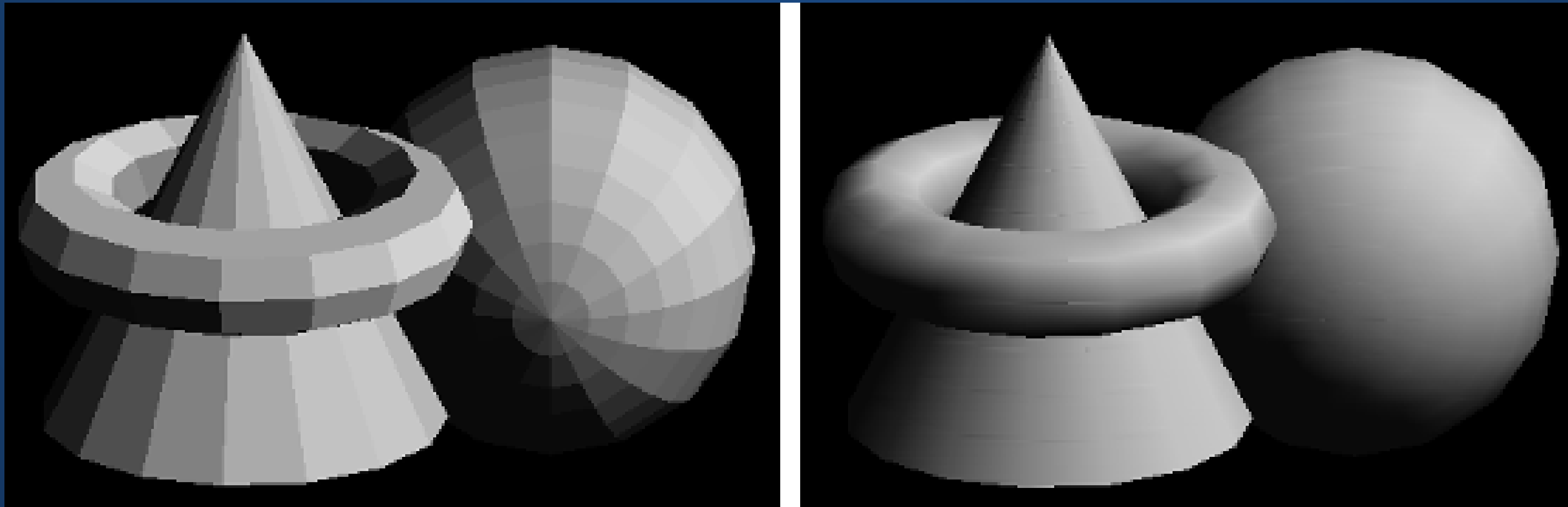
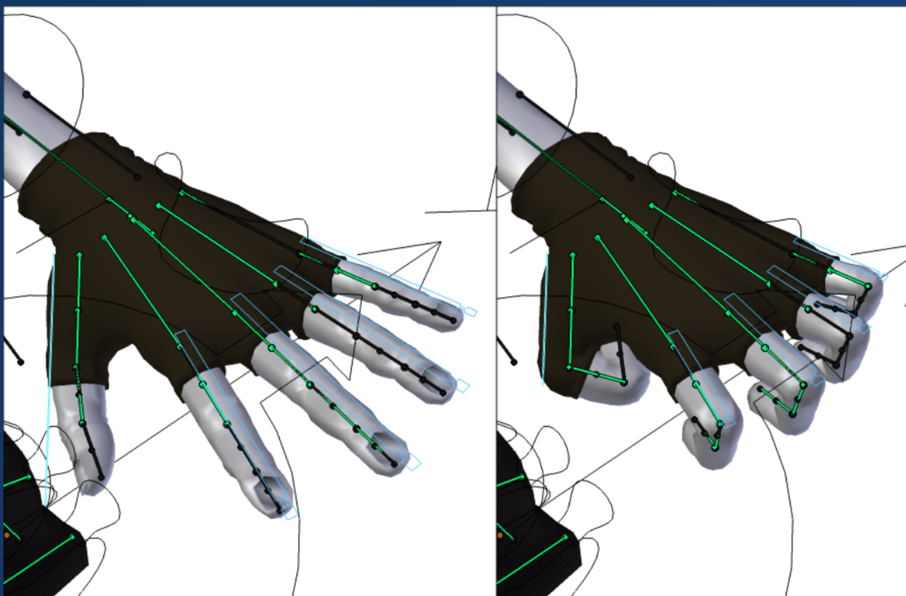
Why are triangles faster to compute than other shapes?

Triangles are guaranteed to be completely planar.

Wouldn't using triangles/polygons give the end result a blocky/flat look instead of a smooth one?



How do you skin a rig?



The process for this is surprisingly complex!

The general idea is this:

- Take the bones and a mesh in a default pose (e.g. the infamous T-pose).
- Give each point on the skin an association to the bones near them.
  - Example: This point on the skin is distance 8 from bone A and distance 2 from bone B. The point position will be 80% controlled by bone B and 20% by bone A.
- When the bones move, recalculate the skin positions using the weights.

How does a scene hierarchy save time or space?

It doesn't.

It makes it easier to think about what's going on in a scene.

How do we decide which objects go together in a scene hierarchy? Can they change on-the-fly?

Scene hierarchies are designed, not prescribed.

You can choose how you want to group shapes. You can move shapes between groups. Then, use it and see whether it works or not!

Can we program gravity/physics in Processing?

Absolutely! In fact, we'll see how to do this starting next week.

How do you figure out where the triangles go?

Similar to how you figure out where the shapes go in 2D. You do some programming, which is backed by some math, and then you see if the results look right.

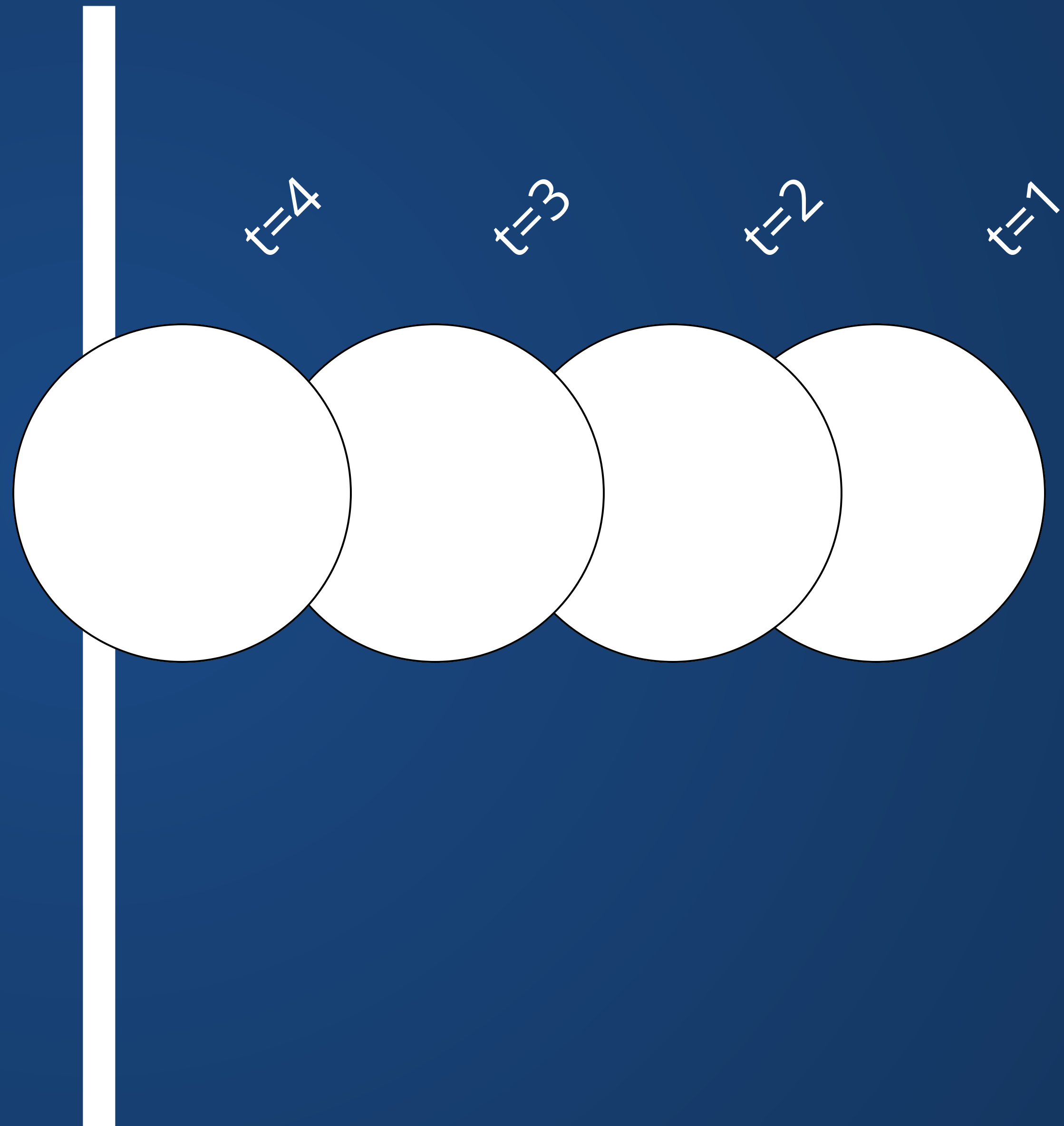
To generate the shapes in the first place, you usually use 3D modeling software like Maya or Blender.

