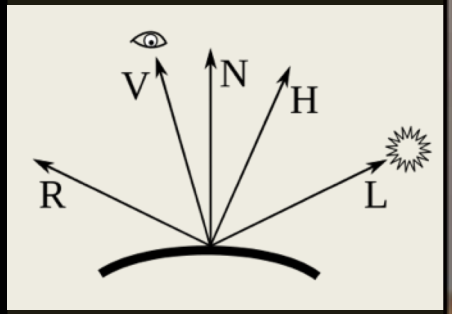
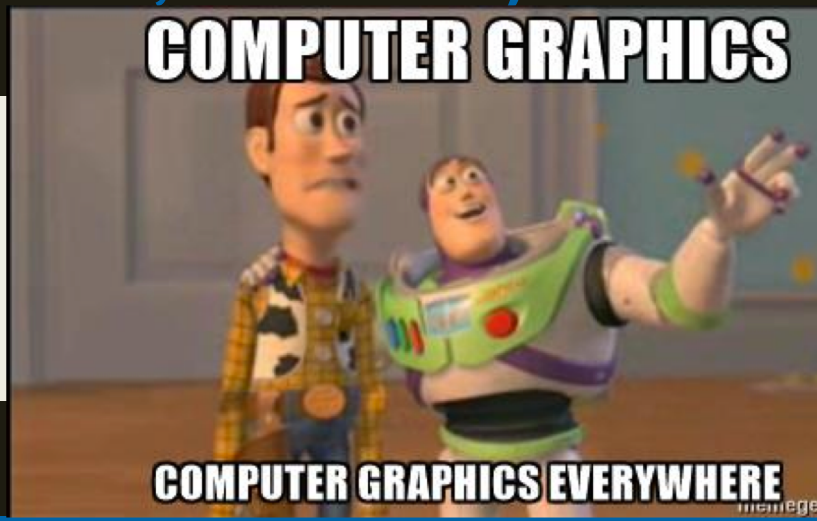
TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Multimedia – OpenGL (Shader, Texturen)

