4. Particle Systems

박종화
suakii@gmail.com

Particle Systems

Particle Systems

- A particle system is a technique in game physics, motion graphics, and computer graphics that uses a large number of very small sprites, 3D models, or other graphic objects to simulate certain kinds of "fuzzy" phenomena, which are otherwise very hard to reproduce with conventional rendering techniques - usually highly chaotic systems, natural phenomena, or processes caused by chemical reactions. - wikipedia

Particle Systems

- ‘간단한 모양 또는 점으로 표시되는 독립적인 물체의 집합‘
- 단일 입자 : Particle
- 입자의 집합 : Particle Systems

Why We Need Particle Systems

```
ParticleSystem ps;

void setup() {
  size(640,360);
  ps = new ParticleSystem();
}

void draw() {
  background(255);
  ps.run();
}
```

A Single Particle

```
class Particle {
//A "Particle" object is just another name for our "Mover." It has location, velocity, and acceleration.
  PVector location;
  PVector velocity;
  PVector acceleration;

  Particle(PVector l) {
    location = l.get();
    acceleration = new PVector();
    velocity = new PVector();
  }

  void update() {
    velocity.add(acceleration);
    location.add(velocity);
  }

  void display() {
    stroke(0);
    fill(175);
    ellipse(location.x,location.y,8,8);
  }
}
```

A Single Particle

```
class Particle {
  PVector location;
  PVector velocity;
  PVector acceleration;
  //A new variable to keep track of how long the particle has been "alive"
  float lifespan;

  Particle(PVector l) {
    location = l.get();
    acceleration = new PVector();
    velocity = new PVector();
    //We start at 255 and count down for convenience
    lifespan = 255;
  }

  void update() {
    velocity.add(acceleration);
    location.add(velocity);
    //Lifespan decreases
    lifespan -= 2.0;
  }

  void display() {
  //Since our life ranges from 255 to 0 we can use it for alpha
    stroke(0,lifespan);
    fill(175,lifespan);
    ellipse(location.x,location.y,8,8);
  }
}
```

isDead?

```
boolean isDead() {
//Is the particle still alive?
   if (lifespan < 0.0) {
     return true;
   } else {
     return false;
   }
 }
```

The ArrayList

- Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

Array Vs ArrayList

- Particle[] parray = new Particle[total];
- ArrayList<Particle> plist = new ArrayList<Particle>();
- Generic^^

ArrayList

- ArrayList<Particle> plist = new ArrayList<Particle>();
- plist.add(new Particle());

ArrayList

```
 for (int i = 0; i < plist.size(); i++) {
        Particle p = plist.get(i);
        p.run();
  }
//원소의 삭제가 있을때는 사용불가
for (Particle p: plist) {
  p.run();
}
```

ArrayList

- Adding one new particle with each cycle through darw().

ArrayList

```
ArrayList<Particle> particles;

void setup() {
  size(640,360);
  particles = new ArrayList<Particle>();
}

void draw() {
  background(255);
//A new Particle object is added to the ArrayList every cycle through draw().
  particles.add(new Particle(new PVector(width/2,50)));

  for (int i = 0; i < particles.size(); i++) {
    Particle p = particles.get(i);
    p.run();//What's the problem?
  }
}
```

