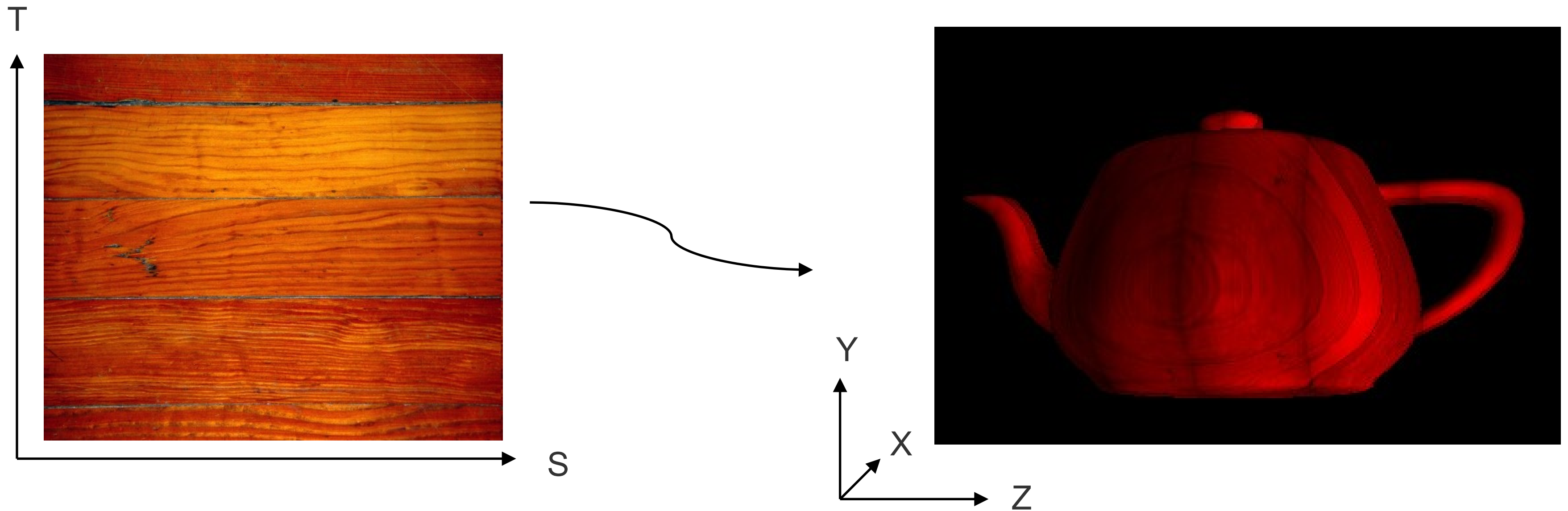
Texture Mapping

Texture Mapping für Fortgeschrittene



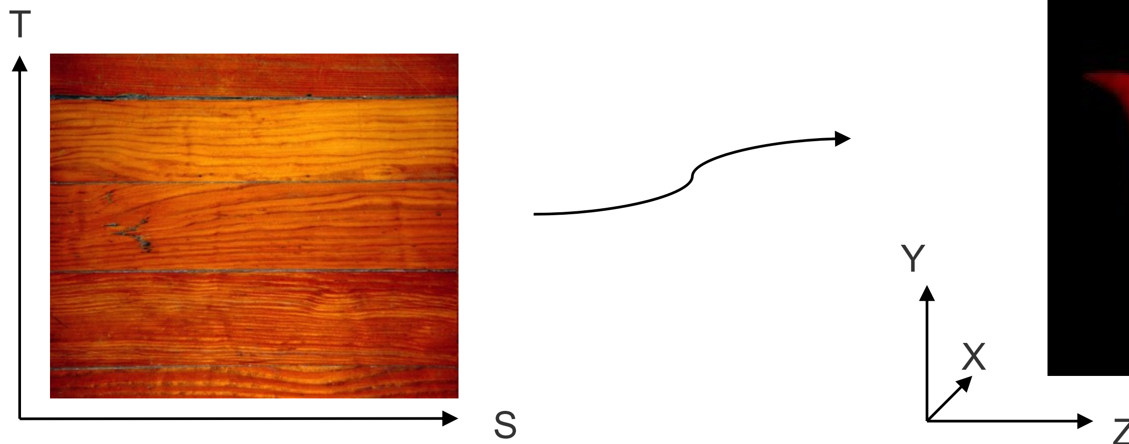
Parametrisierung

```
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
GLfloat sGenParams[4] = {1.0, 0.0, 0.0, 0.0};
GLfloat tGenParams[4] = {0.0, 1.0, 0.0, 0.0};
glTexGenfv(GL_S, GL_OBJECT_PLANE, sGenParams);
glTexGenfv(GL_T, GL_OBJECT_PLANE, tGenParams);
```