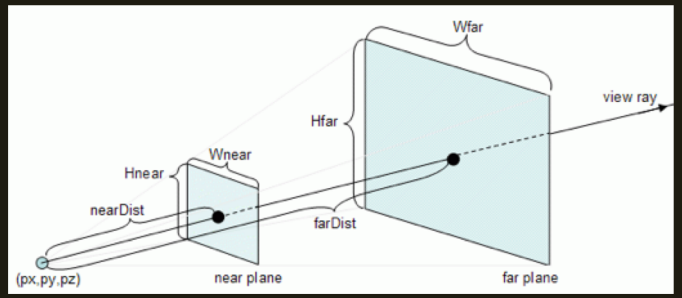


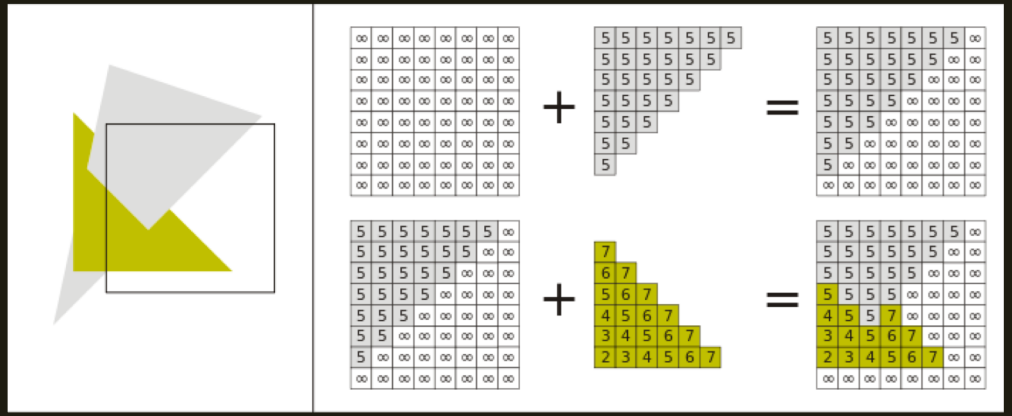


Jonas Treumer / Ben Lorenz

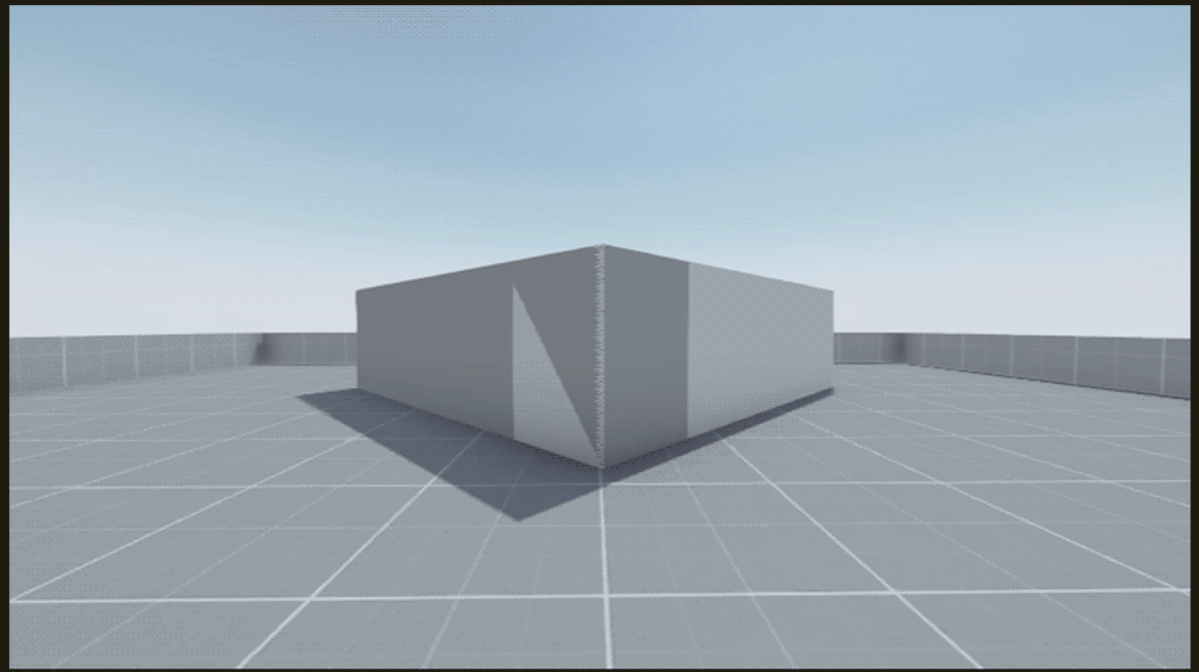
Depth-Test

- ohne Depth-Test werden Polygone der Reihe nach übereinander gezeichnet

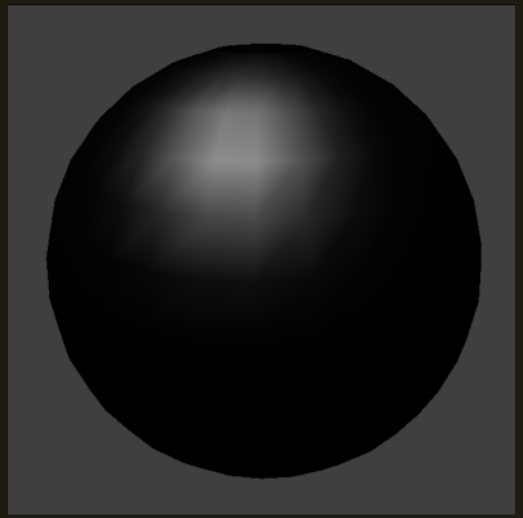


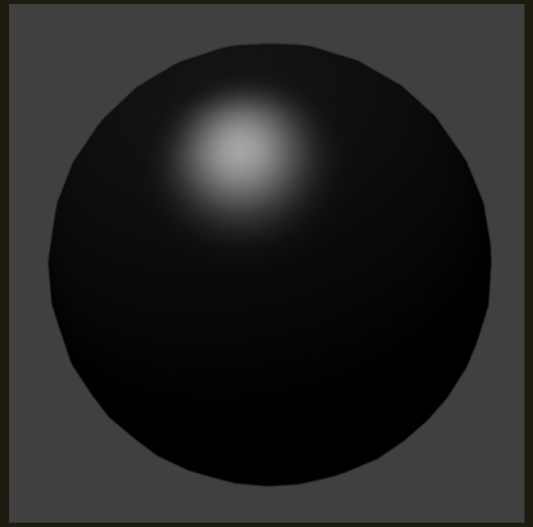


Z-Fighting



Flat vs. Gouraud vs. Phong Shading

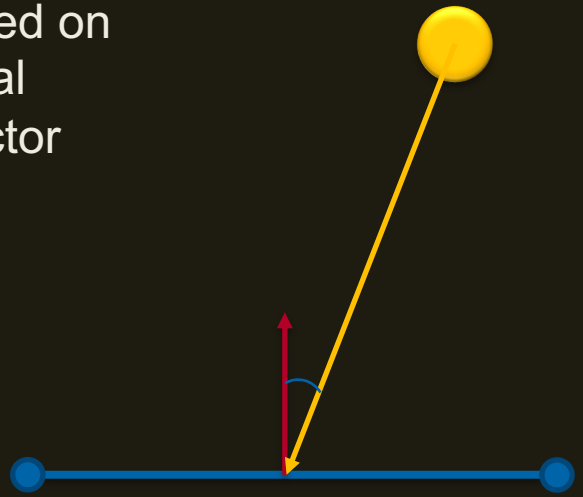




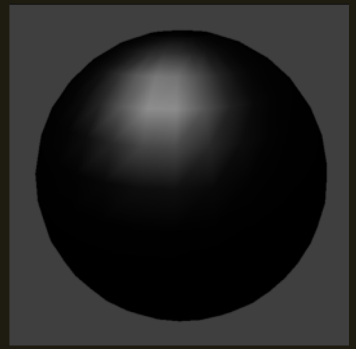
Flat vs. Gouraud vs. Phong Shading

Flat Shading

Hole triangle gets shade based on angle between surface normal and light source direction vector



