Computergraphik

Vorlesung im Wintersemester 2013/14
Kapitel 7: Modernes OpenGL

Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie

Agenda

- Evaluation
- Übungsblätter 10 und 11
- Übungsblatt 12
  - Texturezugriffe
  - Heightmap
  - Gradienten
  - Bumpmap
  - Blending

Evaluation

Evaluation

- Framework ist zu unklar und zu wenig dokumentiert
- Die Rückmeldungen der Testcases sind nicht hilfreich
- In der Übung gibt es zu wenig Feedback zu alten Übungsblättern
- Die Übungsaufgaben sind unterspezifiziert

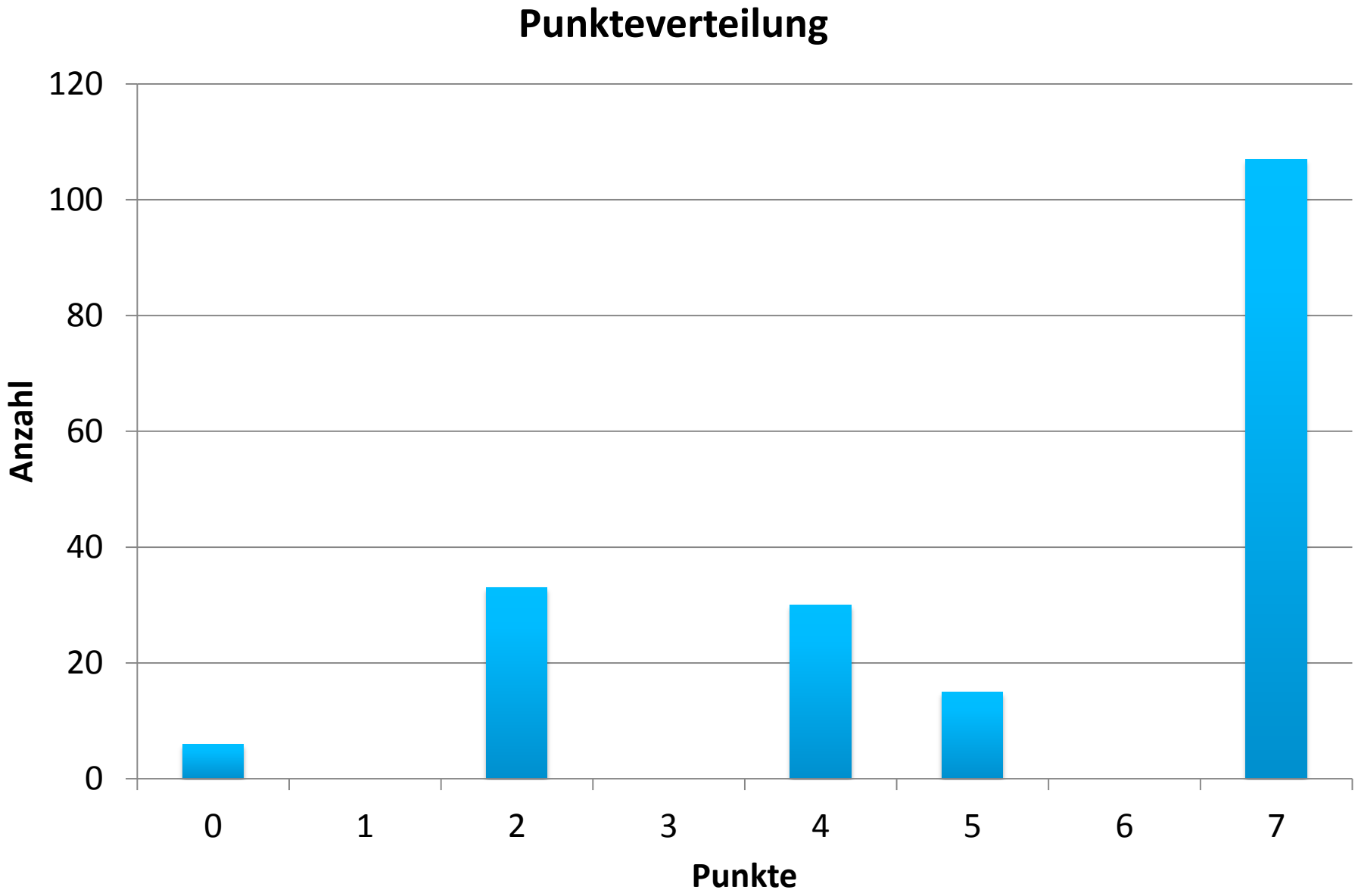
HiWi

Wer hat Lust uns zu helfen?
HiWi für Computergraphik-Übungen gesucht!