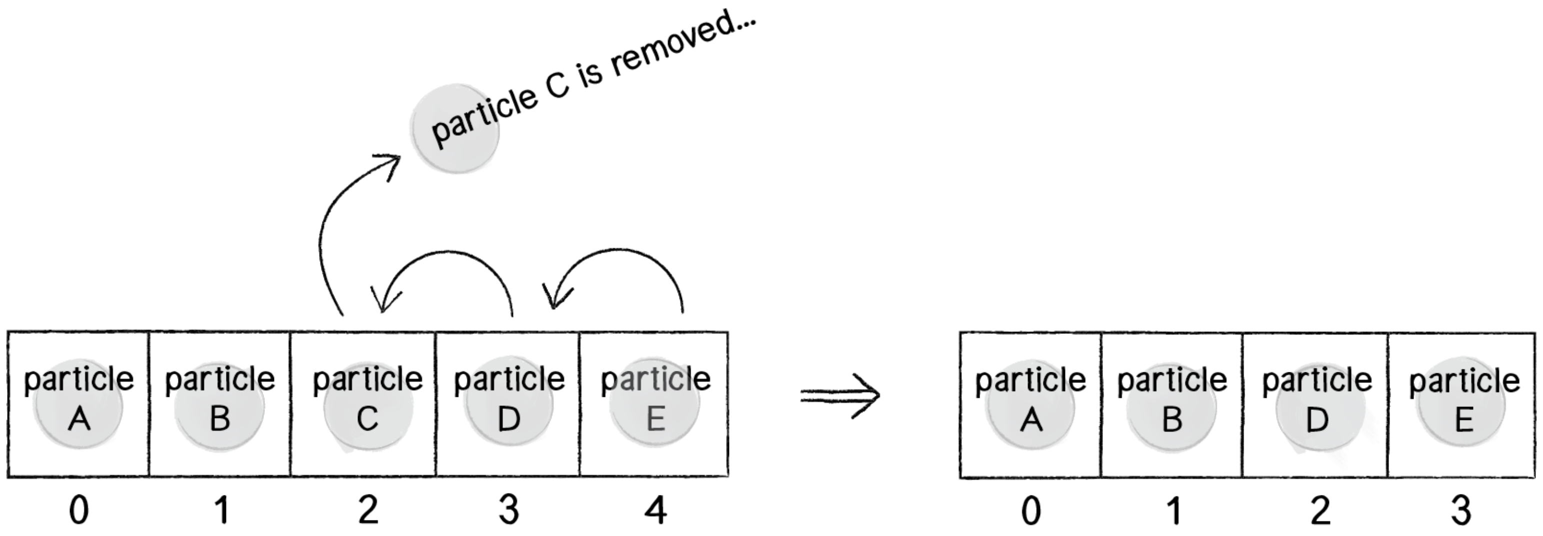
ArrayList

```
for (int i = 0; i < particles.size(); i++) {
    Particle p = particles.get(i);
    p.run();
//If the particle is “dead,” we can go ahead
//and delete it from the list.
    if (p.isDead()) {
      particles.remove(i);
    }
  }
```

ArrayList

```
for (int i = 0; i < particles.size(); i++) {
    Particle p = particles.get(i);
    p.run();
        particles.add(new Particle(new
        PVector(width/2,50)));
 }//What't the problem?
```

ArrayList Remove Problem



ArrayList Remove Problem

- Let’s pretend we are i looping through the ArrayList.

- when i = 0 → Check particle A → Do not delete
- when i = 1 → Check particle B → Do not delete
- when i = 2 → Check particle C → Delete!
- Slide particles D and E back from slots 3 and 4 to 2 and 3
- when i = 3 → Check particle E → Do not delete

Solution – Looping through the list backwards.

```
for (int i = particles.size()-1; i >= 0; i--) {
   Particle p = (Particle) particles.get(i);
   p.run();
   if (p.isDead()) {
     particles.remove(i);
   }
 }
```

Iterator

- Java provides a special class—Iterator—that takes care of all of the details of iteration for you.

Iterator

- Iterator<Particle> it = particles.iterator();

Iterator

```
while (it.hasNext()) {
    Particle p = it.next();
    p.run();
    if (p.isDead()) {
        it.remove();
    }
}
```

The Particle System Class

- // Just one wee ParticleSystem!

```
ParticleSystem ps;


void setup() {
  size(640,360);
  ps = new ParticleSystem();
}

void draw() {
  background(255);
  ps.run();
}
```

ArrayList int the main tab

```
ArrayList<Particle> particles;

void setup() {
  size(640,360);
  particles = new ArrayList<Particle>();
}

void draw() {
  background(255);

  particles.add(new Particle());



  Iterator<Particle> it =
      particles.iterator();
  while (it.hasNext()) {
    Particle p = it.next();
    p.run();
    if (p.isDead()) {
      it.remove();
    }
  }
}
```

ArrayList int the ParticleSystem class

```
class ParticleSystem {
  ArrayList<Particle> particles;


  ParticleSystem() {
    particles = new ArrayList<Particle>();
  }



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

  void run() {
    Iterator<Particle> it =
        particles.iterator();
    while (it.hasNext()) {
      Particle p = it.next();
      p.run();
      if (p.isDead()) {
        it.remove();
      }
    }
  }
}
```

A System of Systems

- And we’ve defined a particle system as a collection of independent objects. But isn’t a particle system itself an object? If that’s the case (which it is), there’s no reason why we couldn’t also have a collection of many particle systems, i.e. a system of systems.

