

3D Shapes

and probably lighting

Shapes

We can use `beginShape()`/`endShape()` combined with `vertex()`/`curveVertex()` to draw shapes.

We can import shapes from SVGs with `loadShape()`, similar to `loadImage()`.



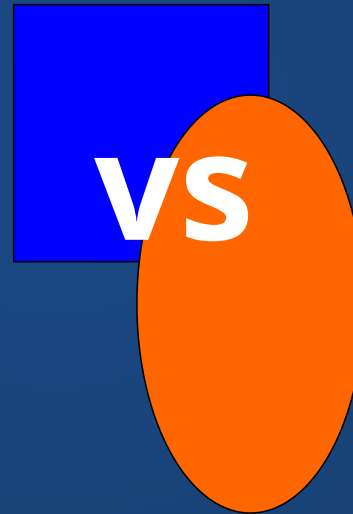
Lerp

We can use the `lerp()` function to smoothly interpolation motion between endpoints.

Interpolation can also be applied to many different types of values.

How does PShape render its shapes? Is it the same as the shapes created in Processing?

```
1 <g
2   inkscape:label="Layer 1"
3   inkscape:groupmode="layer"
4   id="layer1">
5   <rect
6     style="fill:#0000ff;stroke:#000000"
7     id="rect234"
8     width="76.99118"
9     height="77.594948"
10    x="17.603966"
11    y="36.327324" />
12   <ellipse
13     style="fill:#ff6600;stroke:#000000"
14     id="path498"
15     cx="89.078629"
16     cy="126.6972"
17     rx="34.270332"
18     ry="63.077023" />
19 </g>
```

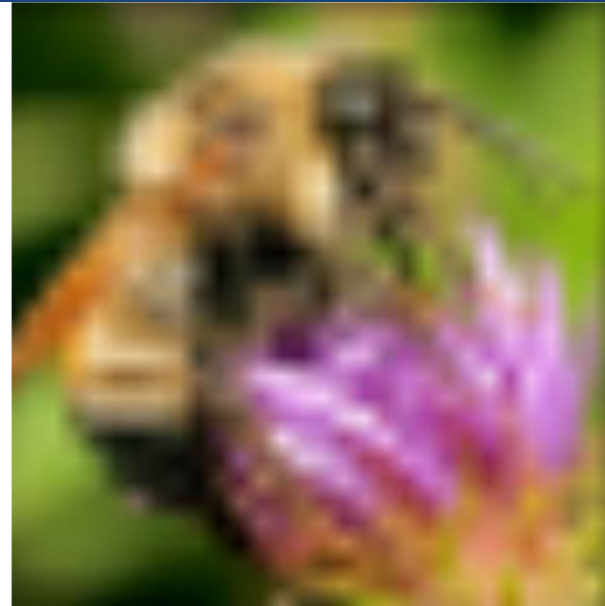


```
1 fill(#0000ff);
2 rect(17.6, 32.3, 76.9, 77.6);
3
4 fill(#ff6600);
5 ellipse(89.1, 126.7, 34.3, 63.1);
```

How can you interpolate between pixels in an image?