Code Review

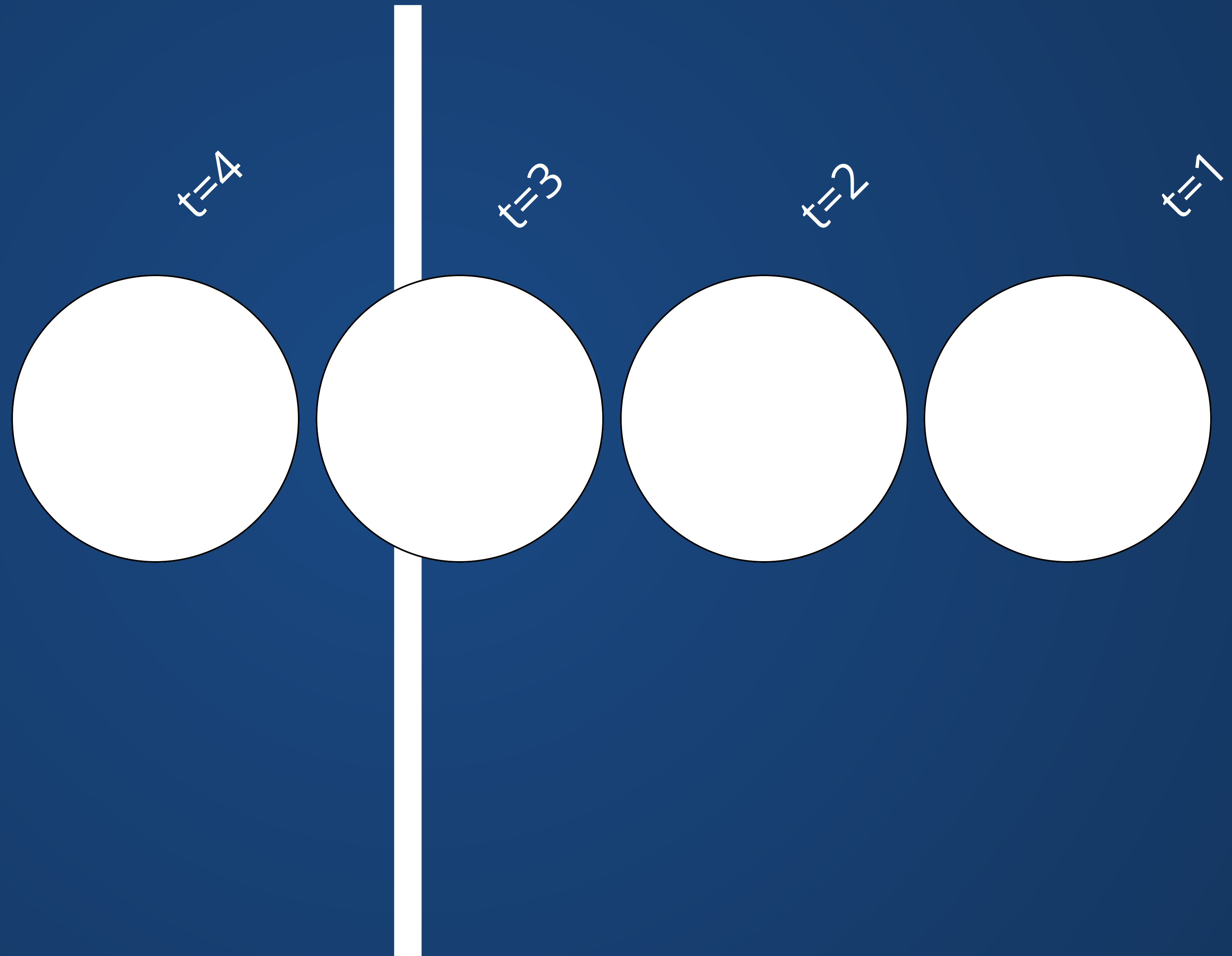
…oh. Right.

Why is collision detection hard??

