

# Computergrafik

Universität Osnabrück, Henning Wenke, 2012-06-18

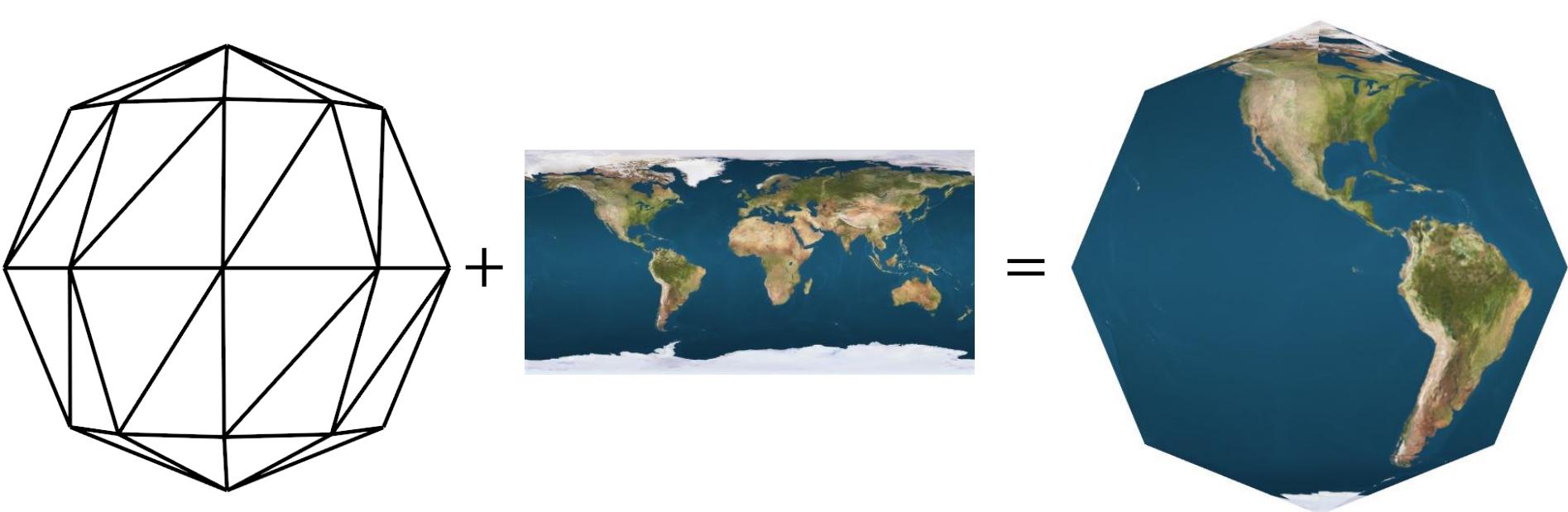
# Kapitel XIII



Texturing

# Definition: Textur (Computergrafik)

- Geometrieunabhängige Oberflächeneigenschaft
- Typischerweise höher aufgelöst als Geometrie, liefert daher zusätzliche Details
- Implementation offen: Daten? Algorithmus?
- Beispiel: Farbtextur



# Textur (OpenGL)

---

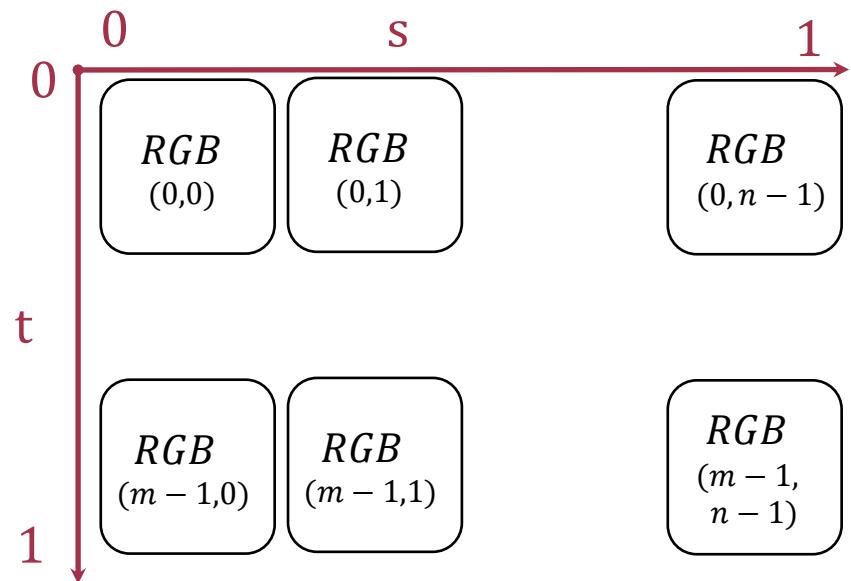
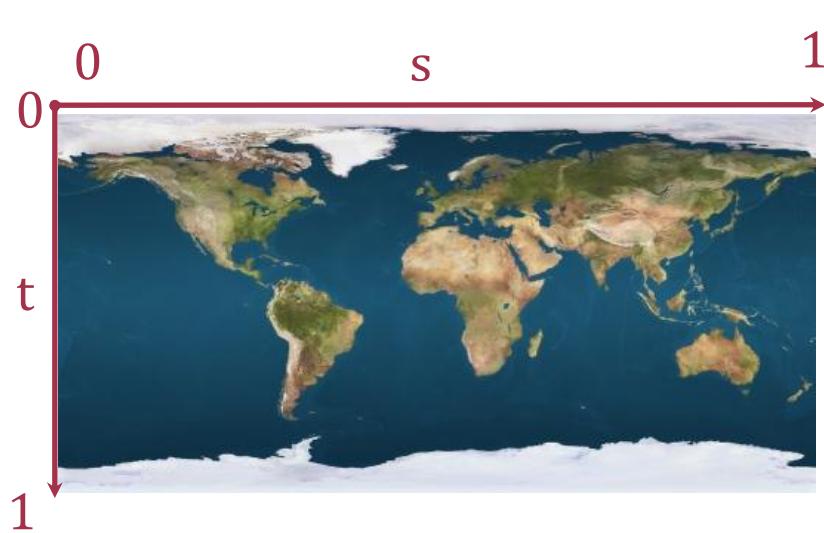
- Datenstruktur, welche Rasterdaten enthält
- Rasterdaten: 1-3 dimensionales, regelmäßiges Gitter
  - Beispiel Bild: Rechteckiges 2D Gitter aus Farbwerten
- Zusätzlich Funktionalität zum Zugriff darauf
  - Beispiel: Kontinuierlicher Zugriff durch Liefern des entsprechend interpolierten Wertes
- Texel:
  - Ein Eintrag der Textur [Tex]ture [El]ement
  - 1-4 Dimensional
- 2D-OpenGL-Textur naheliegende Datenstruktur um Computergrafik Daten-Textur zu implementieren
- Algorithmische bzw. prozedurale Textur mit OpenGL?
  - Nicht mit GL-Textur, lediglich Daten
  - Entsprechenden Shader programmieren