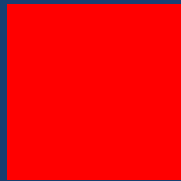


Nearest-neighbor interpolation

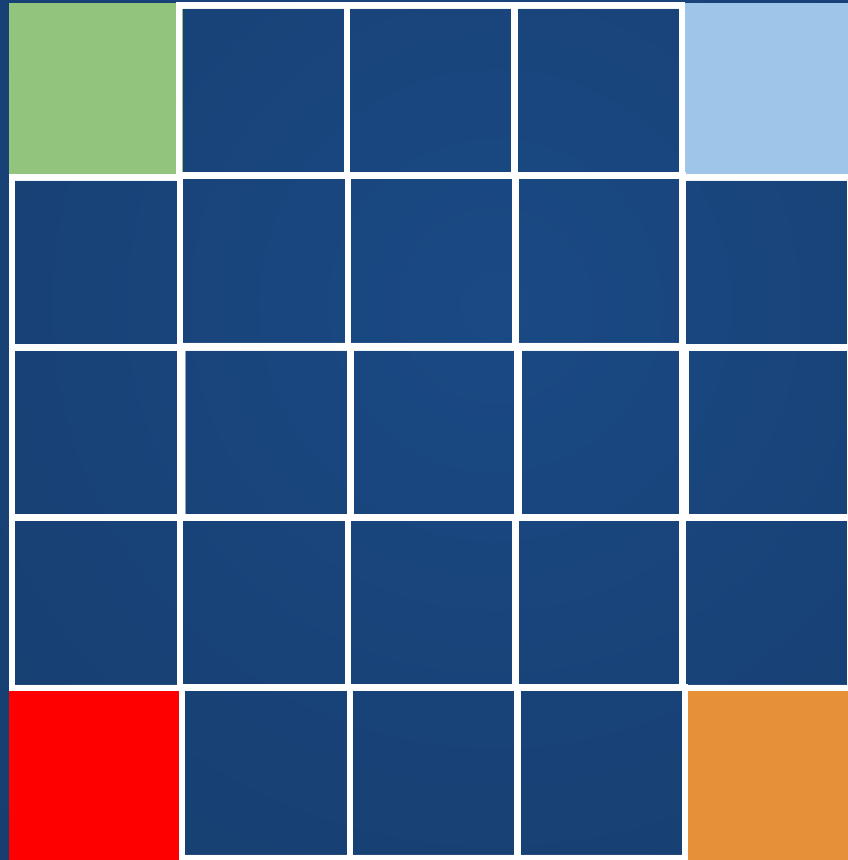


Bilinear interpolation

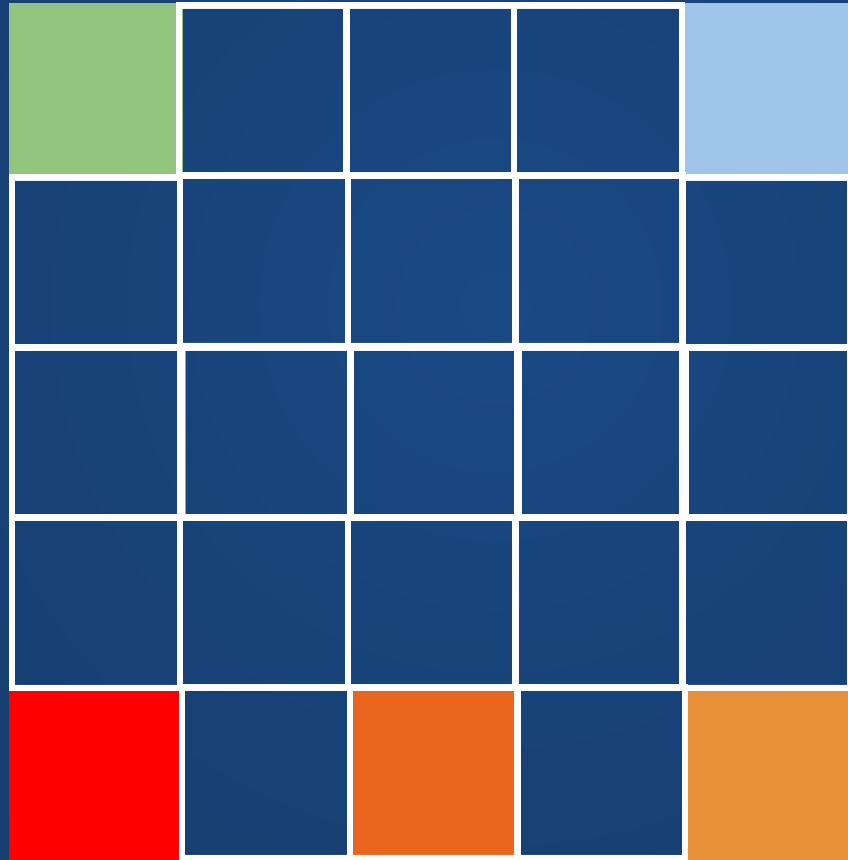
How can you interpolate between pixels in an image?



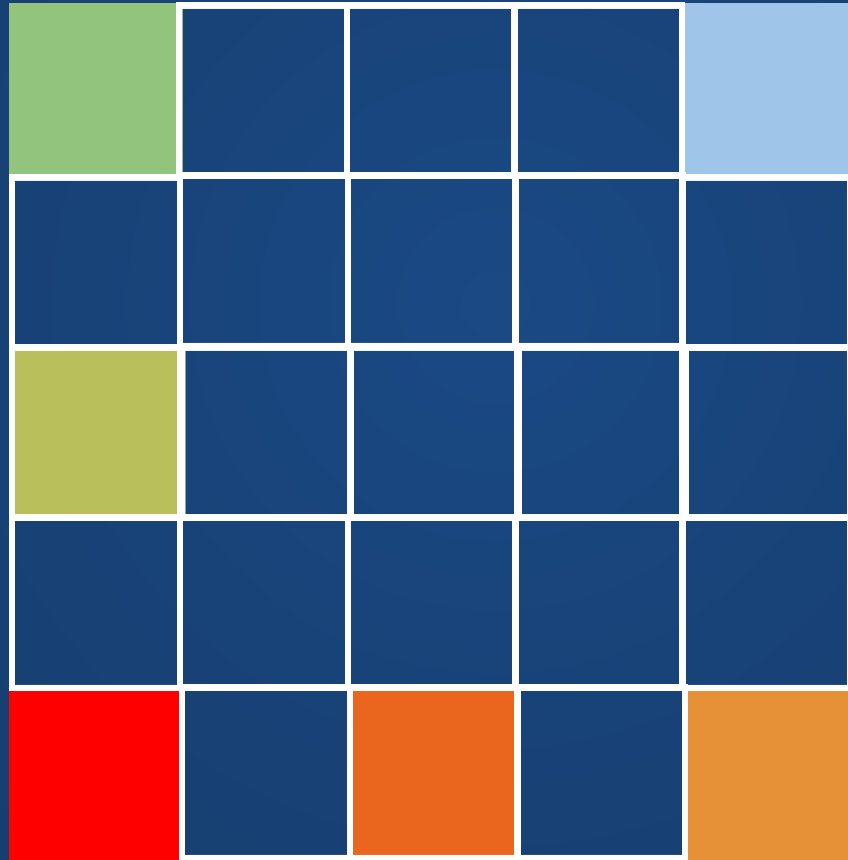
How can you interpolate between pixels in an image?



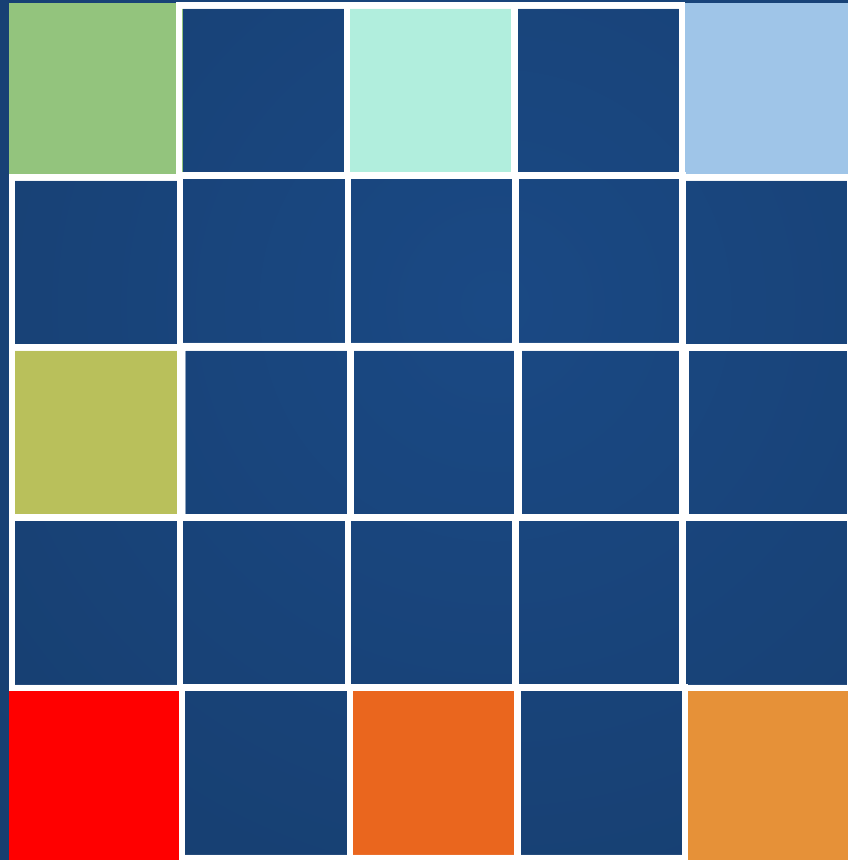
How can you interpolate between pixels in an image?



