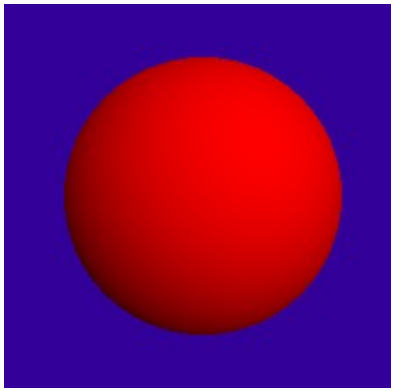
- Single Directional Light Source
- Diffuse Shading
- Ambient Light




$$c_{pixel} = c_{ambient} * c_{object} + dir_{light} \bullet n * c_{light} * c_{object}$$

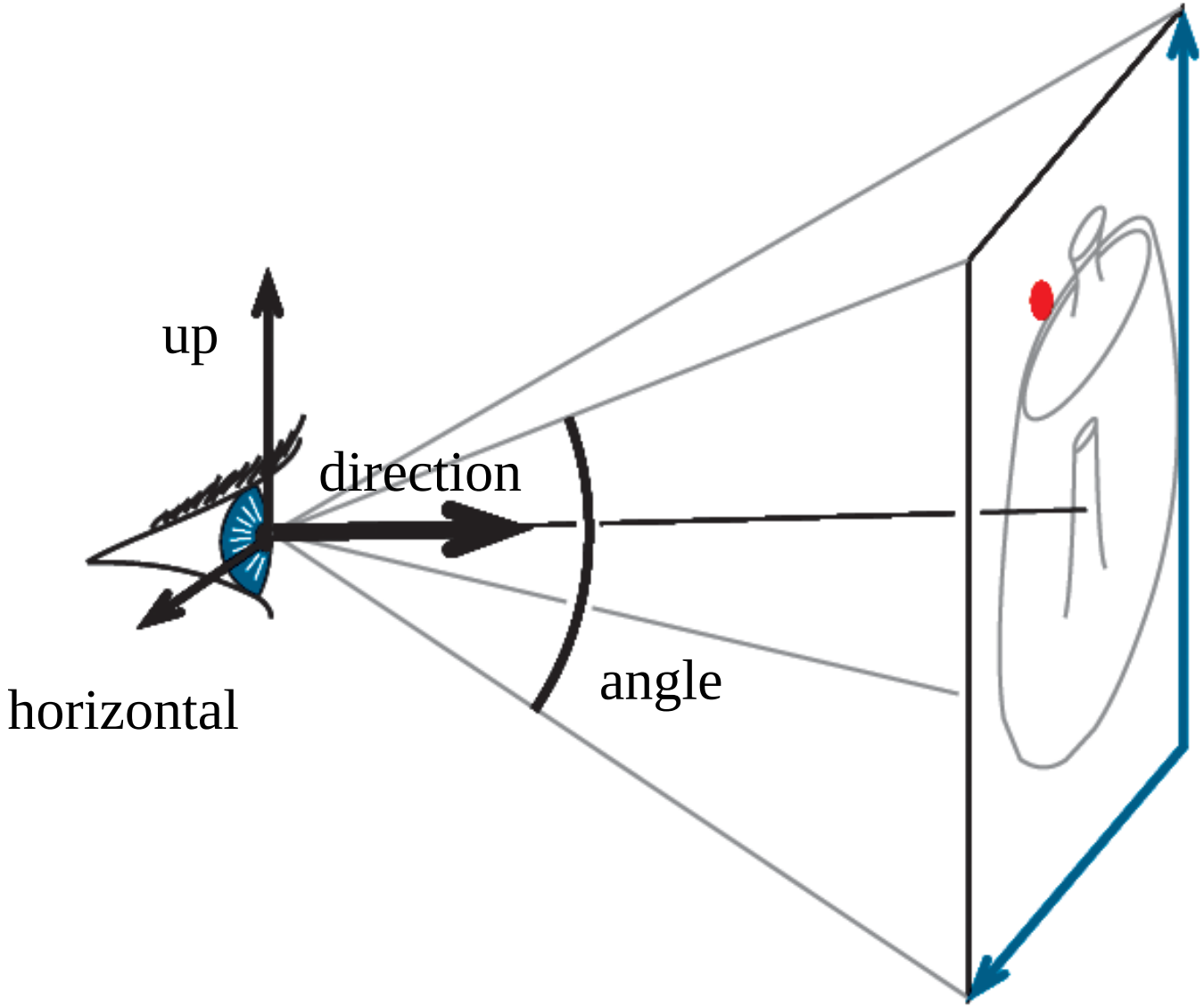


$c_{light} = (1,1,1)$
$c_{ambient} = (0,0,0)$



$c_{light} = (0.5,0.5,0.5)$
$c_{ambient} = (0.5,0.5,0.5)$

# Adding Perspective Camera



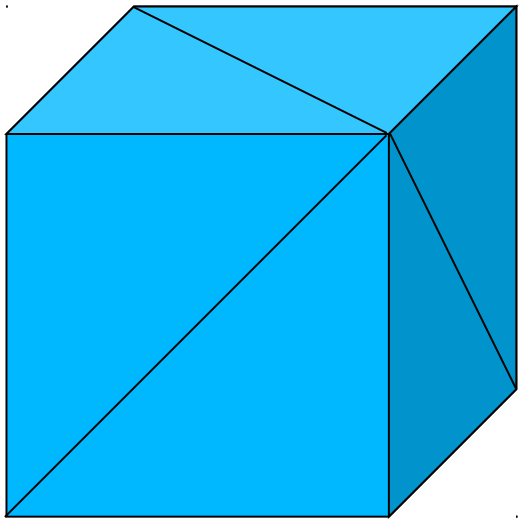