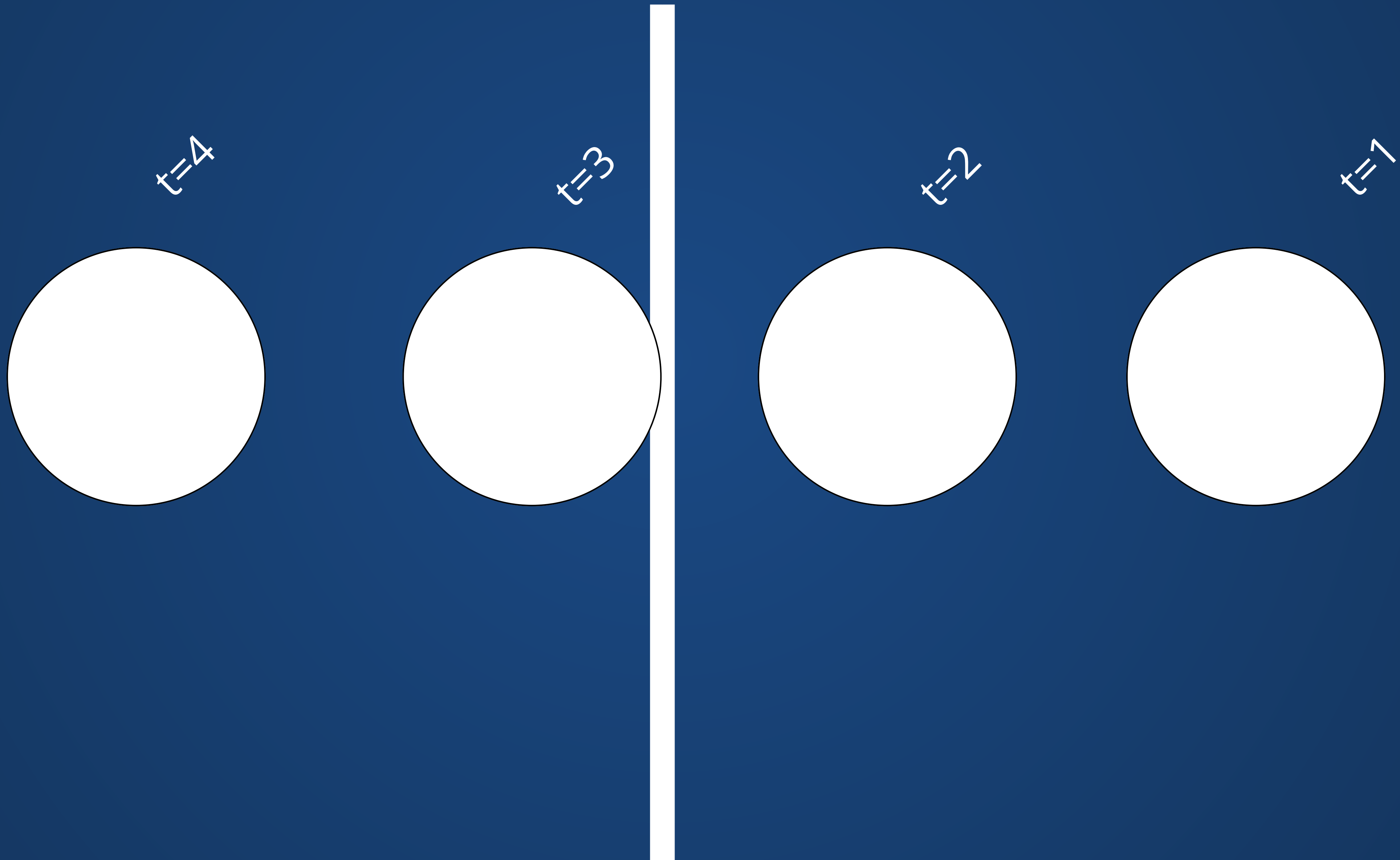
Problem 1: Discrete Timesteps



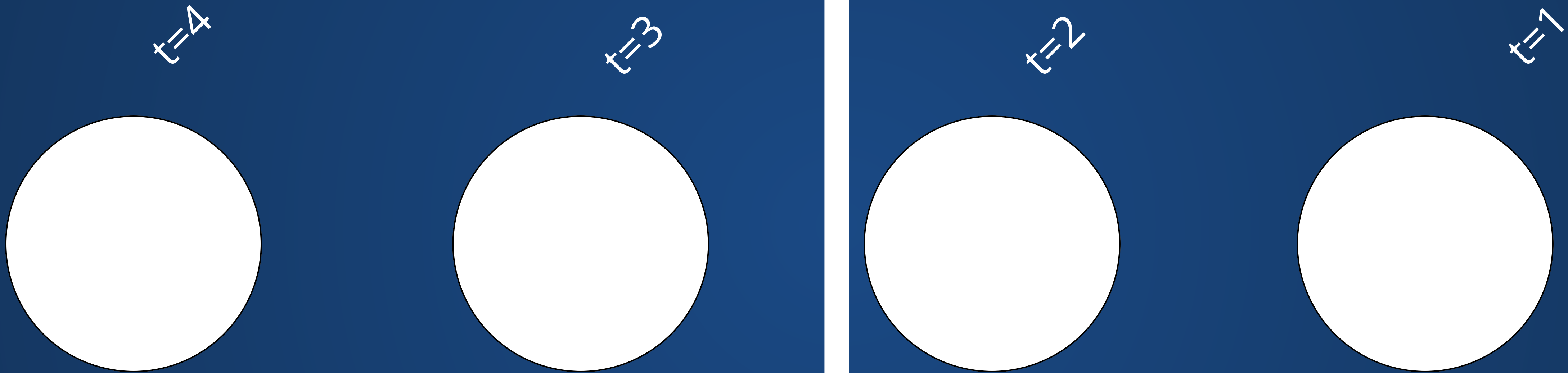
Problem 1: Discrete Timesteps



Problem 1: Discrete Timesteps

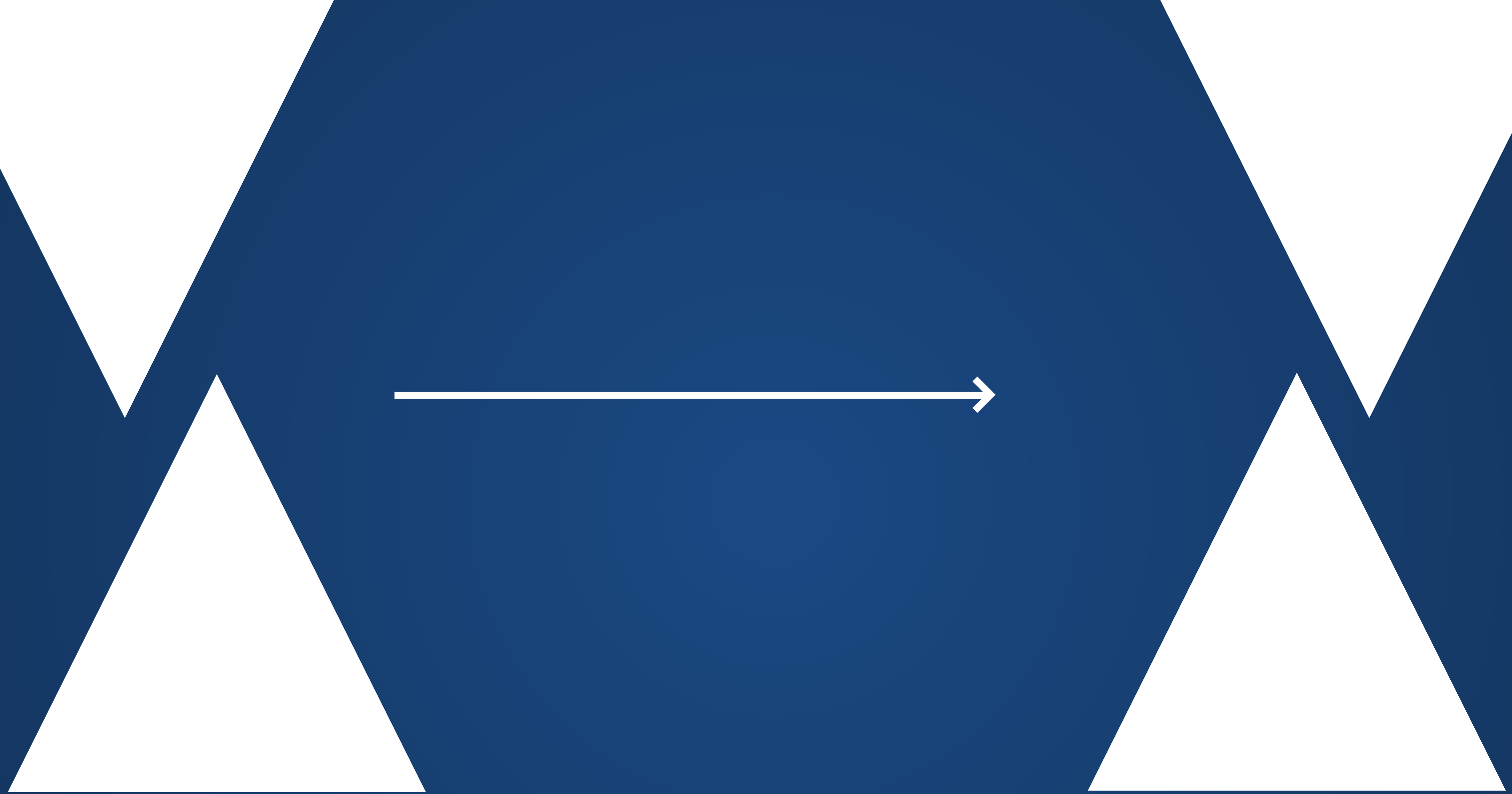


Problem 1: Discrete Timesteps



Wuh woh.

Maybe we can make the timestep smaller?



To check if two triangles collide, need to check all six points.

Problem 2: Combinatorial Explosion

The Stanford Dragon has 871,414 triangles.

How many checks do we need to see if two Stanford dragons collide with each other?

$$6 \times 871414 \times 871414 = 4\,556\,174\,156\,376$$

At 3 GHz, this is about 25 minutes.

Problem 3: Floating Point

```
1  double x;
2
3  void setup(){
4   size(1000, 600);
5   textSize(80);
6   x = 1.0 / 3.0;
7  }
8
9  void draw(){
10   x *= 4.0;
11   x -= 1.0;
12   println(x);
13   if (frameCount % 20 == 0){
14     background(70);
15     text(String.valueOf(x), 50, 300);
16   }
17 }
```

Are these colliding?

Better get it right or your car is going to go straight through a wall without touching it!

What gets used in practice?

- Use partitioning data structures like Bounding Volume Hierarchies or k-d trees (similar to scene hierarchies) to prune unnecessary collision checks.