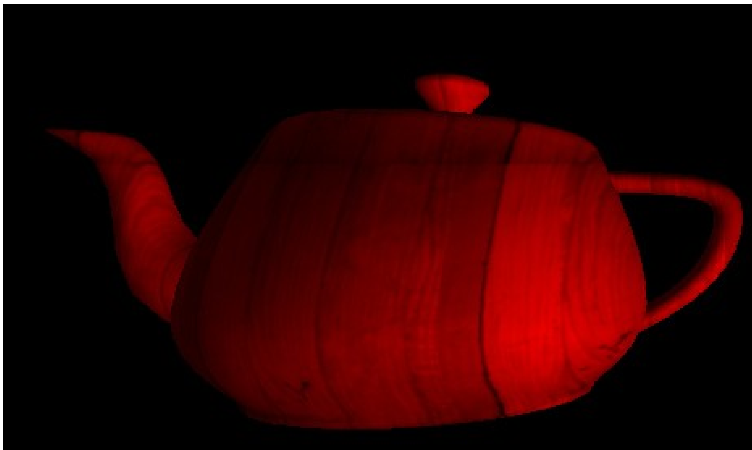
Der 3D Punkt (7.0, 3.0, 8.0) wird abgebildet auf

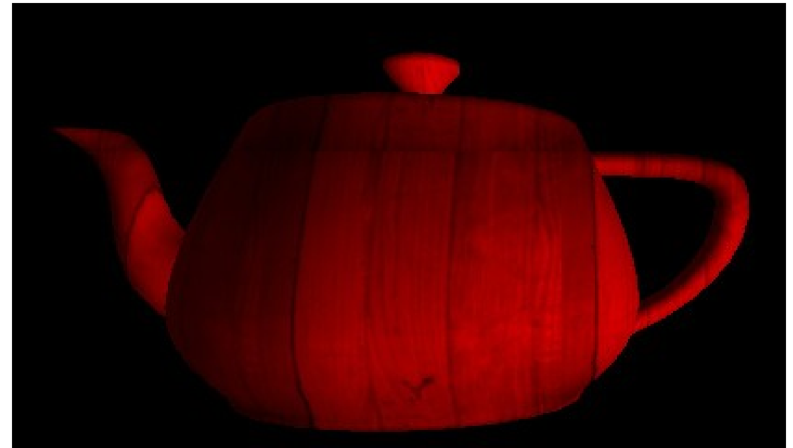
$s = 1.0 * 7.0 + 0.0 * 3.0 + 0.0 * 8.0 + 0.0 * 1.0 = 7.0$

$t = 0.0 * 7.0 + 1.0 * 3.0 + 0.0 * 8.0 + 0.0 * 1.0 = 3.0$



- GL_OBJECT_LINEAR: Die mit glVertex spezifizierten Koordinaten werden verwendet
- GL_EYE_LINEAR: Koordinaten werden mit der Modelview-Matrix multipliziert. Textur bleibt immer an der selben Stelle im Raum, nicht auf dem Objekt





