Partikelsystem zur Visualisierung einer Lawine

Tiras Zemicael, Simon Rininsland

im Rahmen der Lehrveranstaltung 3D-Animation im Sommersemester 2017 an der HS-RM

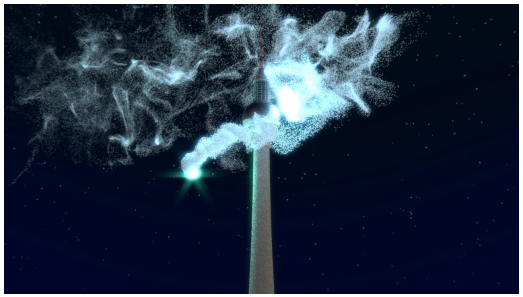
Präsentationsleitfaden

1. Was haben wir gemacht
2. Wie haben wir es gemacht
3. Welche Probleme gab es und wie haben wir sie gelöst

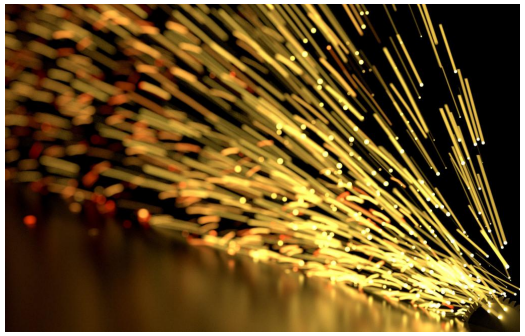
Was haben wir gemacht?

Lawine in einem Partikelsystem

Was ist ein Partikelsystem



- Rauch, Feuer, Explosionen oder Asteroidenfelder
- aber auch Haare oder Gras
- besteht aus Emitterpartikeln mit verschiedenen physikalischen Eigenschaften
  - Krafteinflüsse (Gravitation oder Kollisionen)
  - Geschwindigkeit
  - Dämpfung
  - Lebensdauer
  - Helligkeit
  - Größe/Masse
  - ...



Partikelsysteme im Bezug auf eine Lawine

Physikalische Eigenschaften von Schnee im Partikelsystem:

- Luftwiderstand (fallen mit 4km/h)
  - Oberfläche proportional zur Größe, daher immer etwa gleich
- Reibung
  - sehr hoch bis zu einem bestimmten Wert, dann sehr gering
- Elastizität
  - kaum Elastizität
- Geschwindigkeit
- Position

Wie haben wir es gemacht?

Wie haben wir es gemacht.

1. OpenGL (Python). Endlich etwas sehen!
2. Erste Bewegung
   a. Beschleunigung/Geschwindigkeit
   b. Gravitation
3. Modelle erstellen/einlesen/konvertieren/anzeigen
4. Welt in Datenstruktur abbilden
5. Kollisionen
   a. mit Terrain
   b. mit anderen Partikeln
6. Probleme lösen!!!
   a. Kollision mit Terrain lange nicht richtig.

Wie kamen wir zu diesem Ergebnis?

1. OpenGL (Python). Endlich etwas sehen!
2. Erste Bewegung.
   a. Beschleunigung/Geschwindigkeit
   b. Gravitation
3. Modelle erstellen/einlesen/konvertieren/anzeigen.
4. Welt in Datenstruktur abbilden.
5. Kollisionen
   a. mit Terrain
   b. mit anderen Partikeln
6. Probleme lösen!!!
   a. Kollision mit Terrain lange nicht richtig.