Transformations

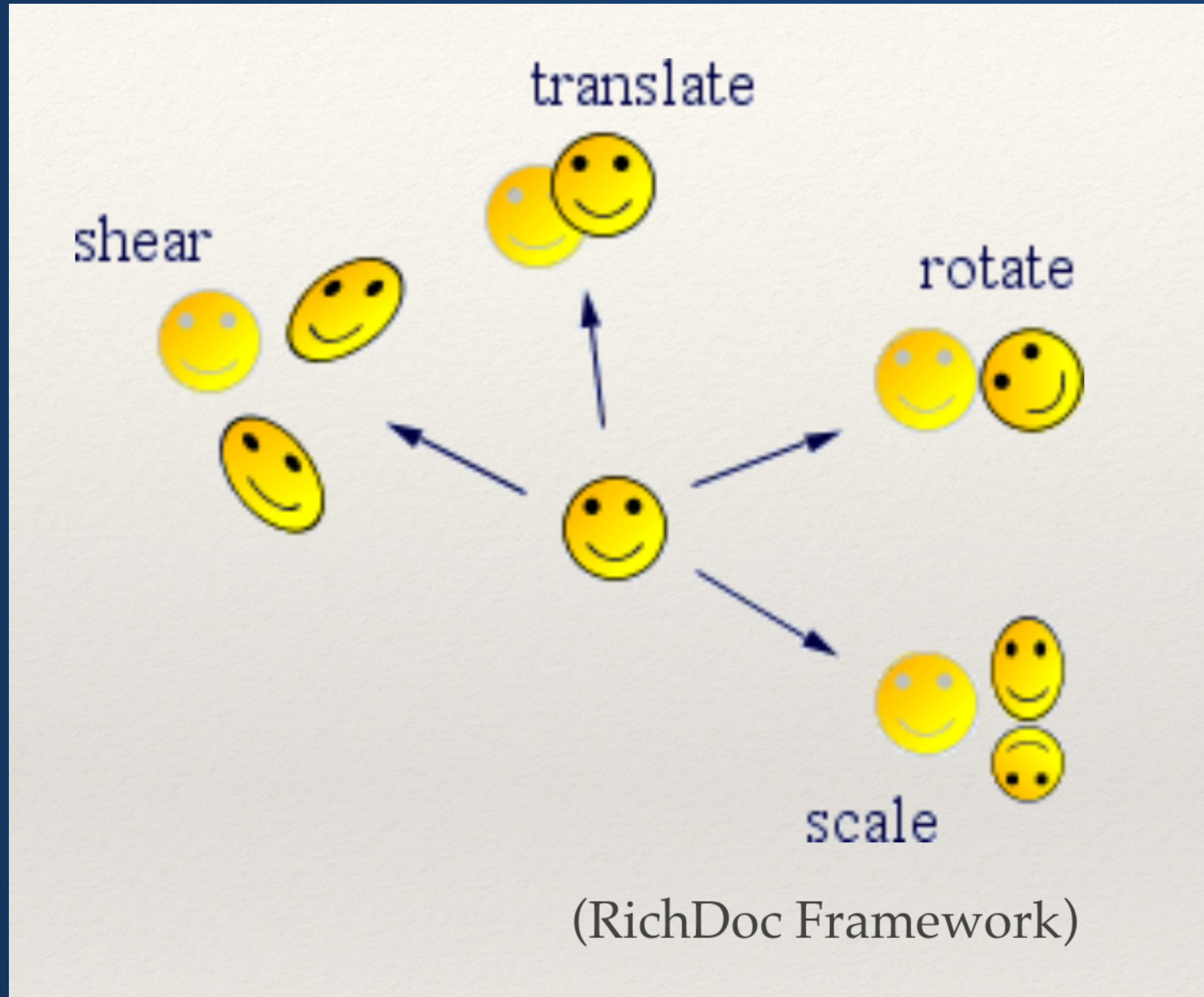
Shapes and Hierarchies

Shapes form complex structures via hierarchies. These make it easier to manipulate structures.

Example: We should be able to just rotate stick person's arm at the shoulder, and ball will move (instead of having to move everything individually).

How do we actually do the manipulations?

Transformations



- One of the foundations of rendering in computer graphics
- Allow us to manipulate objects within a scene

Mathematical Representations

The field of linear algebra has deep connections to transformations in both 2D and 3D

In graphics for CS majors, we have a prerequisite of linear algebra for the course.

Students spend their whole first month struggling with the linear algebra in spite of this.

Representations

A point p is a vector:

$$p = \begin{bmatrix} x \\ y \end{bmatrix}$$

A transformation M is a matrix:

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

We perform matrix multiplication to apply the transformation: $p' = Mp$

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

Matrix Multiplication

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

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

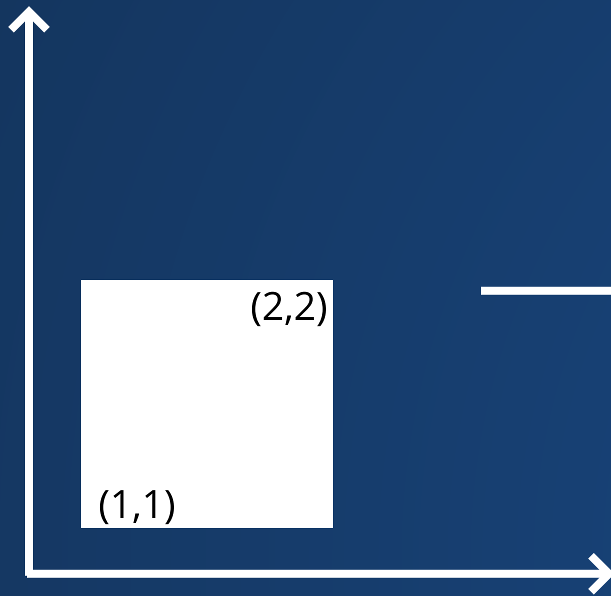
$$x' = ax + by$$

$$y' = cx + dy$$

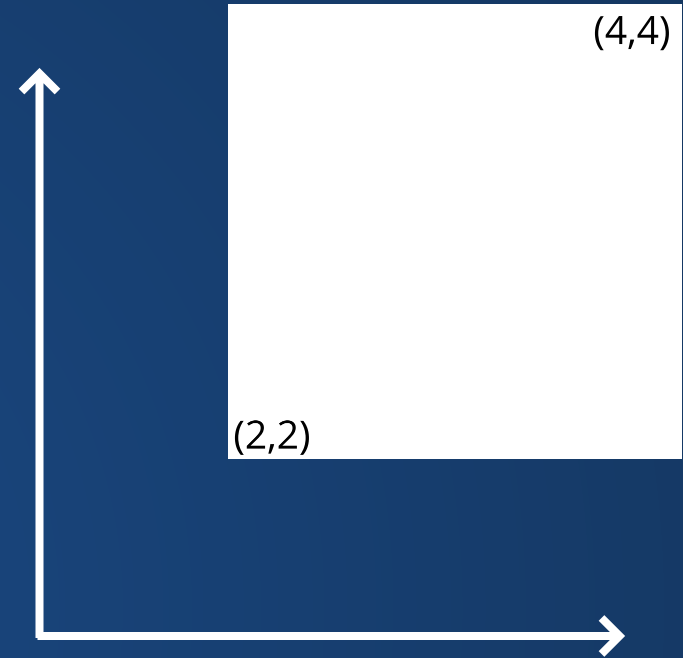
What if we multiply by the so-called identity matrix?

