Praktikum 7: Beleuchtung

Kopieren Sie passenden Code-Dateien aus den letzten Praktika in den des aktuellen Praktikums.

Aufgabe 1 Blinn-Phong Beleuchungsmodell

Implementieren Sie die Methode `BlinnPhongLighting.shade`, welche die Farbe nach dem Beleuchtungsmodell von Blinn-Phong berechnet. Die Produkte aus Lichtfarbe und Materialkoeffizienten sind in den statischen Variablen `diffuseColor` und `specularColor` bereits hinterlegt. Beachten Sie auch die Kommentare der statischen Member-Variablen und die Kommentare in der Methode.

Beachten Sie weiter, dass sich die Kamera an der Stelle $\vec{0} = [0,\ 0,\ 0]^\top$ befindet! Nutzen Sie zur Berechnung die Java-Methoden `Math.sqrt` und `Math.pow`.

Aufgabe 2 Flat-Shading

Stellen Sie eine Methode

```
public static final
void drawTriangleFlat(int[] framebuffer, float[]depthBuffer,
                      int w, int h,
                      int2 pointA, int2 pointB, int2 pointC,
                      float zA, float zB, float zC,
                      int c)
```

bereit, welche ein Dreieck mit konstanter Farbe ‘c“ zeichnet.

Zeichen Sie Ihr Modell mit konstanter Farbe. Berechnen Sie die Farbe für jedes Dreieck mit dem Blinn-Phong-Beleuchtungsmodell! Transformieren Sie dazu eine beliebige Vertex-Position jedes Dreiecks und dessen Normale in den View-Space! Die Normalen pro Vertex sind in dem Array `SimpleMeshModelIO.normals` hinterlegt.

Da unsere Matrix $\mathbf{V}$, welche in den View-Space transformiert *orthogonal* ist, können die Normalen mit der Matrix $\mathbf{V}^{-\top} = \mathbf{V}$ transformiert werden. Da bei Normalen die homogene Kompente $w = 0$ ist, bietet es sich an, eine Methode

```
/**
 * Computes and returns the matrix-vector multiplication of
 * this * [rhs.x, rhs.y, rhs.z, 0]^T, i.e., assuming that
 * - the 4th component of rhs is 0.
 * - setting the last row of this matrix to [0, 0, 0, 0]
```

```
 6   * @param rhs float3 the x, y, z component of as right-hand
 7   *                   side of the product.
 8   * @return float3 The x, y, z component matrix-vector product
 9   *                this * [rhs.x, rhs.y, rhs.z, 0]^T.
10   */
11  public float3 mul_W0(float3 rhs)
```

in `mat4.java` zu implementieren.

