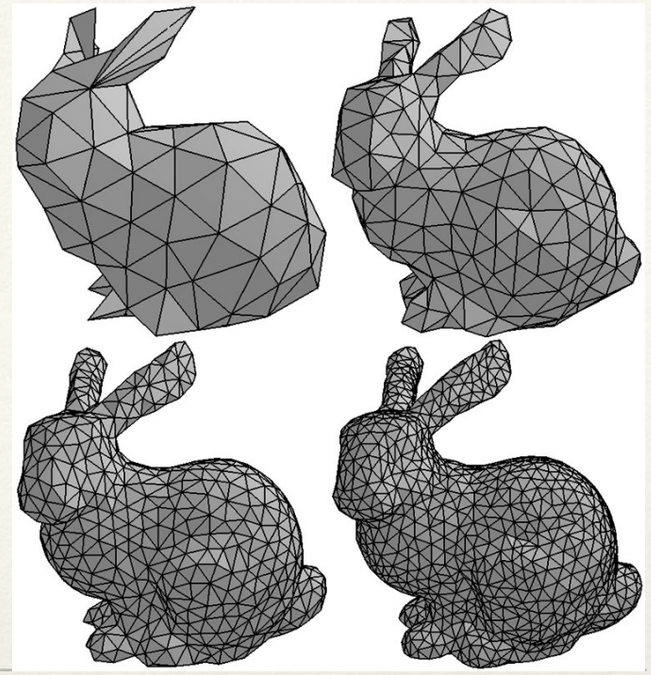
Dr. Sarah Abraham
University of Texas at Austin
Computer Science Department