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

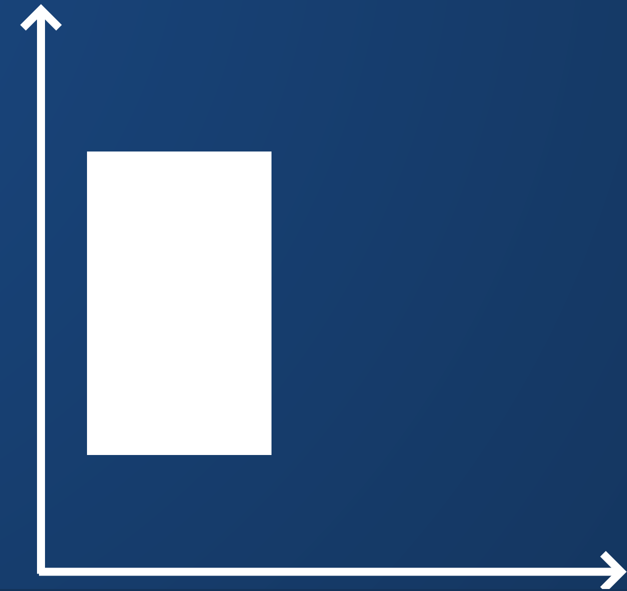
Scaling



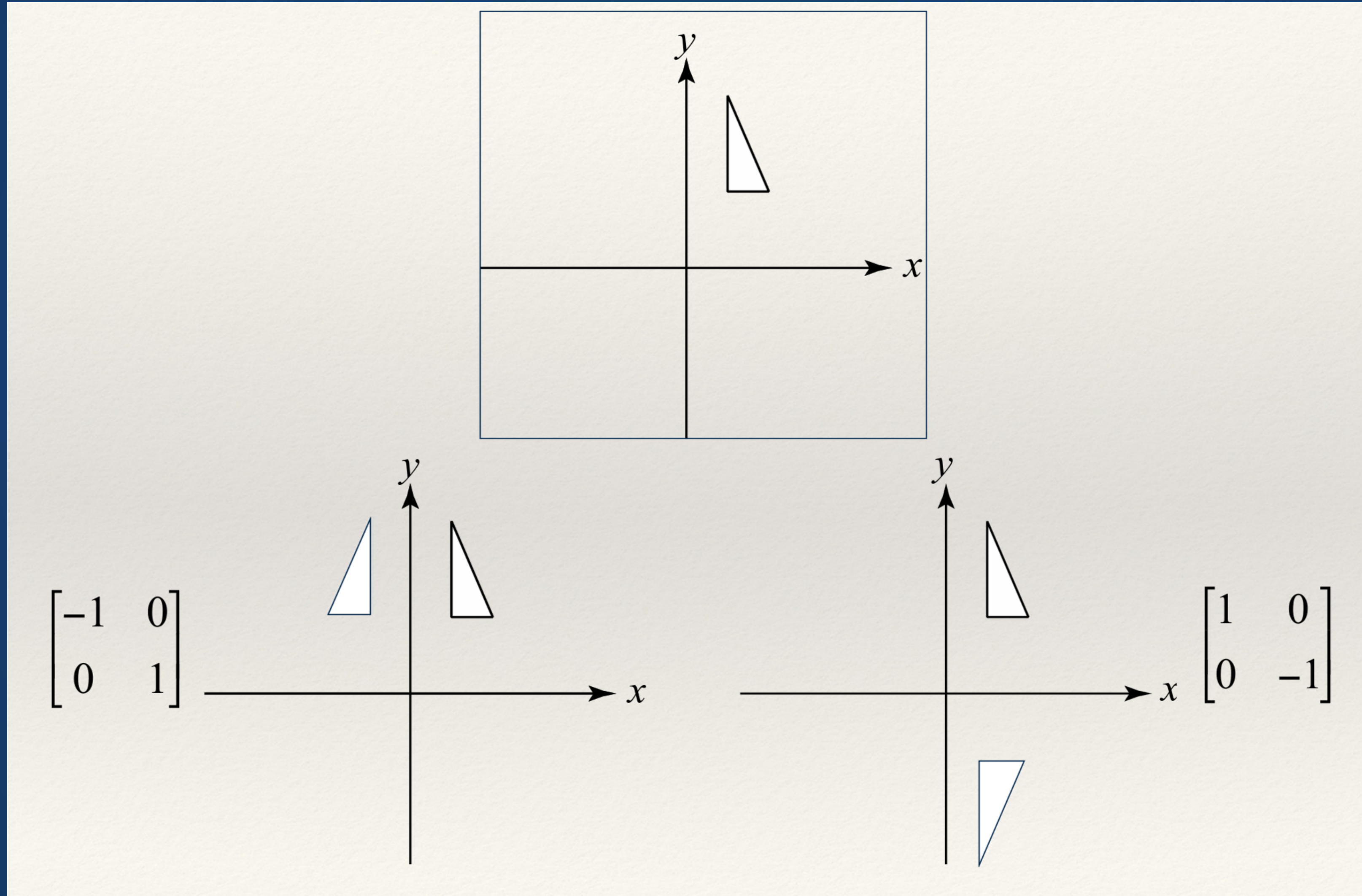
$$M_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$



$$M_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 2 \end{bmatrix}$$



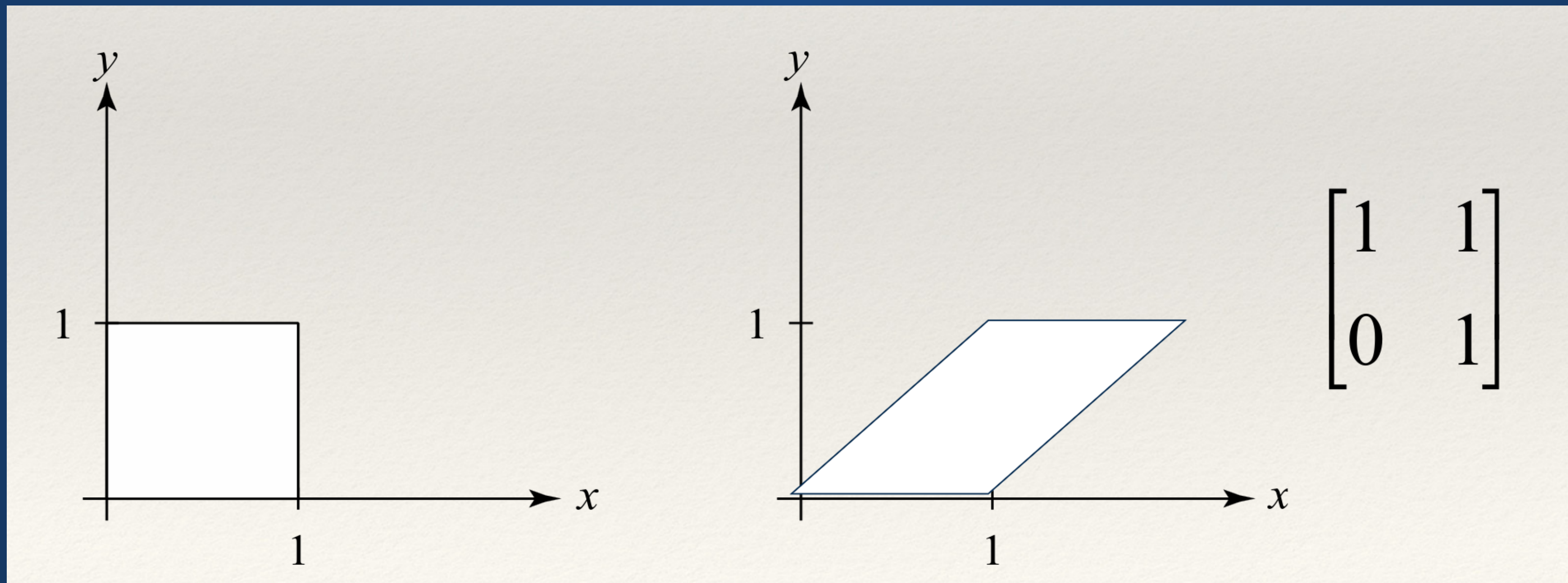
Reflection



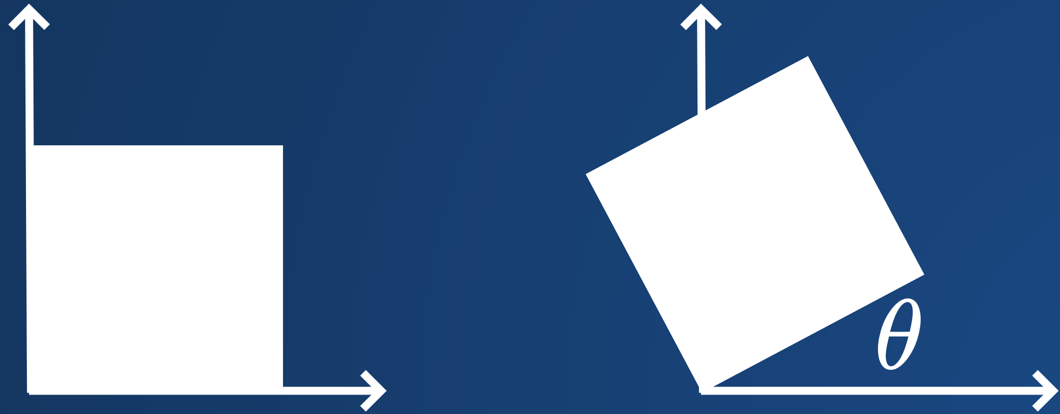
Shear

$$\begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix}$$

$$x' = x + by$$
$$y' = y$$



Rotation



$$\begin{bmatrix}1\\0\end{bmatrix} \to \begin{bmatrix}\cos(\theta)\\ \sin(\theta)\end{bmatrix}$$

$$\begin{bmatrix}0\\1\end{bmatrix} \to \begin{bmatrix}-\sin(\theta)\\ \cos(\theta)\end{bmatrix}$$

$$M_r = R(\theta) = \begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}$$

Linear Transformations

All of these transformations are considered linear transformations:

- Scaling
- Reflection
- Shearing
- Rotation

What very important operation is missing?