Zeitaufwand (1 Tag ~ 7 hrs)

2 Tage

1 Tag davon

1 / 2 Tag

1 / 2 Tag

2 Tage

2 Tage

5 Tage davon

3 Tage

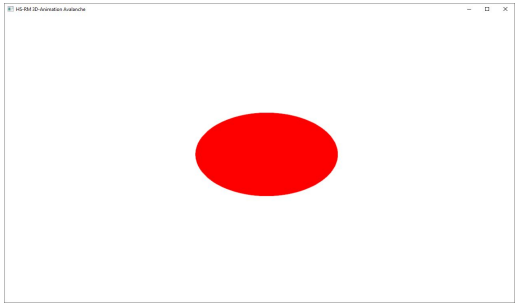
2 Tage

6 Tage davon

6 Tage

Gesamt 18 Tage ~ 126 hrs

ganz kurz OpenGL



Was bietet OpenGL:

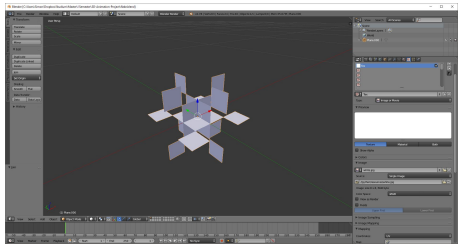
- geometrische Figuren (Geraden, Polygone, Kugeln, Quader ...)
- Transformationen (Rotation, Projektion, Translation …)
- Lichtquellen, Materialien, Texturen, Farbverläufe ...

Eine OpenGL Anwendung folgt immer demselben Prinzip:

- Fenster (mit GLUT)
- Aufbau / Ereignis
  - init Funktion, mousefunc ...
- Hauptereignisschleife
  - display Funktion

Objekte und erste Bewegung / Gravitation

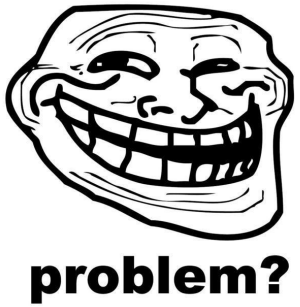
- externer erweiterter OBJ-Loader
- Modelle erstellt mit Blender



- Beschleunigung nur durch Gravitation.
- Kollision zu spät (falsch translatiert)
- Kollision **nur** auf der Y Achse und ab vorgegebenen Y Werten.

Jetzt können wir endlich Objekt- und Terrainkollision implementieren, oder?

Probleme

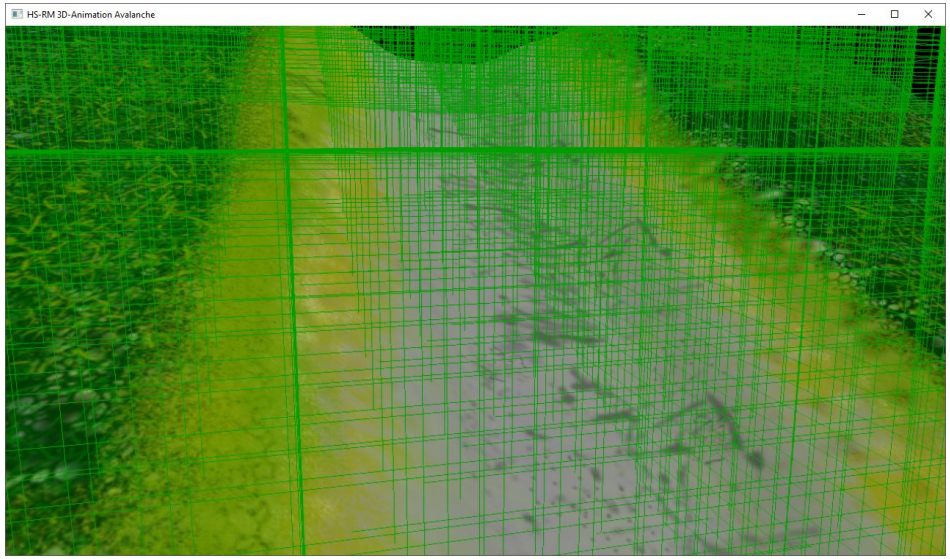


- Wie erkennen wir eine Kollision mit generische Terrains?
  - Kein manuelles nachtragen ab welchen Y Werten man kollidieren will.
- Wie erkennen wir Kollision mit anderen Bällen?
  - prüfen wir die Kollision in jedem Schritt mit allen Bällen?
- Wie reagieren wir überhaupt auf die Kollisionen?
  - Hoppla. Auf einmal wird es kompliziert!

