

Shading/Interpolation

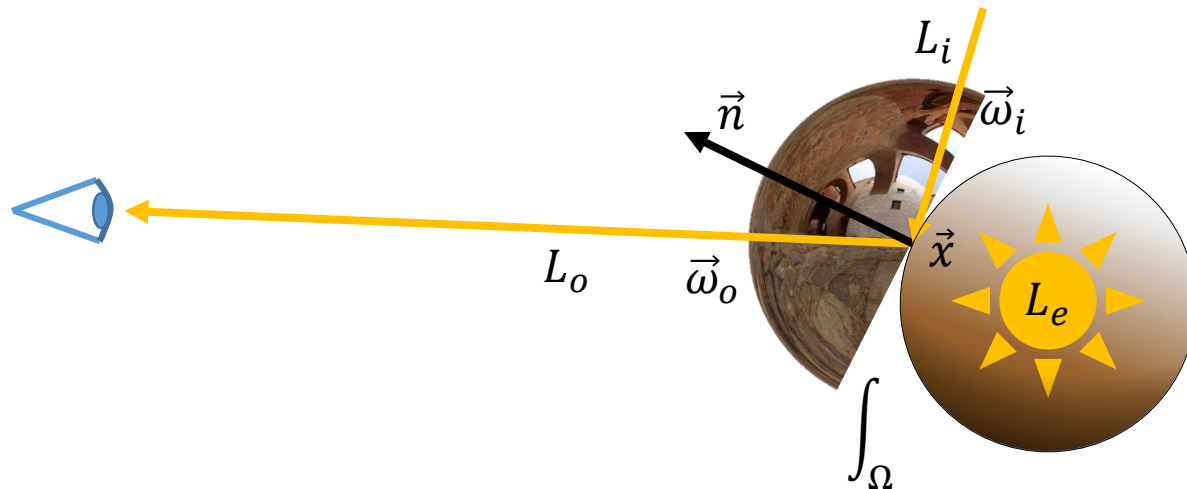
Matthias B. Hullin

Institut für Informatik II, Universität Bonn

Die Renderinggleichung für Oberflächen

$$L_o(\vec{x}, \vec{\omega}_o) = \underbrace{L_e(\vec{x}, \vec{\omega}_o)}_{\text{Emission}} + \underbrace{\int_{\Omega} f(\vec{\omega}_i, \vec{\omega}_o) L_i(\vec{x}, \vec{\omega}_i) \langle \vec{\omega}_i, \vec{n} \rangle d\vec{\omega}_i}_{\text{Reflexion}}$$

Streufunktion einfallende “Cosinus-
(BRDF) Radianz term”



[Kajiya 1986] [Immel et al. 1986]

Eine Lichtquelle

$$L_o(\vec{x}, \vec{\omega}_o) = f(\vec{\omega}_k, \vec{\omega}_o) \frac{I_k}{(\vec{x}_k - \vec{x})^2} \langle \vec{\omega}_k, \vec{n} \rangle$$

Einige Beleuchtungsmodelle (ohne $\cos(\theta)$ Bereinigung):

- Lambert (diffus):

$$\rho_{\text{diff}} \langle \vec{\omega}_i, \vec{n} \rangle$$

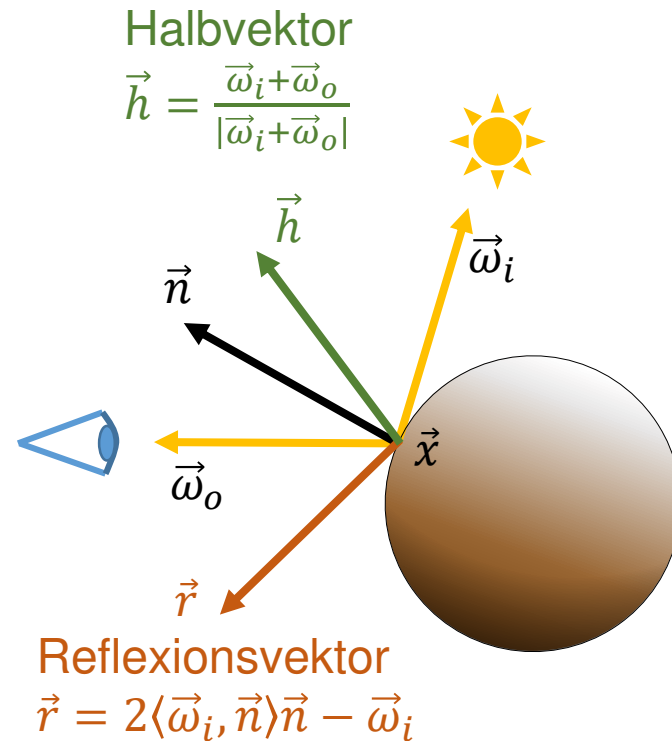
- Phong (glänzend):

$$\rho_{\text{spec}} \langle \vec{r}, \vec{\omega}_o \rangle^c$$

- Blinn-Phong (glänzend):

$$\rho_{\text{spec}} \langle \vec{h}, \vec{n} \rangle^c$$

- ...

