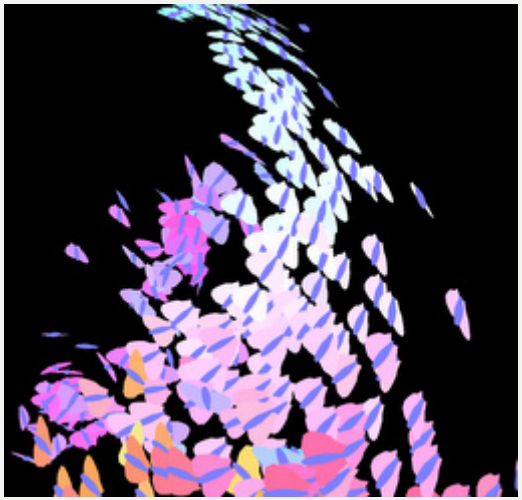
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

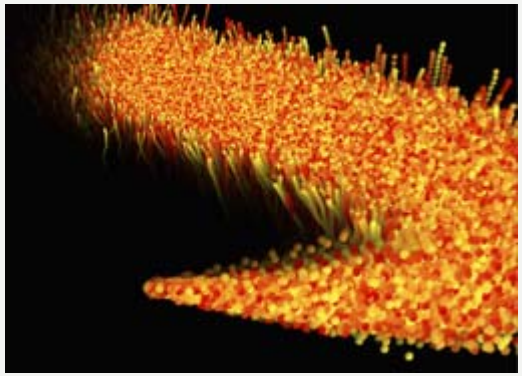
Dipl.Inf. Otmar Hilliges

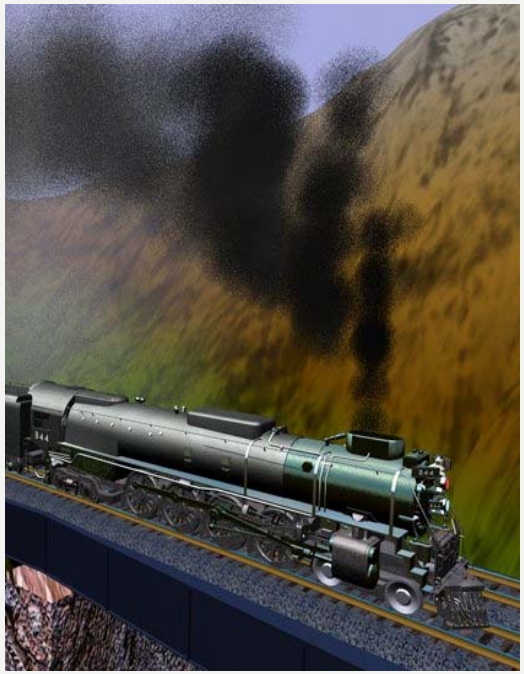
# Programmierpraktikum 3D Computer Grafik

Partikelsysteme, Multipass Rendering

# Kollisionen





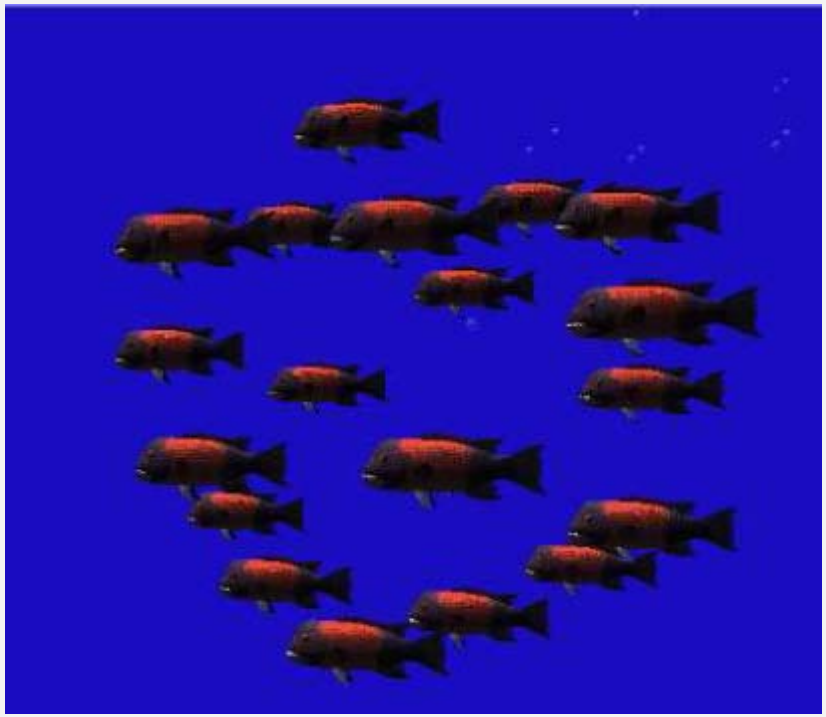


- Rauch
- Feuer
- Vulkane
- Regen
- Tierschwärme

## Eigenschaften

- Partikelsysteme sind Gruppen von Einzelelementen
- Elemente stehen in Beziehung
  - Zueinander
  - Zur Gruppe als ganzes
- Einzelne Partikel haben Attribute:
  - Farbe, Position, Geschwindigkeit, Lebensdauer…
- Äußere Kräfte wirken auf das System
  - Gravitation -> ähnliche Flugbahn für alle Partikel
  - System bewegt sich als ganzes (Rauch einer Lokomotive)
- Innere Kräfte wirken auf das System
  - Vögel eines Schwarms bleiben nahe zusammen ohne zu kollidieren