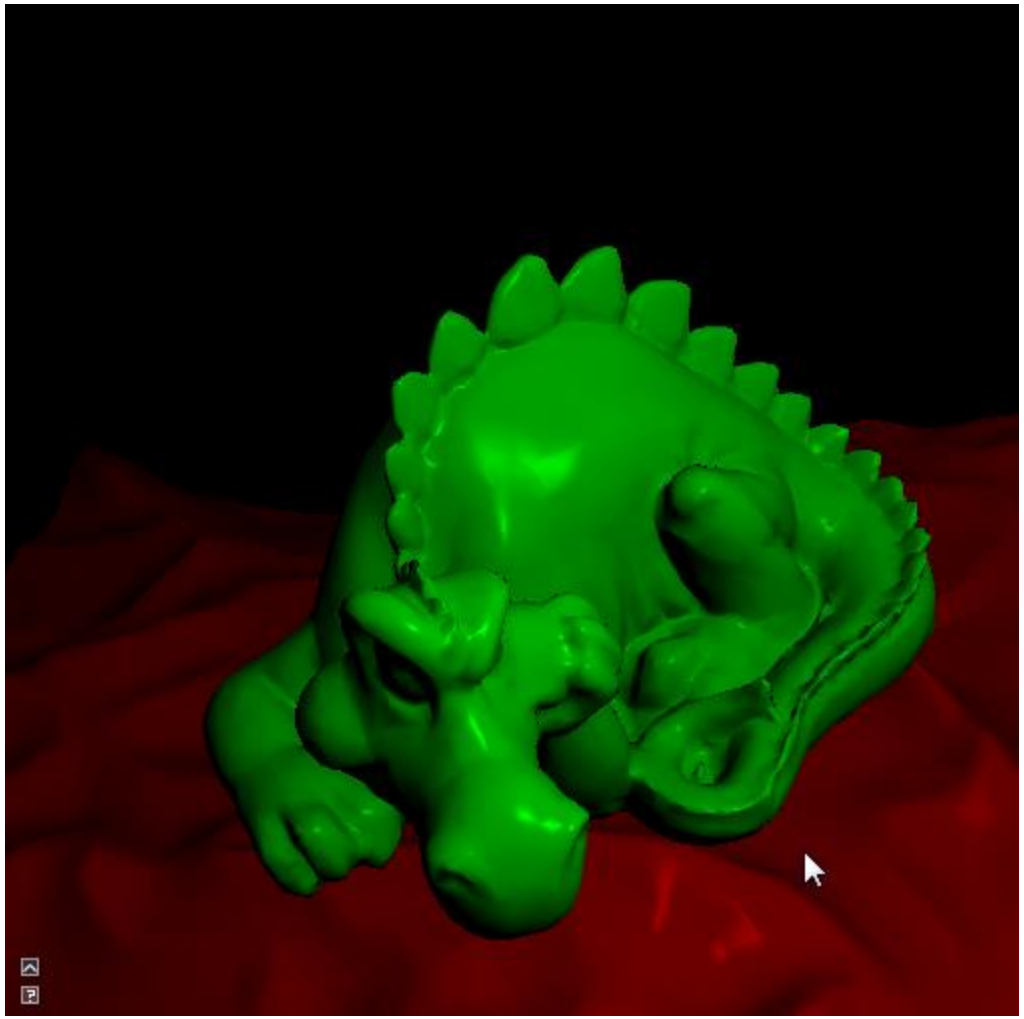
C++-Kenntnisse
Computergrafik

→ Mail an einen Übungsleiter!