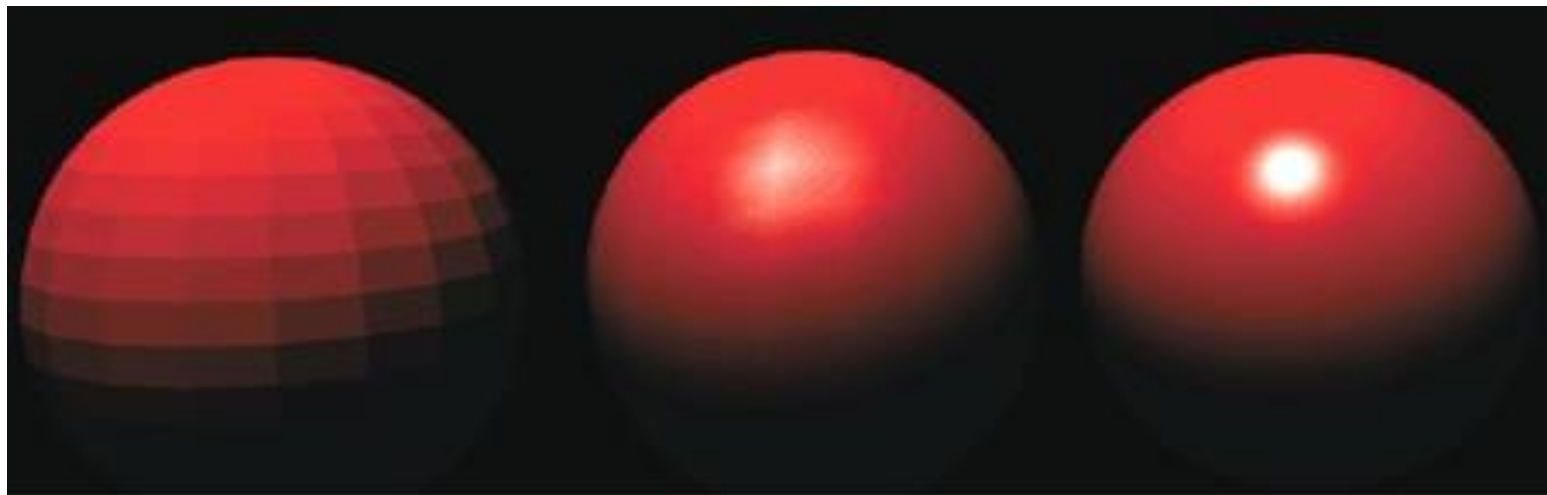


Bilderzeugung

Wir können die Beleuchtung lokal berechnen. Die hierfür erforderlichen Parameter (Normalenvektoren, Farbwerte, usw.) sind jedoch oft nur punktwise gegeben (z.B. als Vertexattribute).

Wie berechnen wir die Beleuchtung eines kompletten Objekts bzw. einer ganzen Szene?

Hierzu müssen Shadingdaten interpoliert werden. Dies kann auf verschiedene Weisen geschehen:



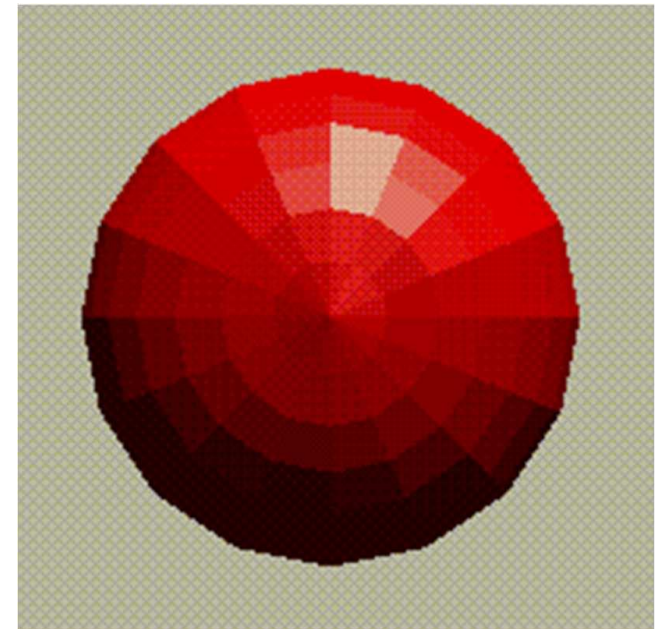
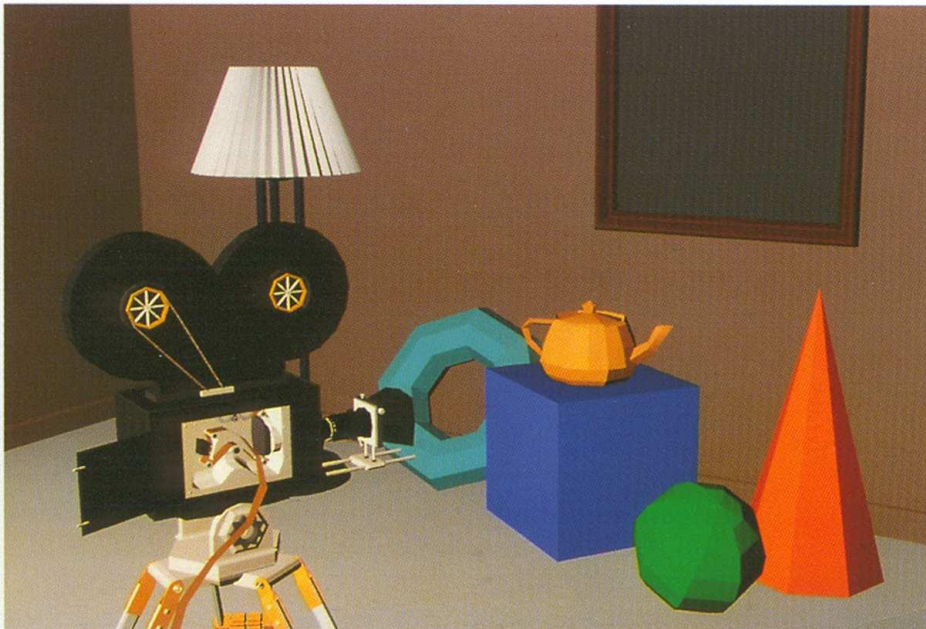
Flat

Gouraud

Phong

Flat Shading

- Die einfachste Interpolationsart ist das **Flat Shading**
- Jedes Polygon erhält einen Farbwert
- Farbe entweder als Attribut des Polygons oder eines seiner Vertices
- Effizient, aber nicht realistisch