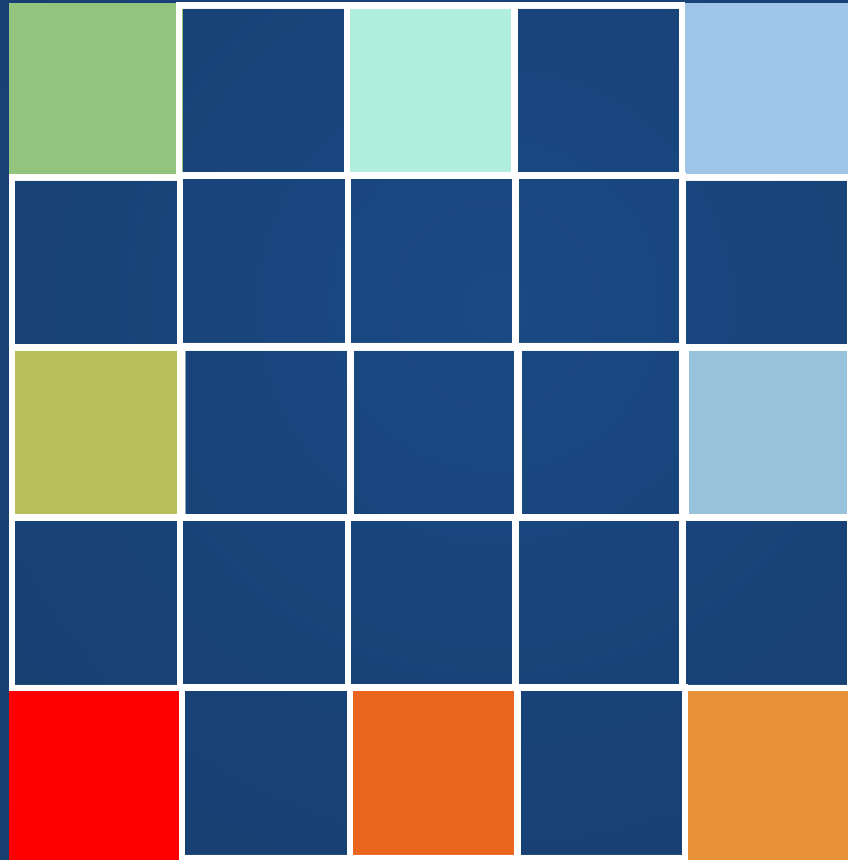
How can you interpolate between pixels in an image?



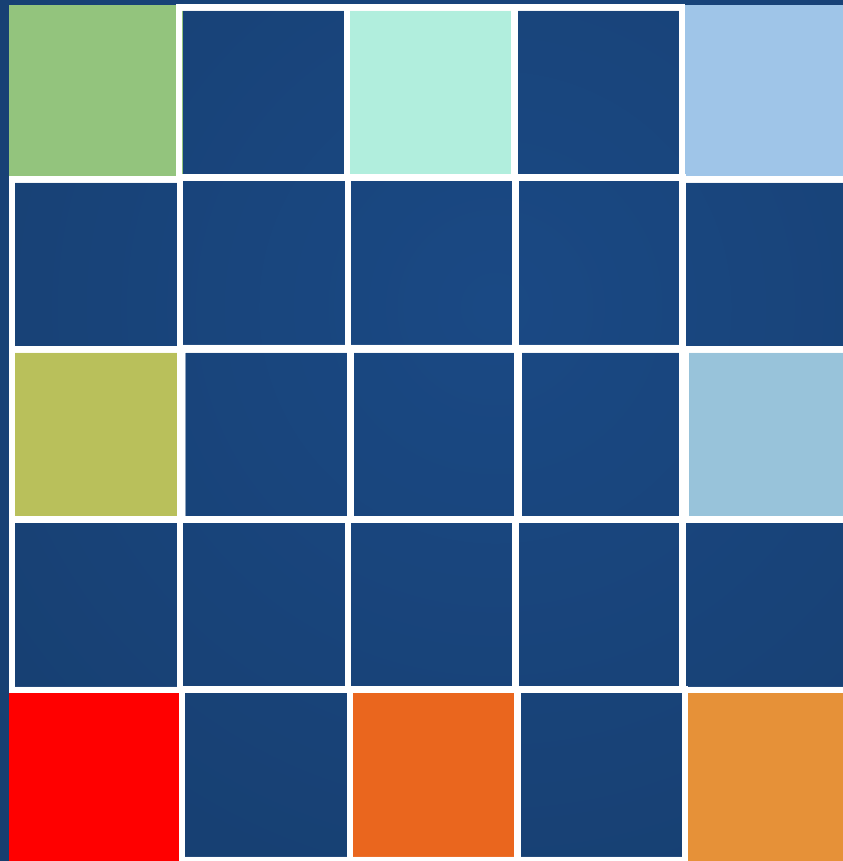
How can you interpolate between pixels in an image?



How can you interpolate between pixels in an image?



How can you interpolate between pixels in an image?



Different from kernel: creates new pixels and fills them in!

Is it possible to draw a sphere with a reasonable number of points?