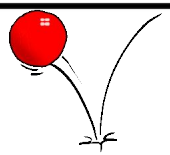
Erst einmal brauchen wir ein Modell, wo die Daten gehalten werden!

Positionsdatenhaltung - unsere Welt

- Welt ist in Grid abgebildet (128x128x128)
  - **Datenhaltung:**
    Modell wird zur Prüfung benötigt
  - **Performance:**
    Kollisionsprüfung nur mit Partikeln im selben Grid
- Terrain hat eigene Struktur
  - 2D Array mit Höhenwerten



Kollisionserkennung mit dem Terrain



- aus Höhenprofil mit X,Z vom Partikel ein **Kollisionsdreieck berechnen** (hier rot)

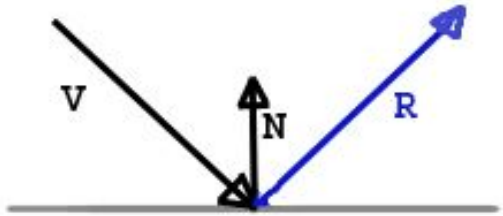
- daraus die **Normale** für das Dreieck

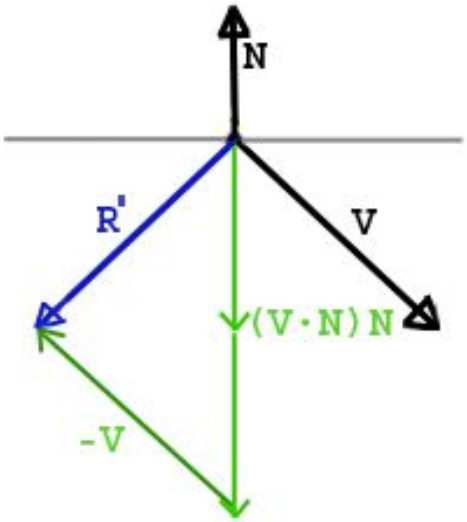
- aus einen Partikelpunkt und 2 Punkten des Kollisionsdreieckes, **Punktdreieck** und **Normale**

- Skalarprodukt beider Normalen ist der Winkel zueinander. Schräge ~ 0 bedeutet Parallelität => Kollisionsreaktion!

ein wenig Mathe: Kollisionsreaktion mit dem Terrain

- aus normale des Kollisionsdreieck (N) und eigenen Vektor (V), Ausgangsvektor berechnen (R):