Übungsblatt 10



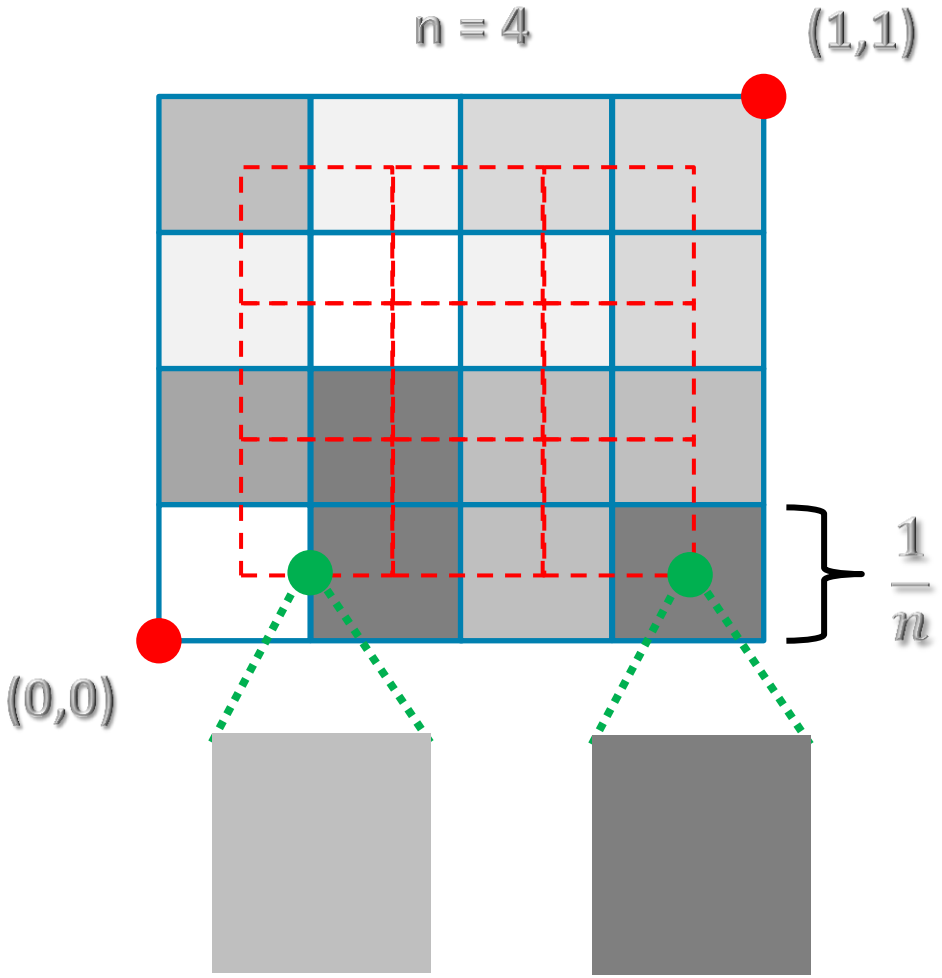
Übungsblatt 11 - Musterlösung



Übungsblatt 12

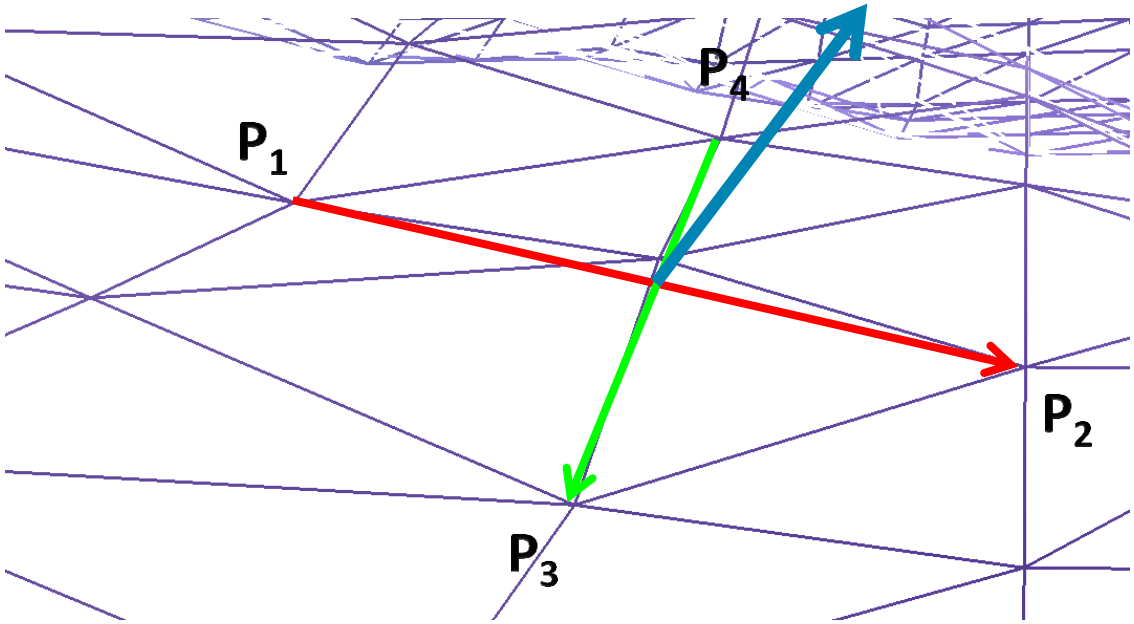
Heightmap

- Höhenwerte in Textur
- Texturzugriff auf Mitte des Texels, ansonsten gefilterte Werte
- Texel-Werte in [0;1] bei Standard-Texturen (vs. Float;Integer-Texturen)



