ARTG 5330
Visualization Technologies

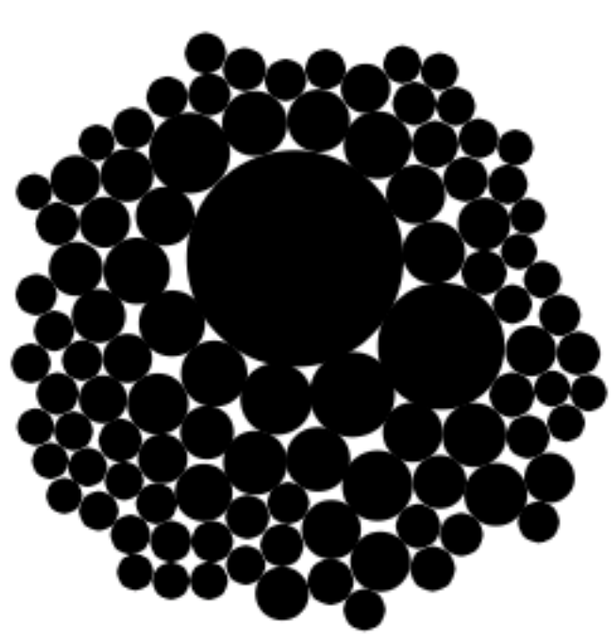
Packed bubbles

Spring semester 2016

Northeastern University
College of Arts, Media and Design

Packed bubbles

1. **Bubbles are attracted to the center**
2. **Bubbles collide with each other**
3. **Initial conditions are paramount**

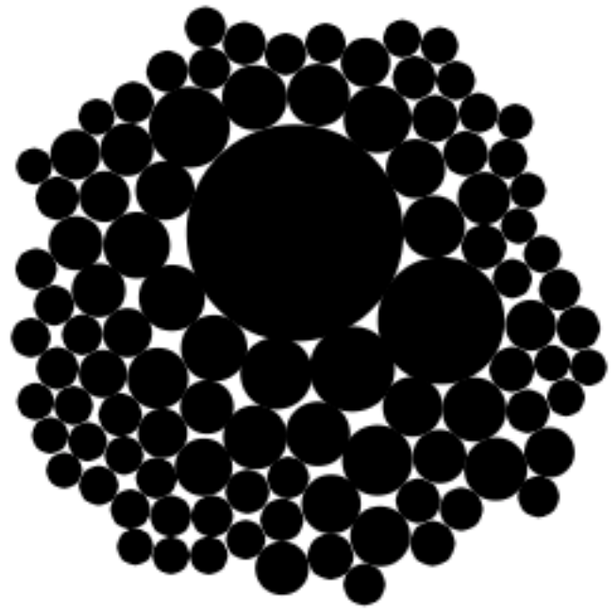


Challenge: first take on interacting with packed bubbles

1. When the mouse is over a circle, it expands.

2. When the mouse exits the circle, it returns to its normal size.

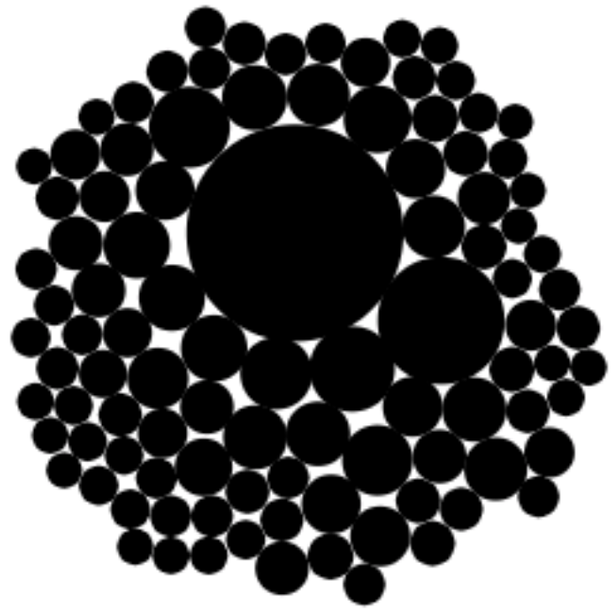
3. When the circle attained its maximum size, it displays the name of the company.



Brainstorm

—Which other types of interaction are suitable for this visualization model and data?