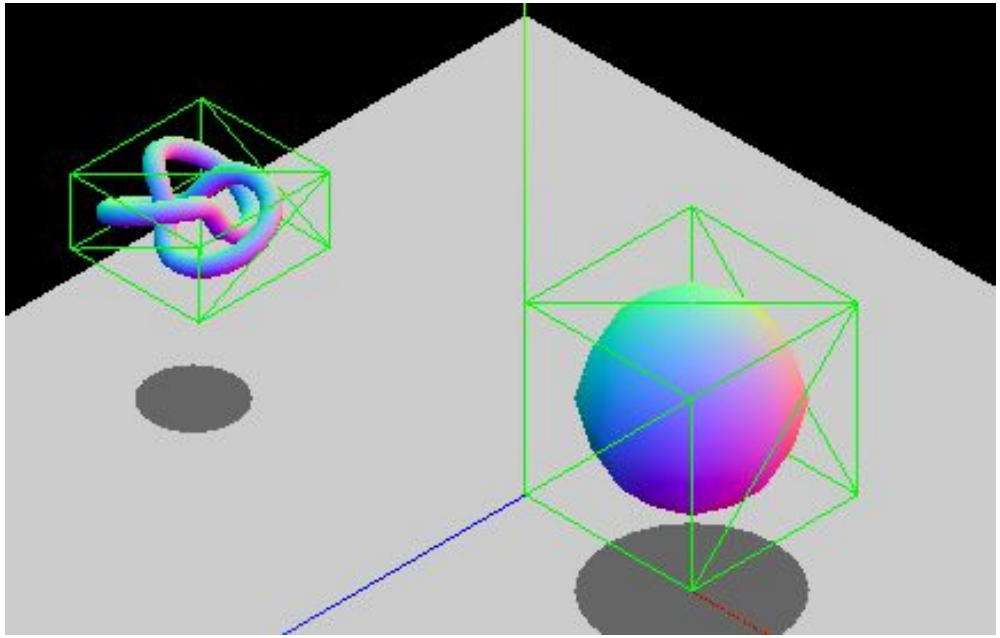


Skalarprodukt projiziert Vektor auf andere Achse! Hier **(V dot N)*N**
R’ = 2* (V dot N)*N - V

- R’ negieren um Ausgangsvektor zu bekommen
Kollisionsvektor(R) = - 2* (V dot N)*N + V
neue Geschwindigkeit des Partikels = Kollisionsvektor * Elastizität

Kollisionserkennung mit anderen Partikeln

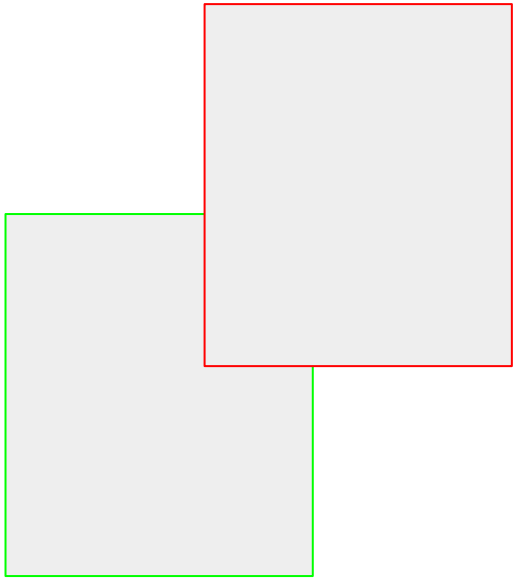
- Verschiedene Ansätze
  - Point Collision
    - Abfrage ob p1 == p2
  - Axis-aligned bounding boxes
  - Bounding Spheres (Bounding volume)



Axis-aligned bounding boxes

- einfaches verfahren in dem eine Box um das objekt erzeugt wird
- Bounding box erhält folgende Eckkanten

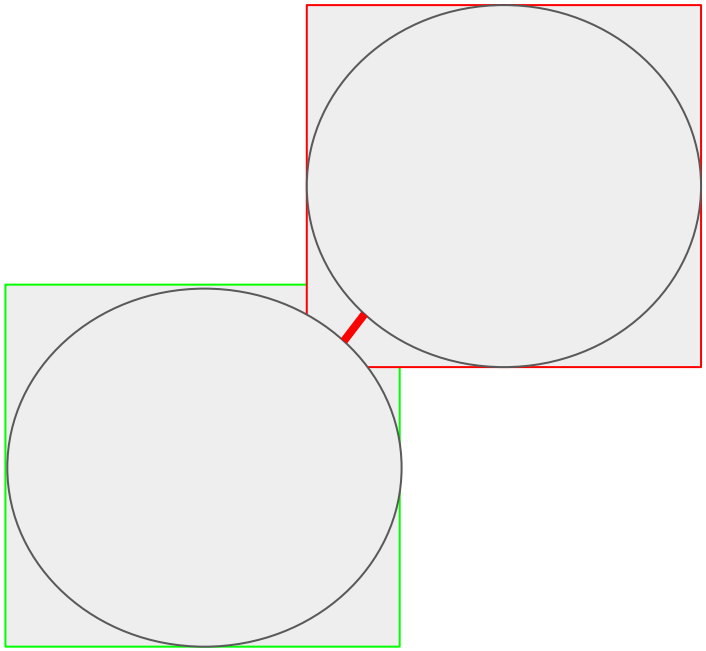
	X	Y
Point 1	min	min
Point 2	min	max
Point 3	max	min
Point 4	max	max