Translation

We want to be able to move objects through space.

But we can't do this using our standard linear algebra...

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

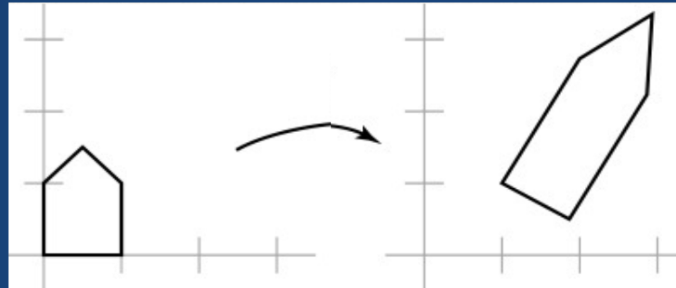
Homogeneous Coordinates

$$p' = Mp$$

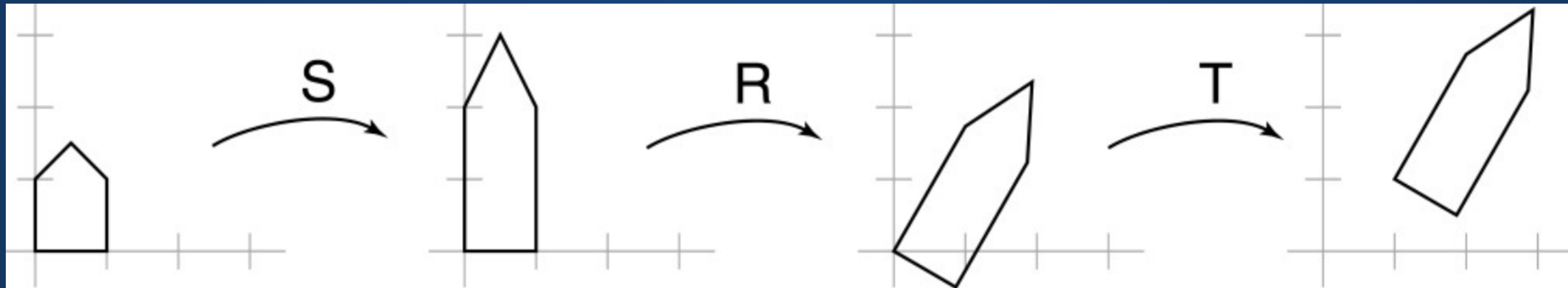
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Combining Transformations

If we have a single, complex matrix, we can simply apply it to our object to get the resulting transform.



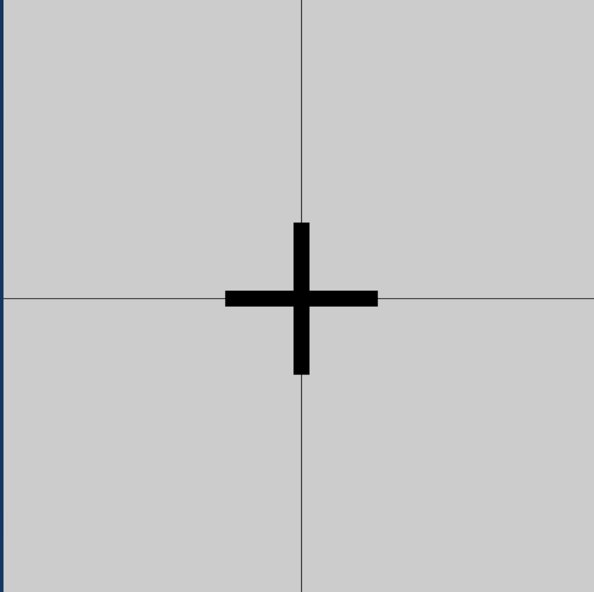
But we can (and often should!) think about it as a sequence of simpler transformations:



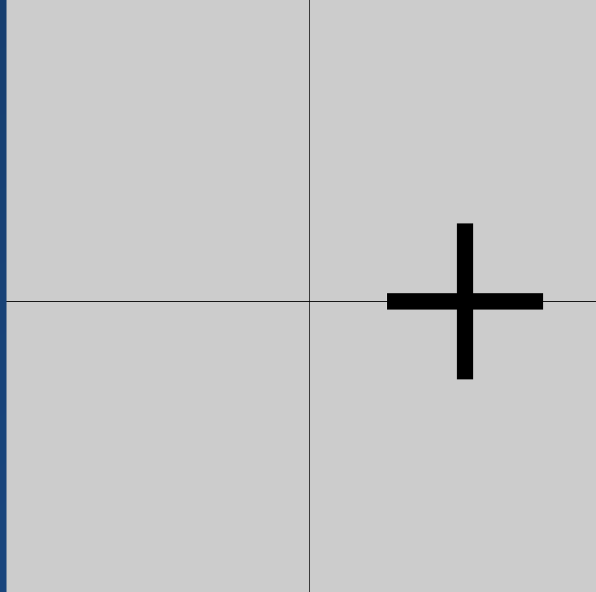
Order Matters!

Mathematical reason: matrix multiplication is generally not commutative.

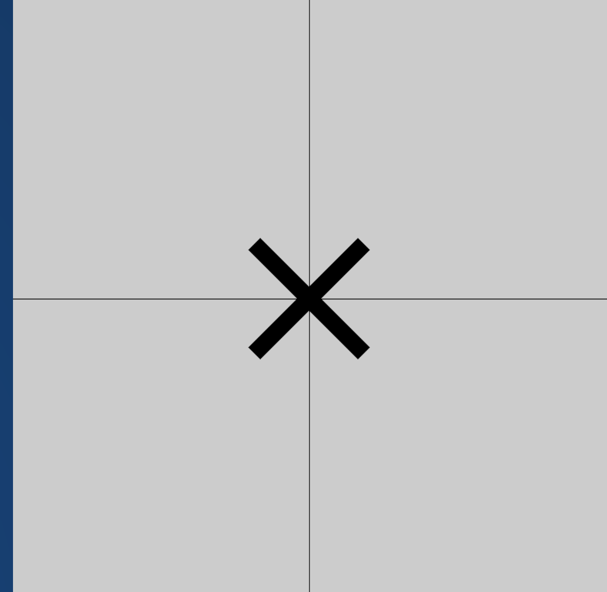
Intuitive: what happens if we translate then reflect vs reflect then translate?



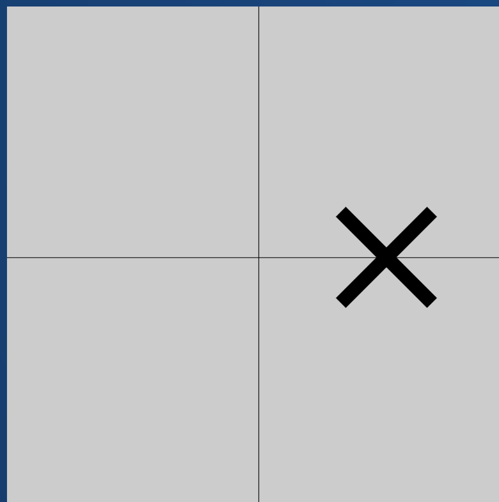
No Transformations Applied



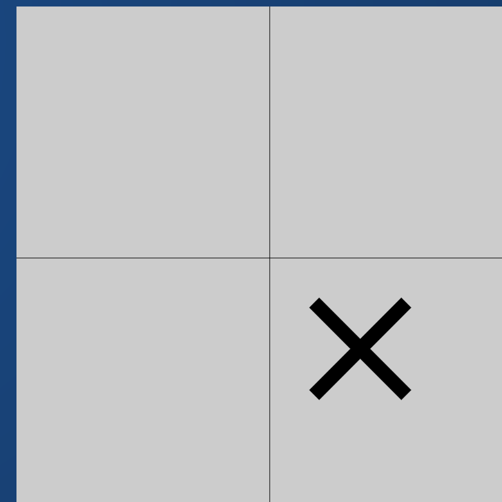
Translate Only



Rotate Only



```
translate()
rotate()
draw()
```



```
rotate()
translate()
draw()
```

Transformations in Processing

- `translate(x, y)`: applies a translation which moves points by (x,y)
- `rotate(θ)`: rotates by θ radians
- `scale(p)`: Scales by p, where 1.0 is no change.

