HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Introduction to Computer Graphics and Animation
Lecture 3 of 5

Prof. Dr. Dennis Allerkamp
December 4, 2024

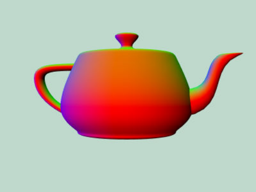
Summary

December 4, 2024

width The following topics will be covered today:

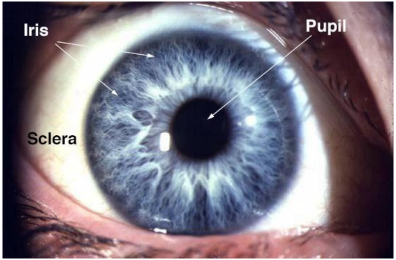
- Colors
- Lighting
- Shading
- Normal matrix

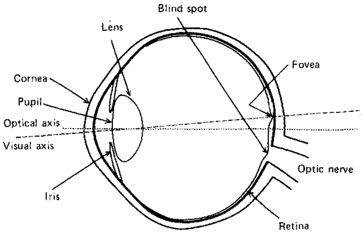
On this day, participants will learn how to use proper shading techniques to realistically illuminate objects in 3D scenes.



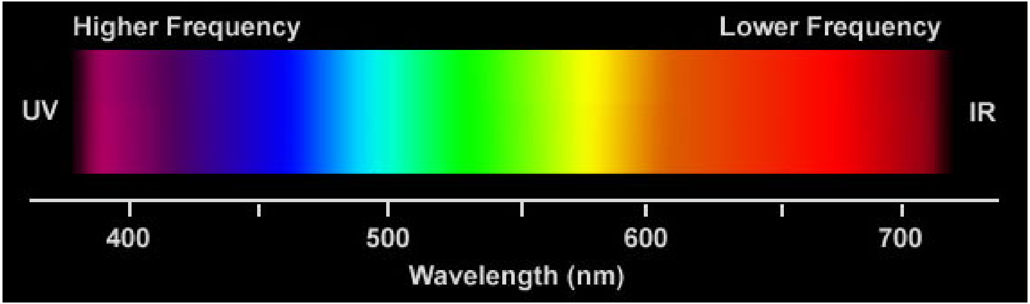
Color Perception and Color Spaces

The Human Eye





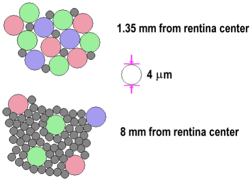
Light = Visually visible part of the electromagnetic spectrum

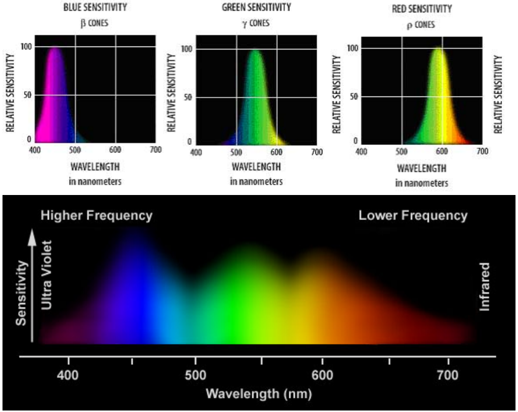


- $\sim$ 380 – 780 nm
- individually different
- outside the visible range: UV, IR, X-ray, microwaves, radio waves, etc.

Color Perception

- On the retina there are
  - Luminance receptors
    about 120 million rods
  - 3 types of color receptors
    about 6 million cones



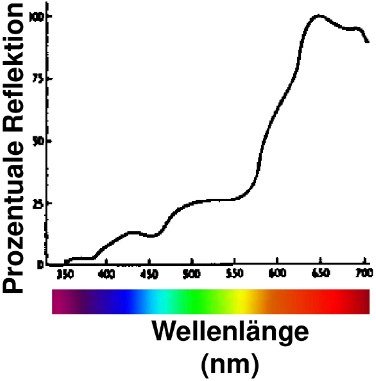


Color Stimulus / Perceived Color

- Color is *not a physical property* but the *interpretation of a stimulus*
- Stimulus is described by
  - Light intensity for each wavelength in the visible spectrum
- Perceived color is described by
  - Hue, e.g. green, blue, red = maximum color component
  - Brightness, Lightness = relative brightness compared to the brightness of white
  - Chromaticity, Saturation = relative chroma compared to brightness

Percentual Reflection

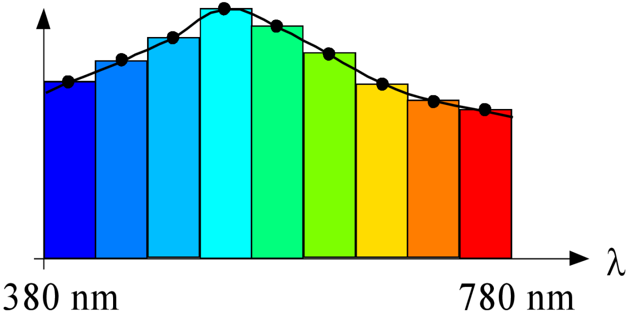
- Reflection spectrum of a tomato: not pure red, also blue light components