—What other visual variables can we use to depict data? (e.g. color)



Data aggregation

Aggregates data in a massive object such as

```
{
  Uber : total_usd,
  Groupon : total_usd,
  …
}
```

Data aggregation

```
for (var r = 0; r < table.getRowCount(); r++){
    var cname = table.getString(r, "company_name");
    var invested = table.getString(r, "amount_usd");
    invested = parseInt(invested);
    if(!isNaN(invested)){
        if(aggregated.hasOwnProperty(cname)){
            aggregated[cname]=aggregated[cname]+invested;
        }else{
            aggregated[cname] = invested;
        }
    }
}
```

Data aggregation

Converts that object into an array of companies

```
var aAggregated = [];
Object.keys(aggregated).forEach(function(name_){
    var company = {};
    company.name = name_;
    company.sum = aggregated[name_]
    aAggregated.push(company);
});
```

Data aggregation

Sorts the array by USD amount

```
aAggregated.sort(function(companyA, companyB){
    return companyB.sum - companyA.sum;
});
```

Creating particles

Creates 100 particles from the array

```
for(var i=0; i<100; i++){
    var p = new Particle(aAggregated[i].name, aAggregated[i].sum);
    particleSystem.push(p);
}
```

Creating particles

The area of the particle should signify the amount of the investment

```
/* in the constructor */
this.radius = sqrt(sum)/4000;
```

```
/* in the draw function */
ellipse(this.pos.x,
        this.pos.y,
        this.radius*2,
        this.radius*2);
```

Modifying particles

The position of a particle should be made public as it will need to be modified externally.

```
/* var pos ——————→ this.pos */
/* var vel ——————→ this.vel */
```

Create the attractor

Creates a central attractor of strength 1

```
var at = new Attractor(createVector(width/2, height/2), 1);
attractors.push(at);
```

Making uniform attraction forces

Attracts particles regardless of their distances.

```
attractors.forEach(function(A){
    var att = p5.Vector.sub(A.pos, this.pos);
    var distanceSq = att.magSq();
    if(distanceSq > 1){
        att.normalize();
        att.div(10);
        //att.mult(this.radius*this.radius/200);
        acc.add(att);
    }
}, this);
this.vel.add(acc);
this.pos.add(this.vel);
acc.mult(0);
```

Making collisions

It is a constraint based problem, where the constraint is that no circle can be juxtaposed to any other.

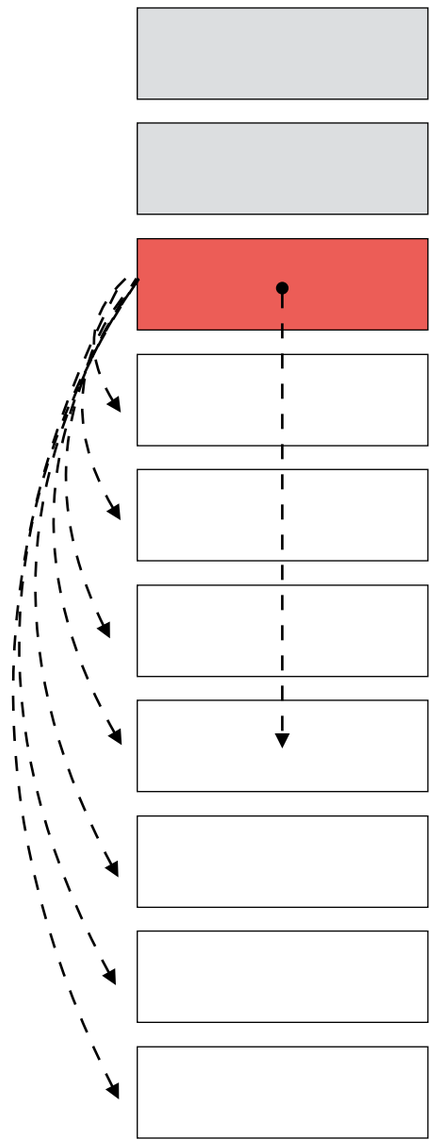
Making collisions

Circles will be compared in pairs, and if the constraints are not met, the system is modified in order to attain some of these constraints.