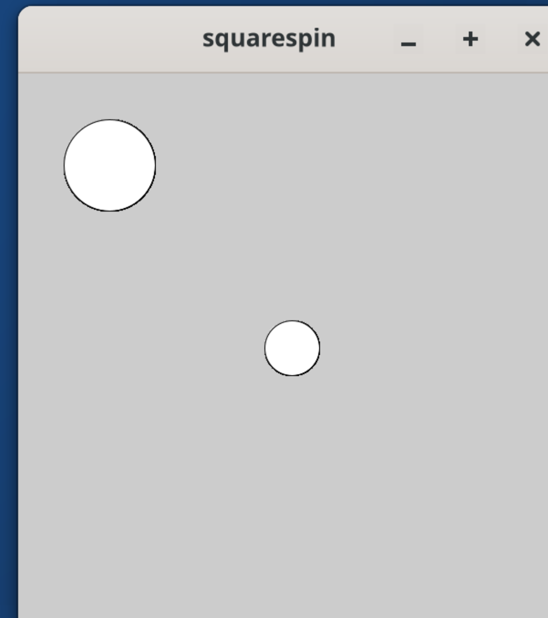
Transformations in Processing

Transformations are a little like modes, where the transformation function affects all future draw calls.

Two major differences:

- Transforms **stack** instead of replacing the previous transform.
- The transform is reset at the top of the draw() loop.

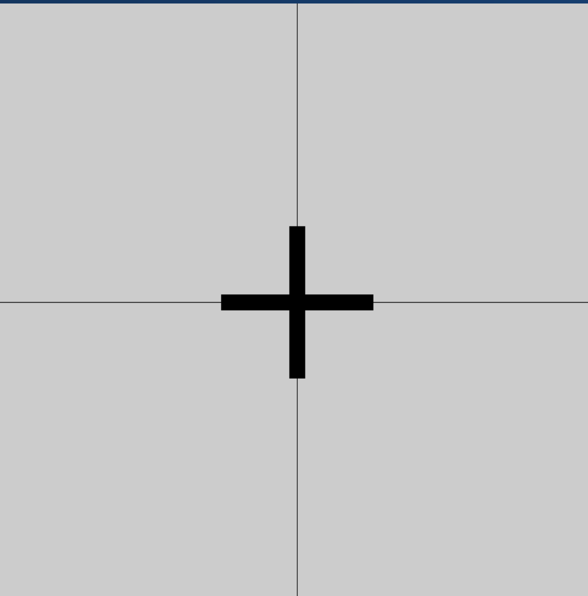
```
1  void draw(){
2    translate(100, 100);
3    ellipse(0, 0, 100, 100);
4    translate(200, 200);
5    ellipse(0, 0, 60, 60);
6    translate(300, 300);
7    ellipse(0, 0, 35, 35);
8  }
```



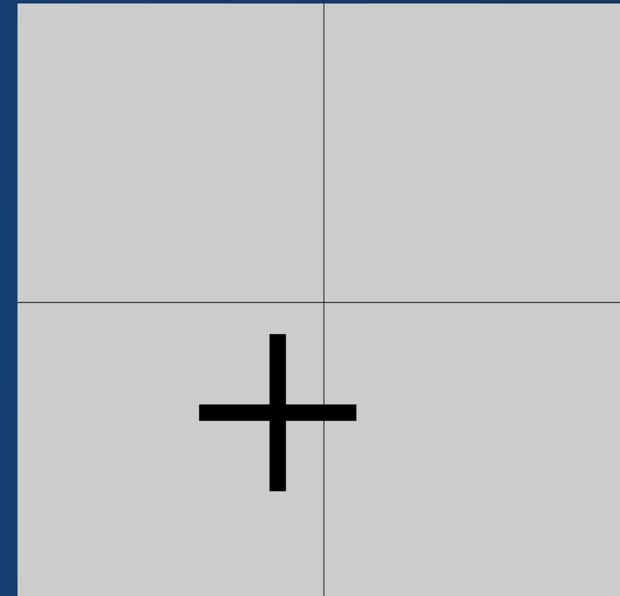
Undoing Transformations

Ctrl+Z 101

Cancelling out a Transform



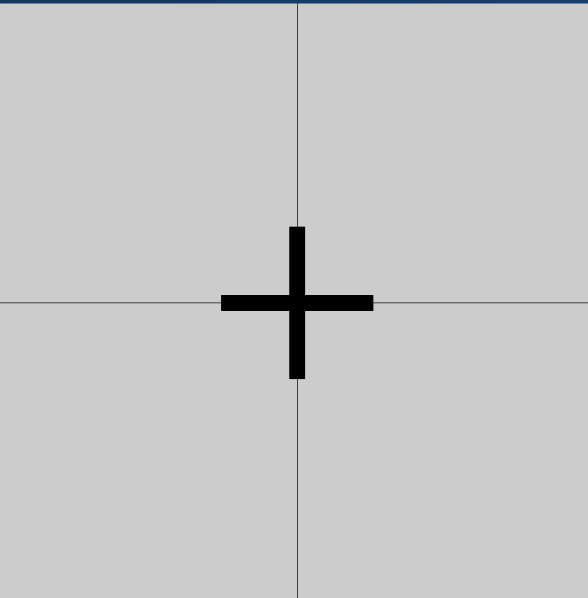
```
1   rotate(radians(45));
2   translate(200, 0);
3
4   // Undo our transformation
5   rotate(radians(-45));
6   translate(-200, 0);
7
8   shape(ref);
```



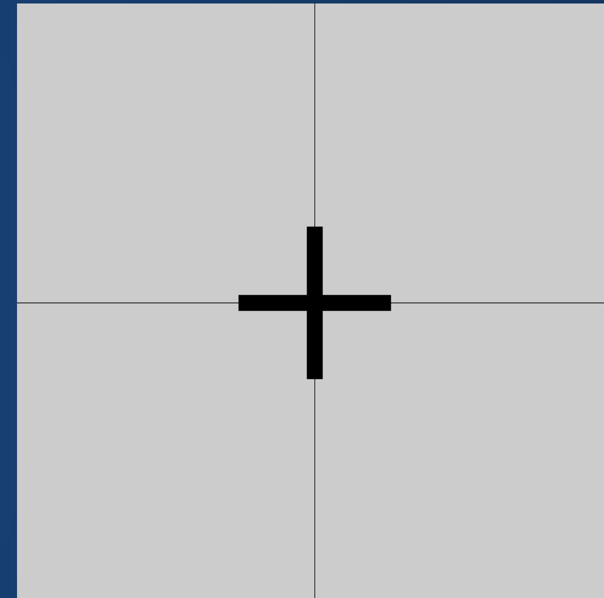
Wuh woh.