green.minX <= red.maxX & green.maxX >= red.minX

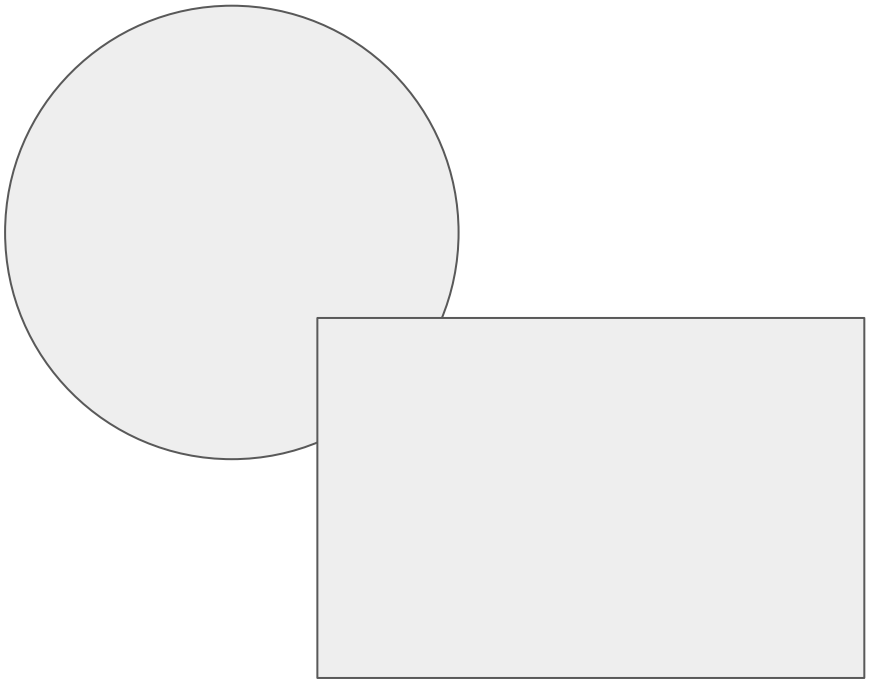
Bounding Box Problem

Spheres Kollidieren trotz nicht vorhandener Kollision



Kombinierter Ansatz

Axis-aligned bounding box vs. bounding Sphere



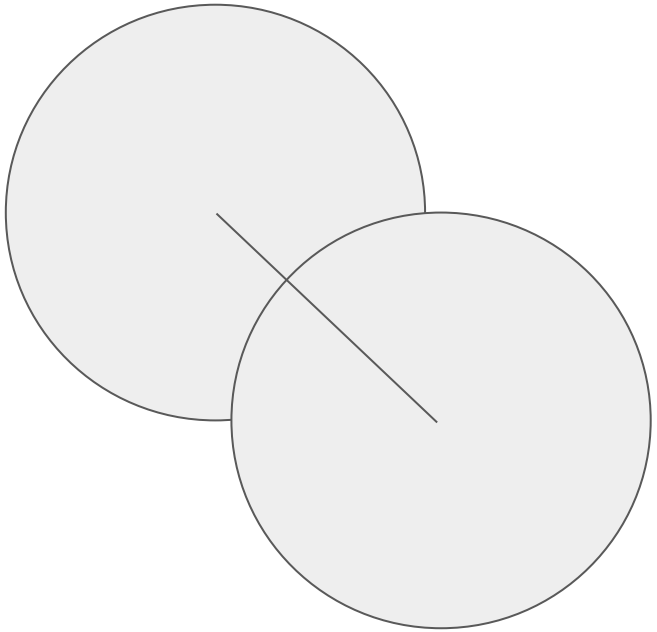
```
function intersect(sphere, box) {
  // get box closest point to sphere center by clamping
  var x = Math.max(box.minX, Math.min(sphere.x, box.maxX));
  var y = Math.max(box.minY, Math.min(sphere.y, box.maxY));
  var z = Math.max(box.minZ, Math.min(sphere.z, box.maxZ));

  // this is the same as isPointInsideSphere
  var distance = Math.sqrt((x - sphere.x) * (x - sphere.x) +
                           (y - sphere.y) * (y - sphere.y) +
                           (z - sphere.z) * (z - sphere.z));

  return distance < sphere.radius;
}
```