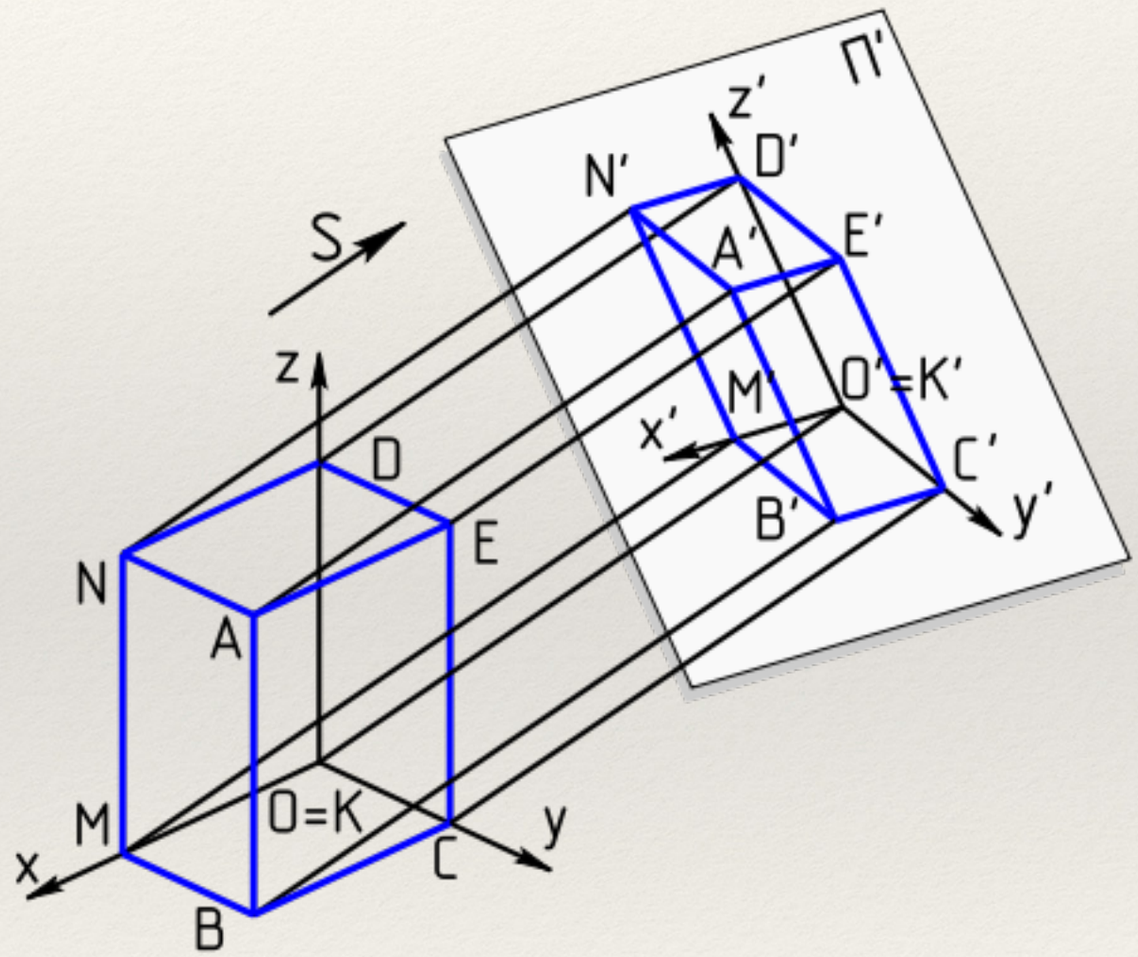


3D Shapes

Elements of Graphics
CS324e

Perspective

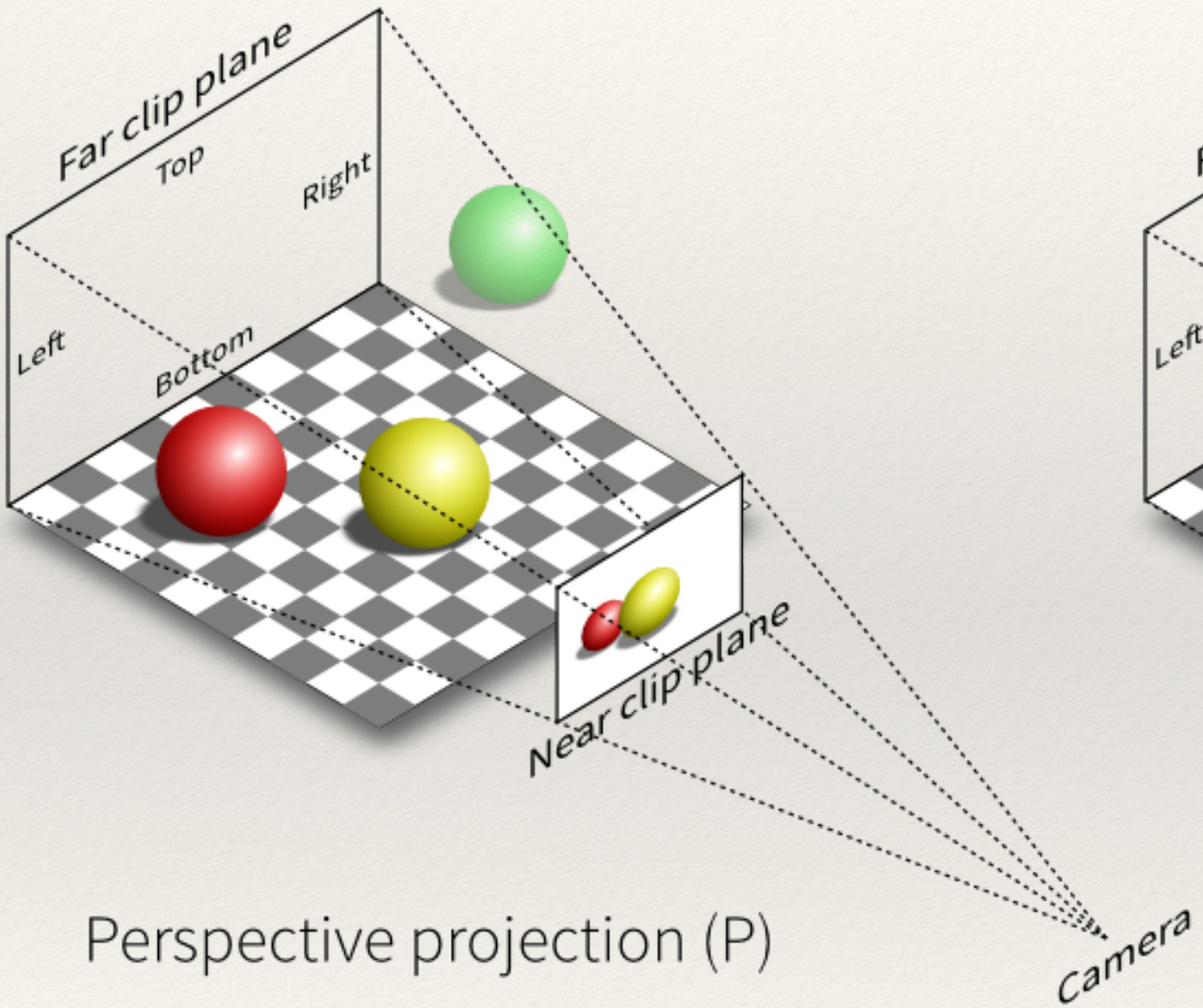
- The representation of depth and object relations on a flat surface
- A technique used by artists and cameras
- Adds realism to a scene by modeling what our eye does automatically