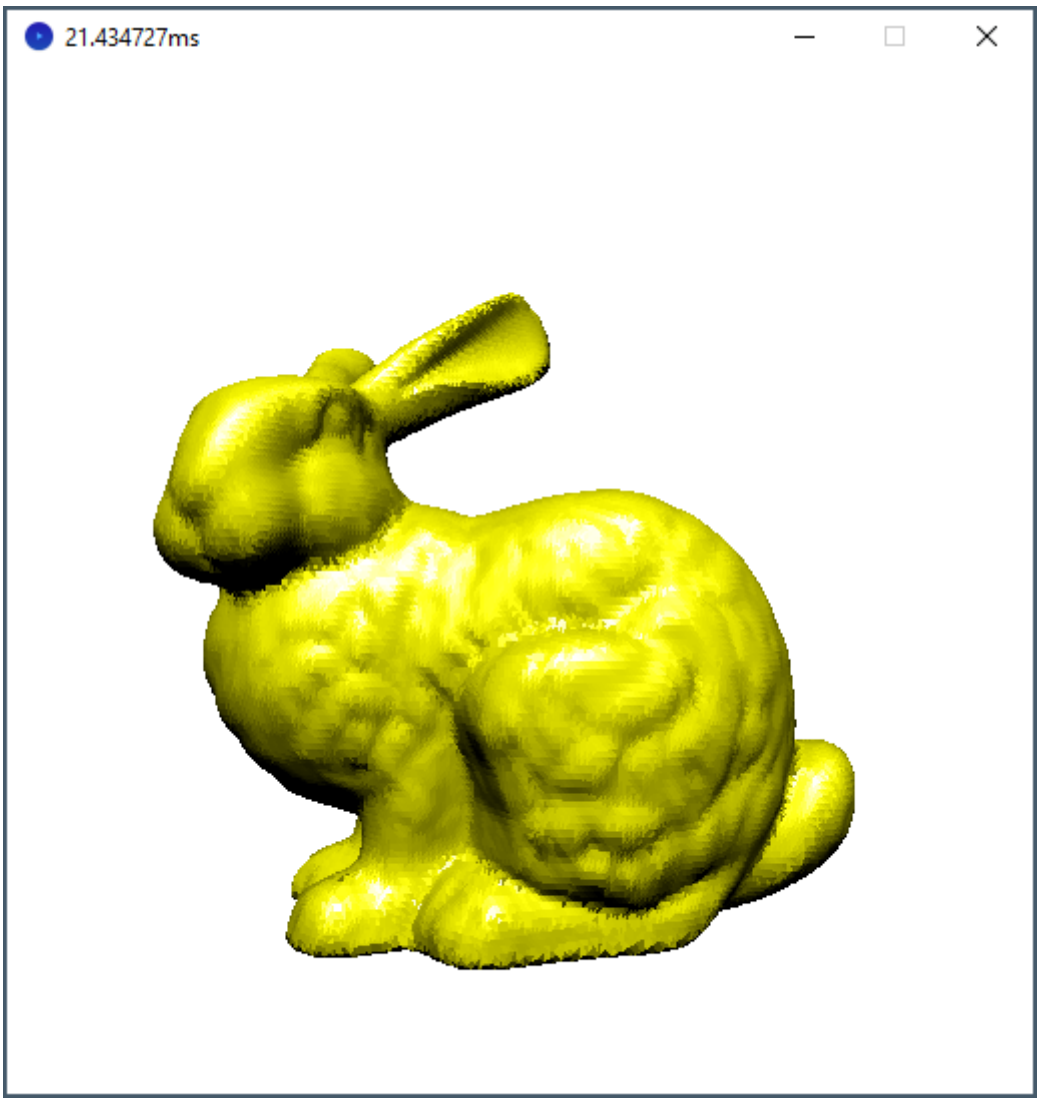

Abbildung 1: Hase im Flatshading

Aufgabe 3 Gouraud Shading

Stellen Sie eine Methode

```
public static final
void drawTriangleGouraud(int[] framebuffer,
                         float[] depthBuffer, int w, int h,
                        int2 pointA, int2 pointB, int2 pointC,
                        float zA, float zB, float zC,
                        float3 cA, float3 cB, float3 cC)
```

welche die Farben `cA`, `cB` und `cC` baryzentrisch über das Dreieck interpoliert, bereit.

Berechnen Sie nun für jeden Vertex die Farbe nach dem Modell von Blinn-Phong und interpolieren Sie die Farben über jedes Dreieck.