Bounding Spheres

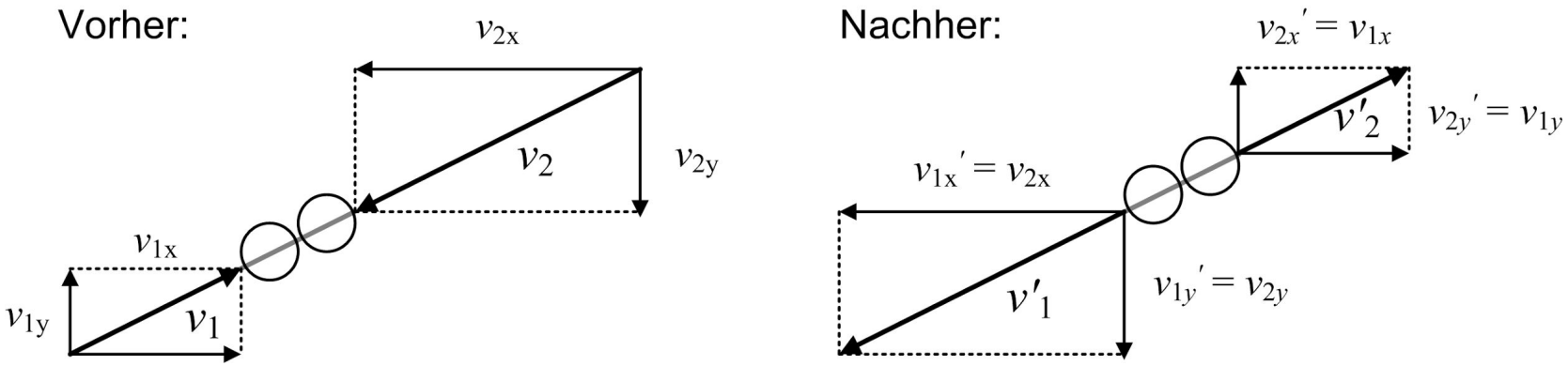
- Kollision zweier Kugeln
- Verfahren nur mit Kugeln Möglich
  - (Eier formen sind nicht möglich)



$$f(A,B) = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2} <= A_{radius} + B_{radius}$$