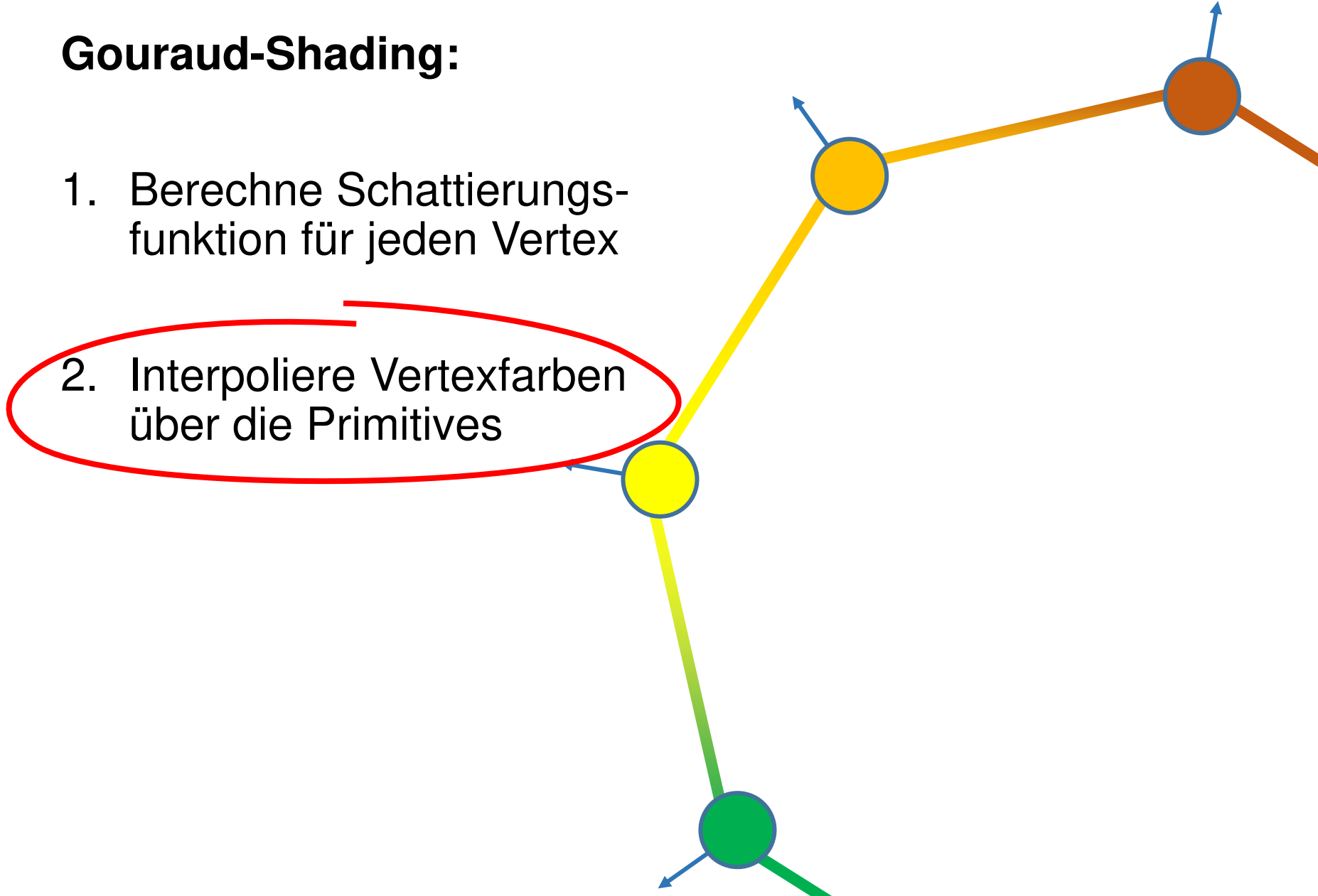


Per-Vertex-Shading [Gouraud 1971]

Gouraud-Shading:

1. Berechne Schattierungsfunktion für jeden Vertex

2. Interpoliere Vertexfarben über die Primitives



Historische Beispiele

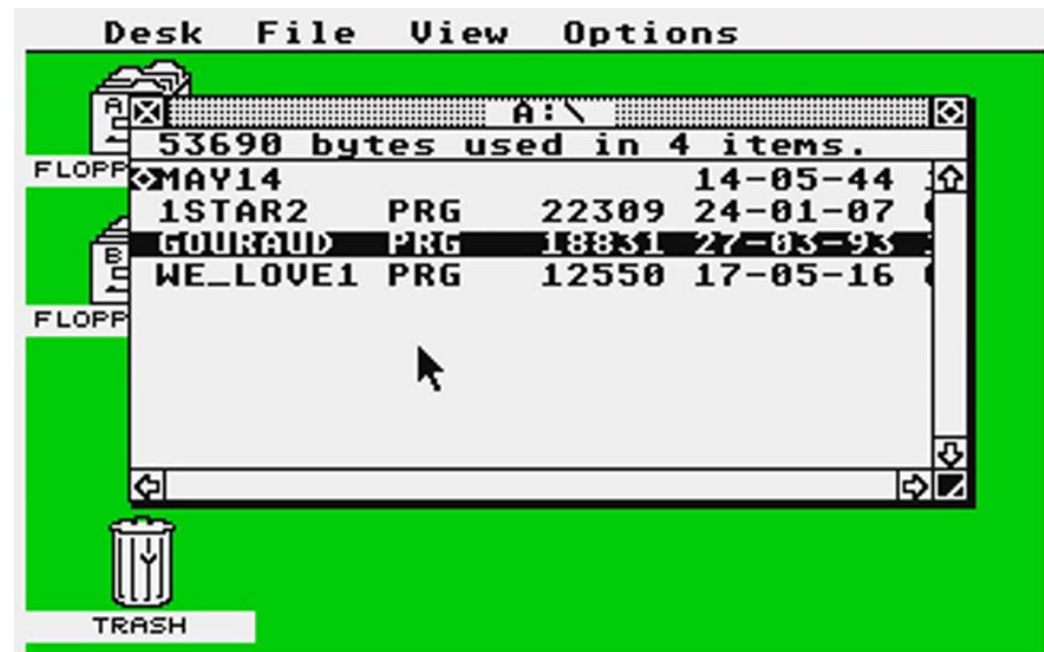
- Ridge Racer
1993, Namco System 22
(aka SimDrive)

Erstes "großes" Spiel mit Gouraud-Shading



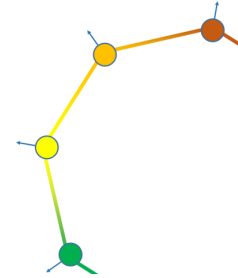
- Gouraud demo
Martin Griffiths (Griff)
1993, Atari STE