## 13.1

---

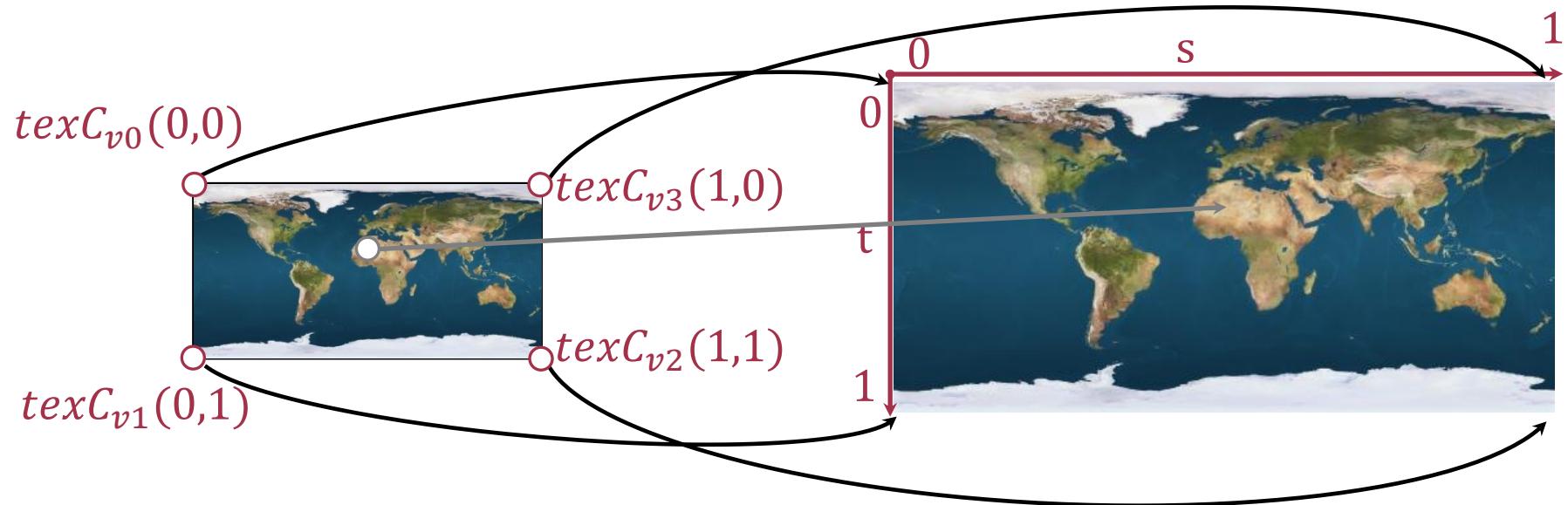
### 2D Texturen

# Beispiel: 2D-RGB-Rasterdaten



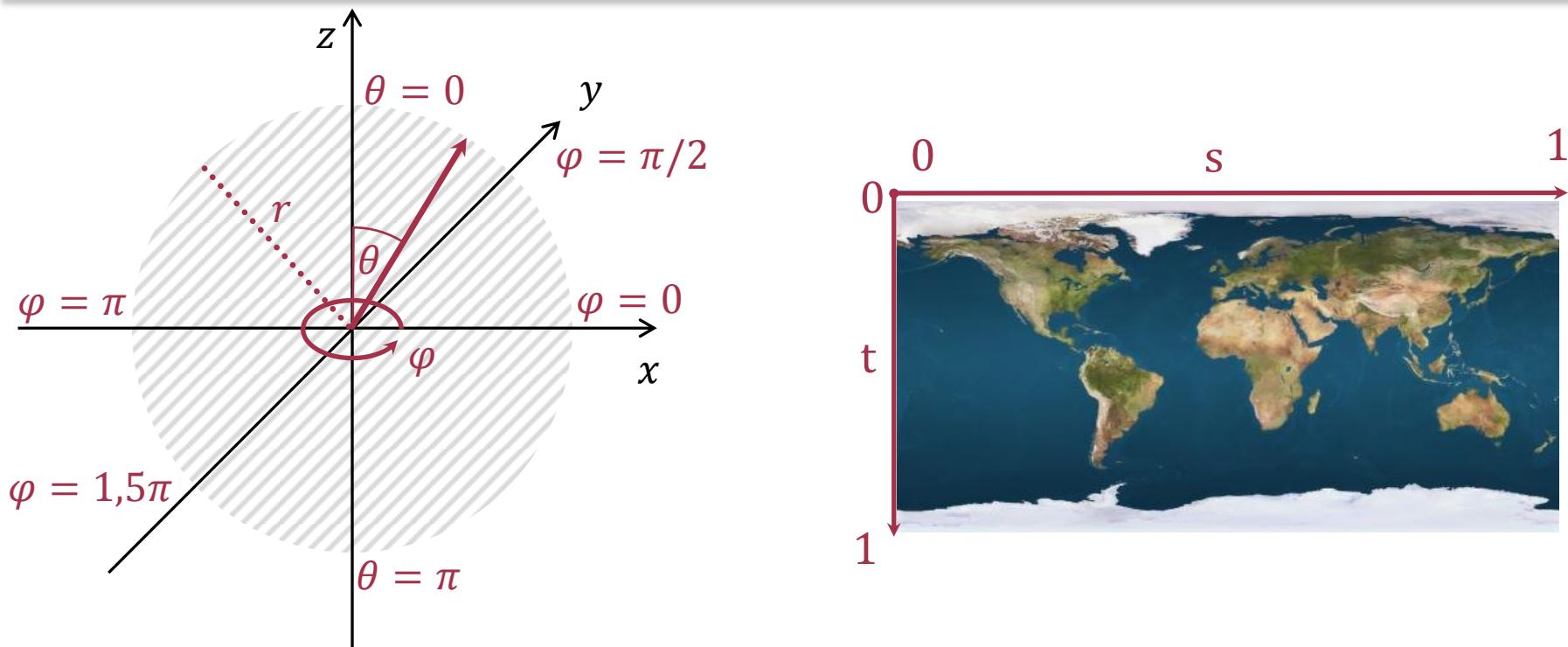
- Auflösung, diskret:  $n \cdot m$
- Koordinaten, kontinuierlich
  - Ursprung, oben links
  - Achsen  $s, t \in [0,1]$