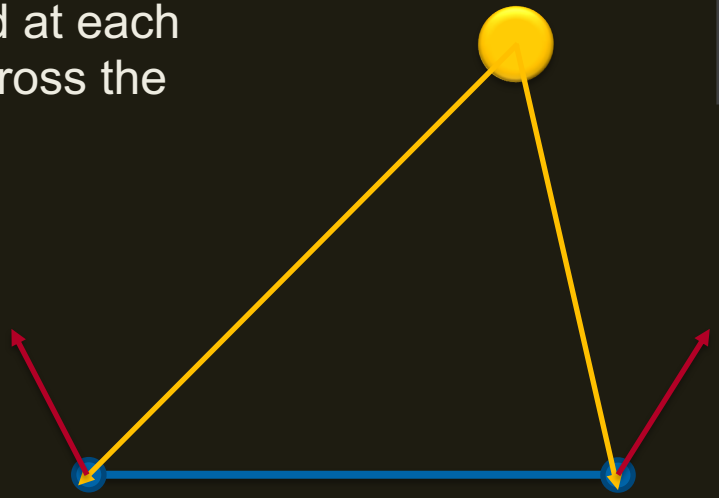
Flat vs. Gouraud vs. Phong Shading



Gouraud Shading

Light intensity is computed at each vertex and interpolated across the surface

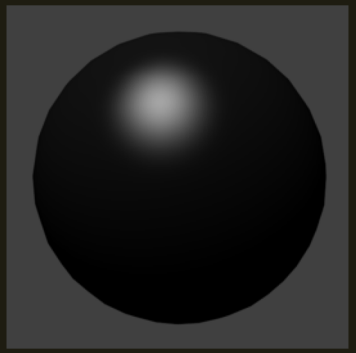
vertex normal = average of surrounding triangle normals

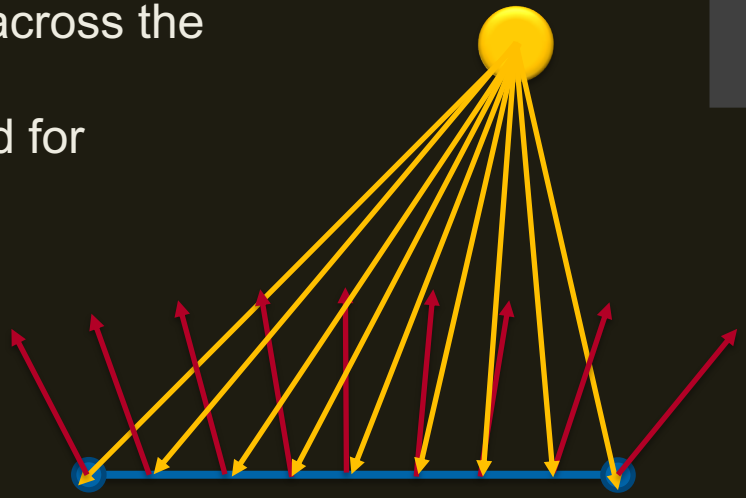


Flat vs. Gouraud vs. Phong Shading

Phong Shading

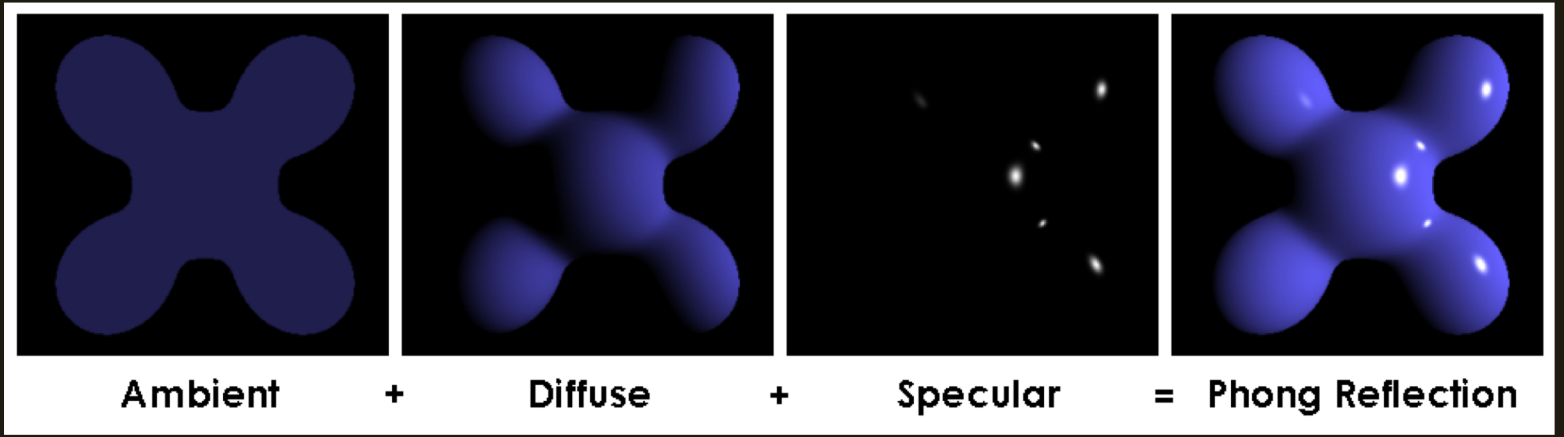
Normals are interpolated across the surface
Light intensity is calculated for each pixel