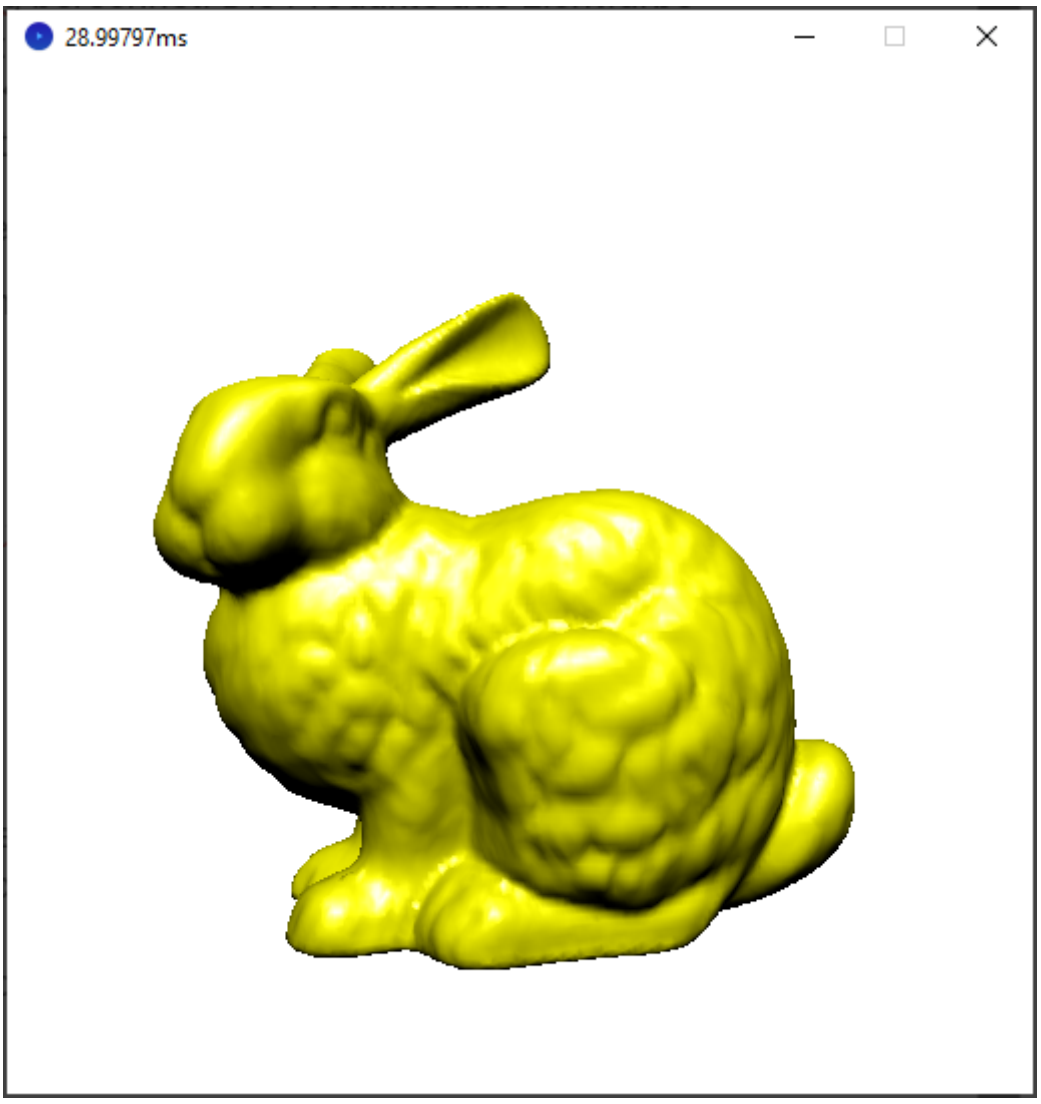

Abbildung 2: Hase mit Gouraud Shading

Aufgabe 4 Phong Shading

Stellen Sie eine Methode

```
public static final
void drawTriangle_Phong(int[] framebuffer,
                        float[] depthBuffer, int w, int h,
                        int2 pointA, int2 pointB, int2 pointC,
                        float zA, float zB, float zC,
                        float3 nA, float3 nB, float3 nC,
                        float3 vA, float3 vB, float3 vC)
```

bereit. Diese soll die View-Space Position `vA`, `vB` und `vC`, sowie die Normalen `nA`, `nB` und `nC` über das Dreieck interpolieren. Für jede interpolierte View-Space Position und View-Space Normale soll die Farbe nach dem Beleuchtungsmodell von Blinn-Phong berechnet werden. Rufen Sie die Methode für jedes Dreieck aus und bewundern Sie das Ergebnis.