Buffers

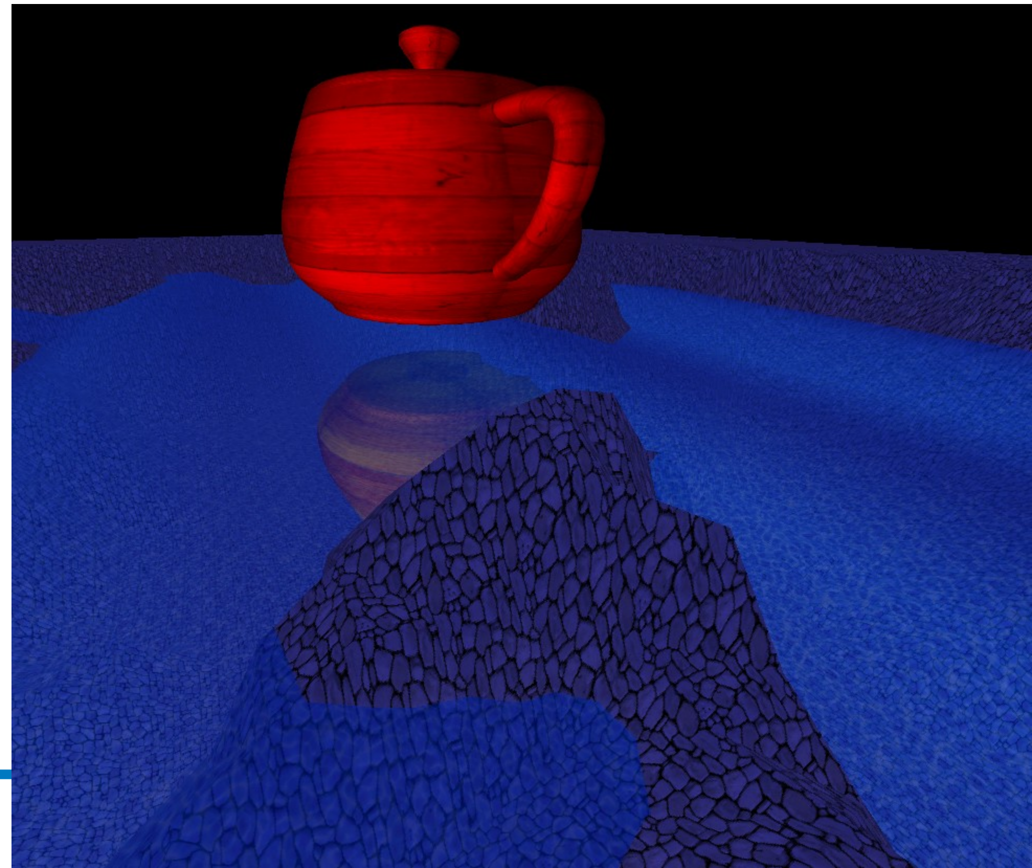
Buffers

- Enthalten Informationen Pixelweise

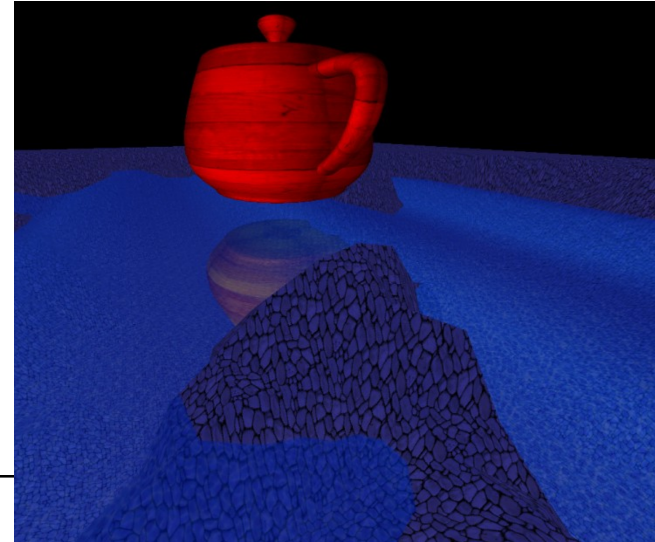
- Color buffer: Enthält das Bild; Farben und Alpha-Werte. Eventuell auch zwei color buffers für Stereo
- Depth buffer: Tiefe für jeden Pixel
- Stencil buffer: Nur bestimmte Teile des Bildschirms zeichnen
- Accumulation buffer: Zusammenbauen von Bildern aus anderen Buffern (z.B. supersampling und averaging)