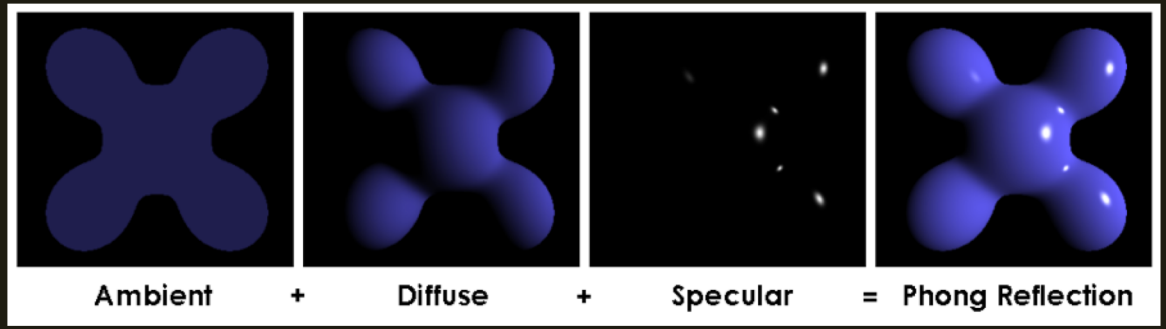


Phong Reflection Model

- empirical, local illumination model
- light sources are points

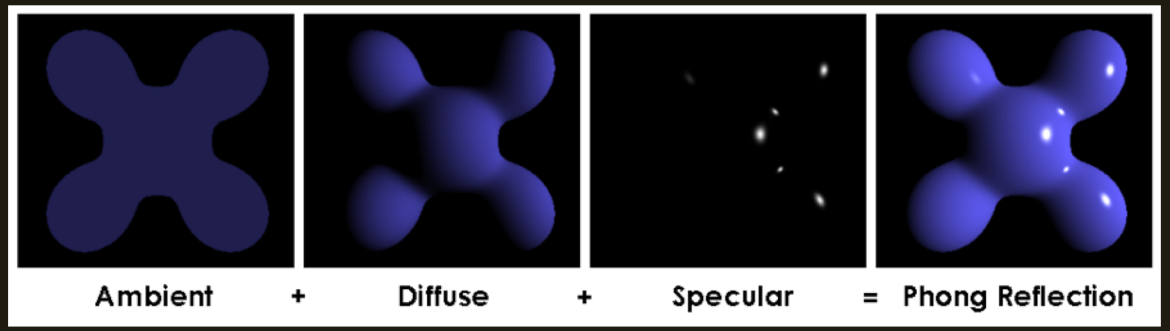


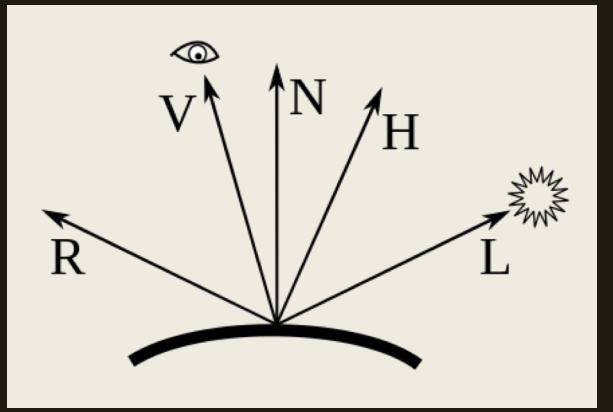
Phong Reflection Model



$$I_{out} = I_{ambient} + I_{diffuse} + I_{specular}$$

Phong Reflection Model

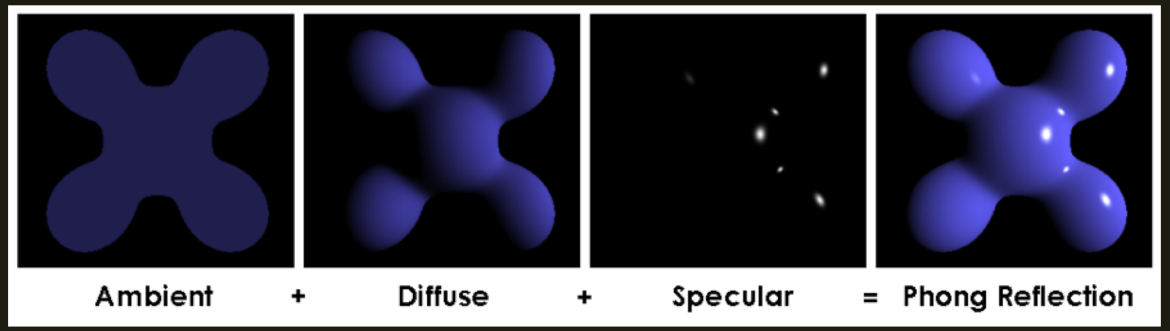


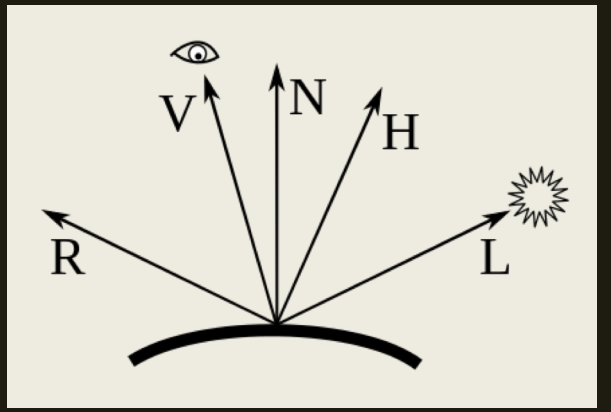


$$I_{out} = I_{ambient} + I_{diffuse} + I_{specular}$$

$$I_{out} = I_{ambient} k_a + I_{in} \left[k_{diffuse} (\vec{L} \cdot \vec{N}) + k_{specular} \frac{n+2}{2\pi} (\vec{R} \cdot \vec{V})^n \right]$$