Making collisions

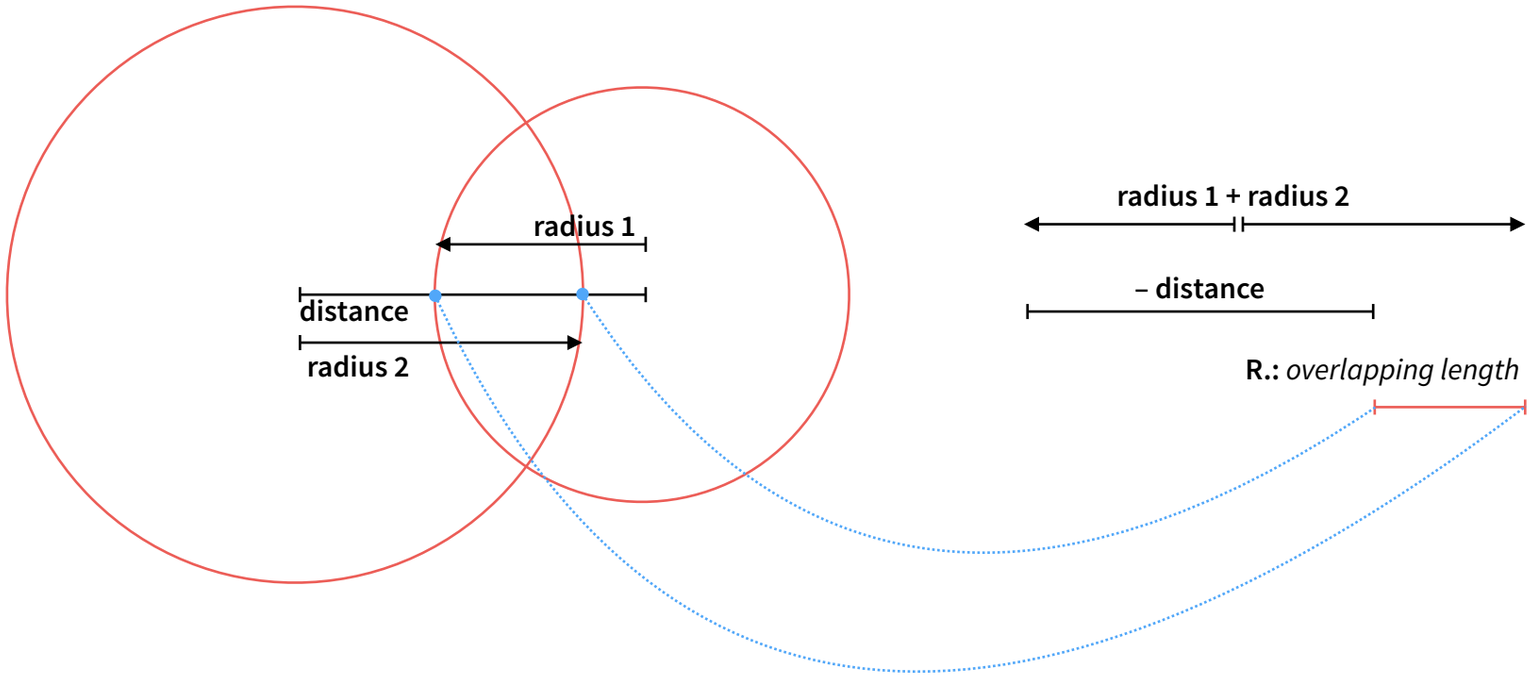
Pair-wise comparison

```
for(var i=0; i<particleSystem.length-1; i++){
    for(var j=i+1; j<particleSystem.length; j++){
        var pa = particleSystem[i];
        var pb = particleSystem[j];
         …
    }
}
```



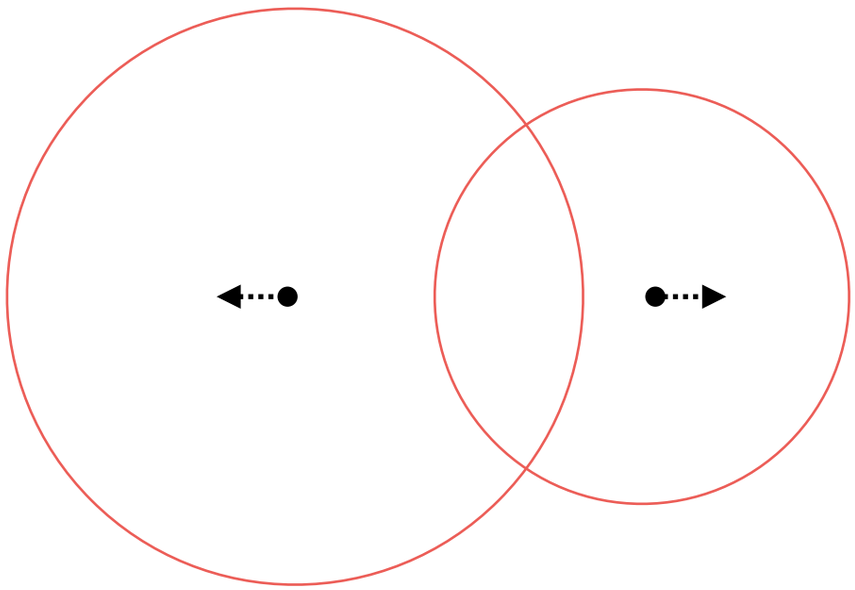
Making collisions

Adjusting the positions for each pair of particles consists of pushing each particle away from the other in a distance that nullifies their overlapping length.