Hier: verwende z-Koordinate zur Schattierung



Gouraud-Shading

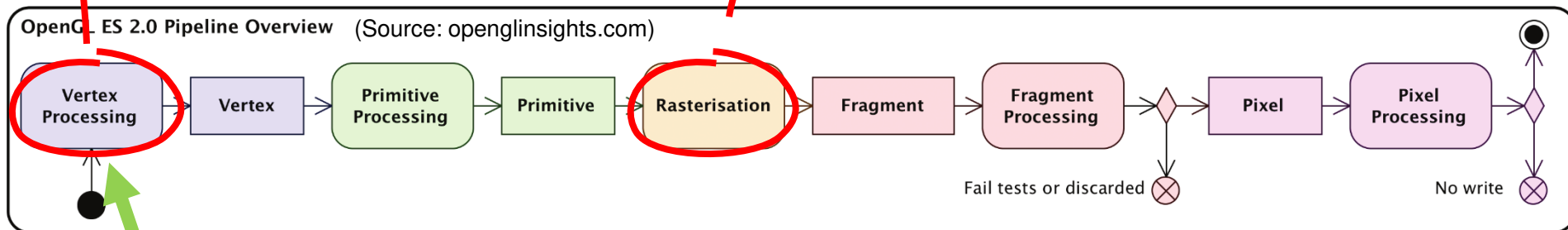
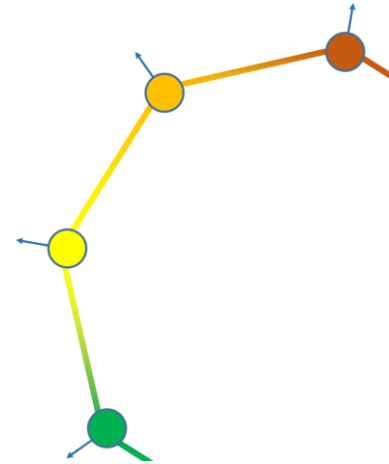
1. Was ist Gouraud-Shading?
- 2. Wie implementieren wir es?**
3. Wie implementiert die GPU es?
4. Probleme und Lösungen



Gouraud-Shading auf der GPU

1. Berechne
Schattierungs-
funktion für jeden
Vertex

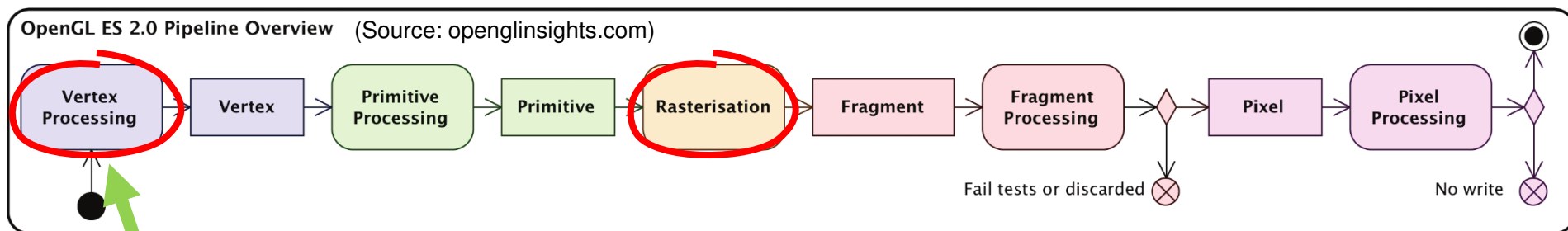
2. Interpoliere
Vertexfarben über
die Primitives



We are here

Beispielimplementierung (GLSL) – Vertexshader

```
uniform mat4 transform;  
layout (location = 0) in vec4 vPosition;  
layout (location = 1) in vec3 vNormal;  
  
out vec4 color;  
vec3 normal;  
  
vec4 shade(vec3 normal) {...} // eval. some BRDF for surface orientation  
void main() {  
    gl_Position = transform * vPosition;  
    normal = mat3(transform) * vNormal;  
  
    color = shade(normal);  
}
```

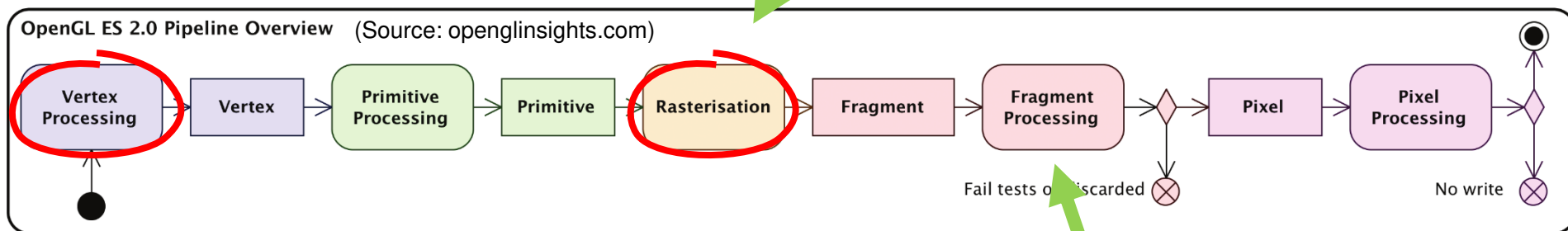


We are here

Beispielimplementierung (GLSL) – Fragmentshader

```
in vec4 color;  
out vec4 fragmentColor;  
  
void main() {  
  
    fragmentColor = color;  
  
}
```

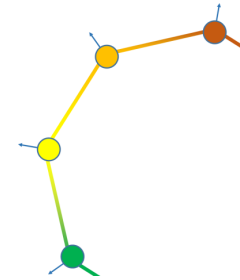
aber was ist hier passiert?