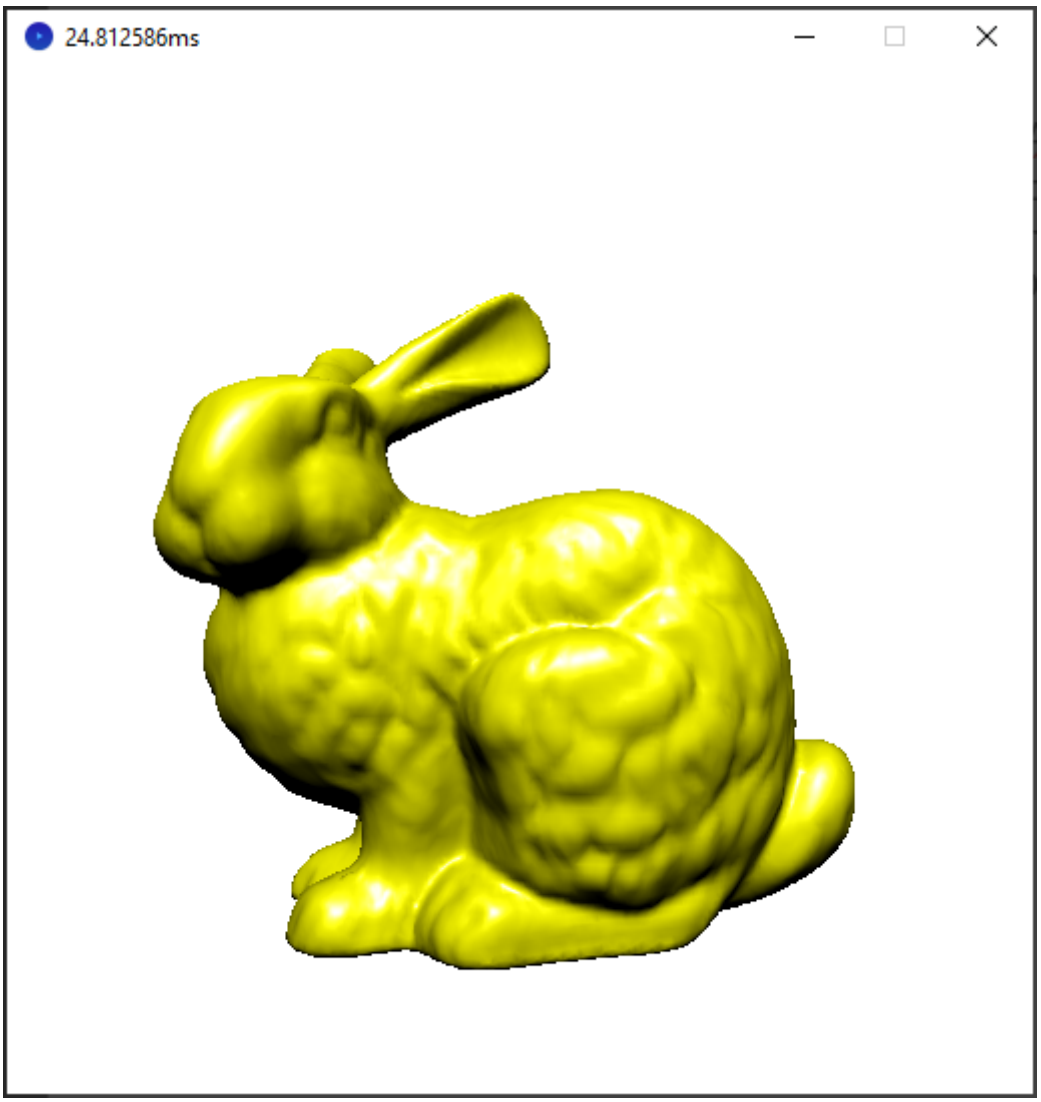

Abbildung 3: Hase mit Phong Shading