Reflektionen

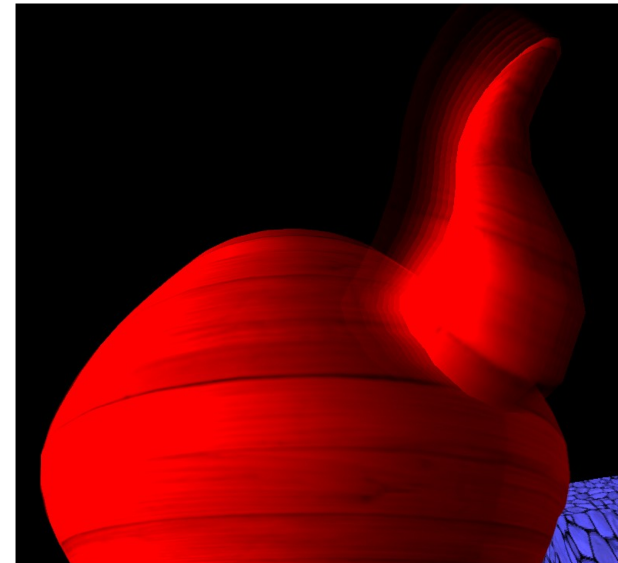
- Reflektierenden Bereich in den Stencil Buffer schreiben
- Reflektion unter die reflektierende Fläche zeichnen
- Reflektierende Fläche zeichnen



```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
…

glEnable(GL_STENCIL_TEST);
glColorMask(0, 0, 0, 0);                    // do not draw to color buffer
glDepthMask(0);                             // do not draw to depth buffer
glStencilFunc(GL_ALWAYS, 1, 1);             // allways pass stencil test
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);  // always set stencil buffer to 1

drawReflectingArea();

glColorMask(1, 1, 1, 1);
glDepthMask(1);
glStencilFunc(GL_EQUAL, 1, 1);              // only pass if stencil buffer is 1
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);     // don't change stencil buffer

glPushMatrix();
glScalef(1, -1, 1);
glTranslatef(0, OBJECT_HEIGHT, 0);
drawObject()                   // reflection
glPopMatrix();
glDisable(GL_STENCIL_TEST);

glEnable(GL_BLEND);
glColor4f(1, 1, 1, 0.7f);
drawReflectingArea();
glDisable(GL_BLEND);
drawObject();                  // real object
```



Accumulation buffer



- Kombiniere mehrere Bilder
- z.B. Motion Blur

```
glClear(GL_ACCUM_BUFFER_BIT);
for (float i=0.0; i<5.0; i+=1.0) {
        paintImage(time - i);
        glAccum(GL_ACCUM, 0.1);
}
paintImage(time);
glAccum(0.7);
glAccum(GL_RETURN, 1.0);
```

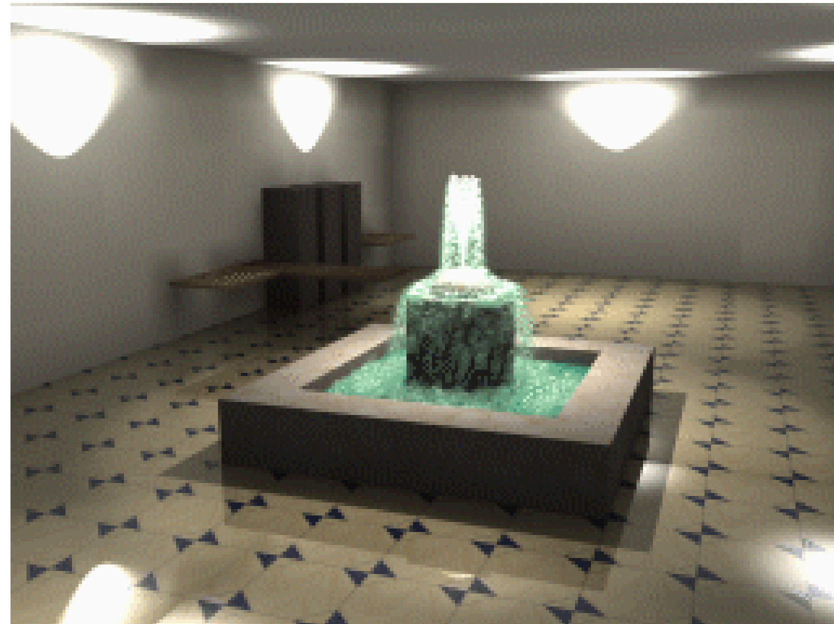
- Supersamling: Kamera mehrmals leicht versetzen
- Schärfentiefe