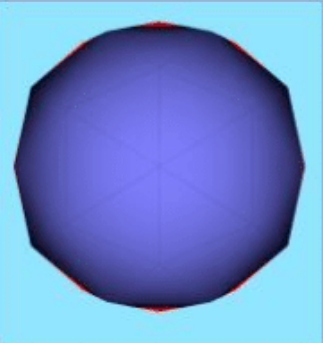
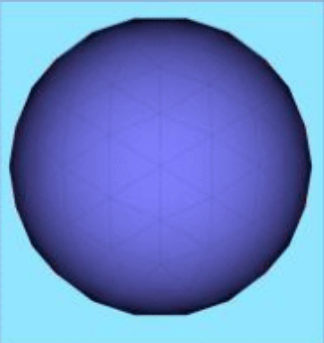
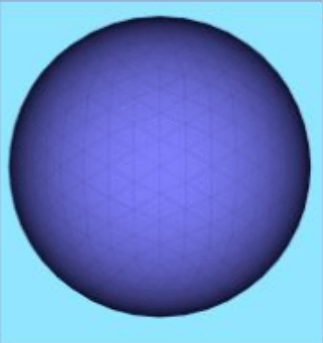
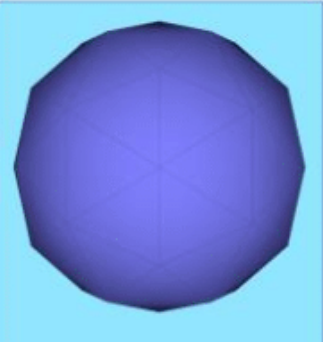
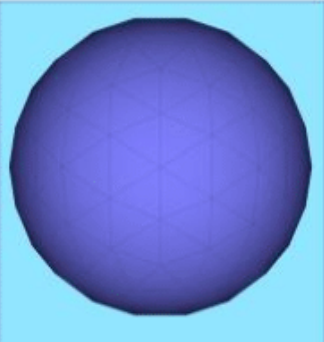
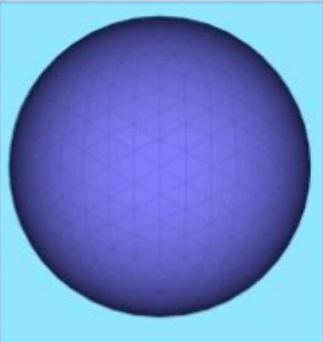
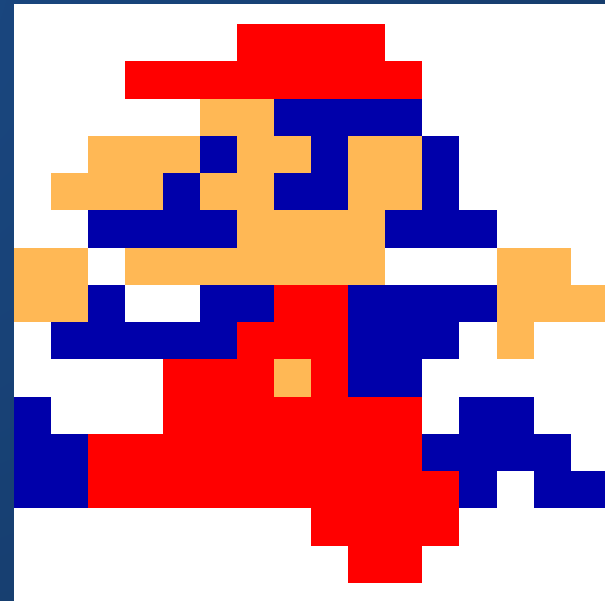
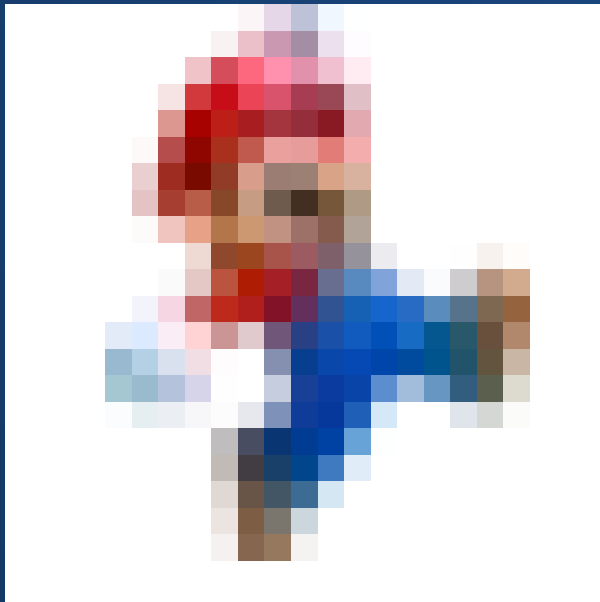
model method	Tessellated icosahedron, 80 triangles	Tessellated icosahedron, 320 triangles	Tessellated icosahedron, 1280 triangles
Phong normals			
Consistent normals			

Figure 9. Increasing surface smoothness results in almost identical shading for Phong and consistent normals. Red color indicates pixels with negative dot products.

How does a scene graph interact with manual (user-initiated) adjustments to an object?

The same way it does to a computer-initiated adjustment.

Do games actually use SVGs for their game assets?