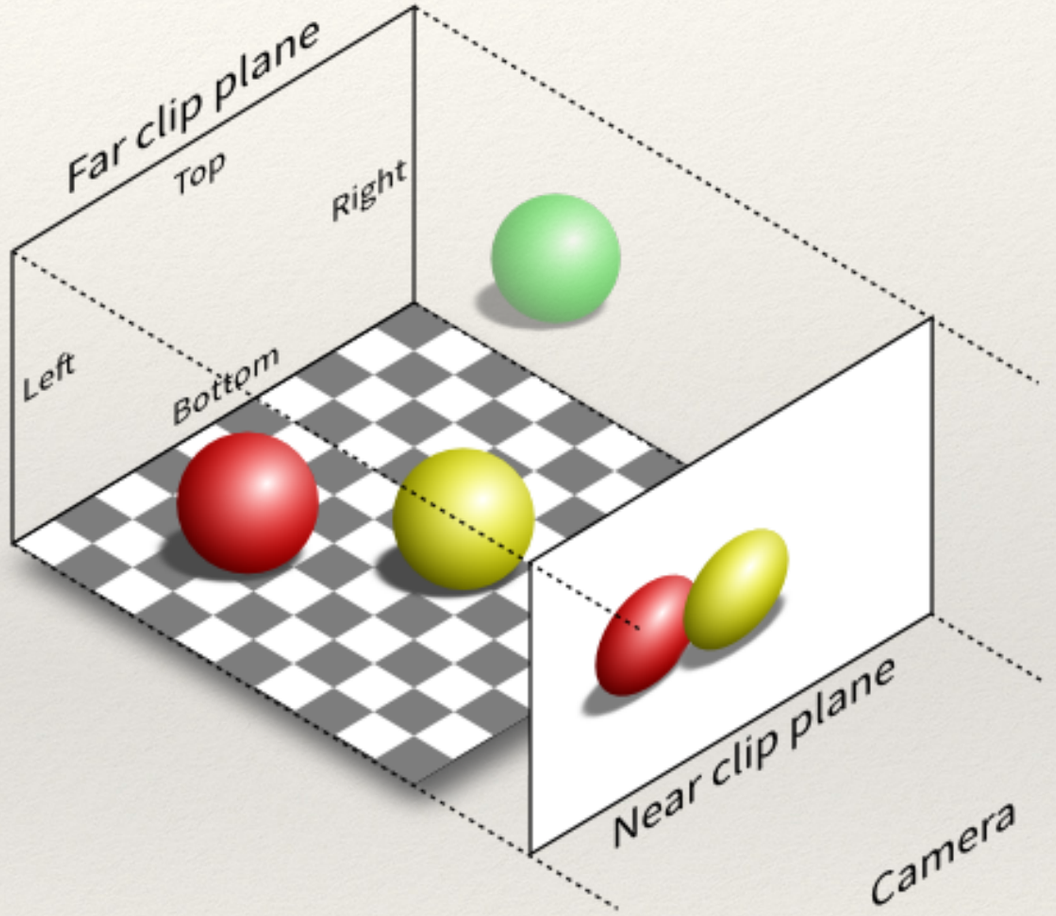
Projections

- Cameras can project in two ways: orthographic or perspective
- Orthographic
  - Distant objects appear at same scale as closer objects
  - Gives a flat, “technical” appearance
- Perspective
  - Distant objects appear at smaller scale than closer objects
  - Gives a physically realistic appearance

Orthographic vs Perspective Projections



Perspective projection (P)