# Texturkoordinaten: Rechteck



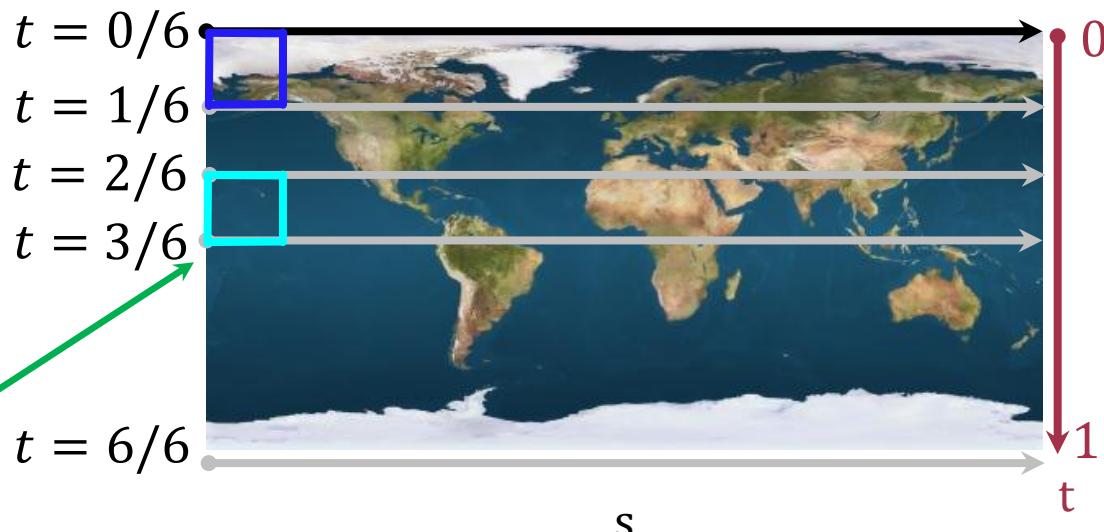
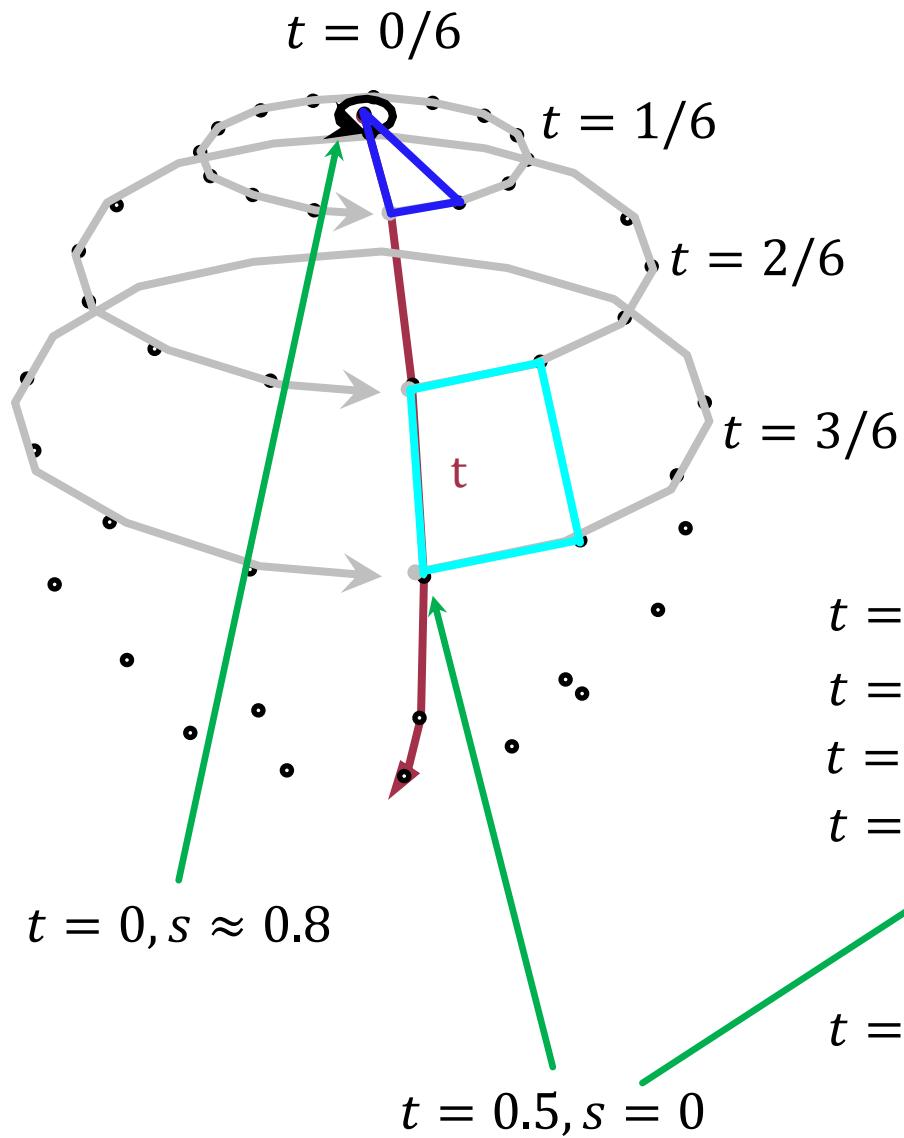
- Gegeben: Rechteck, 4 Vertices, mit Attribut Position
- Ausreichend?
- Nein, zusätzlicher Flächenparameter nötig
- Gibt Position eines Punktes in Relation zur Fläche an
- Texturkoordinate eines Vertex
- Fragment?
  - VS gibt Texturkoordinaten der Vertices aus
  - FS erhält interpolierte Texturkoordinate

# Texturkoordinaten: Kugel I



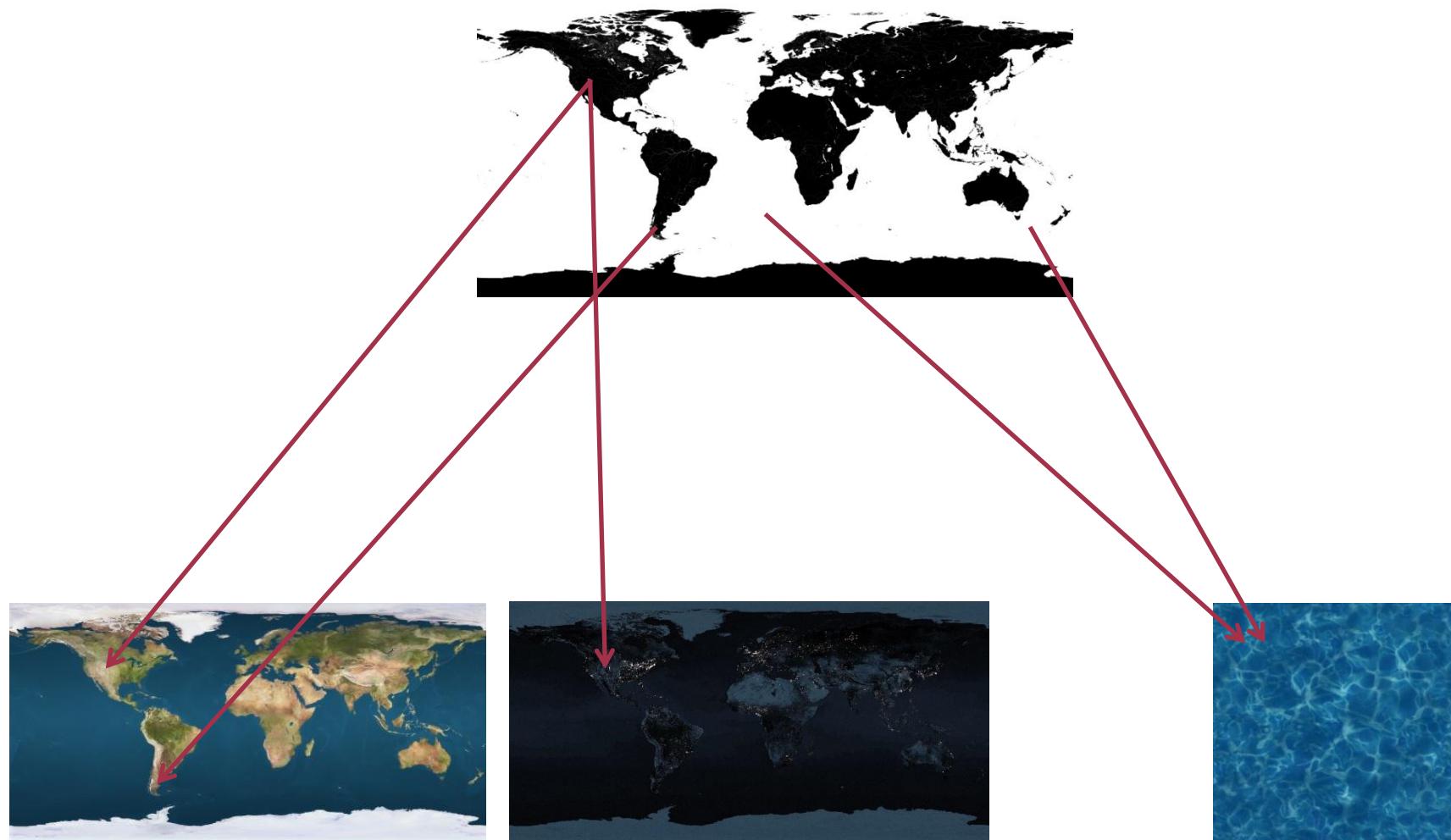
- Berechne Texturkoordinaten aus Kugelkoordinaten gemäß:
- $r$  hat nichts mit relativer Position in Kugeloberfläche zu tun
  - $\frac{\varphi}{2\pi} = s$
  - $\frac{\theta}{\pi} = t$