Code Review

**What are the advantages
of controlling the shape's
movement with
interpolation?**

**What are the advantages
of controlling the shape's
movement with speed?**

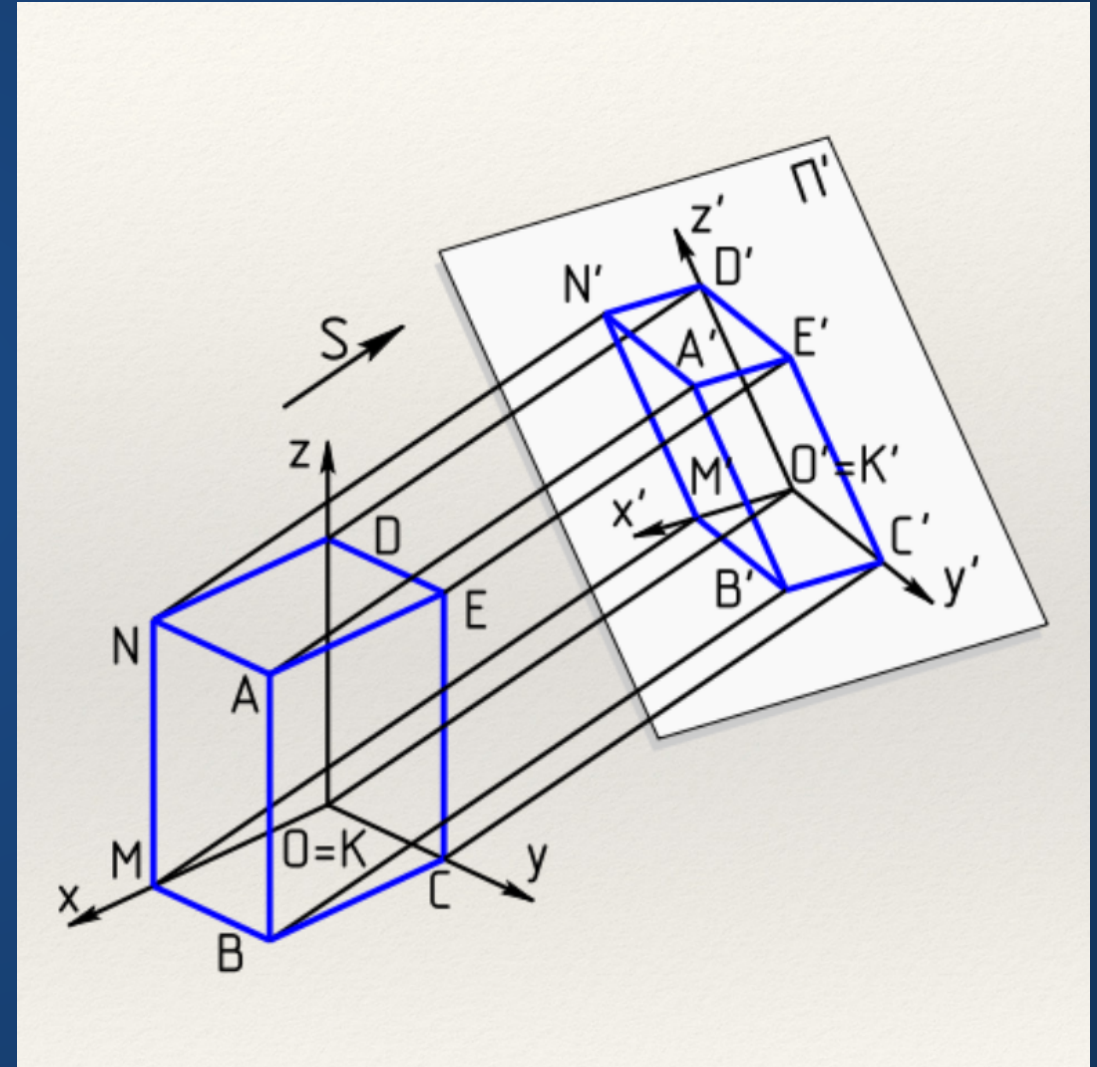
What Changes in 3D?

Our *scene* is now three-dimensional.

But we're still trying to draw 2D images!
Suddenly, we're going to have many more
choices about how to make pictures.