Shoes-Socks Theorem

Cancelling out a Transform, but properly



```
1    rotate(radians(45));   // shoes
2    translate(200, 0);     // socks
3
4    translate(-200, 0);    // isocks
5    rotate(radians(-45));  // ishoes
6
7    shape(ref);
```



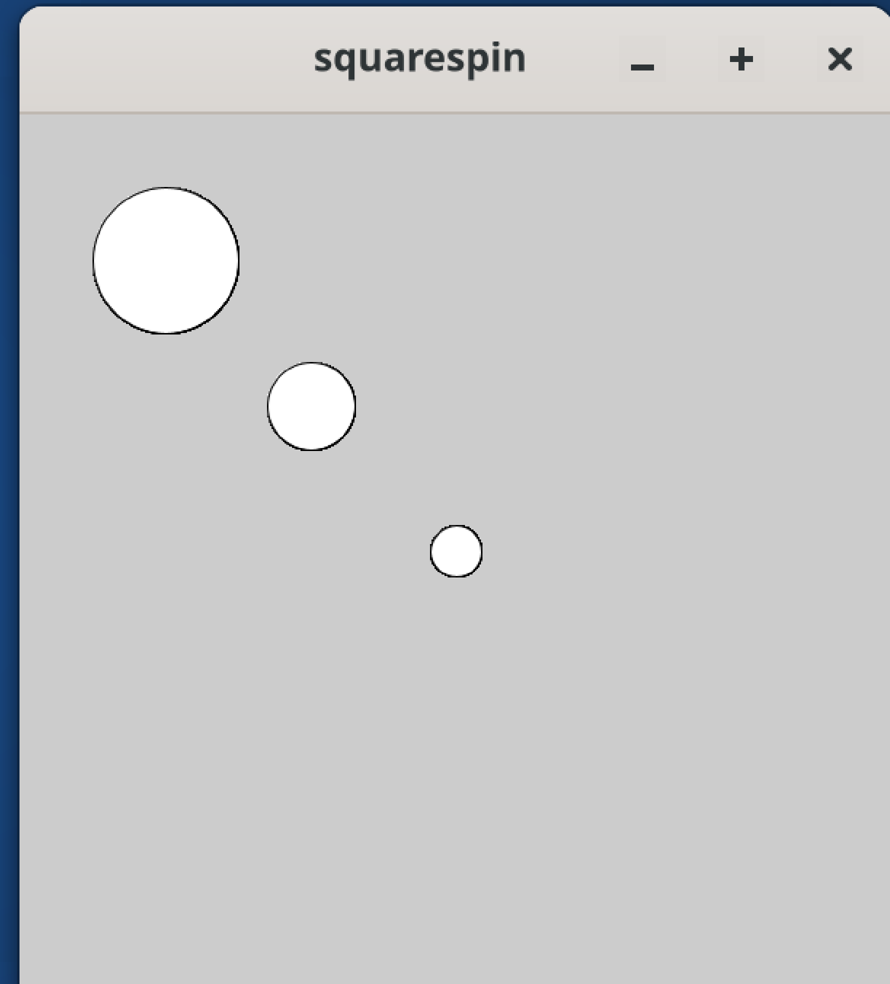
Bookmarking Transformations

- Processing gives us tools to save and restore transformations.
- `pushMatrix()` takes the current global transformation and records it in a matrix stack. You can think of this as a global variable that is hidden from you and can only be accessed with the functions here.
- `popMatrix()` pops a transformation matrix off of the stack and makes it the current transformation matrix.
- These functions let us manipulate objects at different levels in the scene hierarchy.

```
void draw(){
  pushMatrix();
  translate(100, 100);
  ellipse(0, 0, 100, 100);
  popMatrix();

  pushMatrix();
  translate(200, 200);
  ellipse(0, 0, 60, 60);
  popMatrix();

  pushMatrix();
  translate(300, 300);
  ellipse(0, 0, 35, 35);
  popMatrix();
}
```



Why Transformations?

We can already emulate scale by passing parameters to `rect()`. Why not just add a rotation parameter?

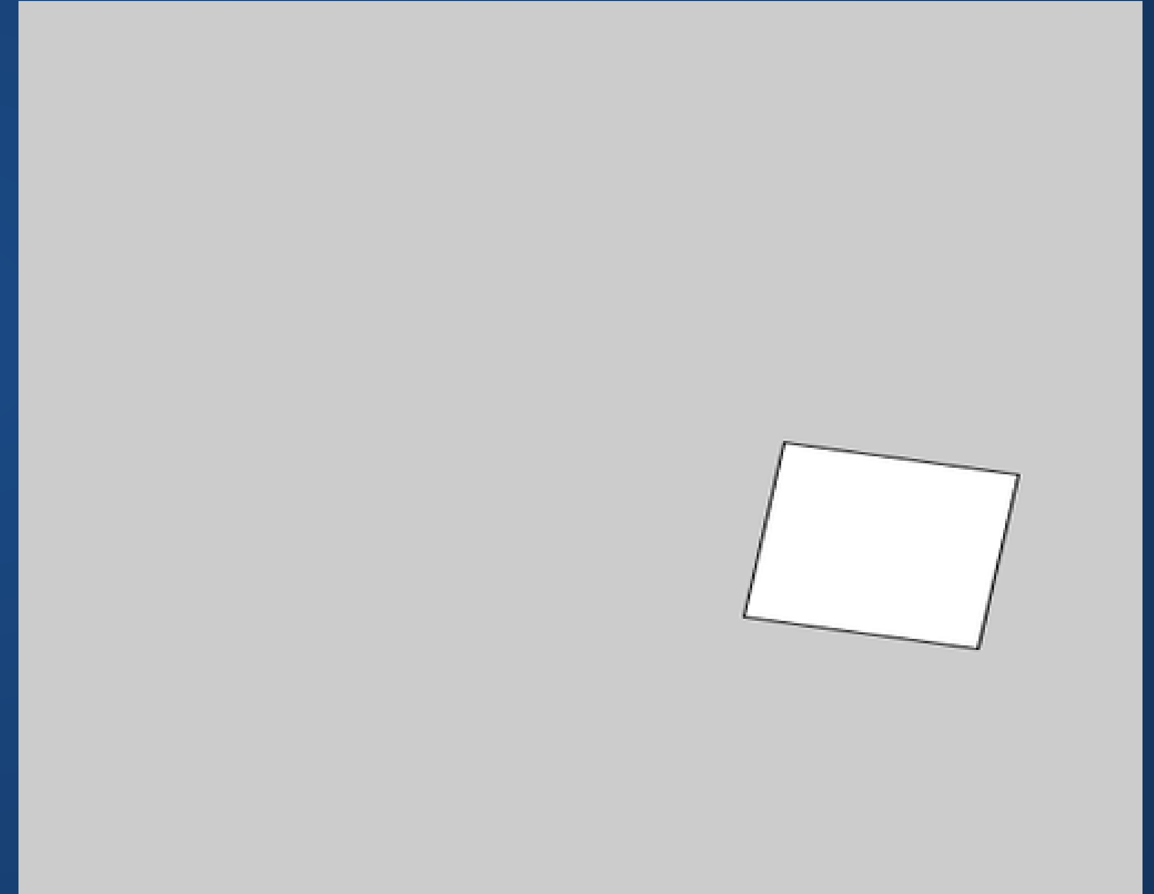
```
rect(0, 0, 100, 20, PI/8)
```

Composing transformations lets us describe very complex motion with simple code.

By author Zorgit on Wikimedia, CC-BY-SA 3.0: https://commons.wikimedia.org/wiki/File:Cycloid_f.gif

Composing transformations lets us describe very complex motion with simple code.

```
1  void draw(){
2    translate(300, 300);
3
4    rotate(frameCount * 0.03);
5    translate(50, 0);
6
7    translate(30, 0);
8    rotate(frameCount * 0.05);
9
10   translate(10, 0);
11   rotate(frameCount * 0.1);
12
13   rect(0, 0, 100, 100);
14 }
```



In 3D, the "rotate" parameter gets complex

We need at least four* parameters to describe a rotation in three dimensions.

We need more spatial arguments as well!

```
box(x, y, z, width, height, depth, rot1, rot2, rot3, rot4);
```

In 3D, transformations are heavily used to avoid parameter explosions:

- Boxes are always created at the origin
- Spheres are always created at the origin
- There are no ellipsoids (you have to scale a sphere)

* terms and conditions apply

Hands-On: Using Transformations

1. Translate the screen so that the origin (0,0) is at the center of the screen. Draw a solid black circle at (0,0) to demonstrate this.
2. Draw a shape of your choice. Before drawing it, translate, then rotate, then scale the shape (pick your own parameters for the transforms).
3. Draw the same shape as you did in step 2, with the same parameters for the transforms. However, this time, apply the scale first, then rotate, then translate.
4. Draw an object of your own choice and apply your own set of transformations to it.
5. OPTIONAL: Animate a rectangle orbiting your black circle using only transformations. See if you can keep the rectangle axis-aligned while it orbits!

The transformation in step 1 should apply to all of the other steps.
For Steps 2-5, you may want to pushMatrix() and popMatrix() so that your transformations don't get too confusing.

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.