Generatives Gestalten

Jan Schill (jasc7047) 590303

Ideas

- NASA API von Meteoriten auf einer 3D Erde platzieren und animieren
- Sonnensystem mit physikalischen Formeln gestalten
- Interaktion mit der Kinect und Planeten/ Sterne durch physikalische Formel

Ich habe mich entschieden die letzte Idee umzusetzen.

Konzeptioneller Ansatz

Der Akteur soll vor der Kinect stehen und mit den Haenden in einem festgelegten Threshold *schwarze Loecher* erzeugen und mit diesen die auf der Zeichenflaechen gezeichneten Planeten interagieren.

Technische Umsetzung

Die Kinect verfuegt ueber einen Tiefenerkennungssensor, der sich fuer meine Projekt ideal eignet. Also lasse ich mir mit

```
depth = kinect.getRawDepth();

for (int x = 0; x < kinect.depthWidth; x++) {
    for (int y = 0; y < kinect.depthHeight; y++) {
      int offset = x + y * kinect.depthWidth;
      int de = depth[offset];

      if (de > min && de < max) {
        display.pixels[offset] = (color(35));
      } else {
        ...
      }
    }
}
```

aus meinen gewuenschten `min` bzw. `max` Treshold die Pixel geben. Ich lasse mir bei diesen Pixeln den Bereich grau einfaerben, alles andere soll schwarz sein.

`display.pixels[offset] = (color(0, 0, 0));`

Um zwischen verschiedenen Haenden zu unterscheiden benoetigt man einen `Blob Detection`-Algorithmus. Mit diesem diffenziere ich zwischen zwei Haenden, die benutzt werden koennen und lasse an den Haenden (im Zentrum der Pixel im Threshold) jeweils ein *schwarzes Loch* erscheinen.

Blob Detection

Am Anfang erzeuge ich eine Konstante mit der ich sage, nach welcher Farbe der Algorithmus suchen soll, da ich das ja *einfach* festlegen kann.
Dann lasse ich mir von jedem Pixel die Farbe geben:

```
color currentColor = display.pixels[offset];
float r1 = red(currentColor);
float g1 = green(currentColor);
float b1 = blue(currentColor);
float r2 = red(trackColor);
float g2 = green(trackColor);
float b2 = blue(trackColor);

float d = distSq(r1, g1, b1, r2, g2, b2);
```

und berechne mir mit diesen einen *Mittelwert*, welcher bestimmt, ob der *abgefangene* Farbwert dem gesuchten entspricht: `if (d < threshold*threshold && totalPixels > 250) {`

Hier ueberpruefen wir, ob wir in unserer *schwarzem Loch*-ArrayList, Pixel haben, welche in Naehe des gefundenen Pixeles sind, so dass wir *nicht* ein neues *schwarzes Loch* erzeugen, sondern ein vorhandenes erweitern.

```
boolean found = false;
  for (BlackHole s : currentBlackHoles) {
    if (s.isNear(x, y)) {
       s.add(x, y);
         found = true;
         break;
       }
  }
```

Wenn der Pixel, nicht in der Naehe eines vorhandenen *schwarzen Loches* ist, wird ein Neues erzeugt:

```
if (!found) {
  BlackHole s = new BlackHole(x, y);
  s.setColor(cols[(int) random(0, cols.length)]);
  s.setDiameter(85);
  s.setMass(random(blackHoleMassMin, blackHoleMassMax));
  currentBlackHoles.add(s);
}
```

Um eine Mindestgroesse festzulegen, entfernen wir einfach die, welche kleiner sind als 500 Pixel:

```
for (int i = currentBlackHoles.size()-1; i >= 0; i--) {
  if (currentBlackHoles.get(i).size() < 500) {
    currentBlackHoles.remove(i);
  }
}
```