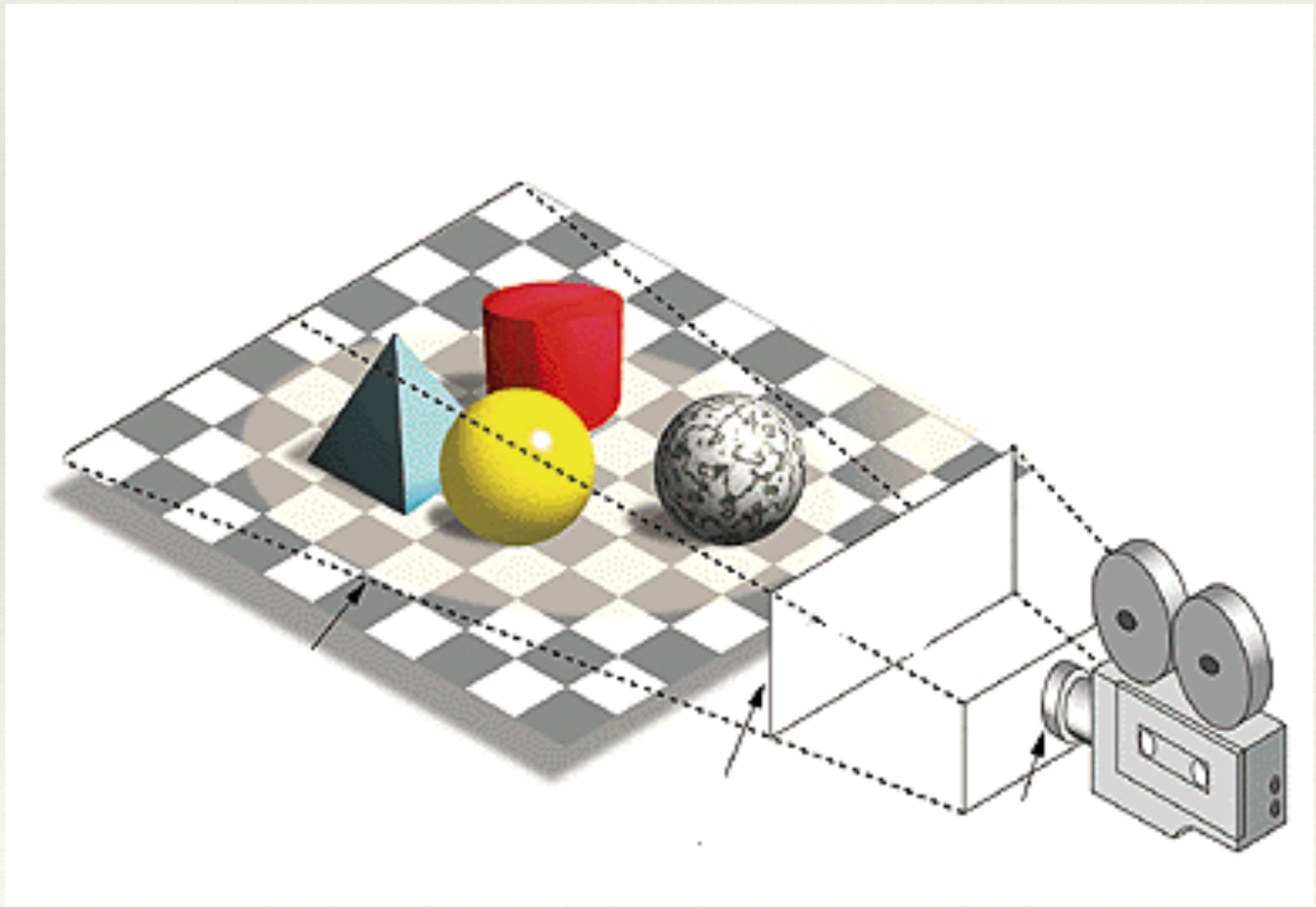


Orthographic projection (O)

(Glumpy)

Consider

- What does the orthographic view of this scene look like? What about the perspective view?



Changing Render Mode

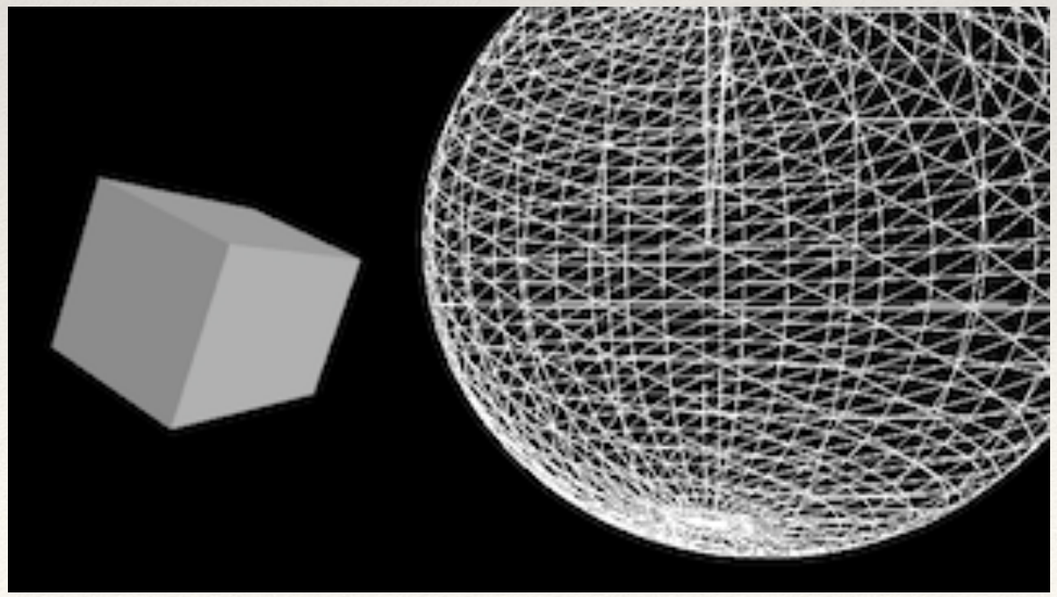
- By default Processing assumes 2 dimensions
- We'll need to notify it that we want to account for depth and perspective projections to work in 3D
- To use the P3D renderer:

```
size(width, height, P3D);
```