We are here

Gouraud-Shading

1. Was ist Gouraud-Shading?
2. Wie implementieren wir es?
- 3. Wie implementiert die GPU es?**
4. Probleme und Lösungen



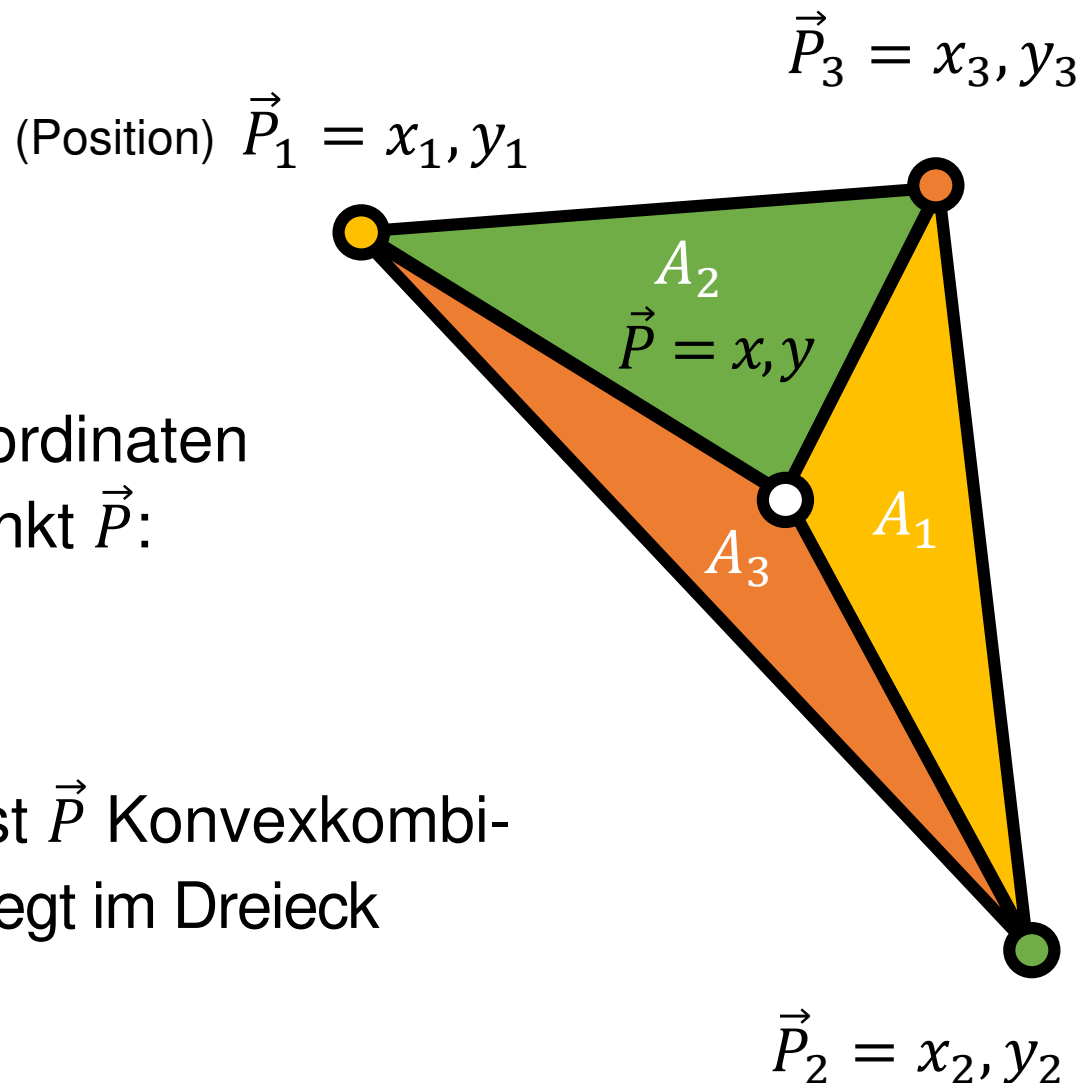
```
fragmentColor = color;
```

Baryzentrische Koordinaten zur Interpolation

- Baryzentrische Koordinaten (Gewichte) von Punkt \vec{P} :

$$\lambda_i = A_i / (A_1 + A_2 + A_3)$$

$$\Rightarrow \vec{P} = \sum_i \lambda_i \vec{P}_i$$
- Wenn $0 \leq \lambda_i \leq 1$, ist \vec{P} Konvexkombination von \vec{P}_i und liegt im Dreieck



Baryzentrische Koordinaten zur Interpolation

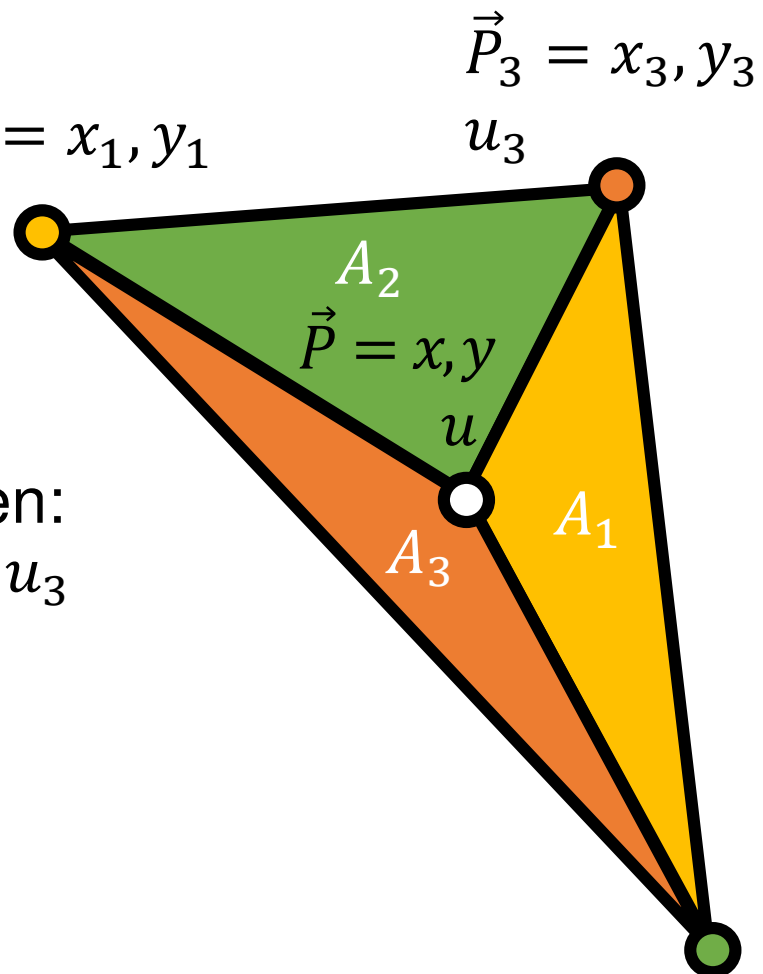


Diagram illustrating Barycentric Coordinates for Interpolation on a triangle.