A System of Systems

```
// This time, the type of thing we are putting
// in the ArrayList is a ParticleSystem itself!

ArrayList<ParticleSystem> systems;
void setup() {
  size(600,200);
  systems = new ArrayList<ParticleSystem>();
}
```

A System of Systems

```
void mousePressed() {
  systems.add(new ParticleSystem(new
PVector(mouseX,mouseY)));
}
```

A System of Systems

```
void draw() {
background(255);
for (ParticleSystem ps : systems) {
    ps.run();
    ps.addParticle();
}


}
```

OOP

- Inheritance: 상속
  - 물려 받자.
- Polymorphism: 다형성
  - 한 객체를 다양한 형태로 다루자. 상위 클래스의 입장
- Override
  - 부모 클래스의 메소드 재정의

Particles with Inheritance

```
class Particle {
  PVector location;
  PVector velocity;
  PVector acceleration;

  Particle(PVector l) {
    acceleration = new PVector(0,0.05);
    velocity = new PVector(random(-1,1),random(-2,0));
    location = l.get();
  }

  void run() {
    update();
    display();
  }

  void update() {
    velocity.add(acceleration);
    location.add(velocity);
  }

  void display() {
    fill(0);
    ellipse(location.x,location.y,8,8);
  }
}
```

Particles with Inheritance

```
class Confetti extends Particle {

We could add variables for only Confetti here.

  Confetti(PVector l) {
    super(l);
  }

//There is no code here because we inherit update() from parent.


//Override the display method.
  void display() {
    rectMode(CENTER);
    fill(175);
    stroke(0);
    rect(location.x,location.y,8,8);
  }
}
```

Particles with Inheritance

- Let's say we want to have the Confetti particle **rotate** as it flies through the air.

Particles with Inheritance

```
void display() {
   float theta = map(location.x,0,width,0,TWO_PI*2);

   rectMode(CENTER);
   fill(0,lifespan);
   stroke(0,lifespan);
        pushMatrix();
                 translate(location.x,location.y);
                 rotate(theta);
                 rect(0,0,8,8);
        popMatrix();
 }
```

Polymorphism Basics

```
ArrayList<Dog> dogs = new ArrayList<Dog>();
ArrayList<Cat> cats = new ArrayList<Cat>();
ArrayList<Turtle> turtles = new ArrayList<Turtle>();
ArrayList<Kiwi> kiwis = new ArrayList<Kiwi>();

for (int i = 0; i < 10; i++) {
  dogs.add(new Dog());
}
for (int i = 0; i < 15; i++) {
  cats.add(new Cat());
}
for (int i = 0; i < 6; i++) {
  turtles.add(new Turtle());
}
for (int i = 0; i < 98; i++) {
  kiwis.add(new Kiwi());
}
```

Polymorphism Basics

```
//Just one ArrayList for all the animals!
ArrayList<Animal> kingdom = new ArrayList<Animal>();

for (int i = 0; i < 1000; i++) {
  if (i < 100) kingdom.add(new Dog());
  else if (i < 400) kingdom.add(new Cat());
  else if (i < 900) kingdom.add(new Turtle());
  else kingdom.add(new Kiwi());
}

for (Animal a: kingdom) {
  a.eat();
}
```

Polymorphism

- Polymorphism (from the Greek polymorphos, meaning many forms) refers to the treatment of a single instance of an object in multiple forms.

- Dog rover = new Dog();
- Animal spot = new Dog();

Particle Systems with Polymorphism

```
class ParticleSystem {
ArrayList<Particle> particles;
  PVector origin;

  ParticleSystem(PVector location) {
    origin = location.get();
    particles = new ArrayList<Particle>();
  }

  void addParticle() {
    float r = random(1);
    if (r < 0.5) {
      particles.add(new Particle(origin));
    } else {
      particles.add(new Confetti(origin));
    }
  }

  void run() {
    Iterator<Particle> it = particles.iterator();
    while (it.hasNext()) {
       Particle p = it.next();
       p.run();
      if (p.isDead()) {
        it.remove();
      }
    }
  }
}
```

Particle Systems with Forces

```
void applyForce(PVector force) {
    PVector f = force.get();
    f.div(mass);
    acceleration.add(f);
 }
```