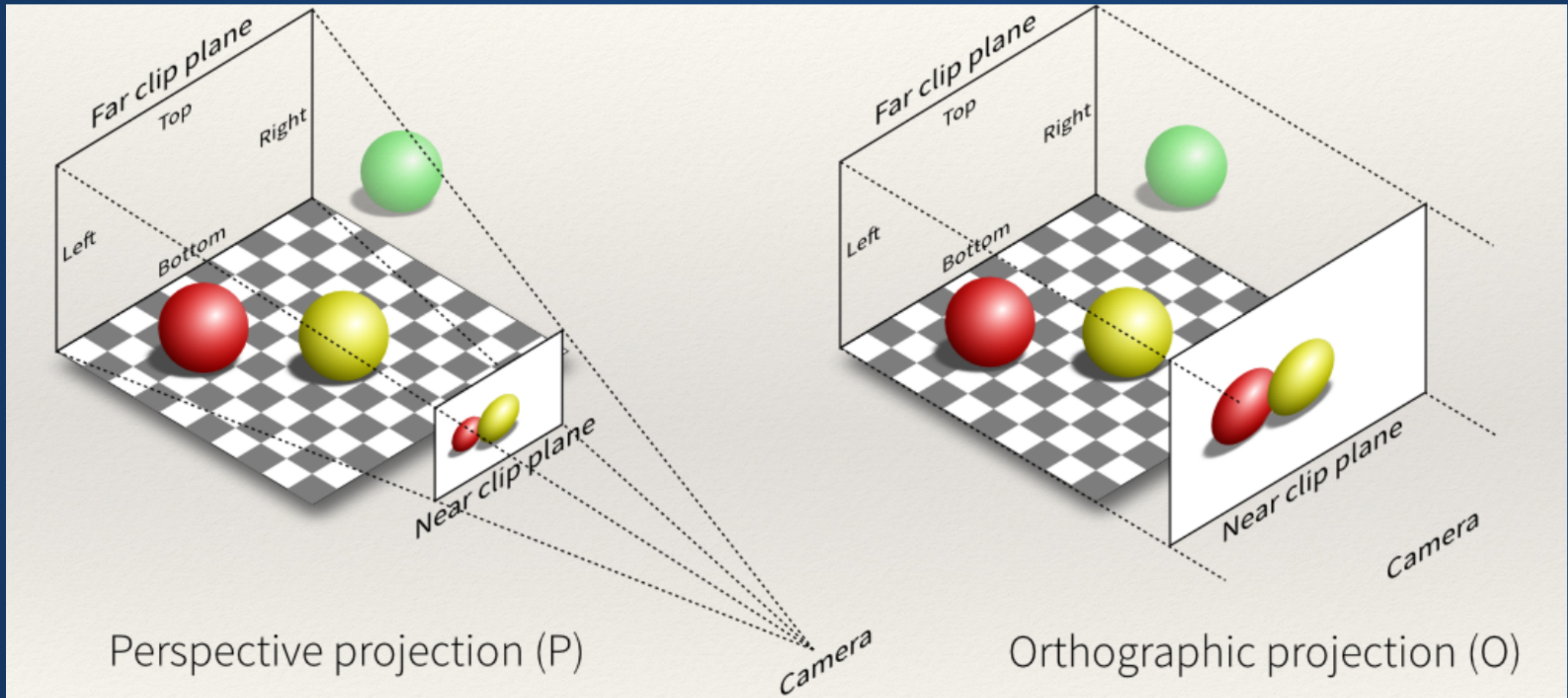
Example: Perspective

- A representation of depth and object relations on a flat surface
- Technique used by artists and cameras
- Adds realism to a scene by modeling what our eyes already do



Perspective Choices

Cameras can project in two ways: orthographic or perspective



Perspective Choices

Orthographic

- Distant objects appear at the same scale as closer objects.
- Gives a flat, technical appearance

Perspective

- Distant objects appear at a smaller scale than closer objects.
- Gives a physically realistic appearance.

There are many more choices in 3D: camera position, direction, lighting, even other choices within perspectives!

3D Shapes in Processing

By default, Processing assumes you want to work in 2 dimensions.
We'll need to tell it we want to work in 3D.

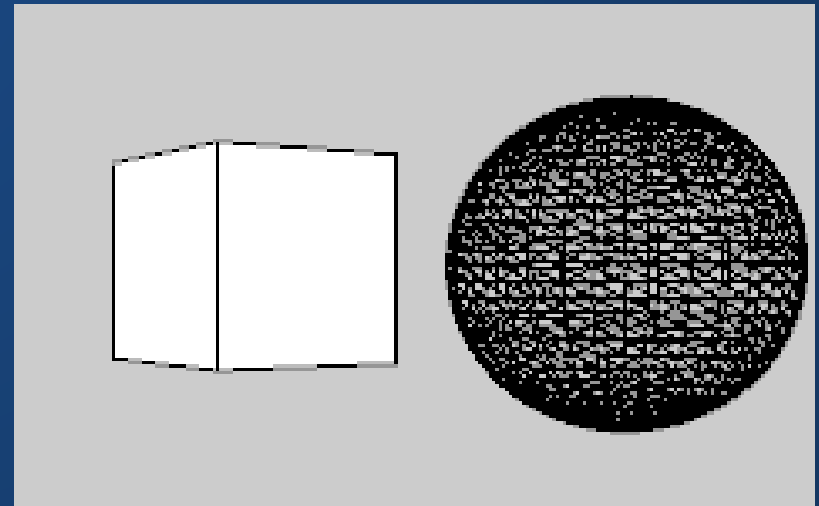
```
size(width, height, P3D);
```

Note: there is also a P2D renderer. P2D and P3D work in OpenGL,
making them faster with more effects.

Replacements for rect()

Now that we're in 3D, `rect()` and `ellipse()` calls won't cut it anymore!
Instead, we can use `box()` and `sphere()` to create boxes and spheres, respectively.
More complex shapes can be made with `beginShape()` and `vertex()`

We can use `fill()` and `stroke()` on these primitives, but cannot set positions---must use affine transforms to do this.



Meshes

Just like with SVGs, we can load meshes into PShape objects.

```
1 PShape object = loadShape("filename.obj");
```

Note: we need to be in 3D rendering mode in order to load .obj files!

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

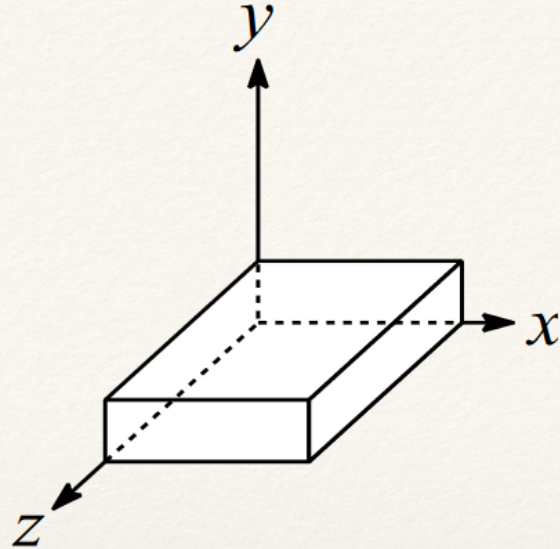
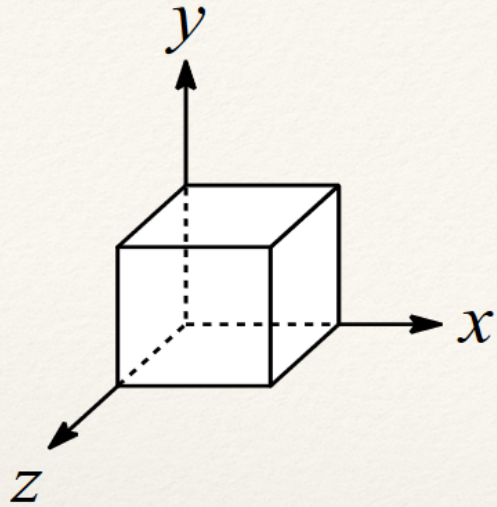

3D Transforms

In 2D, we had access to three basic transformations:

- scale
- rotate
- translate

In 3D, we're going to have access to the same set of transformations. In fact, their mathematical notation looks similar too!