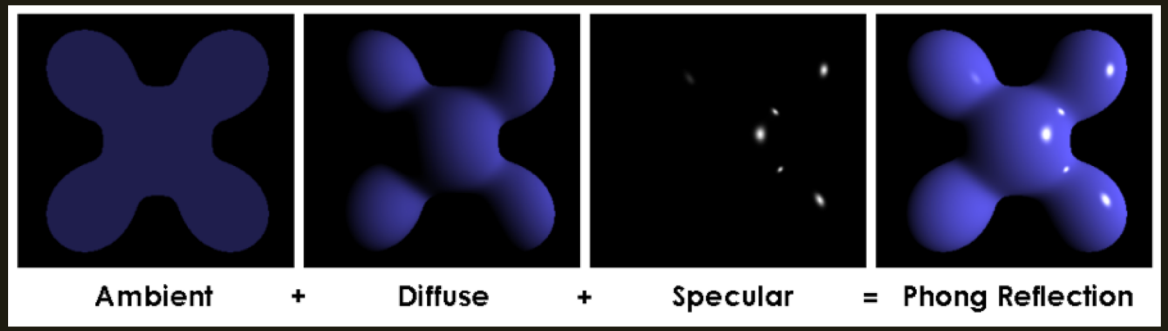
Phong Reflection Model

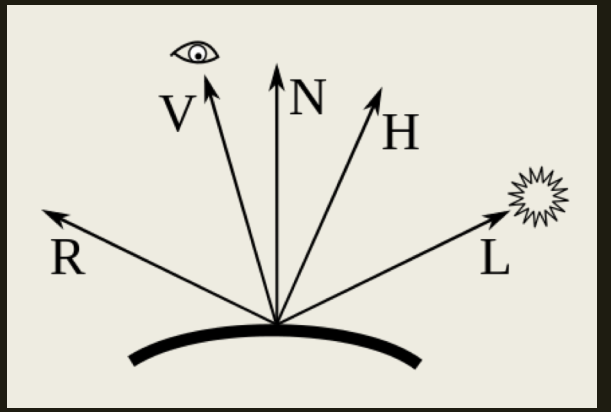




$$I_{diffuse} = I_{in} k_{diffuse} (\vec{L} \cdot \vec{N}) = I_{in} k_{diffuse} * \cos(\varphi)$$