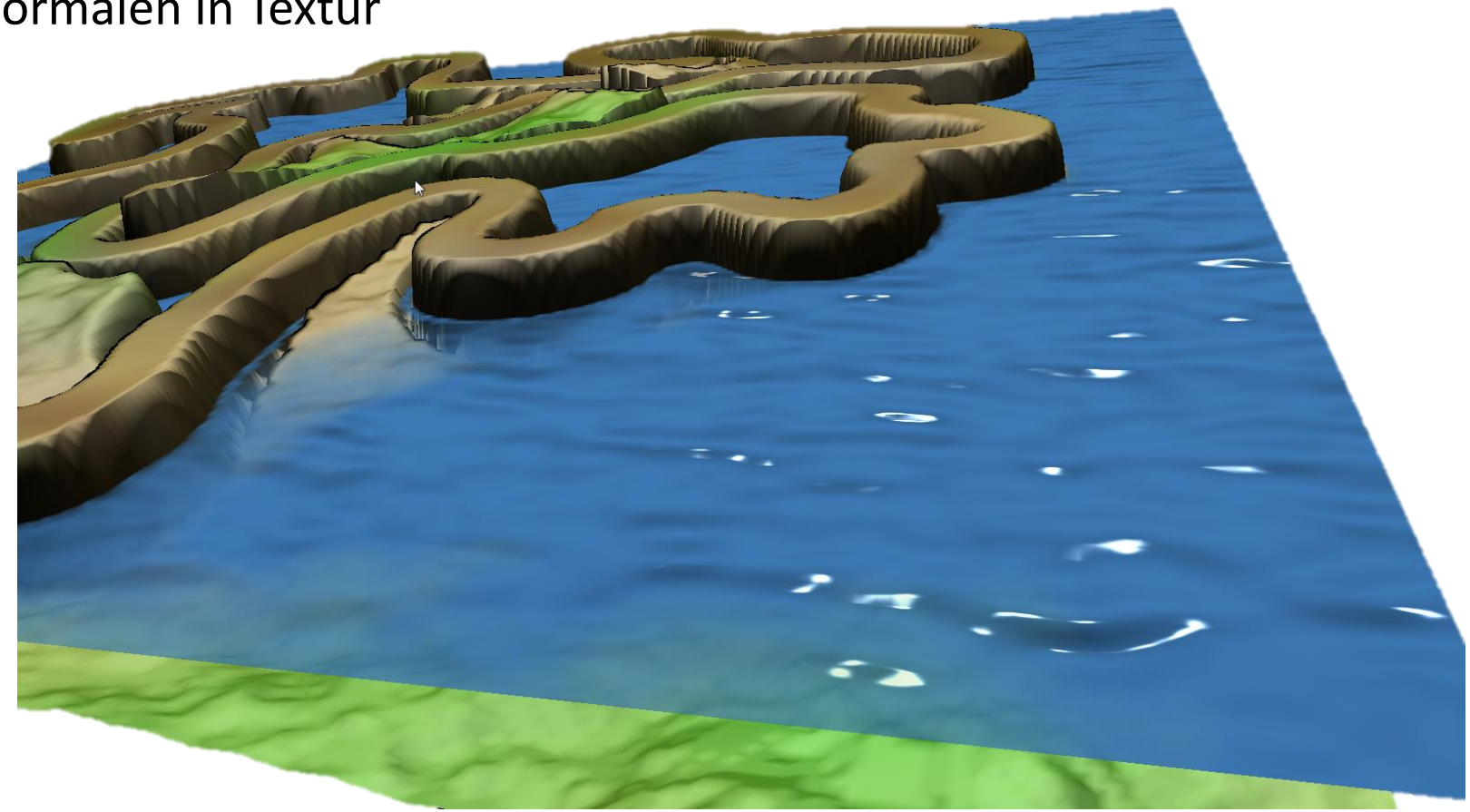
Normalen bei Höhenfeldern

- Aufstellen von Gradienten durch diskrete Differenzen in zwei Richtungen (z.B. u und v)
- Bestimmung der Normale durch Kreuzprodukt
- Richtung beachten



Bump-Mapping

- Analoges Vorgehen um Normalenvektor aus Bump-Map zu bestimmen
- Verwendung als Shading-Normale, z.B. Phong-Modell
- Geometrie wird nicht beeinflusst
