Day 12

Vector Math
Physics
Particle Systems

Day 12 Agenda

Review: Classes and Objects

Introduction to Vectors

Forces and Physics

ArrayList and Particle Systems

Vectors

What is a Vector?

At its core, a vector holds an **X** and a **Y** value.
In 3D a vector holds an **X**, **Y**, and **Z**.

It can represent a **location**, a **direction**, a **speed** (velocity)
as well as **acceleration** and other **forces**.

Why are vectors useful?

By using vectors in your projects, you'll be able to use **procedural animation** techniques like simulating physics.

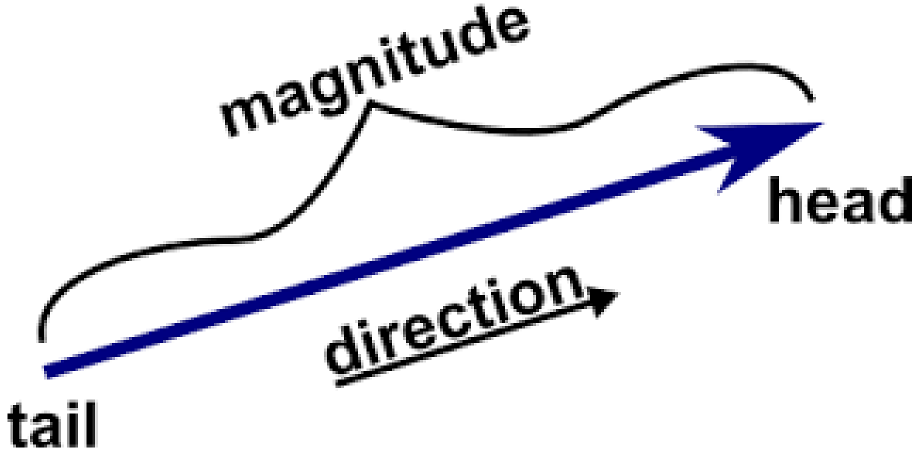
Games, VR, and AR are built on vector math and linear algebra. When you use tools like **Unity**, vectors are *everywhere*.

What is a vector?

A vector is a data structure that has **magnitude** (aka **length**) and **direction**

*When we use a vector to represent **position**, we treat the vector as if it begins at (0,0).*

When we use a vector to represent velocity (speed), we treat the vector as if it begins at the (x,y) of an object's location.



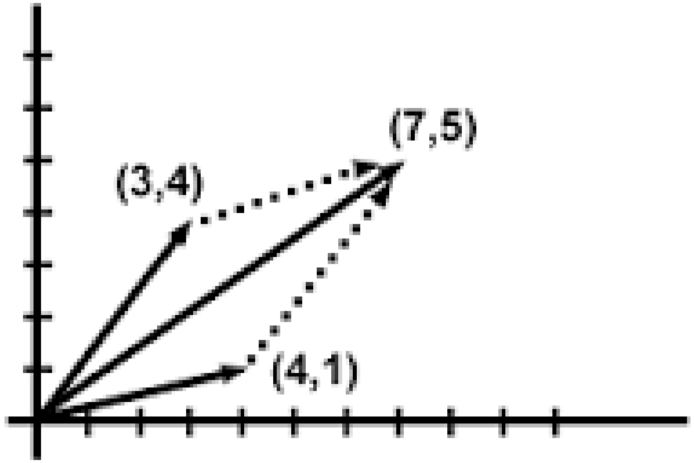
Vector math

We can add **vectors** the same way we add **numbers**
(numbers are called **scalar** values in linear algebra)

Vector **a** = (3, 4)
Vector **b** = (4, 1)

a + **b** = (3, 4) + (4, 1)

a + **b** = (7, 5)



Vectors in Processing

```
PVector v1;
PVector v2;

void setup() {
  // new PVector(x, y)
  v1 = new PVector(40, 20);
  v2 = new PVector(25, 50);
}

void draw() {
  // PVector has x and y properties
  ellipse(v1.x, v1.y, 12, 12);
  ellipse(v2.x, v2.y, 12, 12);
  // and methods like
  // add(), mult(), normalize()
  v2.add(v1);
  ellipse(v2.x, v2.y, 24, 24);
}
```

Why does this matter? I HATE math!

Without vectors it gets complicated to talk about distances and directions in code.

Vectors let us represent an object’s

- Position,
- Velocity,
- Direction,
- or Acceleration

With one value, and then easily add and compare in one line of code.