# Texturkoordinaten: Kugel II



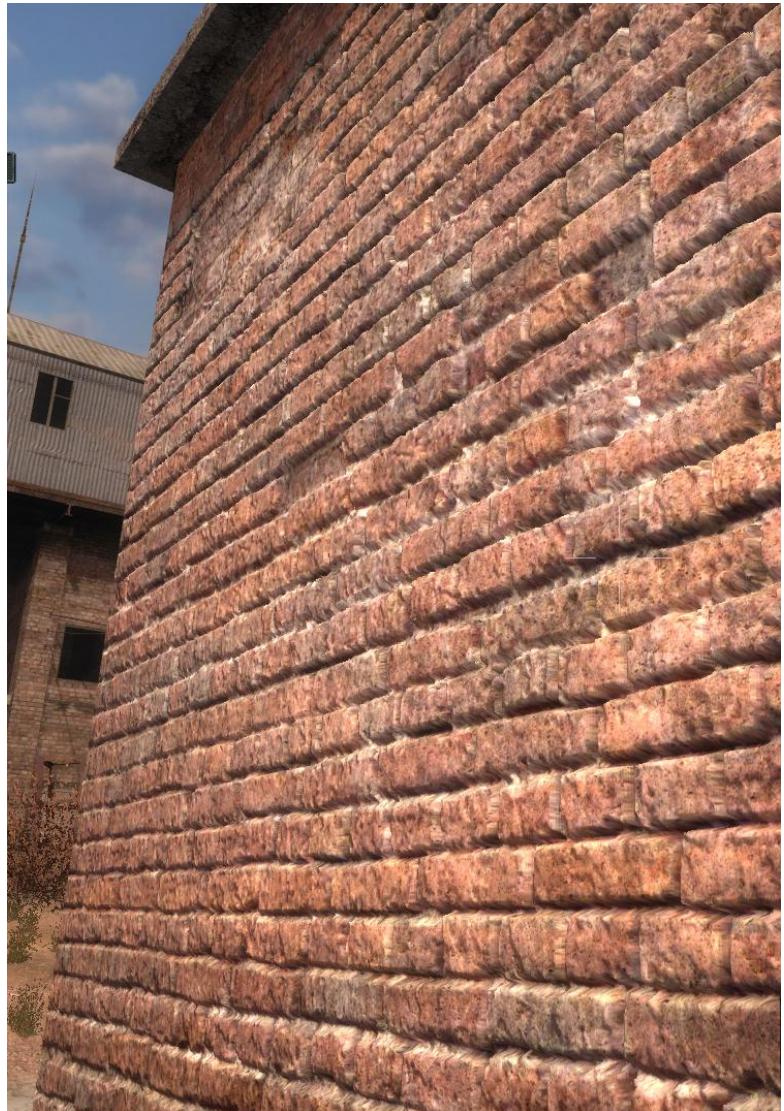
# Beispiel: Wasser oder Land

---



# Beispiele für Oberflächentexturierung

---



# Motivation

---

- Entscheidender Vorteil gegenüber Vertexattributen?
  - Unabhängige Auflösung
  - Wahlfreier Lesezugriff (kann sehr teuer sein)
- Texturen lesen im Fragment Shader...
  - Oft räumlich kohärente Zugriffe  $\Rightarrow$  sehr günstig
  - Texturauflösung oft höher als Geometrieauflösung, kann ausgenutzt werden, wenn Objekt entsprechend nah
- Texturen lesen im Vertex Shader
  - Bei 1:1 Mapping Vertex Attribute sinnvoller
  - Bei Vertexdaten andere Auflösung nicht hilfreich
  - Aber: lesender Zugriff auf Eingangsdaten anderer Vertices
  - Auch sinnvoll bei 1D / 3D Textur Beispielen

## 13.2

---

OpenGL & GLSL Befehle zum Erstellen &  
Verwenden einer Textur

# Zugriff auf 2D-Textur im Shader

```
// Deklaration einer 2D Textur 'colorTex' im Shader
// Texturen immer uniform, werden nicht durch die Pipeline verarbeitet.
// Sampler Typen repräsentieren Texturen im Shader. Muss passend zum
// Texturtyp gewählt werden
uniform sampler2D colorTex;
```

```
// Liefert für Texturkoordinate 'P' interpolierten Wert der Textur 'sampler'
vec4 texture(          // Diese Werte liegen hier zwischen 0 und 1
    sampler2D sampler, // Die 2D-Textur
    vec2 P            // 2D-Texturkoordinate
);
```

```
// FS-Beispiel: Färbe Geometrie mit Farbe aus Textur ein
#version 150 core
uniform sampler2D colorTex;      // Farbtextur, z.B. Erdoberfläche
in vec2 texCrd;                // Texturkoordinate dieses Fragments
out vec4 fragColor;           // Resultierende Fragmentfarbe
void main() {
    // Setzt Farbe des Fragments auf den Wert der Textur für seine Texturecoords
    fragColor = texture(colorTex, texCrd);
}
```

# Erzeugen & Binden

```
// Liefert die id eines, zunächst leeren, Textur Objekts
int glGenTextures()

// Setzt die Textur mit der id 'texture' als aktuell zu bearbeiten-
// des Textur-Objekt des Typs 'target'. Alle Befehle die sich auf
// 'target' beziehen wirken dann auf diese Textur, solange keine
// andere an dieses 'target' angebunden wird.

void glBindTexture(
    int target, // aktuell zu verarbeitender Texturtyp, etwa:
                // GL_TEXTURE_1D
                // GL_TEXTURE_2D (diese Veranstaltung)
                // GL_TEXTURE_3D
                // ...
                // GL_TEXTURE_CUBE_MAP
    int texture // id eines Texture-Objects
);
```

# Texture Unit

---

```
// Aktiviert eine Textureinheit der Grafikkarte zur Verwendung durch
// eine Textur. [Textureinheit - Textur: 1:1 "Beziehung"]
void glActiveTexture (
    int texture // zu aktivierende Textureinheit. GL_TEXTURE0, GL_TEXTURE1, ...
        // Dabei gilt: GL_TEXTURE(n+1) = GL_TEXTUREn + 1
        // Es gibt laut GL-Specification mindestens 80
);
```