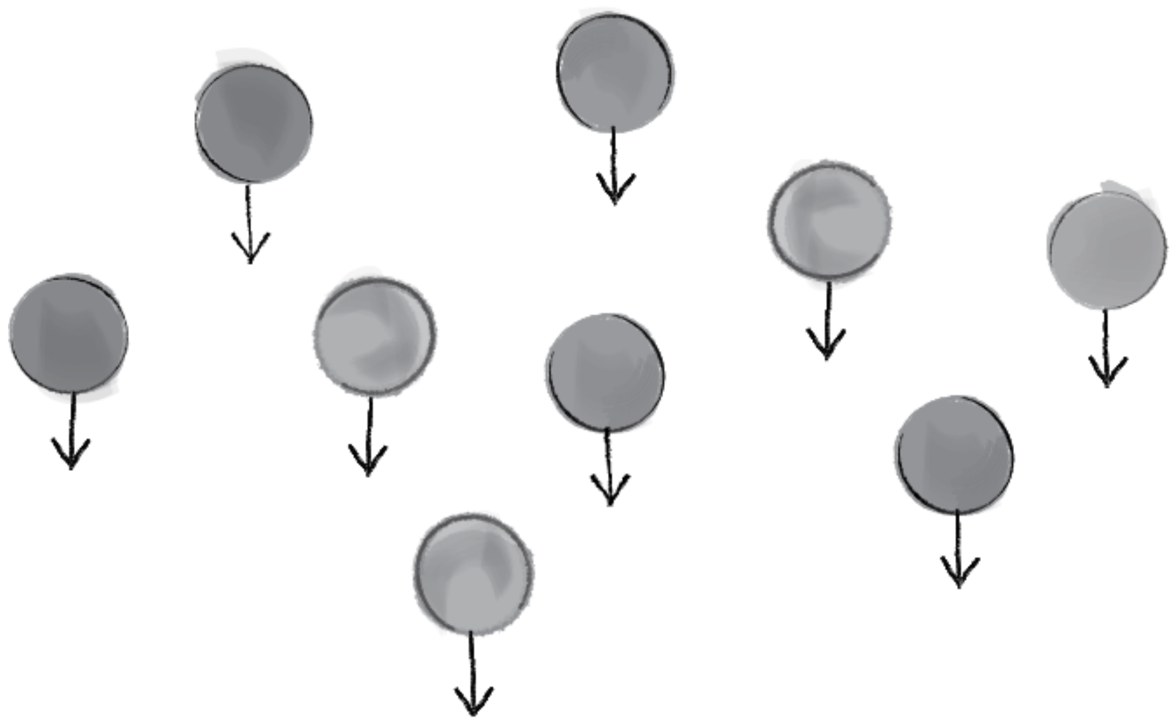
Particle Systems with Forces

```
ParticleSystem ps;

void setup() {
  size(640,360);
  ps = new ParticleSystem(new PVector(width/2,50));
}

void draw() {
  background(100);

Apply a force to all particles.
  PVector gravity = new PVector(0,0.1);
  ps.applyForce(gravity);

  ps.addParticle();
  ps.run();
}
```