# Triangle Meshes (.obj)

vertices

```
v -1 -1 -1
v  1 -1 -1
v -1  1 -1
v  1  1 -1
v -1 -1  1
v  1 -1  1
v -1  1  1
v  1  1  1
```

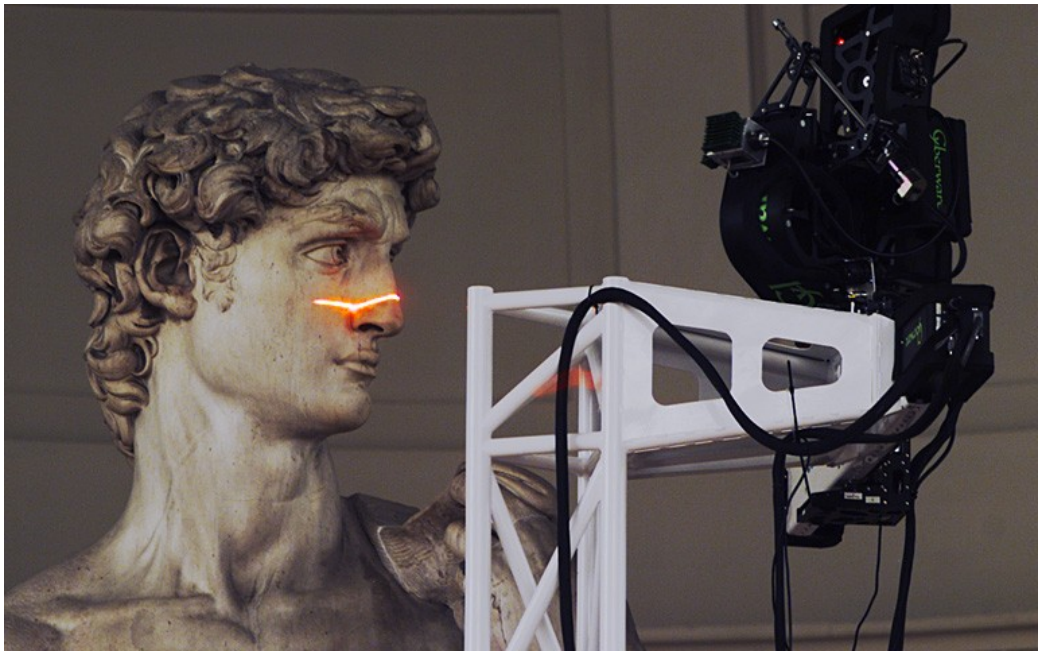
triangles

```
f 1 3 4
f 1 4 2
f 5 6 8
f 5 8 7
f 1 2 6
f 1 6 5
f 3 7 8
f 3 8 4
f 1 5 7
f 1 7 3
f 2 4 8
f 2 8 6
```



# Acquiring Geometry

- 3D Scanning



Digital Michealangelo Project (Stanford)

Cyberware

# Next Week:

Ray Tracing
Surface reflectance