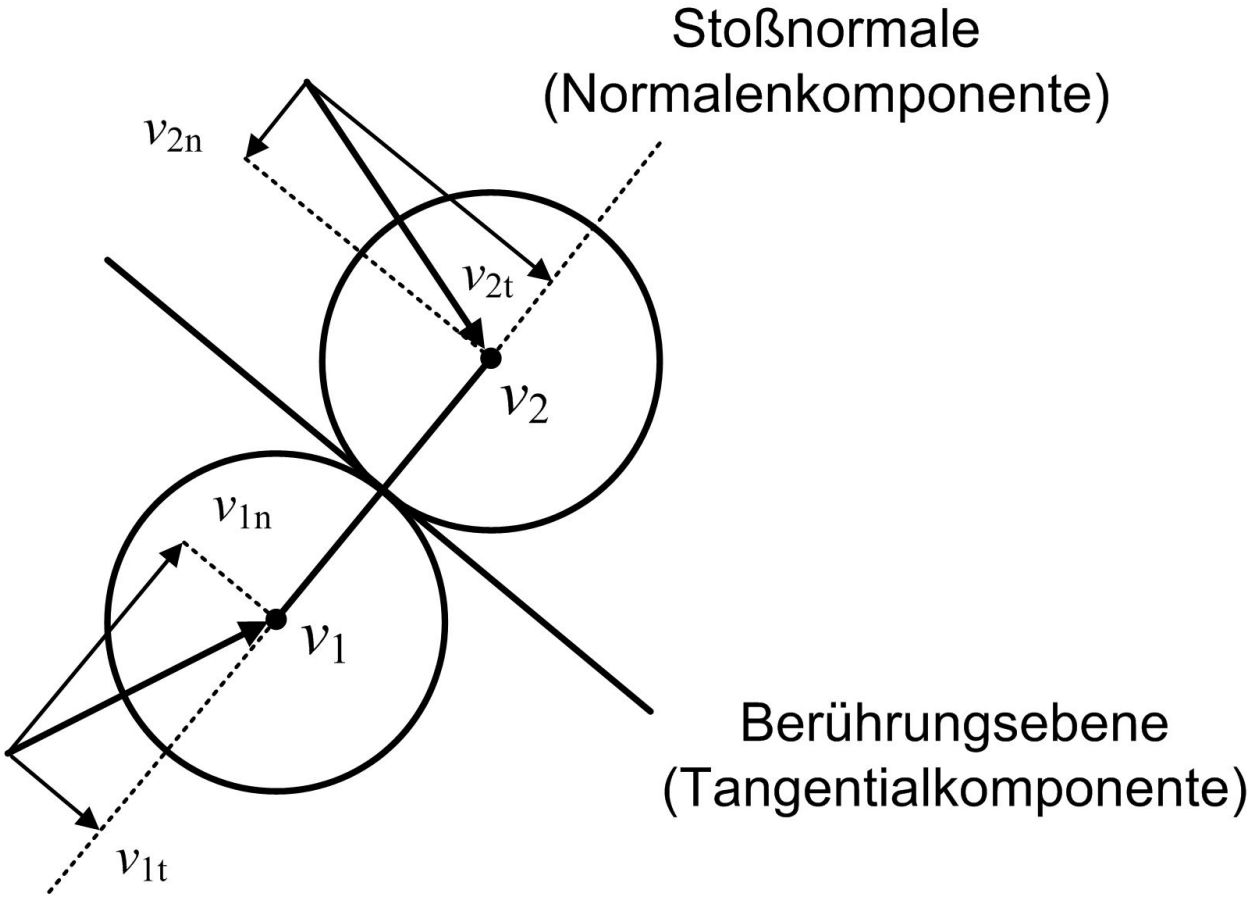
Kollision Reaktion zweier Sphären

- Kollision kommt in verschiedener form
  - Elastischer Stoß
  - Unelastischer Stoß