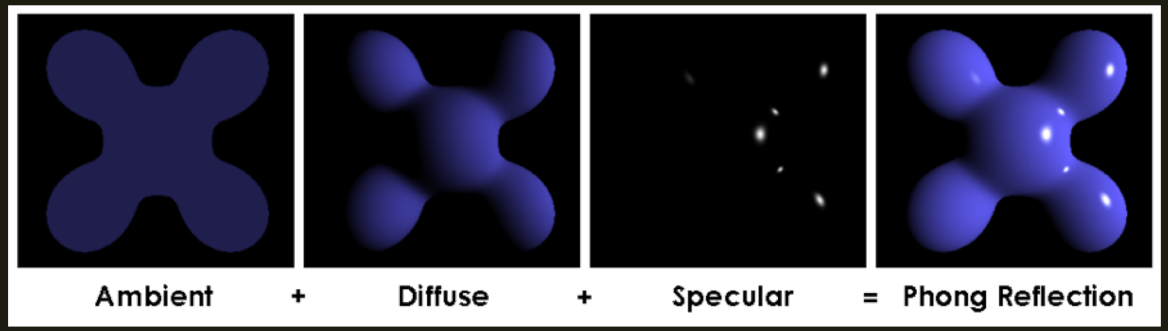
Phong Reflection Model

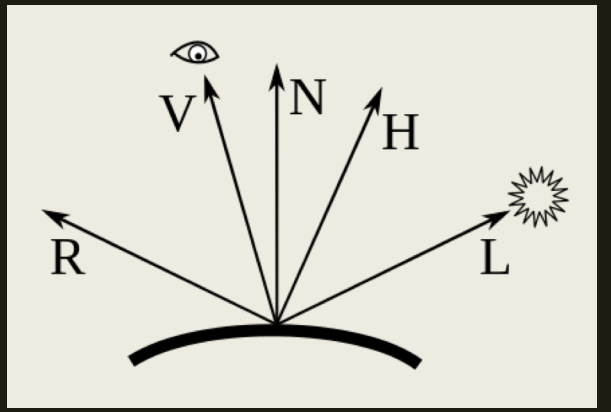




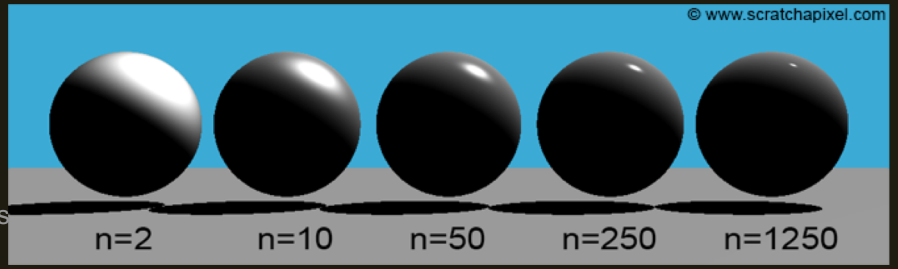
$$I_{specular} = I_{in} k_{specular} \frac{n+2}{2\pi} \left(\vec{R} \cdot \vec{V}\right)^{n} = I_{in} k_{specular} \frac{n+2}{2\pi} * \cos(\theta)^{n}$$

Phong Reflection Model



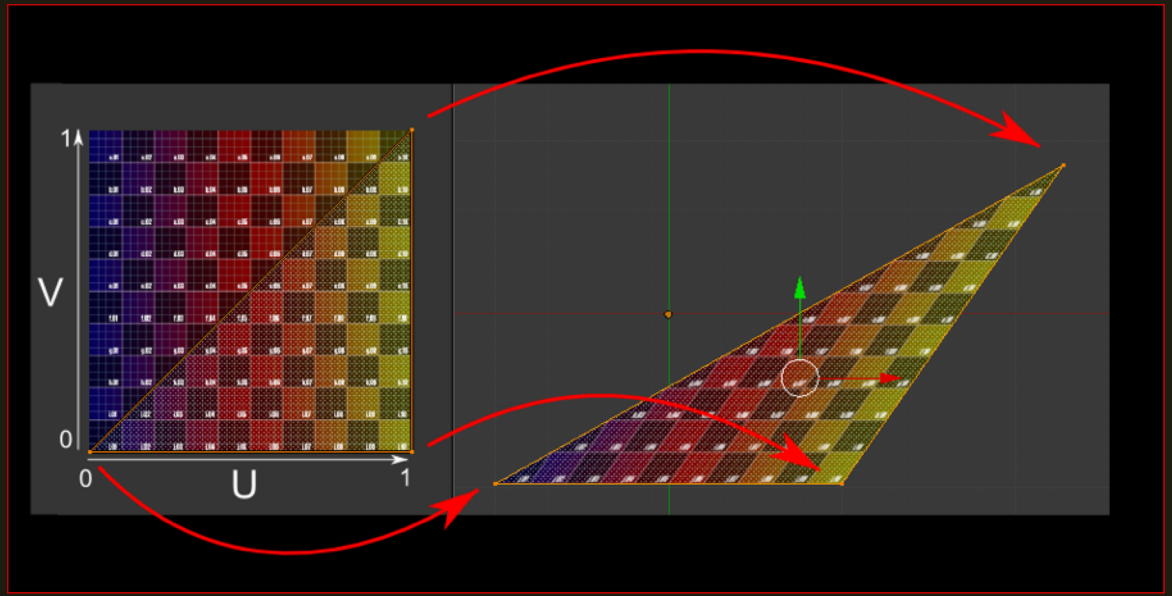


$$I_s = I_{in} k_s \frac{n+2}{2\pi} \left(\vec{R} \cdot \vec{V}\right)^n = I_{in} k_s \frac{n+2}{2\pi} * \cos(\theta)^n$$