- `GL_ACCUM, GL_LOAD, GL_RETURN, GL_ADD, GL_MULT`

Partikelsysteme

Partikelsysteme

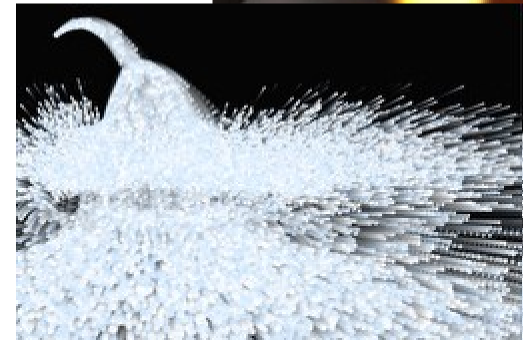
- Simulation eines Effekts durch viele einzelne Partikel
- Alle Partikel verhalten sich “gleich”
- Feuer, Rauch, Wasser, Gras, Schnee, Haare, Autos, Menschen



Partikelsysteme

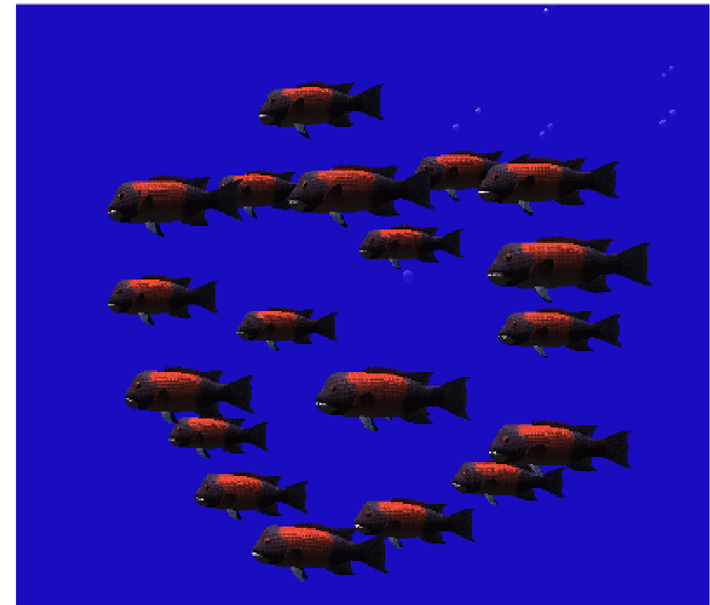
- Ein Emitter erzeugt Partikel
- Jedes Partikel hat eigene Attribute: Farbe, Größe, Geschwindigkeit, Lebensdauer
- Attribute werden bei der Erschaffung eines Partikels aus einer Wahrscheinlichkeitsverteilung gezogen, die für alle Partikel gleich ist
- Das Verhalten von Partikeln hängt von Regeln, Attributen und äußeren Einflüssen ab

- Beispiel: Wasser
  - alle erzeugten Partikel haben gleiche Farbe, aber unterschiedliche Größe und Richtung als Attribut
  - Schwerkraft wirkt auf alle Partikel



Fische als Partikelsystem

- Emitter: Nur am Anfang wird eine bestimmte Zahl emittiert
- Attribute: Größe, Farbe, Richtung, Geschwindigkeit
- Richtung und Geschwindigkeit ändern sich teils zufällig
- Richtung und Geschwindigkeit ändern sich auch durch Schwarmbildung und Kollisionsvermeidung
- Fische können selbst Emitter sein: Luftblasen



Mögliche Implementierungen

- Particle Engine regelt alles
  - Attribute in Arrays gespeichert
  - Particle Engine ändert die Attribute und zeichnet die Partikel

- Jedes Partikel ist die Instanz einer Klasse
  - Particle Engine ist nur der Emitter, erzeugt Instanzen und legt Attribute fest
  - Particle Engine ruft dann nur noch Particle::update() auf
  - Jede Instanz ändert ihre eigenen Attribute und zeichnet sich selbst

Vorschlag zur Realisierung in C++

- `draw()` in der Hauptklasse:

```
for (int i=0; i<particles.size(); i++)
    particles[i].update();
```

- `update()` in den Partikelklassen:

```
Vec3f direction;
float angle;
m_position_particle += m_direction_particle *
    m_speed_particle +  m_direction_global *
    m_speed_global ;
```