- Heute eher: Normal-Mapping: explizite Speicherung der Shading-Normalen in Textur



Texturzugriffe in GLSL

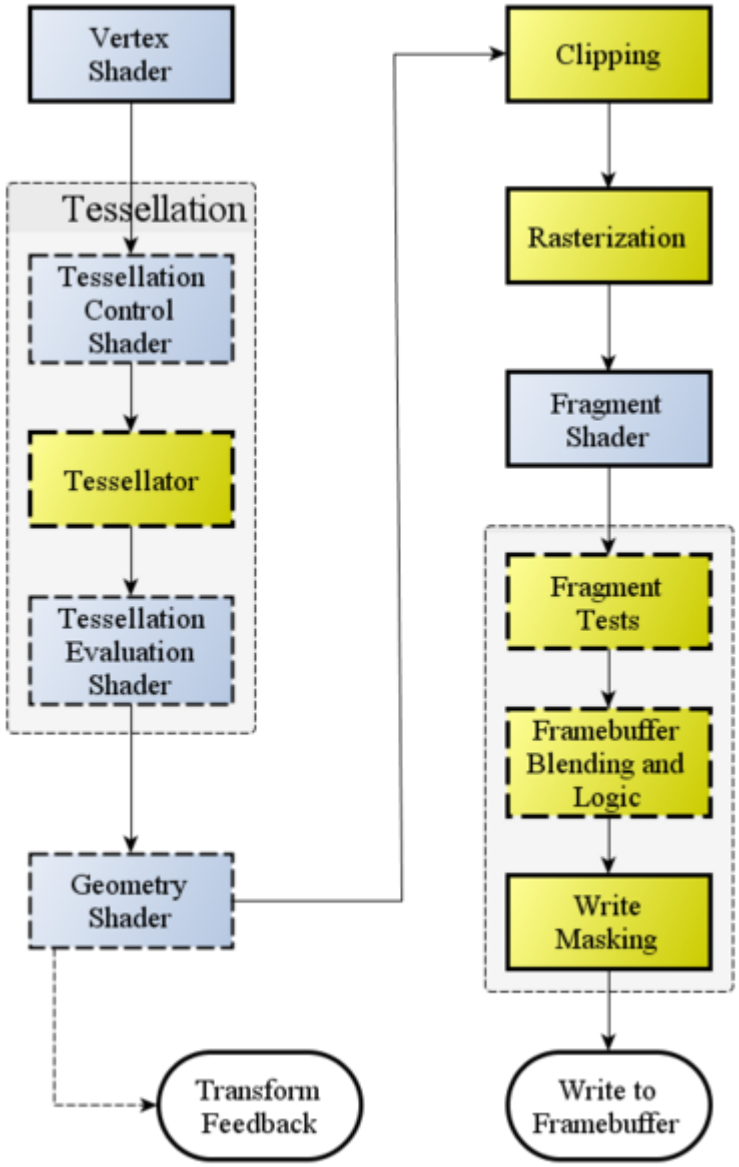
- Zugriff über ‚sampler'-Uniforms
  - Repräsentiert eine Textureinheit
  - Filtert die Textur!
  - **`sampler1D, sampler2D, sampler3D`**
- Aus dem Hauptprogramm als Uniform int setzen mit **`glUniform1i`**
- Der Typ des Samplers muss zum Typ der Textur passen
- vgl. auch **`Sampler Objects`** in OpenGL