# Texturdaten

```
// Übergibt 2D-Daten an die gerade an 'target' angebundene Textur, sowie
// die gerade aktive Textureinheit und beschreibt die Daten.

void glTexImage2D (
    int target, // Konstante für Ziel-Texturtyp, etwa GL_TEXTURE_2D
    int level, // Mipmap-Level, falls kein MM: 0 (hier)
    int internalFormat, // Konstante für Farbkomponentenanzahl, etwa:
                        // GL_RED für 1, GL_RG für 2, GL_RGB für 3, GL_RGBA für 4
                        // Optional: Angaben der Größe und des Datentyps, etwa:
                        // GL_RGBA32F: 4 Komponenten, 32 bit pro Komponente, Float
    int width, // Texturbreite in Texeln
    int height, // Texturhöhe in Texeln
    int border, // Bedeutungslos, muss 0 sein
    int format, // Konstante für Anordnung der Ausgangs Farbkomponenten.
                // Muss zu internalFormat passen. Für iF GL_RGB etwa:
                // GL_RGB, GL_BGR. Wird zu RGB umsortiert
    int type, // Konstante für Datentyp der Ausgangsdaten, etwa:
              // GL_BYTE, GL_INT, GL_FLOAT
    <Float>Buffer data // von java.nio.Buffer abgeleitete Klasse mit
                       // den Daten. Müssen bei einer 2D-Textur zeilenweise vorliegen
);
```

# Beispiel

```
int texId = glGenTextures();           // Erzeuge leere Textur
glBindTexture(GL_TEXTURE_2D, texId);   // setze 'texId' als aktuell zu
                                         // bearbeitende 2D-Textur
glActiveTexture(GL_TEXTURE7);         // Aktiviere Textureinheit 7.
                                         // Nachfolgende Befehle beziehen sich dann darauf.

// Übergibt 2D-Daten an texId bzw. die Textureinheit GL_TEXTURE7
void glTexImage2D(
    GL_TEXTURE_2D, // Manipuliere aktuelle 2D-Textur
    0,             // Mipmaplevel 0, diese Veranstaltung immer
    GL_RGB8,       // Internes Format:
                  // 3-Komponenten
                  // 8 Bit pro Komponente Genauigkeit
    8192, 4096    // Auflösung: 8192 x 4096
    0,             // Immer 0, toter Parameter.
    GL_RGB,        // Ausgangsdaten liegen im rgb Format...
    GL_FLOAT       // ...und als Float-Datentyp vor
    <daten>        // Die Daten selbst. Einlesen etc. -> Übung
);
// Binde an sampler2D "colorTex", Folie 14, im aktiven Program Object.
// Binde 7 für Textureinheit GL_TEXTURE7 an. Selbst achten: n - GL_TEXTUREn
glUniform1i(getUniformLocation("colorTex"), 7 );
```

## 13.3

---

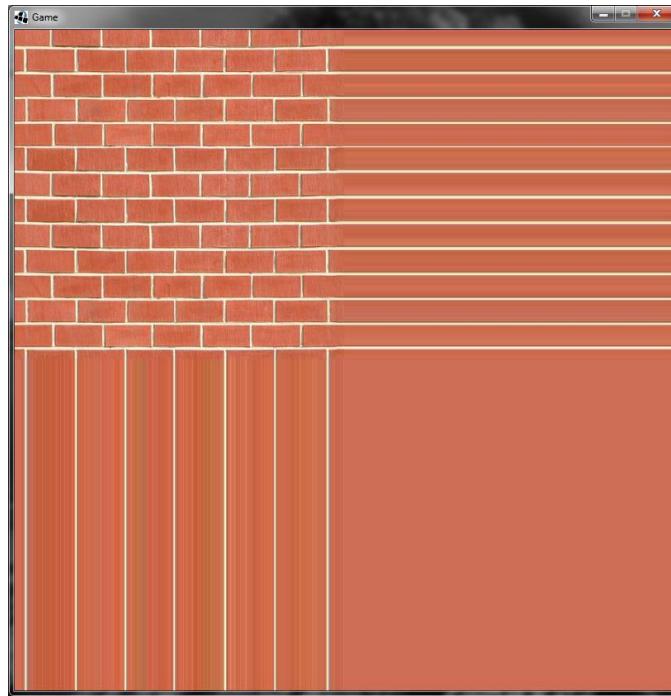
Funktionen zum Zugriff auf die Textur

# Verhalten an den Rändern

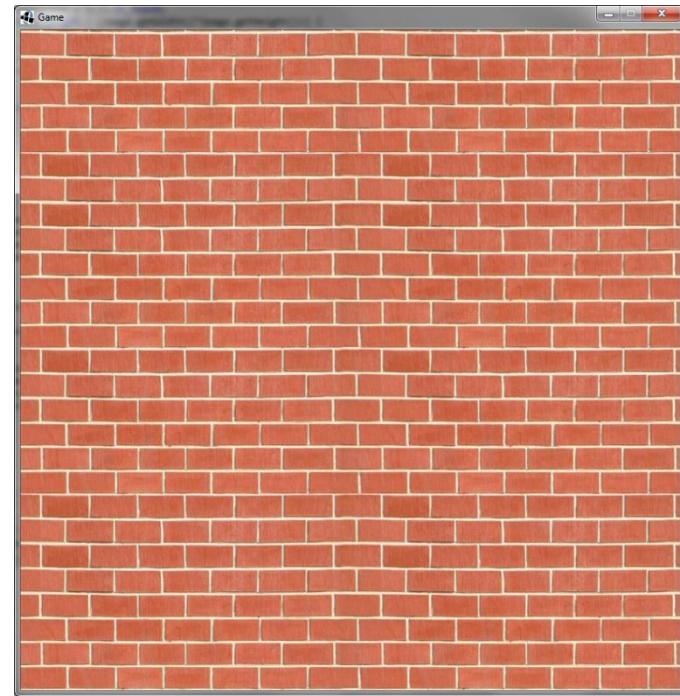
➤ Gegeben Viereck mit Texturkoordinaten s,t:

- $s, t \in [0, 2]$
- Textur muss für GL\_REPEAT ‘tileable’ sein

GL\_CLAMP\_TO\_EDGE



GL\_REPEAT



# Setzen der OpenGL Texturparameter

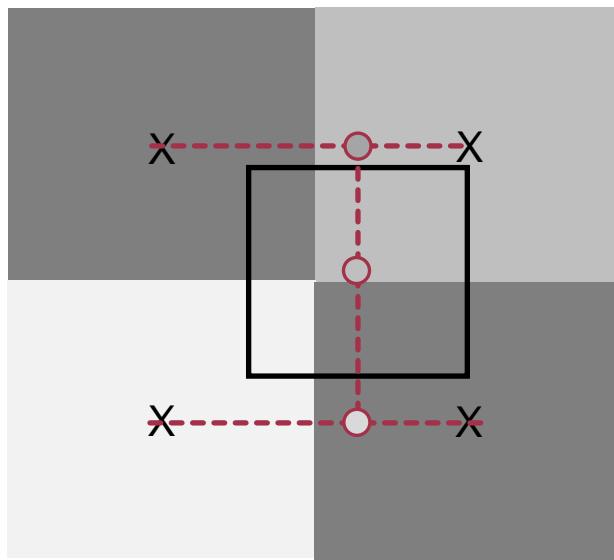
```
// Setzt verschiedene Parameter für die gerade an 'target' angebun-
// dene Textur und die gerade aktive Textureinheit.

void glTexParameterI(
    int target, // Konstante für Texturtyp, etwa GL_TEXTURE_2D
    int pname,  // Konstante für den zu settenden Parameter, etwa:
                //     GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T,
                //     GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER
                // ...
    int param   // Konstante für Parameterwert, etwa:
                //     GL_REPEAT, GL_CLAMP_TO_EDGE für GL_TEXTURE_WRAP_S
);
```