Vertices and their attributes:

- Vertex 1: $\vec{P}_1 = x_1, y_1$ (Position), u_1 (Attribut)
- Vertex 2: $\vec{P}_2 = x_2, y_2$ (Position), u_2 (Attribut)
- Vertex 3: $\vec{P}_3 = x_3, y_3$ (Position), u_3 (Attribut)

The triangle is divided into three sub-triangles by lines connecting each vertex to the opposite side:

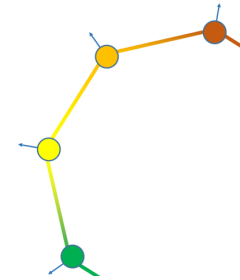
- A_1 (Yellow)
- A_2 (Green)
- A_3 (Orange)

A point $\vec{P} = x, y$ is shown inside the triangle, with its barycentric coordinate u indicated.

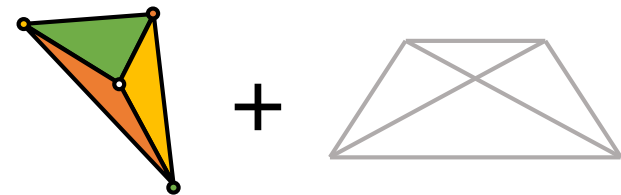
- Verwende λ_i um zwischen Vertexattributen zu interpolieren:

$$u = \lambda_1 \cdot u_1 + \lambda_2 \cdot u_2 + \lambda_3 \cdot u_3$$
- Im Raytracer: sehr einfach umzusetzen
- Im Rasterisierer wird dies von spezialisierter Hardware übernommen
- Interpolation von λ_i : siehe letzte Vorlesung

1. Was ist Gouraud-Shading?
2. Wie implementieren wir es?
3. Wie implementiert die GPU es?
4. **Probleme und Lösungen**



```
fragmentColor = color;
```



Visuelle Probleme

- Das menschliche Auge ist kein "Messgerät"
- Cornsweet-Effekt:

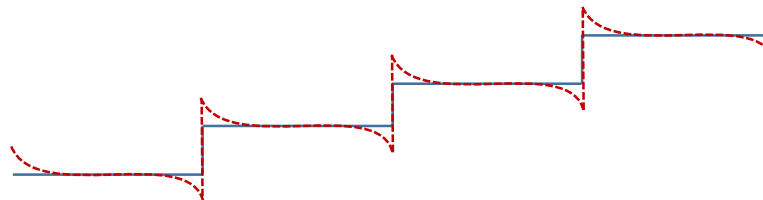


Tatsächliches Bild

Was wir zu sehen glauben

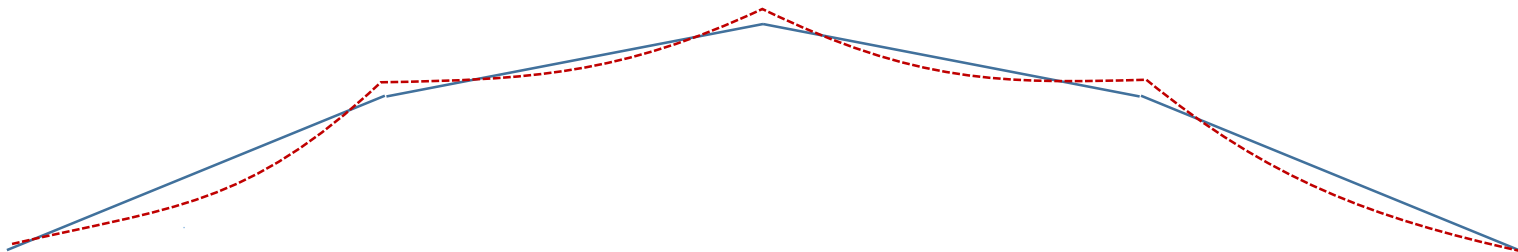
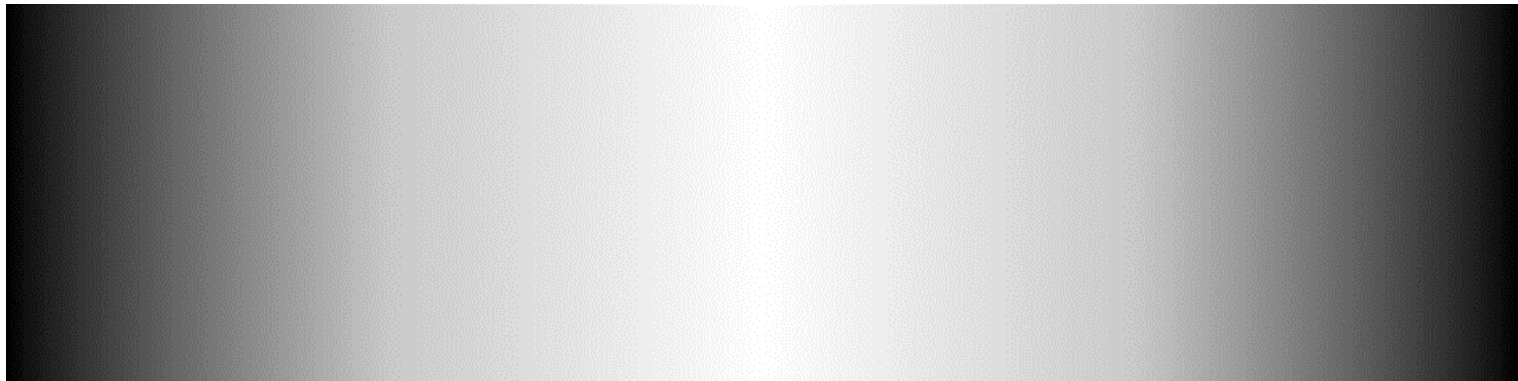
Visuelle Probleme

- Das menschliche Auge ist kein "Messgerät"
- "Mach-Banding"