Texturzugriffe in GLSL

- Zugriffsfunktionen
  - Textur-Koordinaten in [0, 1]^2
  - Andere Werte sind zulässig (clamp/repeat)
  - **`texture`**: Normaler Zugriff
  - **`textureProj`**: Texturkoordinate wird durch .w geteilt
  - **`textureLod`**: Expliziter Zugriff auf MipMap-Stufe
  - **`textureOffset`**: Zusätzliche Verschiebung *in Texels*
  - **`textureGrad:`** Explizite Gradienten der Texturkoordinaten (für MipMap-Level)
  - Kombinationen…

Alternativ: Hardware-Tesselation

- Verfügbar ab OpenGL 4
- Zwei neue Shader:
  - Tesselation Control
    - Arbeitet auf Kontrollpunkten
    - bestimmt den Grad der Unterteilung
  - Tesselation Evaluation
    bestimmt die Position der erzeugten Vertices; z.B. Displacement
- Der Tesselator selbst ist **nicht** programmierbar
- Vorteil: Es müssen nur wenige Daten an die GPU übertragen, unabhängig vom Grad der Unterteilung



Blending

- Zeichnen von transparenten Objekten in OpenGL
  - Blending einschalten: **`glEnable(GL_BLEND)`**
  - Blending-Operation setzen:
    - **`glBlendEquation(GL_ADD)`**
    - **`glBlendFunc(GL_SRC_APLHA, GL_ONE_MINUS_SRC_ALPHA)`**
  - Alpha-Werte als Vertex-Attribut oder durch Shader bestimmt in **`gl_FragColor`** ausgeben

Blending Aufgabe

Eine Szene mit opaken und semitransparenten Dreiecksnetzen soll möglichst **effizient** gezeichnet werden. Dabei soll ein **nicht-kommutativer** Blending-Operator verwendet und ein möglichst korrektes Resultat erzielt werden.

Der Tiefentest ist bei Eintritt in diesen Programmteil **angeschaltet**; die anderen OpenGL-Zustände sind **nicht definiert**.

Zur Verfügung stehen:

- (1) **glDepthMask(GL_TRUE);**
- (2) **glDepthMask(GL_FALSE);**
- (3) **glEnable(GL_DEPTH_TEST);**
- (4) **glDisable(GL_DEPTH_TEST);**
- (5) **glEnable(GL_BLEND);**
- (6) **glDisable(GL_BLEND);**
- (7) **drawTrans();** zeichnet alle transparenten Dreiecke in unbestimmter Reihenfolge.
- (8) **drawTransSorted();** sortiert alle transparenten Dreiecksnetze und zeichnet sie in der Reihenfolge von hinten nach vorne.
- (9) **drawOpaque();** zeichnet alle opaken Dreiecke in unbestimmter Reihenfolge.
- (10) **drawOpaqueSorted();** sortiert alle opaken Dreiecksnetze und zeichnet sie in der Reihenfolge von hinten nach vorne.