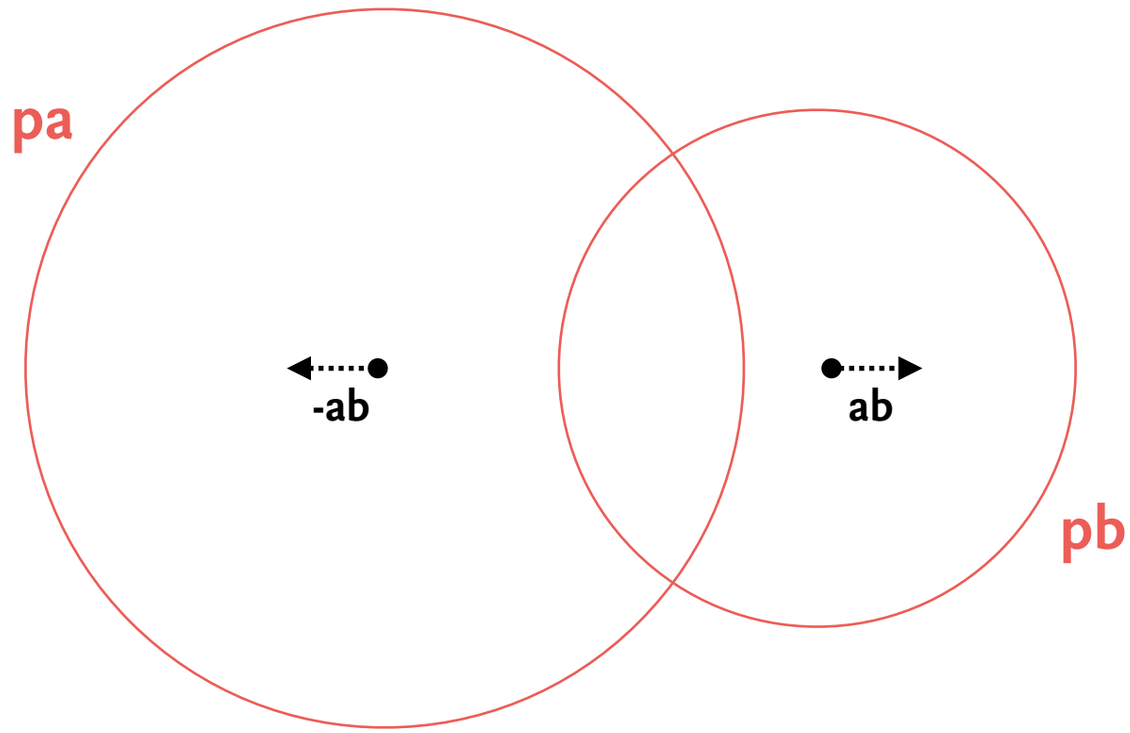
Making collisions

In the end it should result into two vectors that shift the positions of each particle in opposing directions, having each of these vectors a length that is half of the total overlapping length.



Making collisions

```
var pa = particleSystem[i];
var pb = particleSystem[j];
var ab = p5.Vector.sub(pb.pos, pa.pos);
var distSq = ab.magSq();
if(distSq <= sq(pa.radius + pb.radius)){
    var dist = sqrt(distSq);
var overlap = (pa.radius + pb.radius) - dist;
ab.div(dist); //ab.normalize();
ab.mult(overlap*0.5);
pb.pos.add(ab);
ab.mult(-1);
pa.pos.add(ab);
```



Making collisions – relaxation

```
for(var STEPS = 0; STEPS<3; STEPS++){
    for(var i=0; i<particleSystem.length-1; i++){
        for(var j=i+1; j<particleSystem.length; j++){
            var pa = particleSystem[i];
            var pb = particleSystem[j];
            var ab = p5.Vector.sub(pb.pos, pa.pos);
            var distSq = ab.magSq();
            if(distSq <= sq(pa.radius + pb.radius)){
                var dist = sqrt(distSq);
                var overlap = (pa.radius + pb.radius) - dist;
                ab.div(dist); //ab.normalize();
                ab.mult(overlap*0.5);
                pb.pos.add(ab);
                ab.mult(-1);
                pa.pos.add(ab);
            }
        }
    }
}
```

Making collisions – dumping

```
for(var STEPS = 0; STEPS<3; STEPS++){
    for(var i=0; i<particleSystem.length-1; i++){
        for(var j=i+1; j<particleSystem.length; j++){
            (…)
            if(distSq <= sq(pa.radius + pb.radius)){
                (…)

                pa.vel.mult(0.97);
                pb.vel.mult(0.97);

            }
        }
    }
}
```

Initial conditions

Arrange circles radially while making bigger circles stay the center of the canvas, and the smallest in the periphery.

```
var tempAng = random(TWO_PI);
this.pos = createVector(cos(tempAng), sin(tempAng));
this.pos.div(this.radius);
this.pos.mult(1000);
this.pos.set(this.pos.x + width/2, this.pos.y + height/2);
```