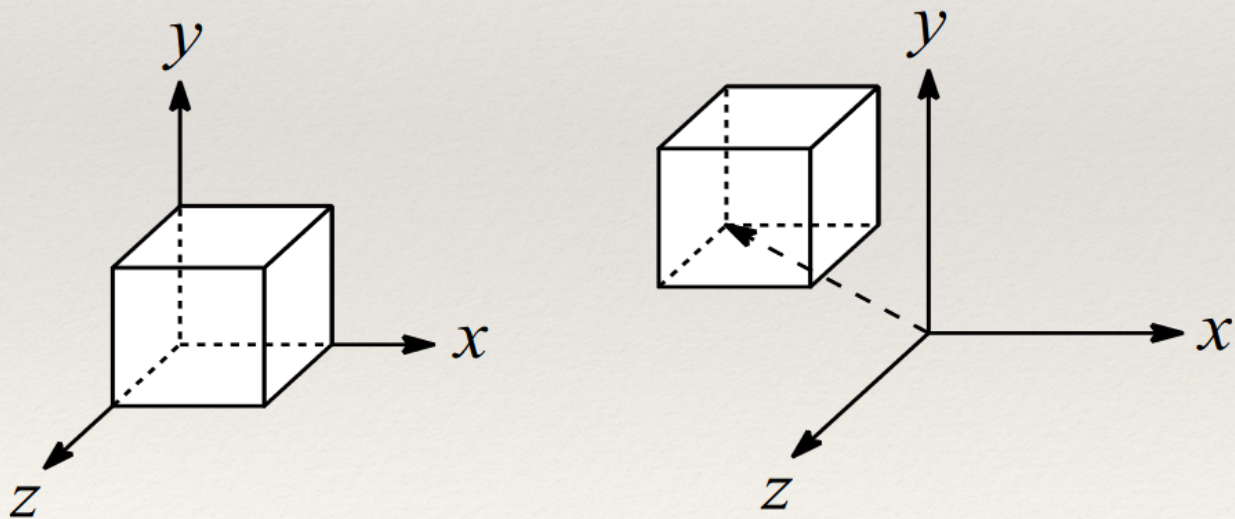
Scaling



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

Scaling

Translation



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

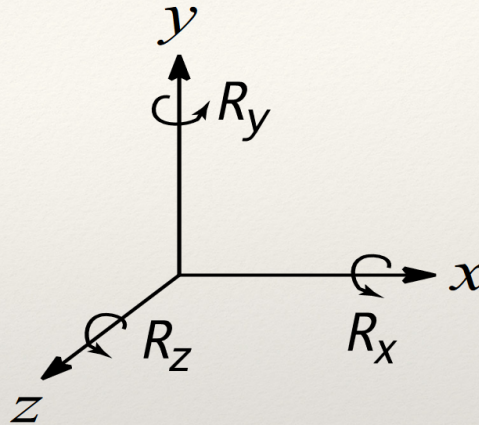
Translation

....Rotation

Okay, so it turns out that rotations in 3D can get a little tricky...

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

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



$$R_{\tilde{\omega}}(\theta) = e^{\tilde{\omega}\theta}$$

$$= I + \tilde{\omega} \sin \theta + \tilde{\omega}^2 (1 - \cos \theta)$$

$$= \begin{bmatrix} \cos \theta + \omega_x^2 (1 - \cos \theta) & \omega_x \omega_y (1 - \cos \theta) - \omega_z \sin \theta & \omega_y \sin \theta + \omega_x \omega_z (1 - \cos \theta) \\ \omega_z \sin \theta + \omega_x \omega_y (1 - \cos \theta) & \cos \theta + \omega_y^2 (1 - \cos \theta) & -\omega_x \sin \theta + \omega_y \omega_z (1 - \cos \theta) \\ -\omega_y \sin \theta + \omega_x \omega_z (1 - \cos \theta) & \omega_x \sin \theta + \omega_y \omega_z (1 - \cos \theta) & \cos \theta + \omega_z^2 (1 - \cos \theta) \end{bmatrix},$$

Summary of Transformations

Translate and scale function basically the same as their 2D counterparts, but with an additional function argument.

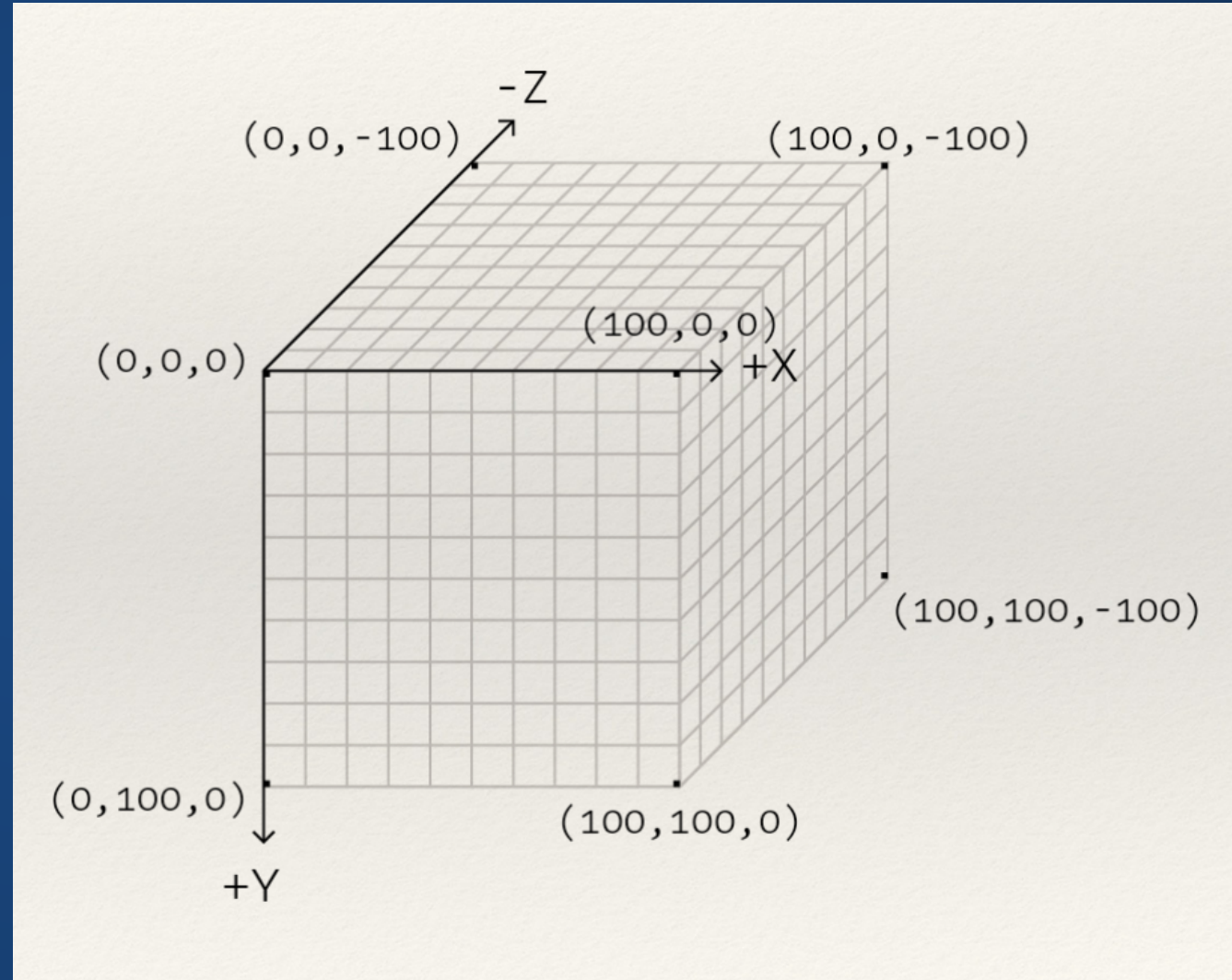
Rotate is replaced by three functions which each take a parameter.