Vectors are the core concept behind modern procedural animation *(as found in games, simulations, creative coding, mobile interfaces and more)*

Position as a PVector

```
PVector pos;

void setup() {
  pos = new PVector(40, 20);
}

void draw() {
  ellipse(pos.x, pos.y, 12, 12);
}
```

Velocity as a PVector

```
PVector pos;
PVector vel;

void setup() {
  pos = new PVector(40, 20);
  vel = new PVector(1, 0);
}

void draw() {
  // each frame x increases by 1
  pos.add(vel);
  ellipse(pos.x, pos.y, 12, 12);
}
```

Acceleration as a PVector

```
PVector pos;
PVector vel;
PVector accel;

void setup() {
  pos = new PVector(40, 20);
  vel = new PVector(0, 0);
  accel = new PVector(0.01, 0.01);
}

void draw() {
  vel.add(accel);
  pos.add(vel);
  ellipse(pos.x, pos.y, 12, 12);
}
```

Limiting PVectors with Limit()

(Keep your objects from flying off the screen)

```
PVector pos;
PVector vel;
PVector accel;
float topSpeed = 5.0;

void setup() {
  pos = new PVector(40, 20);
  vel = new PVector(0, 0);
  accel = new PVector(0.01, 0.01);
}

void draw() {
  vel.add(accel);
  vel.limit(topSpeed);
  pos.add(vel);
  ellipse(pos.x, pos.y, 12, 12);
}
```

Gravity as PVector

(Gravity is a change in velocity in the y direction)

```
PVector pos;
PVector vel;
PVector accel;
PVector gravity;
float topSpeed = 5.0;

void setup() {
  pos = new PVector(40, 20);
  vel = new PVector(0, 0);
  accel = new PVector(0.01, 0.01);
  gravity = new PVector(0, 0.1);
}

void draw() {
  accel.add(gravity);
  vel.add(accel);
  vel.limit(topSpeed);
  pos.add(vel);
  ellipse(pos.x, pos.y, 12, 12);
}
```

Forces and Particle Systems

Other forces (e.g. wind) operate like gravity – just in a different direction.

If you can represent **forces** as **PVectors** you have all the tools to create attractors, repellers, all the elements of an interesting particle system.

With **Classes**, **PVectors**, and **Arrays** we have almost all the tools we need to make a particle system. Our only problem is that **Arrays** are too inflexible to make an interesting simulation.

Enter the **ArrayList**.

dissolving particle shapes

VEX Particle Count 10 - 30 Million

So why ArrayList for Particle Systems?

Notice how those particles in the previous example are fading out and disappearing while new ones emerge?

Because the number of items at an Array is set at **compile time** instead of **runtime**, a particle system built with an array will always have 500, or 1000, or 1,000,000 particles – no more and no less.

ArrayList vs Arrays

ArrayList stores a **variable** number of items

an Array stores a **static** number of items

```
ArrayList<int> numbers = new ArrayList<int>();
int[] otherNumbers = {5, 10};

void setup() {

  // we can't do this with an array
  numbers.add( 5 );
  numbers.add( 10 );

  println(numbers.get(0));
  println(numbers.get(1));
  numbers.remove(1);
}
```

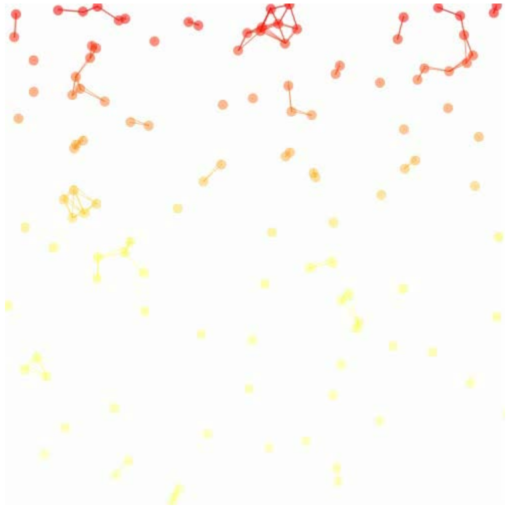
ArrayList and Particle Systems

```
ArrayList<Particle> particles = new
ArrayList<Particle>();

void setup() {
  particles.add( new Particle() );
}

void draw() {
  for (int i = 0; i < particles.size(); i++)
  {
    Particle p = particles.get(i);
    if (p.isDead == true) {
      particles.remove(i);
    } else {
      p.draw();
    }
  }
}
```

In-class Exercise
Let’s build our first particle system

Using these techniques, break into pairs and build a particle system.

First, create a **Particle** Class (like Ball) containing
PVector **position**
PVector **velocity**
PVector **acceleration** - hint: think gravity
float **fade** - hint: alpha value in fill



In your main sketch file, you will need an **ArrayList** of **Particles**

What next?

Read Daniel Shiffman's amazing book *The Nature of Code* (or visit natureofcode.com) to learn about simulating the natural world through physics, and mathematics.

Take Justin Bakse's **legendary** class *Computational Form* in Spring of 2019 (and check out compform.net) to get more tools around **random**, **noise**, and more.