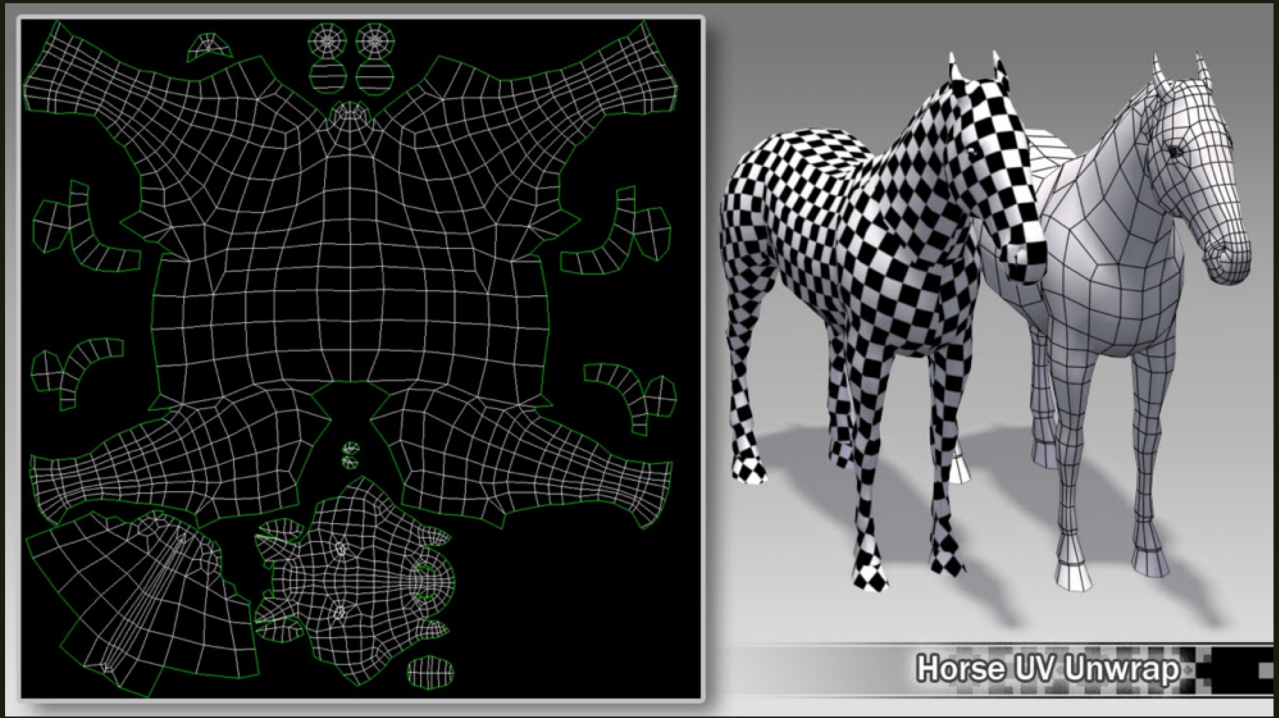


Textures

- Zuordnung Vertex (x,y,z) → Punkt in Texturebene (u,v) [0,1]



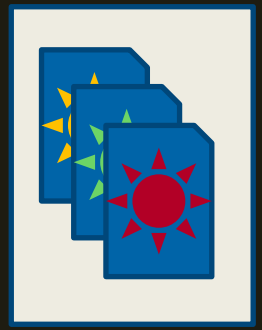
Textures



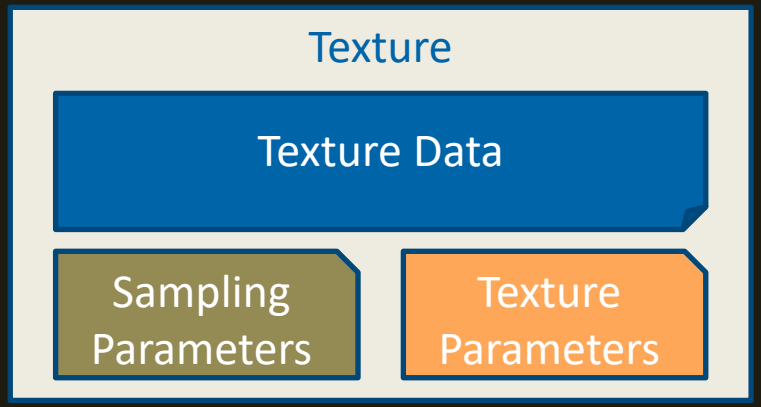
Textures - OpenGL

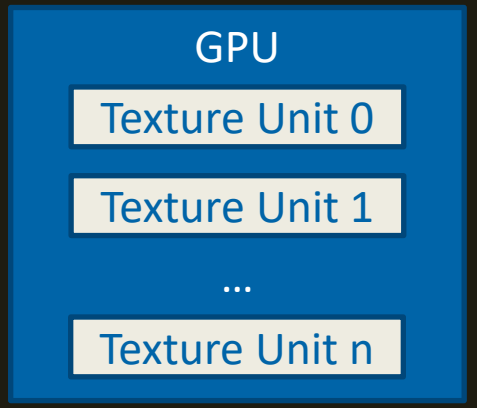
- Bild = Pixel-Array (1D, 2D, 3D; Größe; Format)

- Textur = Container für Bilder (Constraints für Bilder!)
    - Texture Type (GL_TEXTURE_1D/2D/3D/2D_ARRAY…)
    - Texture Size (Power of 2, Device Limit)
    - Image Format (RGB, RGBA, ARGB…)



Textures - OpenGL






```
glActiveTexture(GL_TEXTURE0);
```

```
GLuint texture;
glGenTexture(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
```

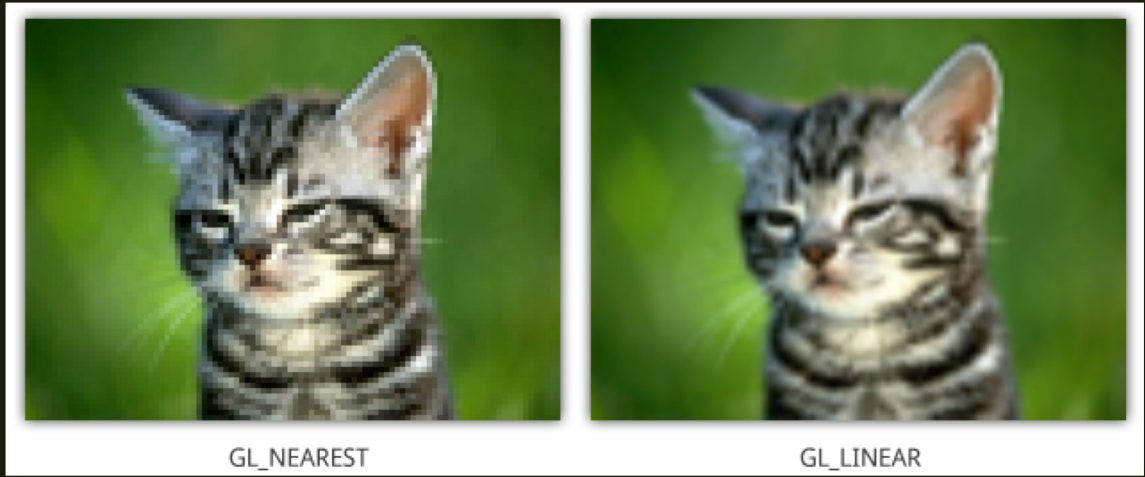
Textures – Wrapping (Sampling Params)