(Note that there is also a P2D renderer. P2D and P3D renderers access OpenGL making them faster and with more effects)

3D Primitive Shapes

- `box()` and `sphere()` are 3D primitives
- More complex shapes can be made with `vertex()`
- `fill()` and `stroke()` work on these meshes
- Note: cannot set position with `box` and `sphere` -- must use affine transformations!



Importing Meshes

- Meshes can be loaded into PShape objects
- Once we're in the 3D rendering mode, we can import .obj files using:

```
PShape object =
loadShape("objectname.obj");
```

Displaying Meshes

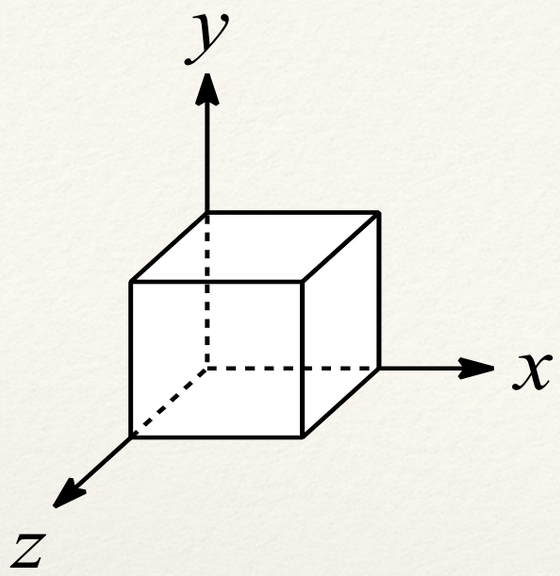
- To display this object, we call `shape()` in the `draw()` function:

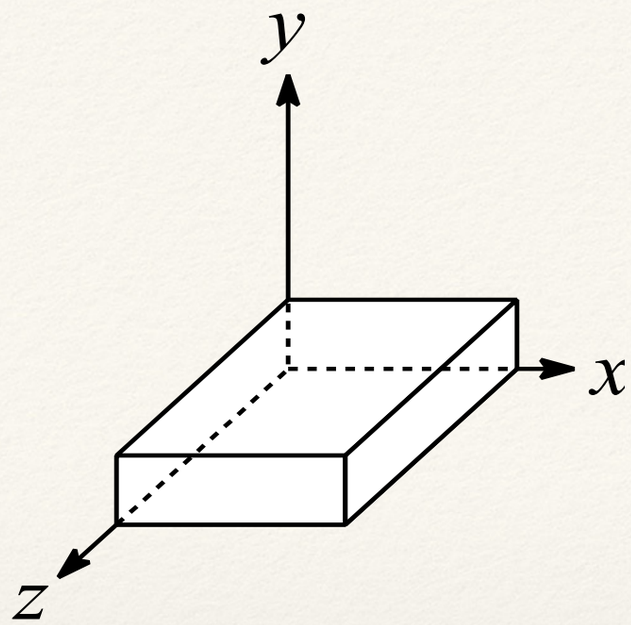
```
   shape(object, 0, 0, object.width,
object.height);
```

- Note: the object might not be oriented for our screen size, so you may have to scale/rotate to see the image…