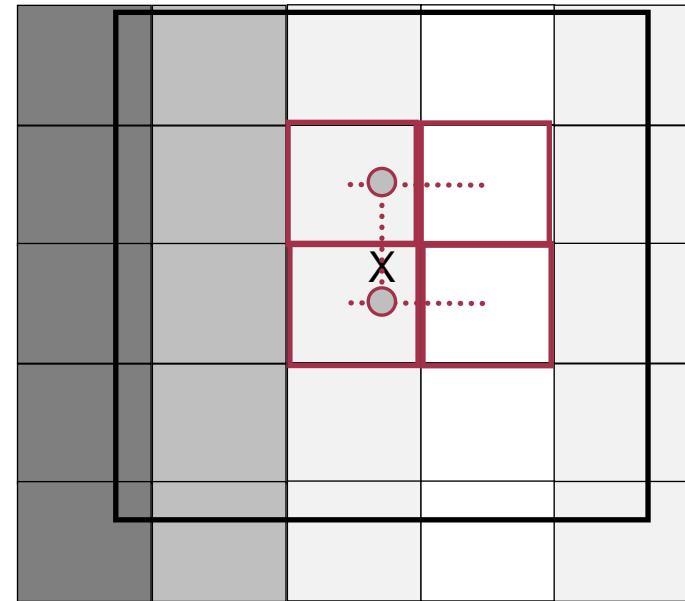
```
// Beispiel: Textur in s & t Richtung wiederholen lassen
glTexParameterI(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterI(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

# Bilinear Filter

Magnification



Minification



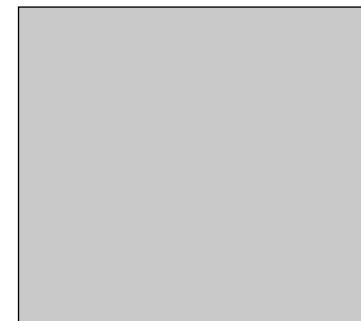
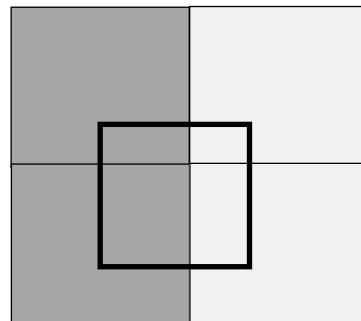
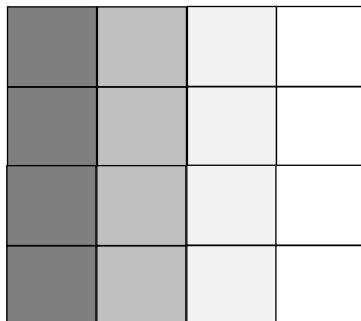
## ➤ Probleme bei Minification?

- Rauschen, da “zufällig” Werte herausgepickt werden und
- Andere Werte sobald sich Kamera bewegt
- Ungünstige Speicherzugriffsmuster

# Mipmapping / Trilinear Filter

---

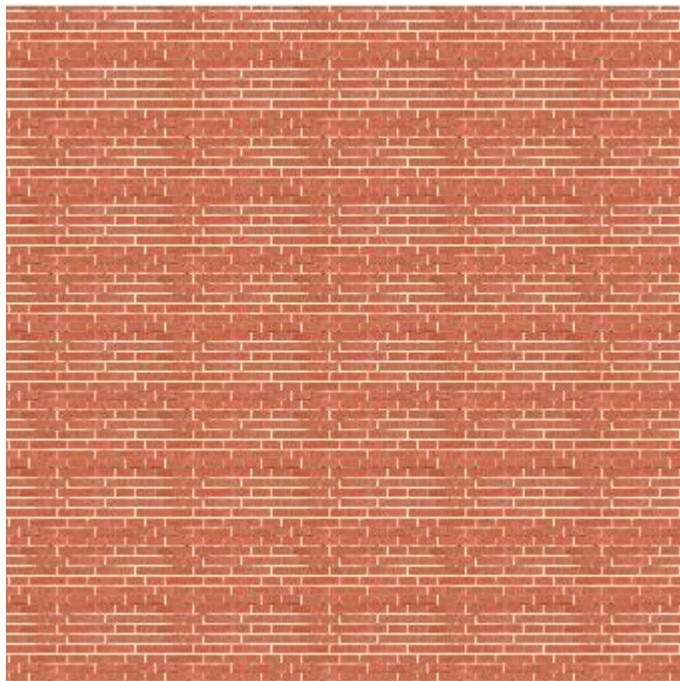
- Erzeuge zusätzliche Textur-Level mit jeweils  $\frac{1}{4}$  Auflösung (hier: Box-Filter, low Quality)
- Lese aus Level, in welchem Pixel gerade etwas kleiner als Texel ist
- Bewirkt sichtbaren Auflösungswechsel, daher zusätzliche Interpolation zwischen zwei Leveln: Trilinear Filter
- Problem: Vor allem schräge Texturen sehr unscharf



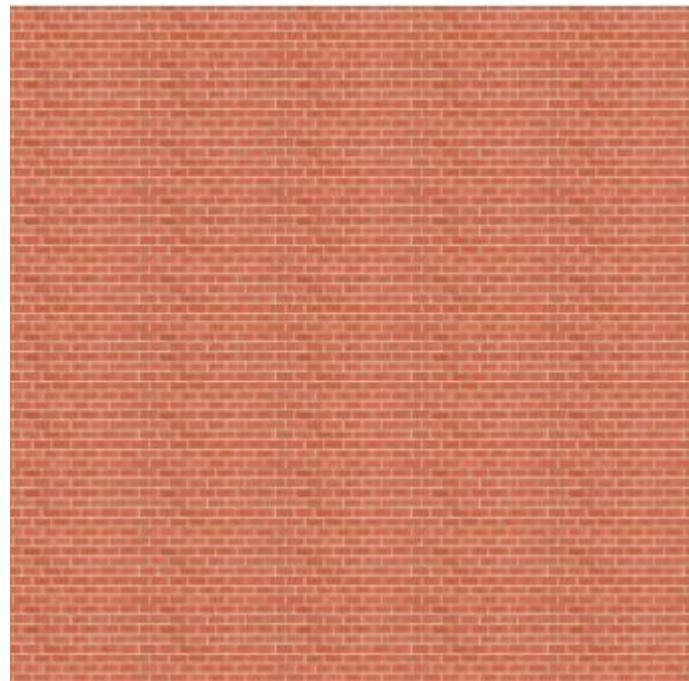
# Demonstration

---

Linear gefiltert



Trilinear gefiltert



# Filter einstellen

---

```
// Beispiel: Verwende zum Vergrößern & Verkleinern Bi-Linearen Filter
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// Mipmapping Beispiel
// Lasse OpenGL Mipmaps erzeugen. Dafür muss Textur an target
// GL_TEXTURE_2D angebunden, mit Daten versehen und texUnit aktiv sein
glGenerateMipmap(GL_TEXTURE_2D);

