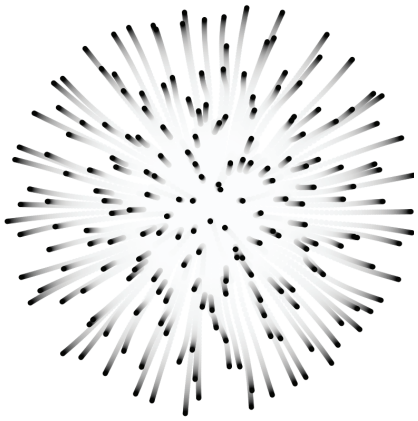
Animated Points

http://www.generative-gestaltung.de

DESCRIPTION

Distribute nodes on the display by letting them repel each other.

CODE



```
Node[] nodes = new Node[200]; // an array for the nodes

void setup() {
  size(800, 800);
  background(255);
  smooth();
  noStroke();

  for (int i = 0 ; i < nodes.length; i++) {  // init nodes
    nodes[i] = new Node(width/2+random(-1, 1), height/2+random(-1, 1));
    nodes[i].setBoundary(5, 5, width-5, height-5);
  }
}

void draw() {
  fill(255, 20);
  rect(0, 0, width, height);
  // let all nodes repel each other
  for (int i = 0 ; i < nodes.length; i++) {
    nodes[i].attract(nodes);
  }
  // apply velocity vector and update position
  for (int i = 0 ; i < nodes.length; i++) {
    nodes[i].update();
  }
  // draw node
  fill(0);
  for (int i = 0 ; i < nodes.length; i++) {
    ellipse(nodes[i].x, nodes[i].y, 10, 10);
  }
}

void keyPressed(){
  if(key=='r' || key=='R') {
    background(255);
    for (int i = 0 ; i < nodes.length; i++) {
      nodes[i].set(width/2+random(-5, 5), height/2+random(-5, 5), 0);
    }
  }
}

class Node extends PVector {

  // ------   properties ------
  // if needed, an ID for the node
  String id = "";
  float diameter = 0;

  float minX = -Float.MAX_VALUE;
  float maxX = Float.MAX_VALUE;
  float minY = -Float.MAX_VALUE;
  float maxY = Float.MAX_VALUE;
  float minZ = -Float.MAX_VALUE;
  float maxZ = Float.MAX_VALUE;
```

```
// animatedPoints.pde
// Node.pde
//
// Generative Gestaltung, ISBN: 978-3-87439-759-9
// First Edition, Hermann Schmidt, Mainz, 2009
// Hartmut Bohnacker, Benedikt Gross, Julia Laub, Claudius Lazzeroni
// Copyright 2009 Hartmut Bohnacker, Benedikt Gross, Julia Laub, Claudius Lazzeroni
//
// http://www.generative-gestaltung.de
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

/**
 * distribute nodes on the display by letting them repel each other
 *
 * KEYS
 * r          : reset positions
 * s          : save png
 */
```

```
  PVector velocity = new PVector();
  PVector pVelocity = new PVector();
  float maxVelocity = 10;

  float damping = 0.5f;
  // radius of impact
  float radius = 200;
  // strength: positive for attraction, negative for repulsion (default for Nodes)
  float strength = -1;
  // parameter that influences the form of the function
  float ramp = 1.0f;


  // ------ constructors ------
  Node() {
  }

  Node(float theX, float theY) {
    x = theX;
    y = theY;
  }

  Node(float theX, float theY, float theZ) {
    x = theX;
    y = theY;
    z = theZ;
  }

  Node(PVector theVector) {
    x = theVector.x;
    y = theVector.y;
    z = theVector.z;
  }

  // ------ rotate position around origin ------
  void rotateX(float theAngle) {
    float newy = y * cos(theAngle) - z * sin(theAngle);
    float newz = y * sin(theAngle) + z * cos(theAngle);
    y = newy;
    z = newz;
  }

  void rotateY(float theAngle) {
    float newx = x * cos(-theAngle) - z * sin(-theAngle);
    float newz = x * sin(-theAngle) + z * cos(-theAngle);
    x = newx;
    z = newz;
  }

  void rotateZ(float theAngle) {
    float newx = x * cos(theAngle) - y * sin(theAngle);
    float newy = x * sin(theAngle) + y * cos(theAngle);
    x = newx;
    y = newy;
  }

  // ------ calculate attraction ------
```

```
  void attract(Node[] theNodes) {
    // attraction or repulsion part
    for (int i = 0; i < theNodes.length; i++) {