3D Transformations

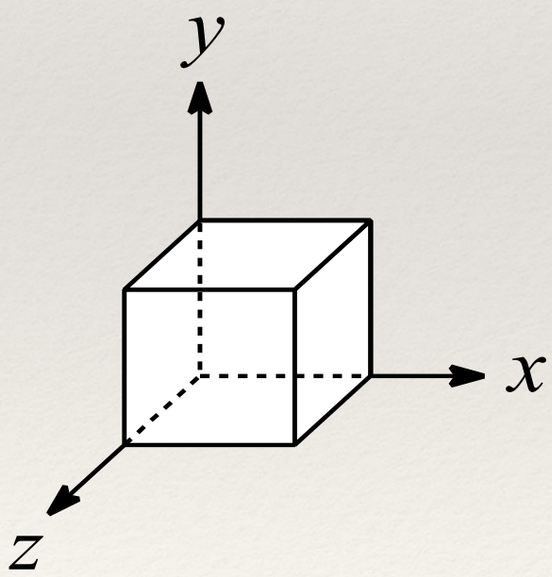
- We have access to the same affine transformations in 3D as in 2D:
  - Scale
  - Rotate
  - Translate
- Their mathematical notion looks similar as well!

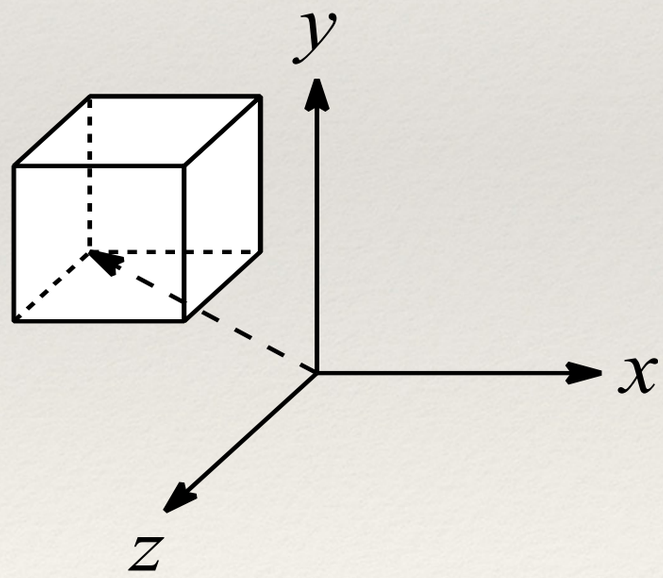




$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Scaling





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