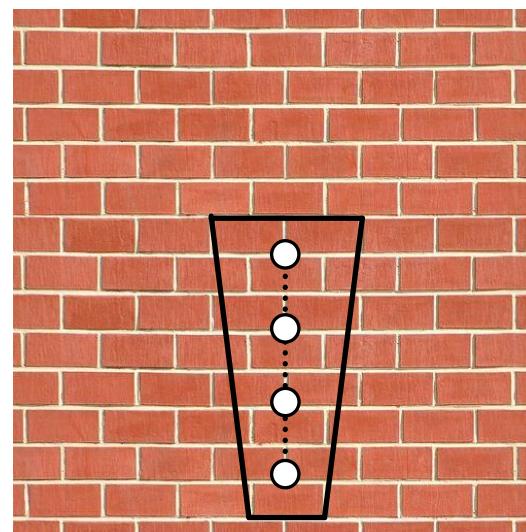
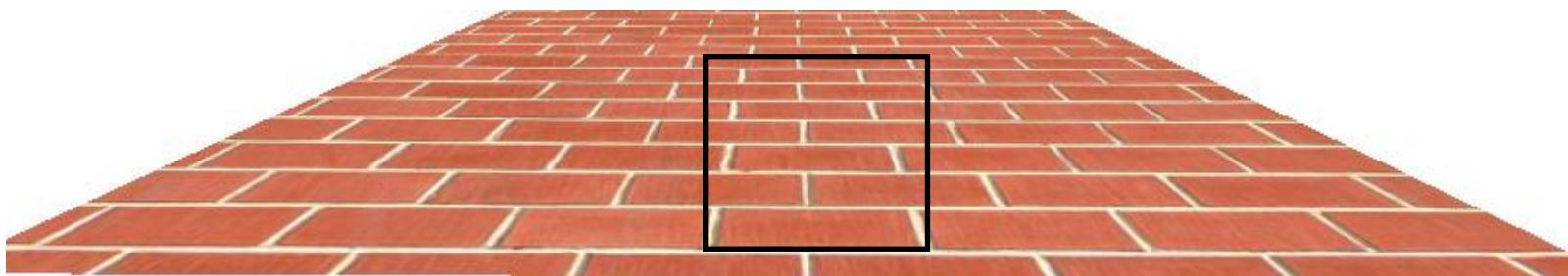
// Aktiviere Tri-Linearen Filter für Verkleinerung
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);

// Aktiviere Bi-Linearen Filter für Vergrößerung
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

# Anisotropischer Filter

---

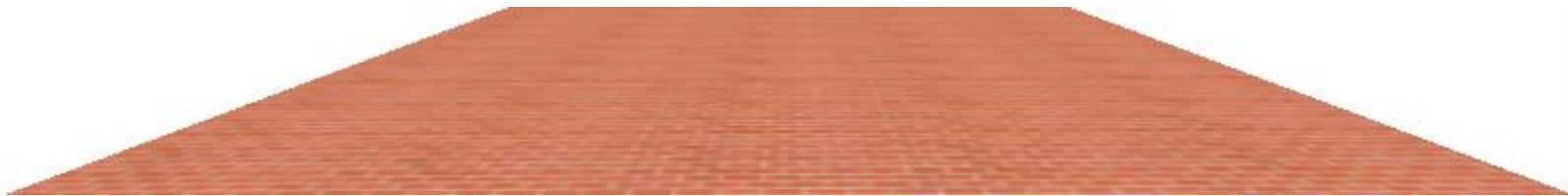
- Projiziere Fragment in Textur
- Wähle längere Richtung aus
- Nimm entlang dieser Strecke mehrere Samples