# Beispiel



## Fischwarm:

- Fische und Luftblasen als Partikel
- Äußere Kräfte:
  - Wegstrecke
  - Auftrieb
  - Kollisionen mit der Umwelt
- Innere Kräfte:
  - Schwarmbildung
  - Kollisionsvermeidung
  - Orientierung

# Implementierungsvorschlag

**Initialisierung:**

- Generierung einer bestimmten Anzahl von Partikeln
- Zuweisung der (individuellen) Attribute: Farbe, Position, Richtung, Beschleunigung…

**Zur Laufzeit:**

- Aktivkriterium prüfen (Lebenszeit, Position)
  - Tip: Partikel wiederverwenden (neu initialisieren)
- Aktualisierung der Attributswerte: Postionsänderung, Farbänderung, etc.
- Neu zeichnen

## Modellierung in C++:

- Partikelklasse (Innere Kräfte) erbt von Partikelsystemklasse (Äußere Kräfte)
- `draw()` in der Hauptklasse

```
for (int i= 0; i<particles.size(); i++)
      particles.at(i)->update();
```

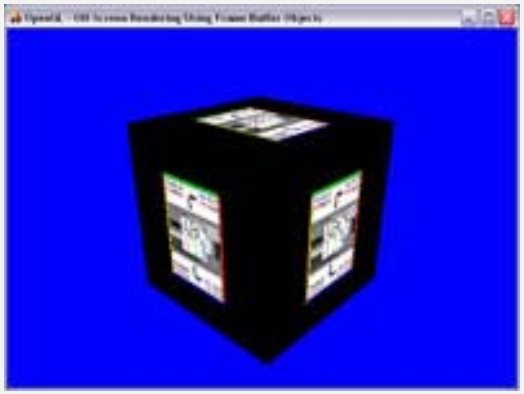
- `update()` in der Partikelklasse

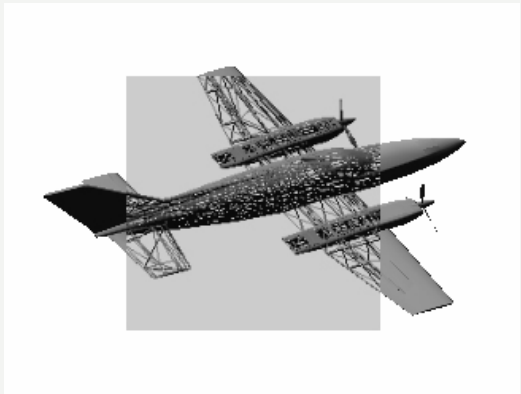
```
vec3f direction;
float angle;
m_position_particle += m_direction_particle *
    m_speed_particle + m_direction_global *
    m_speed_global;
```

# Literatur

- Nehe Particle Tutorial:
  http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=19
- Advanced Tutorials (Point Sprites etc.):
  http://www.codesampler.com/oglsrc/oglsrc_6.htm

# Offscreen Rendering





- Technik zum Erstellen von Spezialeffekten
- Objekte können teilweise modifiziert werden
- Spezielle Beleuchtungs- und Bewegungseffekte
- Basiert auf dynamischen Texturen

- Mehrere Techniken zur Realisierung von Offscreen-Rendering: pBuffer, render_texture, framebuffer_object
- FBOs sind performanteste Version und relativ einfach zu programmieren
- Vorraussetzung EXT_framebuffer_object
- Bietet mehrere attachment points:
  - Farbtexturen (color render target)
  - Tiefen(Höhen) Texturen (depth render target)
  - Stencil Texturen (stencil render target)

## Vorgehensweise

- Im ersten Durchgang wird FBO verwendet um eine Textur zu erzeugen
- (Rendere das Ergebniss auf ein Viereck mit identischer Größe wie der Viewport (orthographische Projektion))
- (Verwende Shader zum Manipulieren der Pixel)
- Rendere Ergebnis als Textur in den Framebuffer (oder ein anderes Offscreen Render Target)

# FBO Beispiel

```
GLuint fbo, color;
//Create FBO:
glGenFramebuffersEXT(1,&fbo);
//Create Offscreen color texture:
glGenTextures(1,&color);
glBindTexture(GL_TEXTURE_2D, color);
glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA8,width,height,0,GL_RGBA,GL_UNSIGNED
   _BYTE,NULL);
//Bind the FBO and attach color Texture to Render target:
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,GL_COLORATTACHMENT0_EXT,GLT
   EXTURE_2D,color,0);
//Render to Texture:
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glPushAttrib(GL_VIEWPORT_BIT);
glViewport(0,0,width, height);


// Render as normal here
// output goes to the FBO and it's attached buffers

glPopAttrib();
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```

# Literatur

- CodeSampler Off-Screen Rendering Tutorial

  http://www.codesampler.com/oglsrc/oglsrc_14.htm
- FBO 101:

  http://www.gamedev.net/reference/articles/article2331.asp