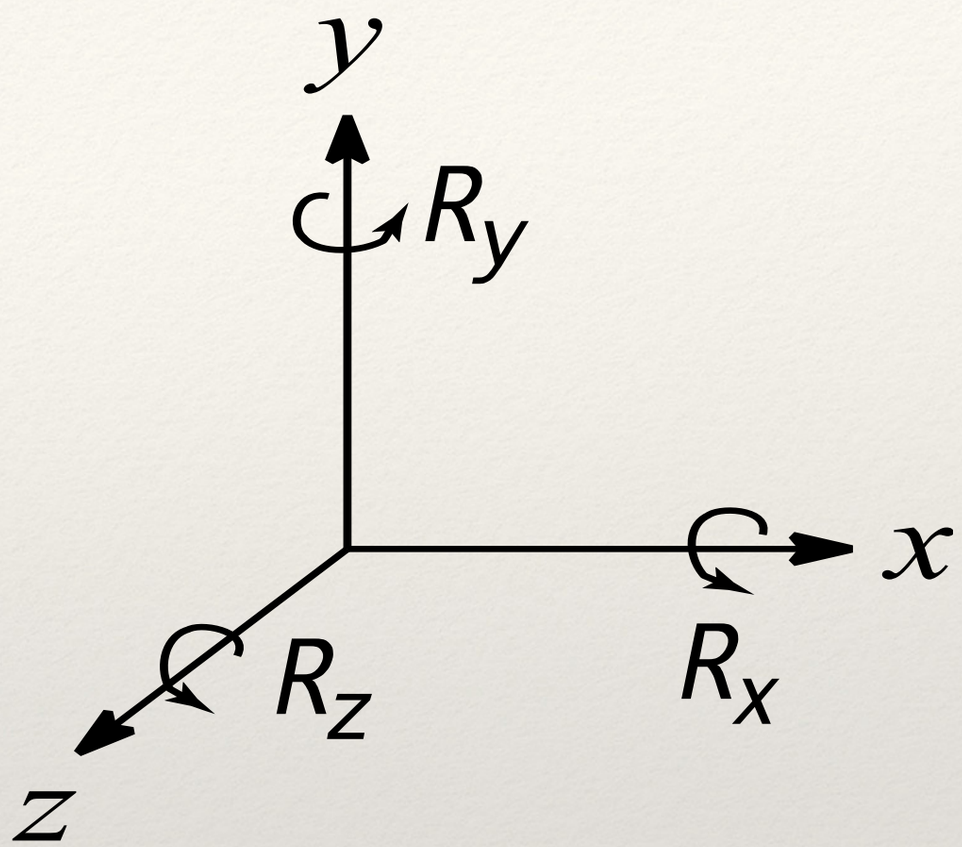
Translation

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

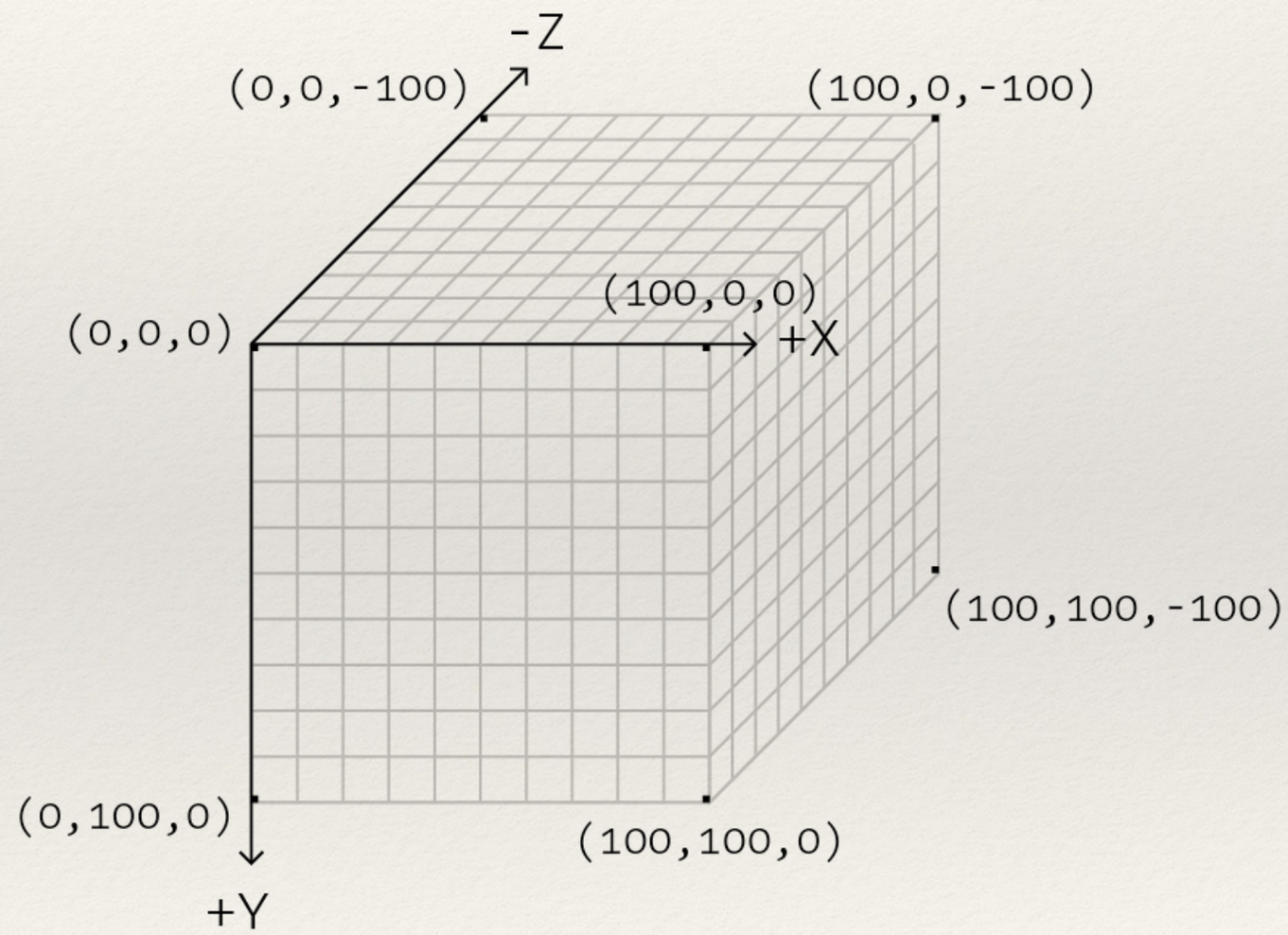


(Use right hand rule)

Well, except that…

Processing Coordinate System

- Processing uses a "left-handed" coordinate system
- Same concept, just make sure the model is clear in your head before trying things!