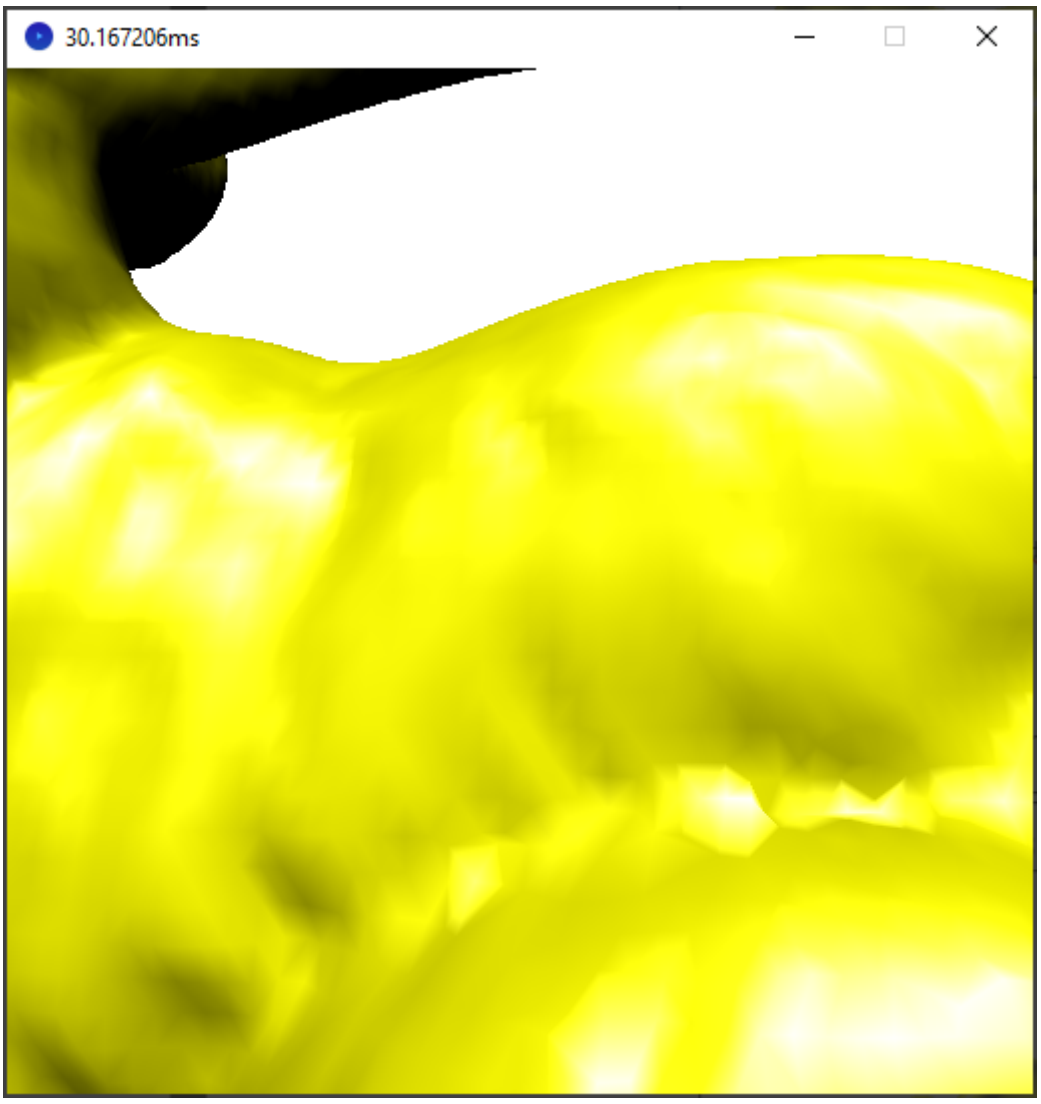

Abbildung 4: Vergrößerter Hase mit Gouraud Shading