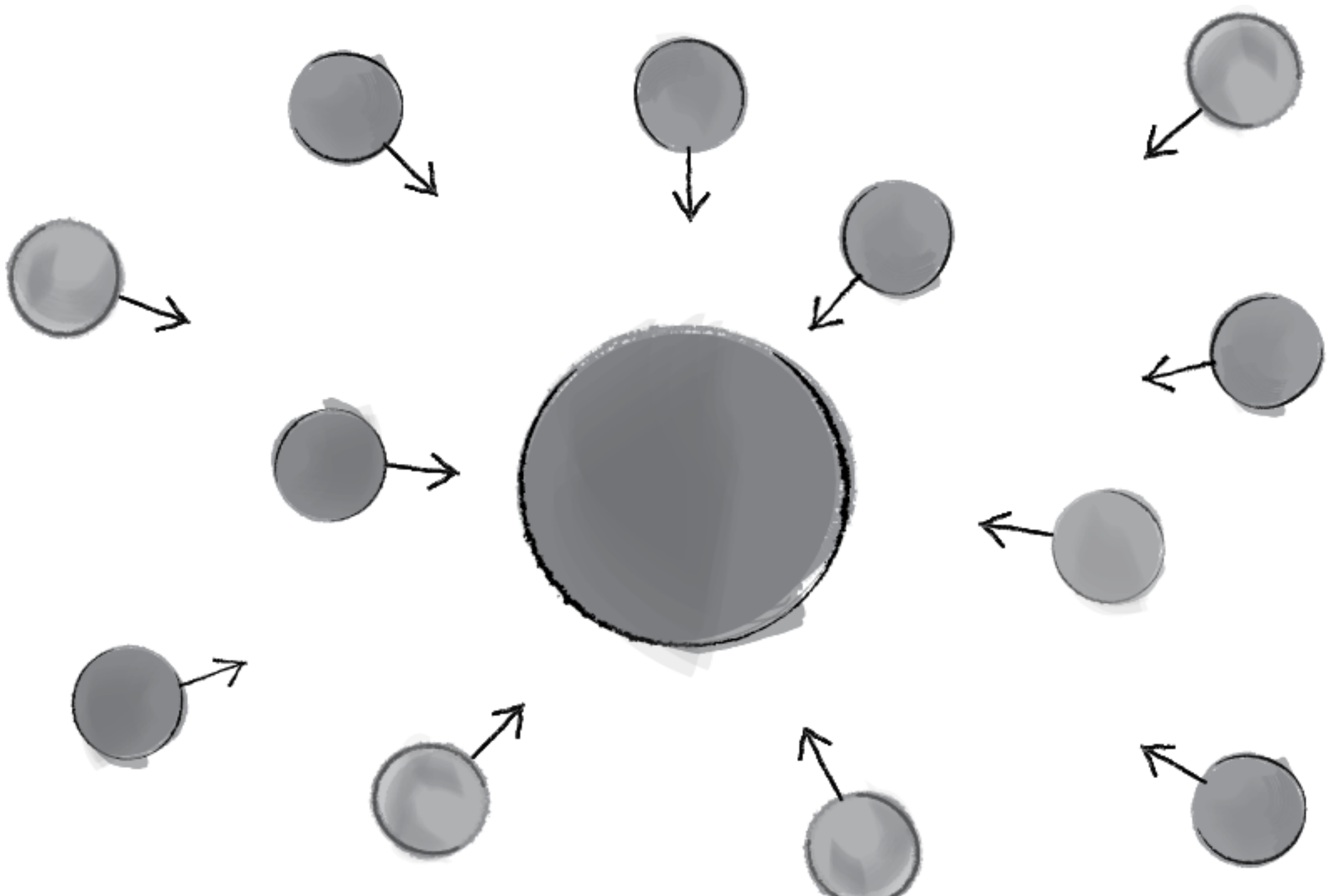
Particle Systems with Forces

```
class ParticleSystem {
  ArrayList<Particle> particles;
  PVector origin;

  ParticleSystem(PVector location) {
    origin = location.get();
    particles = new ArrayList<Particle>();
  }

  void addParticle() {
    particles.add(new Particle(origin));
  }

  void applyForce(PVector f) {
for (Particle p: particles) {
      p.applyForce(f);
    }
  }

  void run() {
Iterator<Particle> it = particles.iterator();
    while (it.hasNext()) {
      Particle p = (Particle) it.next();
      p.run();
      if (p.isDead()) {
        it.remove();
      }
    }
  }
}
```

Particle Systems with Repellers



Particle Systems with Repellers



Particle Systems with Repellers

- Repellers: inverser of the Attractor
- A Repeller object
- A function that passes the Repeller object into the ParticleSystem so that it can apply force to each particle object.

Repeller Class

```
class Repeller {
PVector location;
float r = 10;
Repeller(float x, float y) {
location = new PVector(x,y);
 }
void display() {
            stroke(255);
            fill(255);
            ellipse(location.x,location.y,r*2,r*2);
}
}
```

Repeller Class

applyForce()	applyRepeller
`void applyForce(PVector f) { for (Particle p: particles) { p.applyForce(f); } }`	`void applyRepeller(Repeller r) { for (Particle p: particles) { PVector force = r.repel(p); p.applyForce(force); } }`

Repeller

```
PVector repel(Particle p) {
        PVector dir = PVector.sub(location,p.location);
        float d = dir.mag();
        d = constrain(d,5,100);
        dir.normalize();
        float force = -1 * G / (d * d);
        dir.mult(force);
        return dir;
}
```