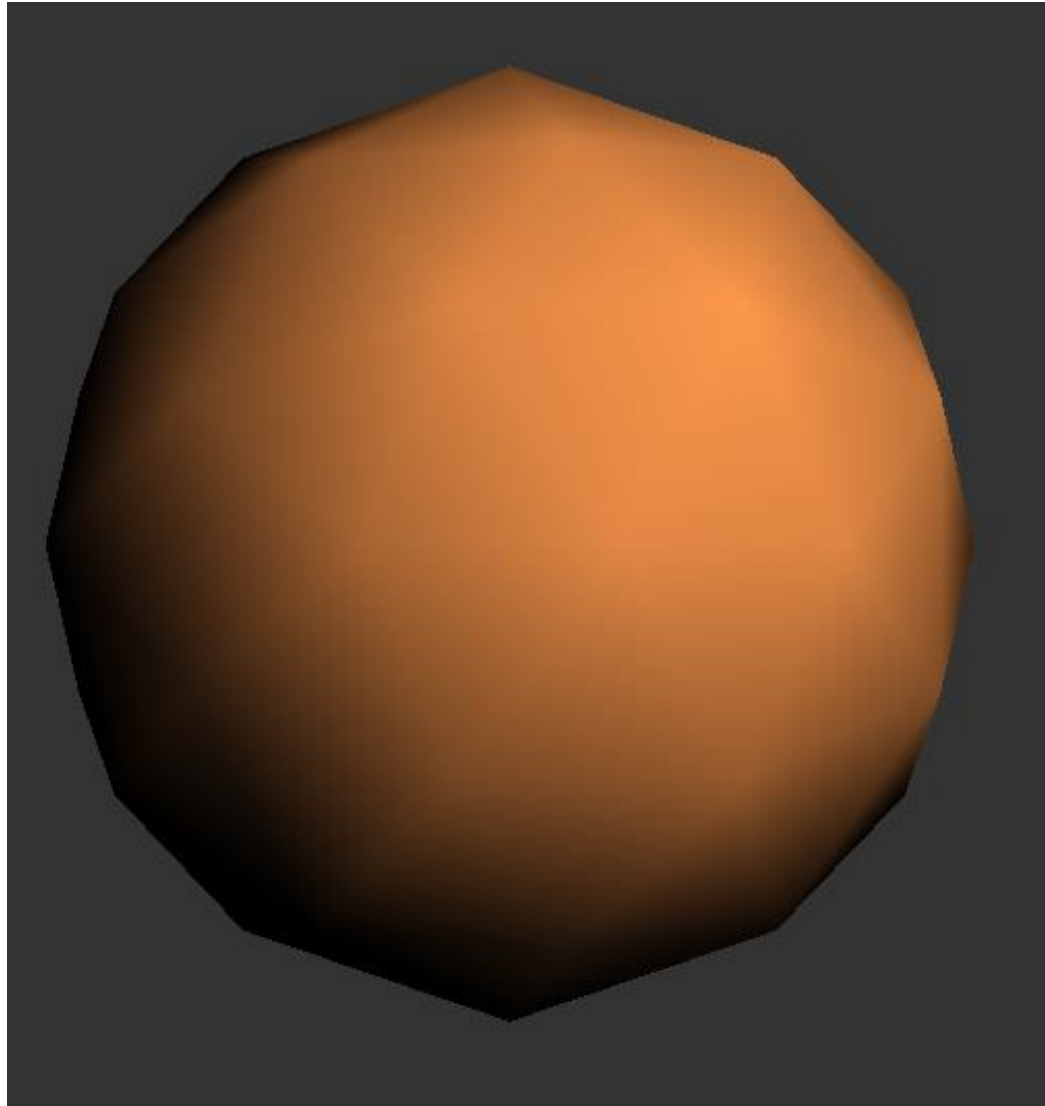


Visuelle Probleme

- Das menschliche Auge ist kein "Messgerät"
- "Mach-Banding" auf Gouraud-interpolierten Flächen