- `translate(x, y, z)`
- `scale(x, y, z)`
- `rotateX(θ)`
- `rotateY(θ)`
- `rotateZ(θ)`

Quick Aside: 3D Coordinates

Most coordinate systems are "right-handed". Processing uses a left-handed coordinate system.

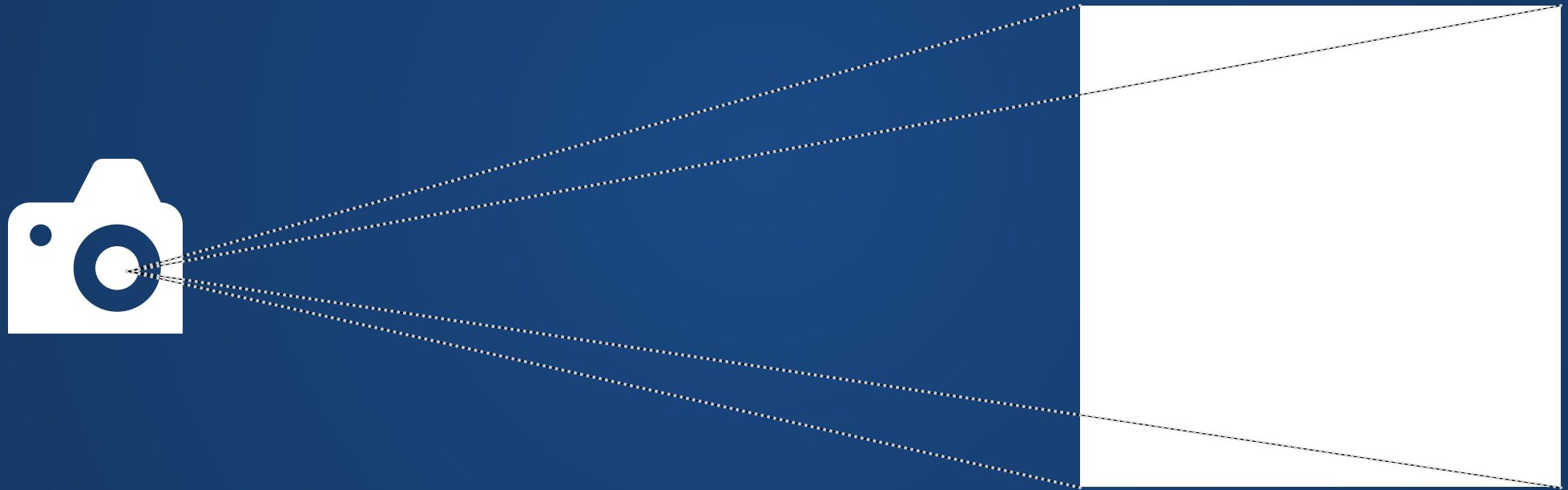
The concepts are the same, just be sure you have the right picture in your head before trying things!



Example

Camera

Camera in 2D



Camera in 3D

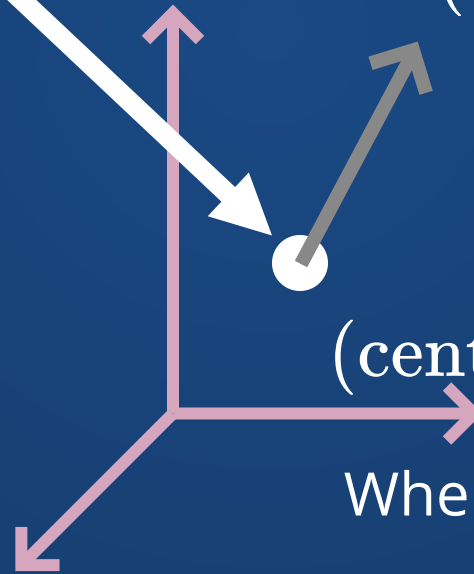
Where is the camera?

(eye_x, eye_y, eye_z)



What direction is the camera turned?

(up_x, up_y, up_z)



$(center_x, center_y, center_z)$

Where is the camera looking?

Camera in Processing

```
camera(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);
```

Example 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: 3D Shapes

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

Remember, you need to pass `P3D` into `size()` to get a 3D environment!

Index Cards!

1. Your name and EID.
2. One thing that you learned from class today. You are allowed to say "nothing" if you didn't learn anything.
3. One question you have about something covered in class today. You *may not* respond "nothing".
4. (Optional) Any other comments/questions/thoughts about today's class.