Repeller

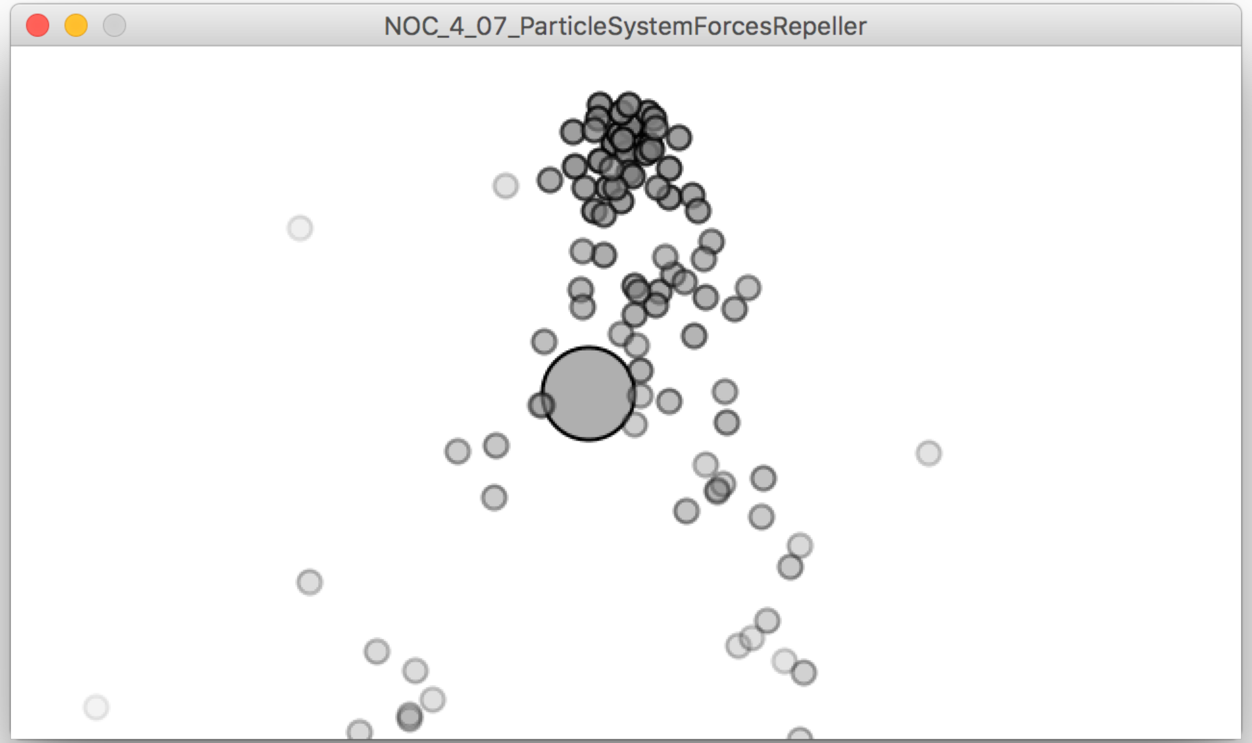


Image Textures and Additive Blending

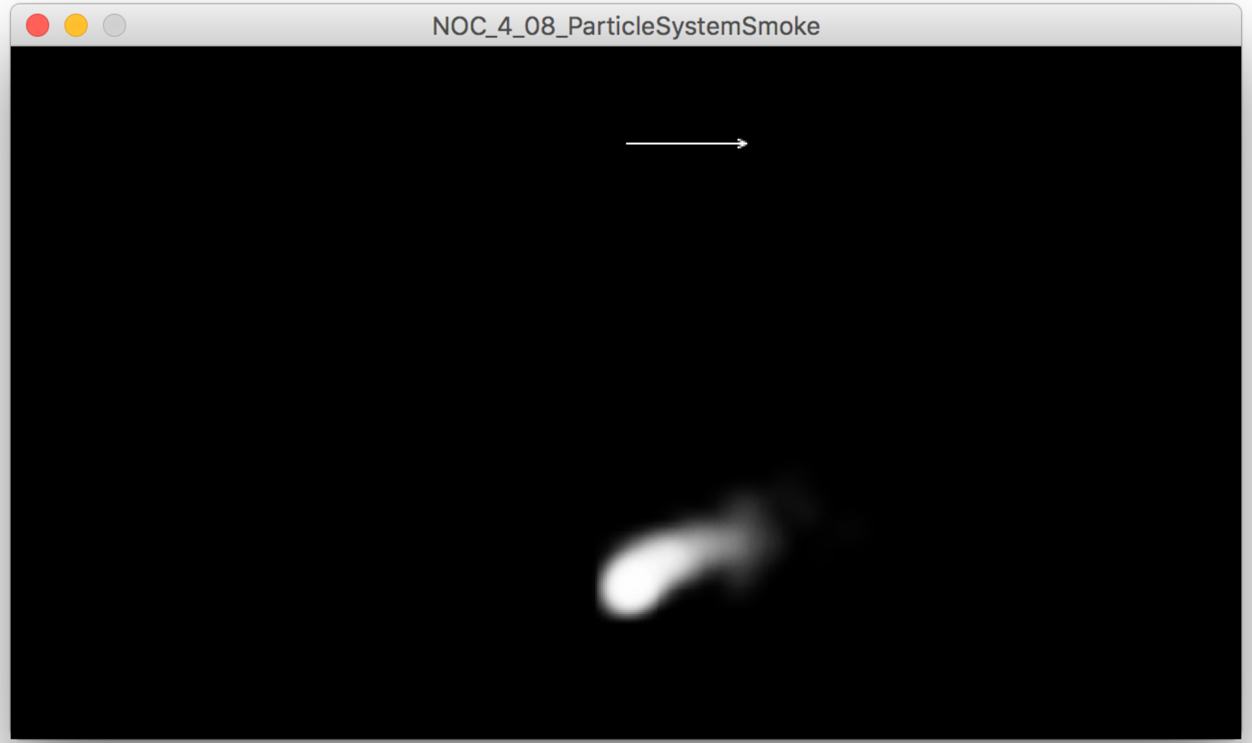


Image Textures and Additive Blending