Elastischer gerader Stoß

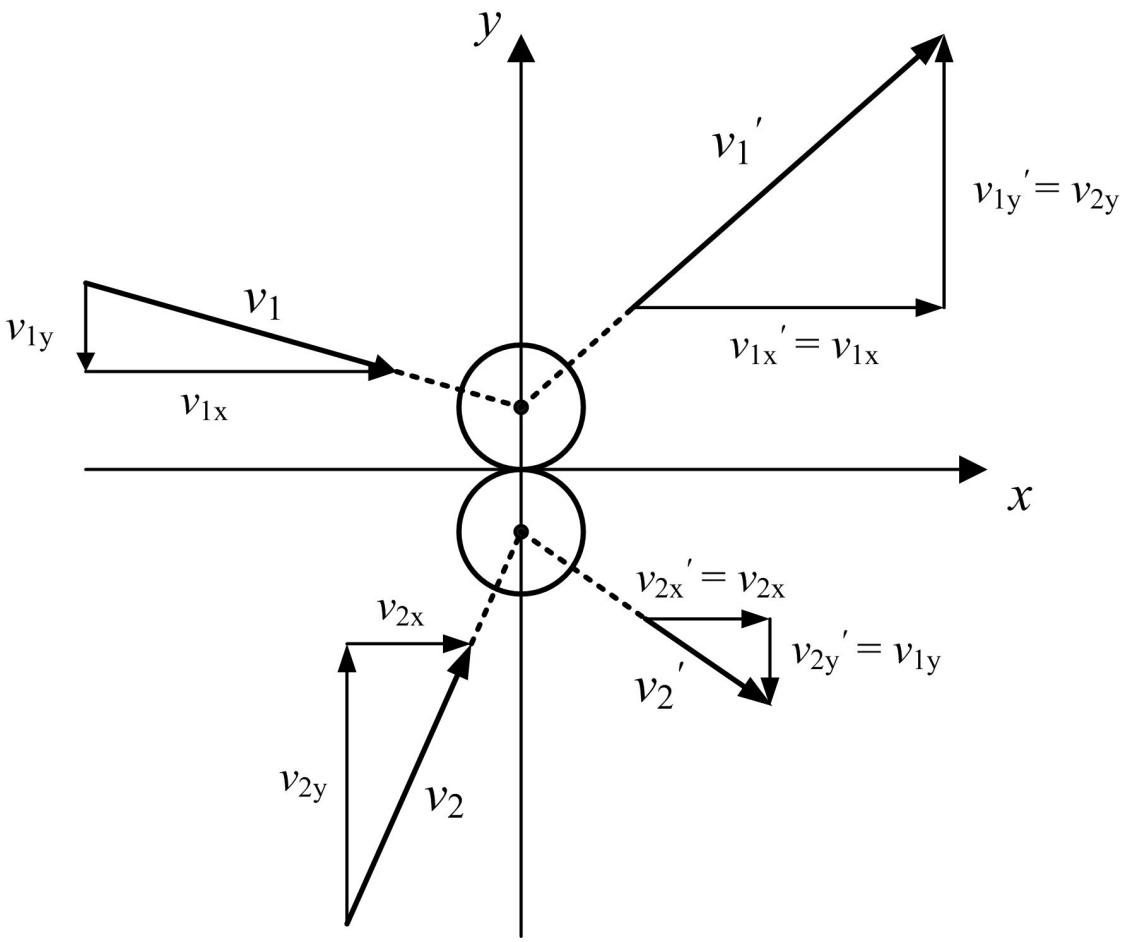
Elastischer Schiefer Stoß



Elastischer Schiefer Stoß

x-achse = Berührungsebene

y-achse = Stoßnormale



Interessante Erkenntnisse

- Berechnung der Position pro Frame. Mathematisch nie genau, immer mit Thresholds rechnen
- Performance > 1000 Flakes sehr schlecht.
- Ladezeit zu Beginn des Programms entsteht aus dem schreiben der Höhenmap in das Array (128x128 Stellen)

Quellen

Informationen:

- https://de.wikipedia.org/wiki/Partikelsystem
- https://de.wikipedia.org/wiki/Schnee
- https://www.informatik.uni-augsburg.de/de/lehrstuehle/dbis/pmi/lectures/vorhergehende_semester/ss06/graphikprogrammierung/script/opengl.pdf
- https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection
- http://www.peterloos.de/index.php/m-wpf/m-wpf-animations/70-a-wpf-elasticimpact

Bilder/Videos

- http://wemakemedia.de/3d-rendering-funken-mit-blender-2-68-cycles-sparks-particle/
- http://blenderdiplom.com/de/tutorials/508-tutorial-advanced-particle-trail-in-blender-264.html
- https://de.wikibooks.org/wiki/Blender_Dokumentation:_Partikel
- https://www.youtube.com/watch?v=0pXYp72dwl0
- https://www.quora.com/Will-a-glass-ball-bounce-higher-than-a-rubber-ball
- https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection
- http://www.peterloos.de/index.php/m-wpf/m-wpf-animations/70-a-wpf-elasticimpact