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

https://open.gl/textures

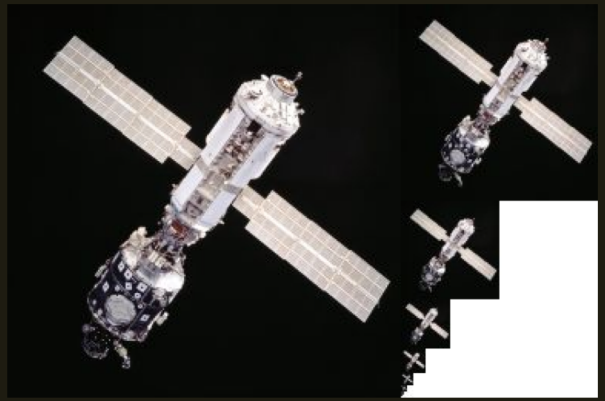
Textures – Filtering (Sampling Params)

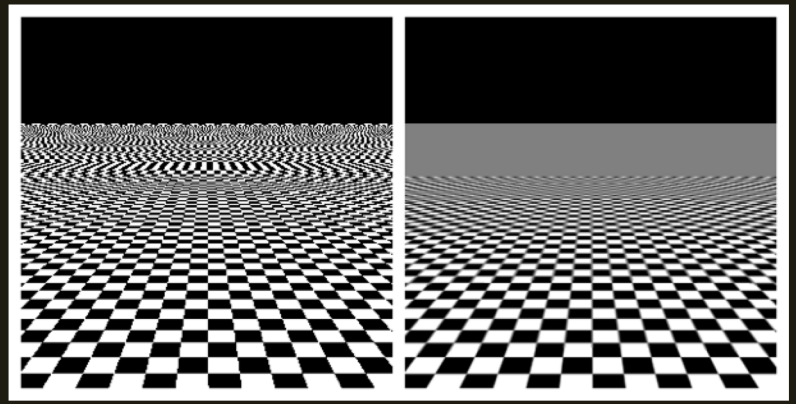
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```



Textures – Filtering – Mip Mapping

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glGenerateMipmap(GL_TEXTURE_2D);
```



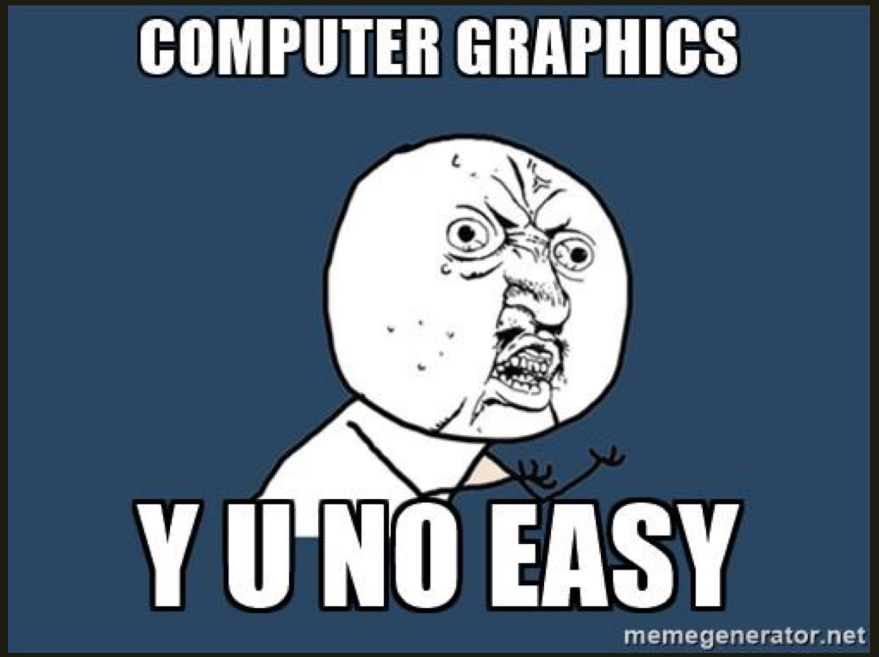


Textures – Laden

- Achtung: Texturkoordinatensystem vs. Bildkoordinatensystem

```
float pixels[] = {
    0.0f, 0.0f, 0.0f,   1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,   0.0f, 0.0f, 0.0f
};
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 2, 2, 0, GL_RGB, GL_FLOAT, pixels);
```

```
//...
//by default bound to texture unit 0
uniform sampler2D tex;
void main()
{
    outColor = texture(tex, Texcoord) * vec4(Color, 1.0);
}
```


COMPUTER GRAPHICS
Y U NO EASY
memegenerator.net