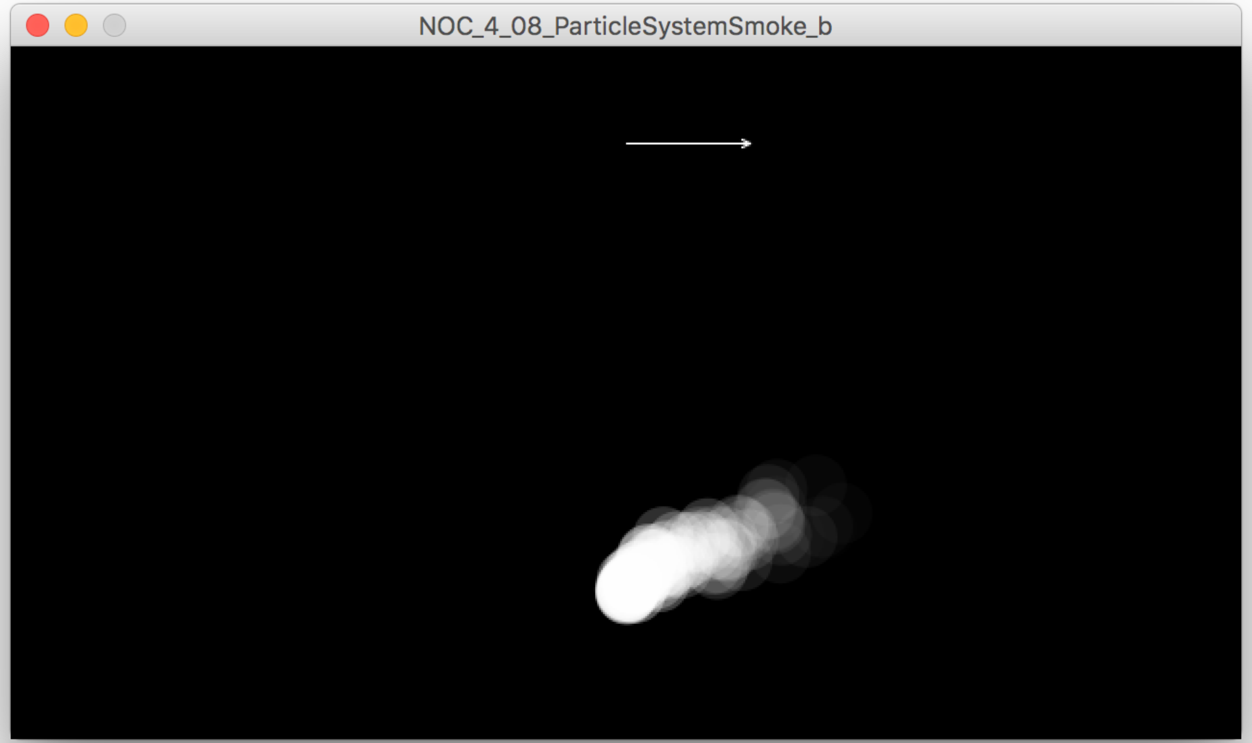


Image Textures and Additive Blending

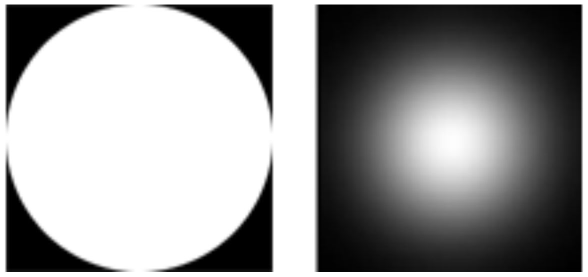
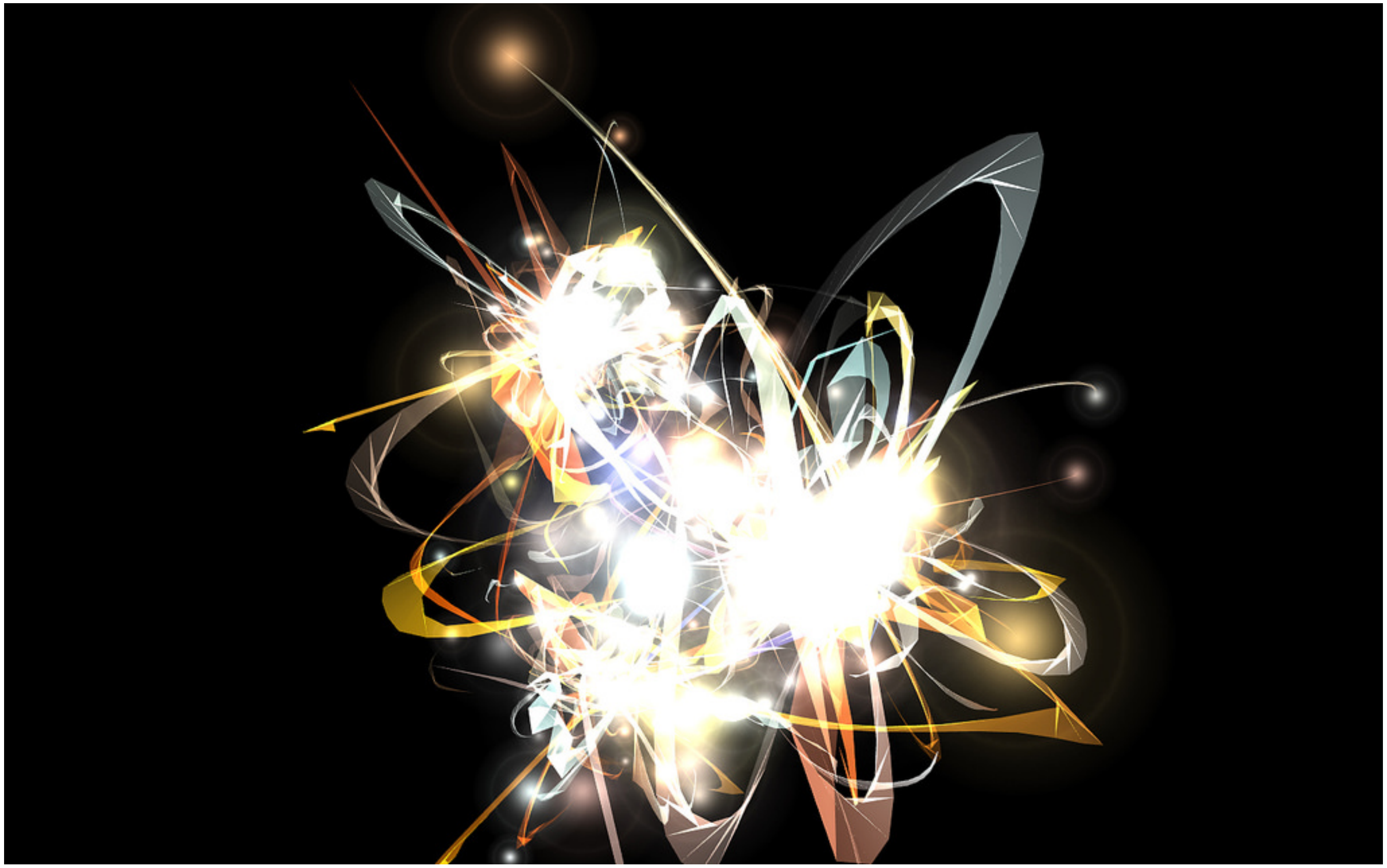


Image texture particle system

- Check ParticleSystemSmoke
- Check ParticleSystemSmokeb

Additive blending



Additive blending

- To achieve additive blending in Processing, you 'll need to use the P2D or P3D renderer.