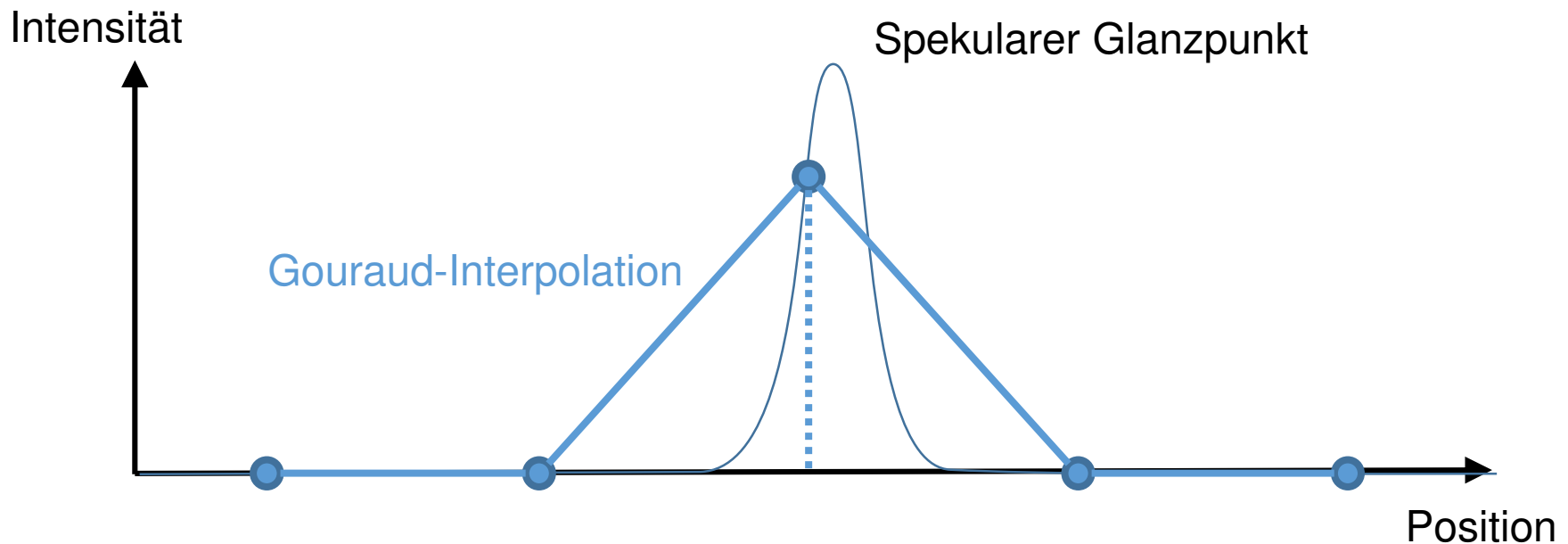


Mach banding



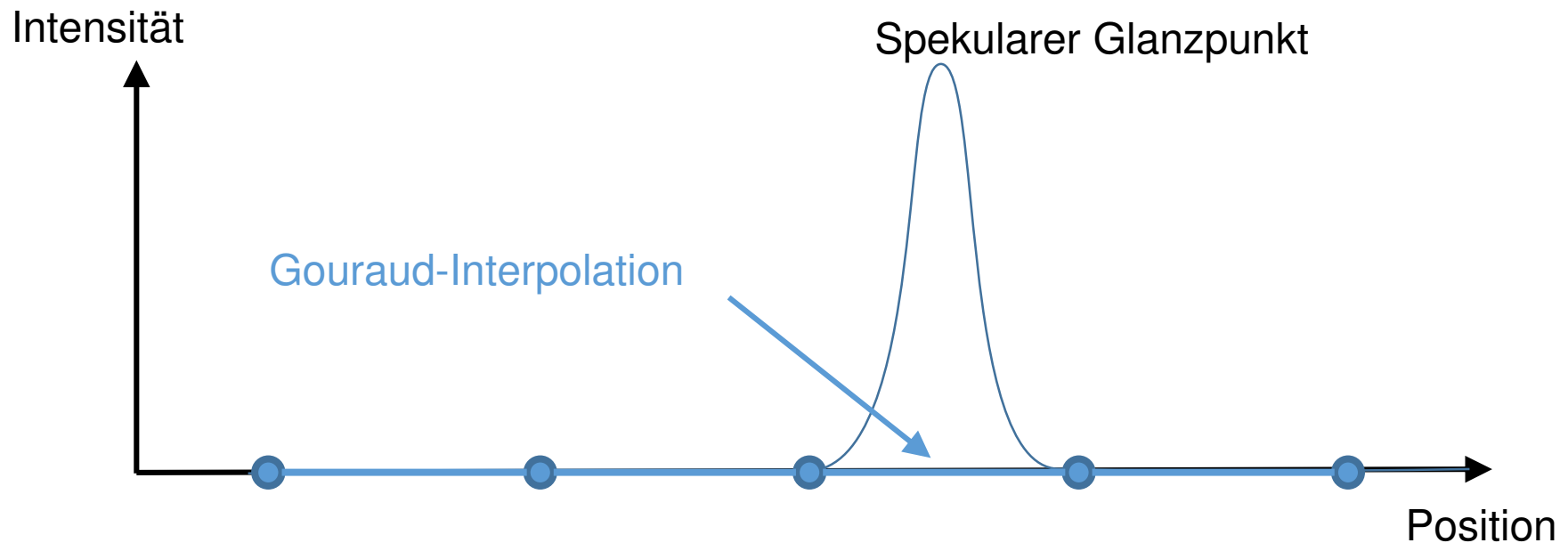
Abtastprobleme

- Gouraud-Interpolation = lineares Rekonstruktionsfilter
- Nyquist-Regel! Spiegelnde BRDFs womöglich nicht hinreichend dicht abgetastet



Abtastprobleme

- Gouraud-Interpolation = lineares Rekonstruktionsfilter
- Nyquist-Regel! Spiegelnde BRDFs womöglich nicht hinreichend dicht abgetastet

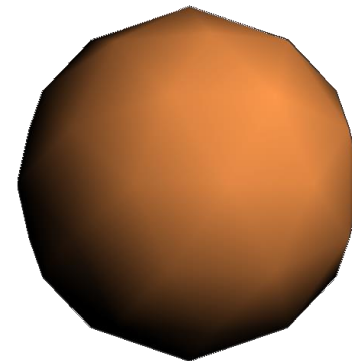
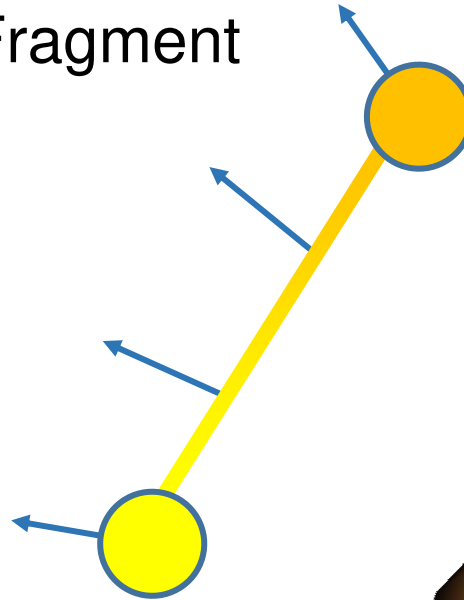


Was bedeutet dies?

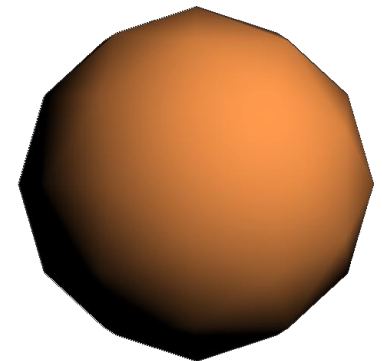
- Glanzlichter auf Vertices breiten sich unnatürlich aus
- Glanzlichter innerhalb eines Polygons verschwinden
- Demo

Phong-Shading [Phong 1975]

- Interpoliere **Vertexnormalen** übers Dreieck
- Schattiere jedes Fragment



Gouraud-style



Phong-style

interpolation

Literatur

Wie interpolieren NVIDIA-GPUs Vertexattribute?

- [Lindholm et al. 2008] Erik Lindholm et al., NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro 28 (2), pp. 39-55, 2008

Etwas schwer verdauliche Abhandlung über GPU-Interna:

- [Giesen 2011] Fabian Giesen, A trip through the Graphics Pipeline 2011. <https://fgiesen.wordpress.com/2011/07/09/a-trip-through-the-graphics-pipeline-2011-index/> - Parts 6 and 7, retrieved 08/2017

Gute Herleitung von perspektivisch korrekter Interpolation (falls Sie es wirklich *verstehen* wollen):

- [Scratchapixel] Unknown Author(s), Scratchapixel 2.0 – Rasterization: a Practical Implementation. <http://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation>, retrieved 08/2017