In den folgenden Zeilen bestimmen wir, welche *Blobs*, zusammengefuegt werden koennen:

```
if (blackHoles.isEmpty() && currentBlackHoles.size() > 0) {
  for (BlackHole s : currentBlackHoles) {
    s.id = blackHolesCounter;
    blackHoles.add(s);
    blackHolesCounter++;
  }
} else if (blackHoles.size() <= currentBlackHoles.size()) {
    for (BlackHole s : blackHoles) {
      float recordD = 1000;
      BlackHole matched = null;
      for (BlackHole cb : currentBlackHoles) {
        PVector centerB = s.getCenter();
        PVector centerCB = cb.getCenter();
        float d = PVector.dist(centerB, centerCB);
        if (d < recordD && !cb.taken) {
          recordD = d;
          matched = cb;
        }
      }
      matched.taken = true;
      s.become(matched);
    }

    for (BlackHole s : currentBlackHoles) {
      if (!s.taken) {
        s.id = blackHolesCounter;
        blackHoles.add(s);
        blackHolesCounter++;
      }
```

```
    }
  } else if (blackHoles.size() > currentBlackHoles.size()) {
    for (BlackHole s : blackHoles) {
      s.taken = false;
    }

    for (BlackHole cb : currentBlackHoles) {
      float recordD = 1000;
      BlackHole matched = null;
      for (BlackHole s : blackHoles) {
        PVector centerB = s.getCenter();
        PVector centerCB = cb.getCenter();
        float d = PVector.dist(centerB, centerCB);
        if (d < recordD && !s.taken) {
          recordD = d;
          matched = s;
        }
      }
      if (matched != null) {
        matched.taken = true;
        matched.lifespan = maxLife;
        matched.become(cb);
      }
    }

  for (int i = blackHoles.size() - 1; i >= 0; i--) {
    BlackHole s = blackHoles.get(i);
    if (!s.taken) {
      if (s.checkLife()) {
        blackHoles.remove(i);
      }
    }
  }
}
```

Am Ende muessen wir alle Pixel einmal updaten und ein Bild zeichnen, damit wir die Illusion erzeugen, dass das Kinectbild groesser ist als nur die `512x424`-Pixel, transformieren wir das Bild ins Zentrum einer groesseren Zeichenflaeche und zeichnen rundherum eine schwarze Flaeche, in der die Sterne (nicht die schwarzen Loecher!) *fliegen* koennen.

```
display.updatePixels();
pushMatrix();
translate(width*0.4, height*0.4);
image(display, 0, 0);

for (BlackHole s : blackHoles) {
  s.setLocation(s.getCenter());
  s.show();
}
```

Nach dem Zeichnen muessen wir fuer die Interaktion zwischen Sternen/Planeten und schwarzen Loechern sorgen. Dies geschieht mit:

```
for (Star s : stars) {
  if (blackHoles.size() > 0)
  {
    for (BlackHole b : blackHoles) {
      PVector force;
      println(attracting);
      if (attracting) {
        force = b.attract(s);
      } else {
        force = b.repel(s);
      }
      s.applyForce(force);
      s.update();
      s.show();
   }
 }
 s.update();
 s.show();
}
```

Entwicklungprozess

Der Entwicklungsprozess bestand darin, dass ich anfangs viel Zeit in die andere beiden Idee investiert habe und dadurch dann weniger Zeit für meine neue und engültige Idee hatte.

Nach dem Ausleihen der Kinect habe ich mich im Internet informiert, wie man die Kinect in ein Processing Sketch einbinden kann und die Daten verarbeiten kann. Ich hatte vorher schon gesehen, dass dies gemacht wurde, es war also keine unüberlegte Idee, die Kinect mit Processing zu verbinden.

Ich habe dann recht schnell die nötigen Libraries gefunden und erforscht, wie ich die Kinect einbinde.

Danach habe ich nach einer *Anziehungs*-Formel gesucht, um im Sketch Kreise per Masse, nach physikalischen Gesetzen mit einander interagieren zu lassen.

Bibliotheken & Tools

```
import org.openkinect.freenect.*;
import org.openkinect.freenect2.*;
```

```
import org.openkinect.processing.*;
```

Mithilfe dieser Bilbiotheken wird eine Verbindung zur per USB-angeschlossenen Kinect aufzubauen.

Mit `Kinect2 kinect = new Kinect2(this);` wird dann ein Objekt erzeugt, mit dem man dann auf den Datenstrom der Kinect zugreifen kann.

```
kinect2.initVideo(); // bekommt die RGB Videostream der Kinect
kinect2.initDepth(); // die Tiefen
kinect2.initIR(); // Infrarot

kinect2.initDevice(); // Datenübertragung starten
```