      Node otherNode = theNodes[i];
      // stop when empty
      if (otherNode == null) break;
      // not with itself
      if (otherNode == this) continue;

      this.attract(otherNode);
    }
  }

  void attract(Node theNode) {
    float d = PVector.dist(this, theNode);

    if (d > 0 && d < radius) {
      float s = pow(d / radius, 1 / ramp);
      float f = s * 9 * strength * (1 / (s + 1) + ((s - 3) / 4)) / d;
      PVector df = PVector.sub(this, theNode);
      df.mult(f);

      theNode.velocity.x += df.x;
      theNode.velocity.y += df.y;
      theNode.velocity.z += df.z;
    }
  }

  // ------ update positions ------
  void update() {
    update(false, false, false);
  }

  void update(boolean theLockX, boolean theLockY, boolean theLockZ)
{

    velocity.limit(maxVelocity);
    pVelocity = velocity.get();

    if (!theLockX) x += velocity.x;
    if (!theLockY) y += velocity.y;
    if (!theLockZ) z += velocity.z;

    if (x < minX) {
      x = minX - (x - minX);
      velocity.x = -velocity.x;
    }
    if (x > maxX) {
      x = maxX - (x - maxX);
      velocity.x = -velocity.x;
    }

    if (y < minY) {
      y = minY - (y - minY);
      velocity.y = -velocity.y;
    }
```

```
  if (y > maxY) {
    y = maxY - (y - maxY);
    velocity.y = -velocity.y;
  }

  if (z < minZ) {
    z = minZ - (z - minZ);
    velocity.z = -velocity.z;
  }
  if (z > maxZ) {
    z = maxZ - (z - maxZ);
    velocity.z = -velocity.z;
  }

  velocity.mult(1 - damping);
}

// ------ getters and setters ------
String getID() {
  return id;
}

void setID(String theID) {
  this.id = theID;
}

float getDiameter() {
  return diameter;
}

void setDiameter(float theDiameter) {
  this.diameter = theDiameter;
}

void setBoundary(float theMinX, float theMinY, float theMinZ,
float theMaxX, float theMaxY, float theMaxZ) {
  this.minX = theMinX;
  this.maxX = theMaxX;
  this.minY = theMinY;
  this.maxY = theMaxY;
  this.minZ = theMinZ;
  this.maxZ = theMaxZ;
}

void setBoundary(float theMinX, float theMinY, float theMaxX,
float theMaxY) {
  this.minX = theMinX;
  this.maxX = theMaxX;
  this.minY = theMinY;
  this.maxY = theMaxY;
}

float getMinX() {
  return minX;
}
```

```
void setMinX(float theMinX) {
  this.minX = theMinX;
}

float getMaxX() {
  return maxX;
}

void setMaxX(float theMaxX) {
  this.maxX = theMaxX;
}

float getMinY() {
  return minY;
}

void setMinY(float theMinY) {
  this.minY = theMinY;
}

float getMaxY() {
  return maxY;
}

void setMaxY(float theMaxY) {
  this.maxY = theMaxY;
}

float getMinZ() {
  return minZ;
}

void setMinZ(float theMinZ) {
  this.minZ = theMinZ;
}

float getMaxZ() {
  return maxZ;
}

void setMaxZ(float theMaxZ) {
  this.maxZ = theMaxZ;
}

PVector getVelocity() {
  return velocity;
}

void setVelocity(PVector theVelocity) {
  this.velocity = theVelocity;
}

float getMaxVelocity() {
  return maxVelocity;
}
```

```
  void setMaxVelocity(float theMaxVelocity) {
    this.maxVelocity = theMaxVelocity;
  }

  float getDamping() {
    return damping;
  }

  void setDamping(float theDamping) {
    this.damping = theDamping;
  }

  float getRadius() {
    return radius;
  }

  void setRadius(float theRadius) {
    this.radius = theRadius;
  }

  float getStrength() {
    return strength;
  }

  void setStrength(float theStrength) {
    this.strength = theStrength;
  }

  float getRamp() {
    return ramp;
  }

  void setRamp(float theRamp) {
    this.ramp = theRamp;
  }

}
```