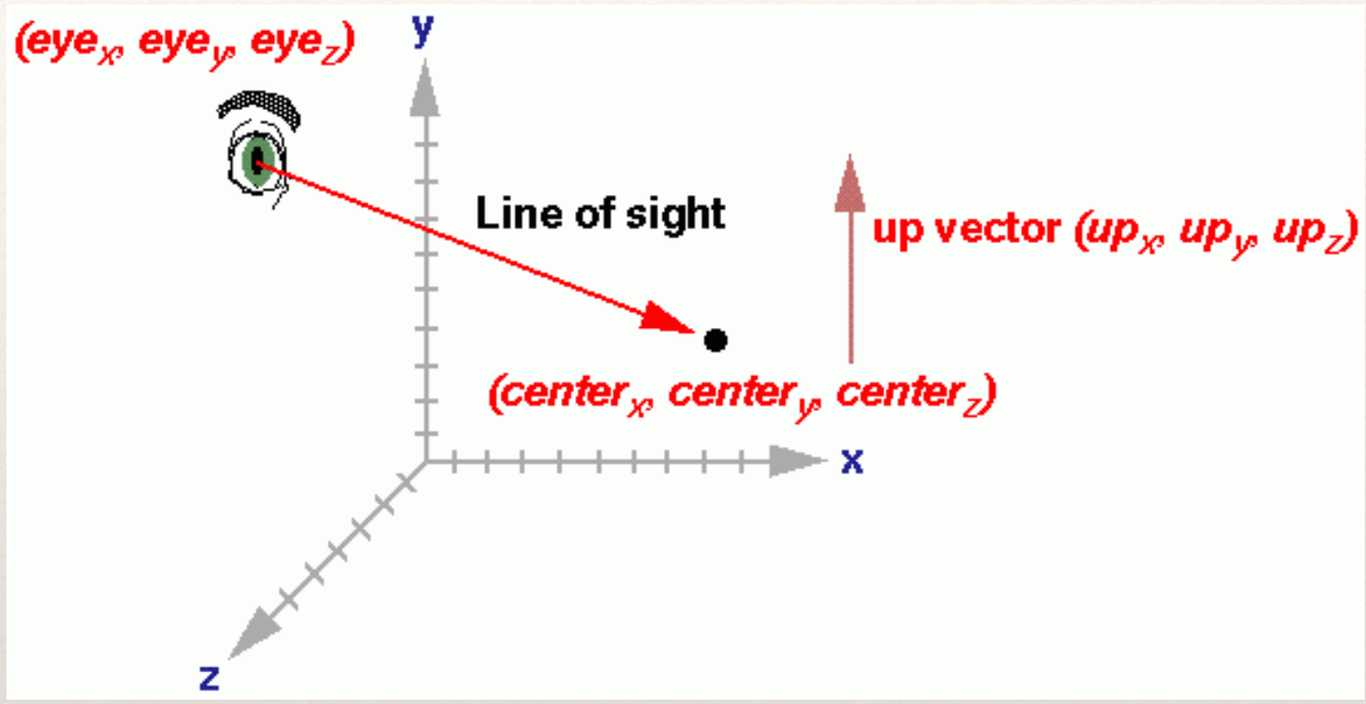
Processing 3D Transformations

- Mostly the same as 2D transformations:
  - `translate(x, y, z);`
  - `scale(x, y, z);`
  - `rotateX(θ);`
  - `rotateY(θ);`
  - `rotateZ(θ);`

3D Example

Camera

- Where is the camera?
  - `eyeX`, `eyeY`, `eyeZ`
- Where is the camera looking?
  - `centerX`, `centerY`, `centerZ`
- How is the camera oriented?
  - `upX`, `upY`, `upZ` (note: this is a direction not a point!)



(http://www.alpcentauri.info/)

Setting the Camera in a Scene

- `camera(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);`
- Example code for changing the height of the camera based on mouse movement:

```
camera(200.0, mouseY, 120.0,
       0.0, 0.0, 0.0,
       0.0, 1.0, 0.0);
```

Hands-on: Moving Cameras

- Today's activities:

  1. Create several 3D shapes
  2. Set up a camera to look at these objects
  3. Experiment with moving the camera along the z, y, and z axes
  4. Experiment with rotating the camera around a point. Note that `beginCamera`/`endCamera` may be useful for this