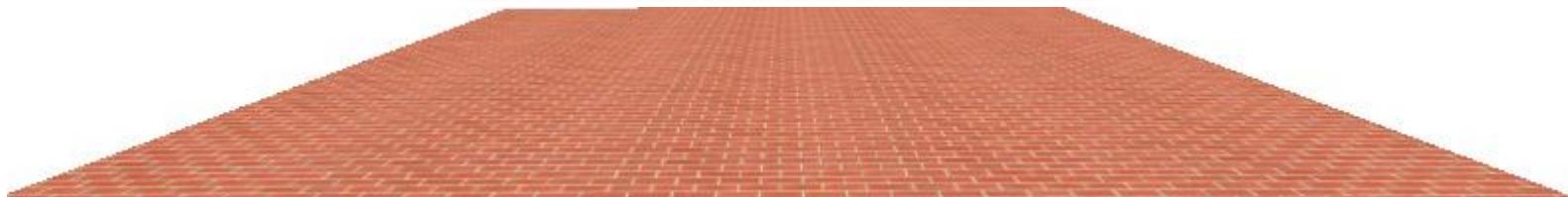
# Demonstration

---

Kein AF



AF, 16 Samples



## 13.4

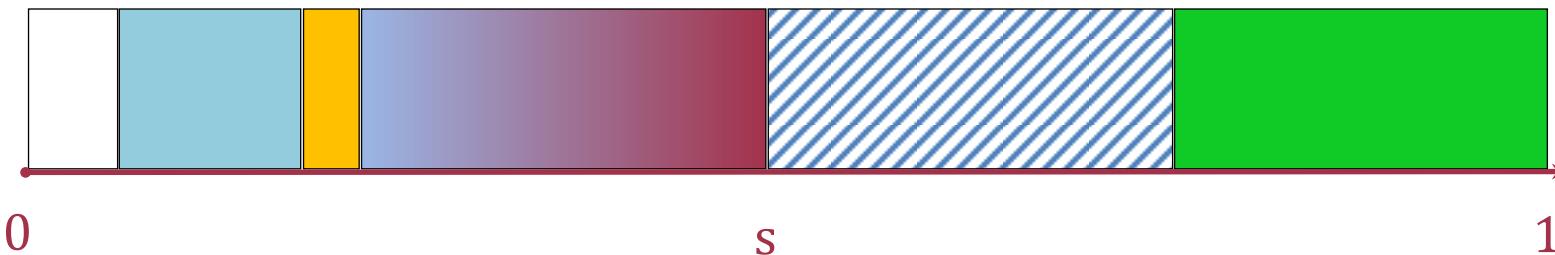
---

### 1D-, 3D- und prozedurale Texturen

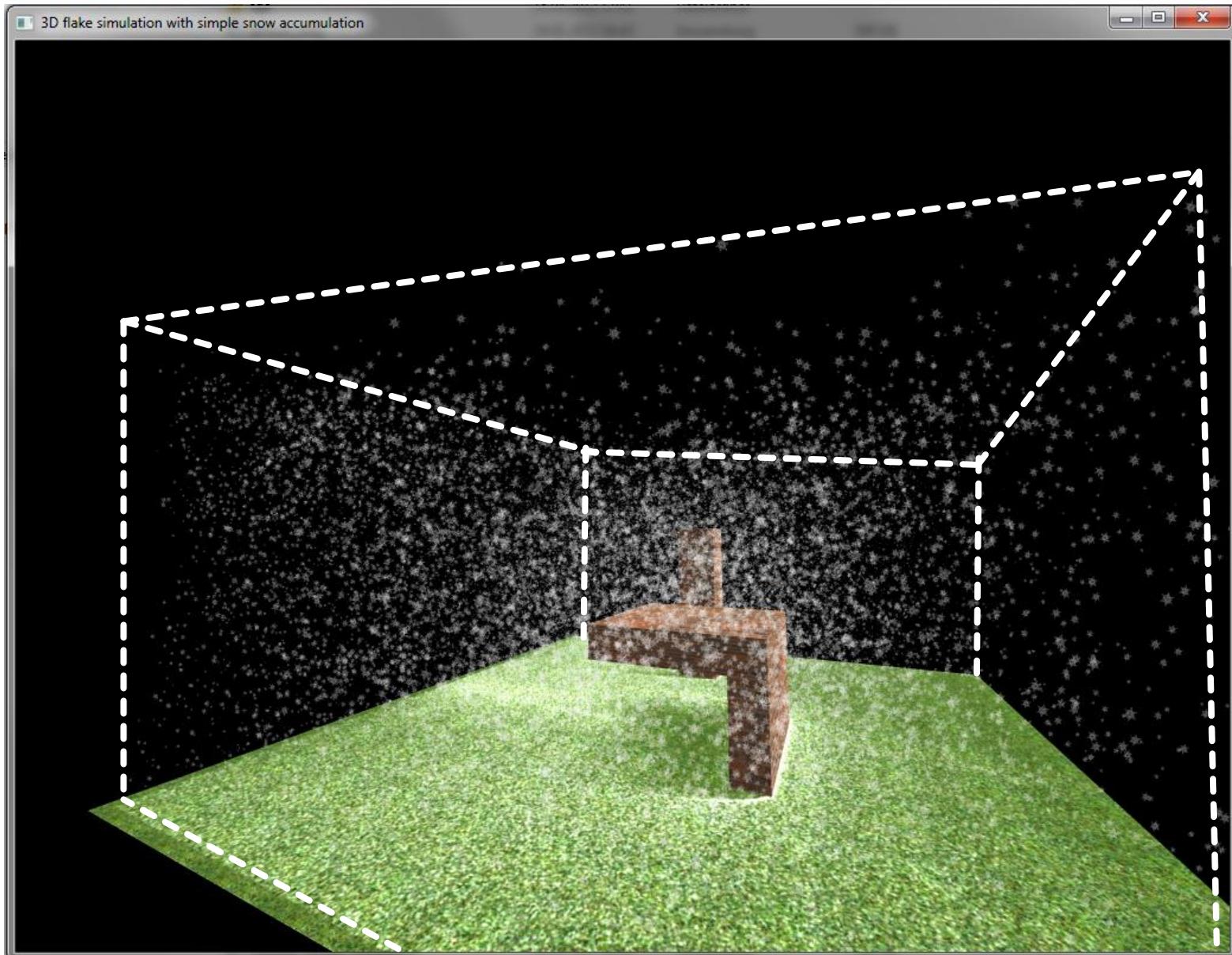
# 1D-Texturen

---

- Können z.B. vorberechnete Werte einer aufwändigen Funktion einer Variablen enthalten
- Beispiel: Bilde Variable auf diskrete unterschiedlich große Intervalle ab. Verwende zum Lesen `GL_NEAREST`
- Anwendung: Visualisierung, Klassifizierung einer Größe
- Alternative Fallunterscheidungen: Können extrem teuer sein
- 2 oder 3 Parameter?

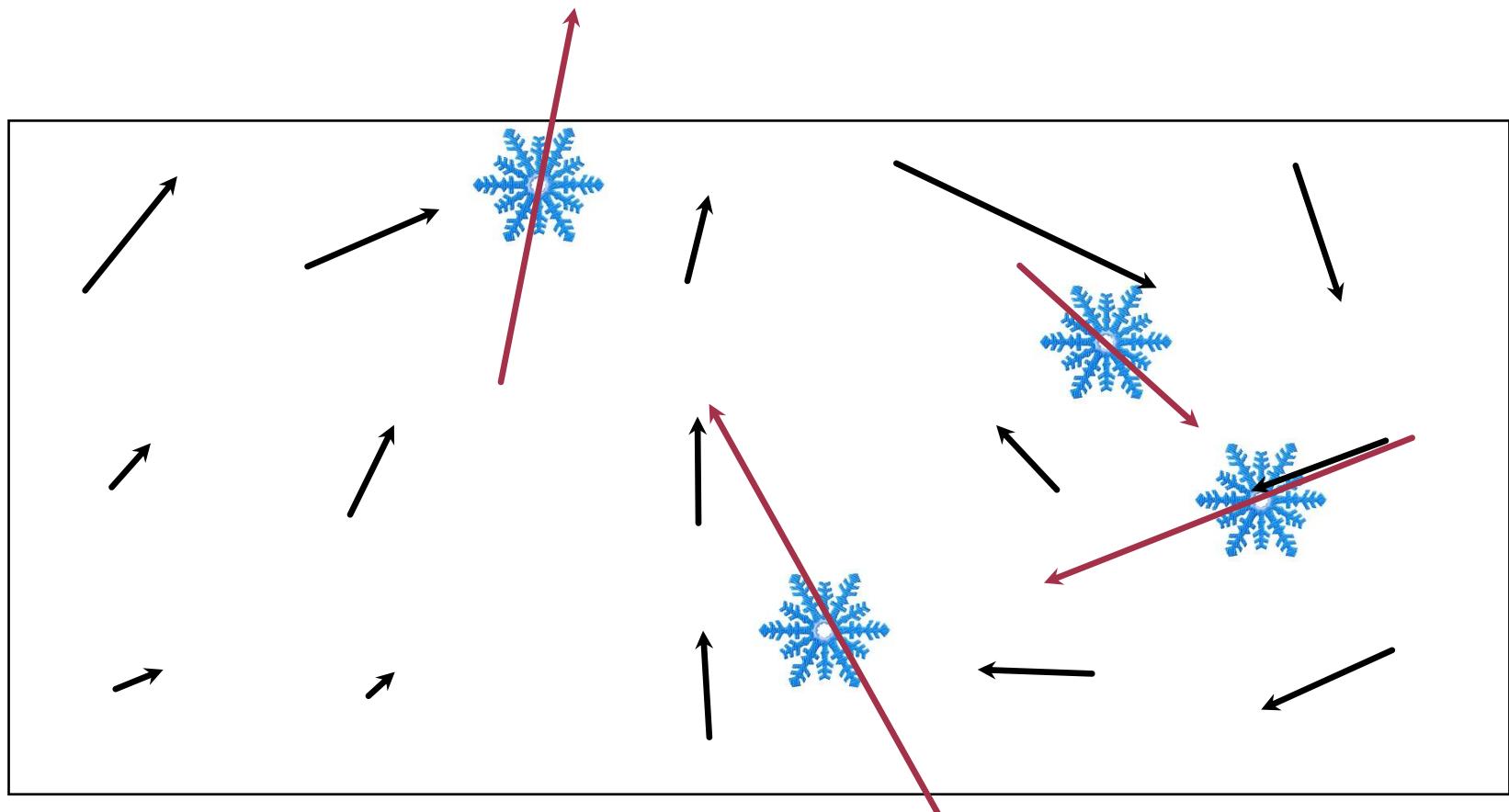


# 3D-Texturen I



# 3D-Texturen II

- Ausgang: Ergebnisse der Echtzeitströmungssimulation von Philipp Middendorf mit OpenCL
- 3D Raster von Vektoren, die Kräfte an diesem Ort darstellen



# Procedural Texturing

---