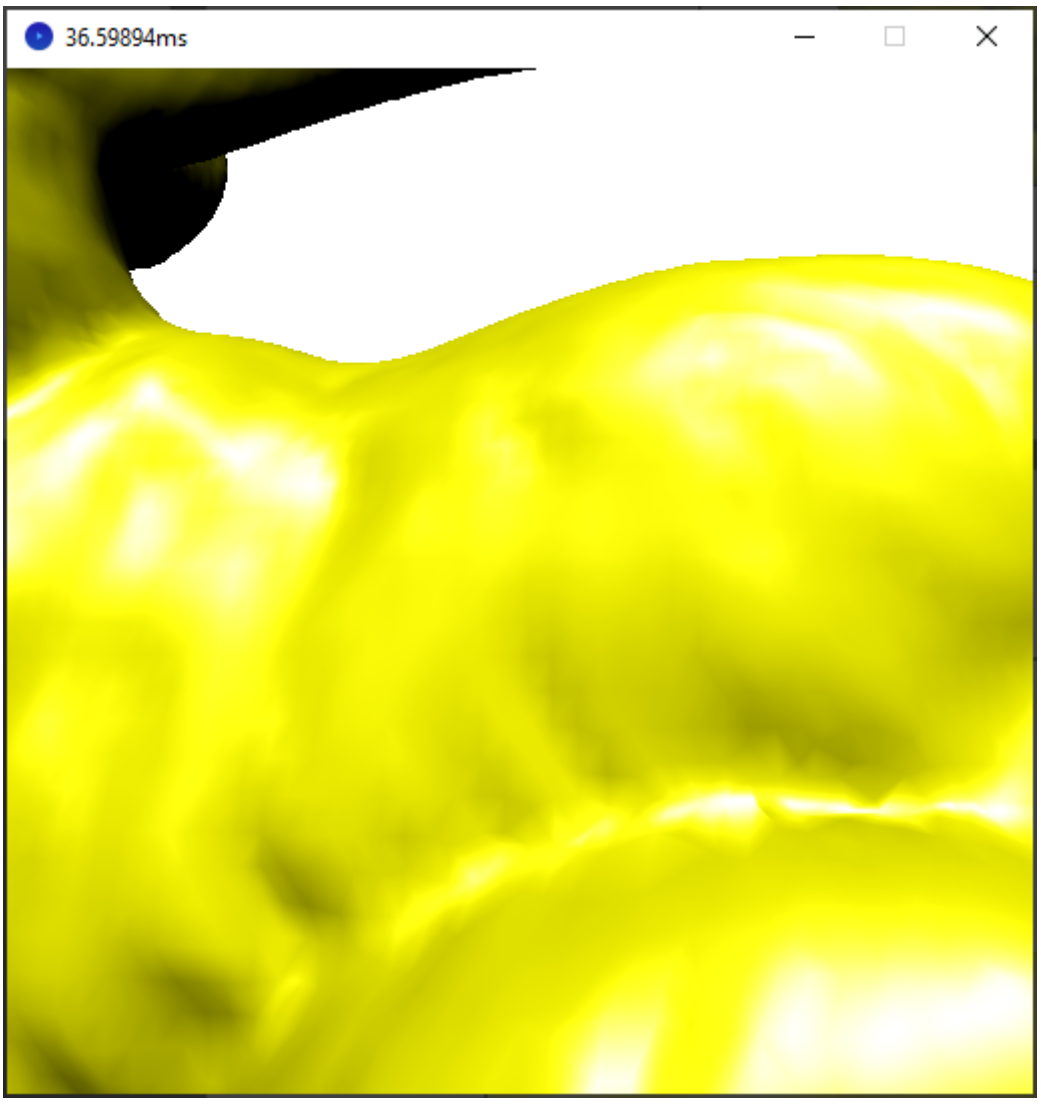

Abbildung 5: Vergrößerter Hase mit Phong Shading