Reflection Spectrum of a Tomato

Spectrum as Color Space

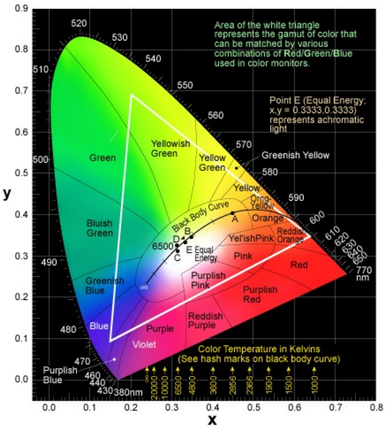
- Color spectrum approximated by 9 (=n) discrete values
- n-dimensional representation difficult to realize



Color Spaces

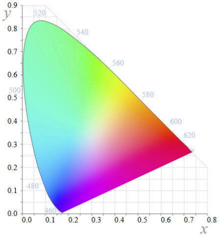
Color spaces represent an ordered representation of all available colors in a spatial arrangement.

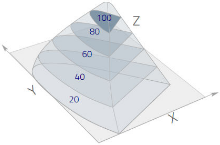
- Device-independent color spaces
  - Spectrum as color space
  - CIE standard in 1931
  - CIE 1976 UCS, LUV, LAB systems
- Device-oriented color spaces
  - RGB system
  - CMYK system
  - HLS (HSV/HSI) system



CIE Color Space

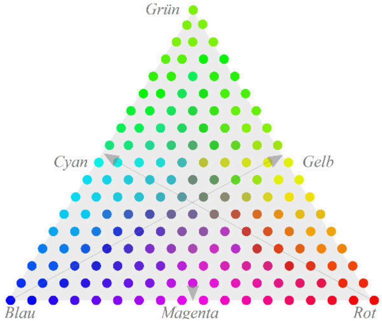
- CIE standard color table
  - Dimensions: X, Y, Z projected to x,y
  - Formed from spectral curve and purple line
- Spectral curve
  - Colors on the edge of the diagram are spectral colors
  - pure, monochromatic, saturated.
- Purple line
  - Non-spectral colors perceivable by humans
- The closer we get to the center, the grayer, whiter, and blacker the colors become
- Neutral point lies at $x = 1/3$, $y = 1/3$





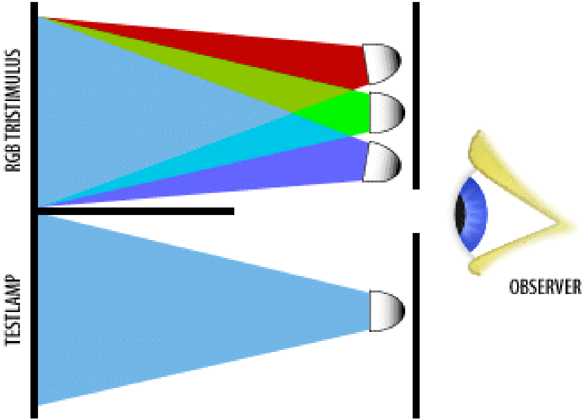
Maxwell's Triangle

- Colors
  - Primary colors
  - Secondary colors
- Complementary colors
- Basis for color spaces, RGB



https://wisotop.de/RGB-Dreieck-CIE-Farbtafel.php

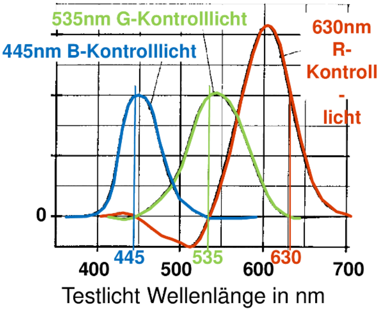
CIE Tristimulus Experiment (1931)



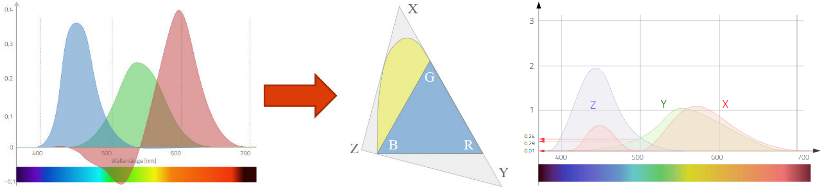
CIE Tristimulus Experiment

Resulting Functions of Color Values

- Intensity of colored light for color matching
- Problem: negative values
  - Certain spectral colors cannot be represented by combinations of control lights
  - Midpoint between blue and green
    - High-saturation cyan not possible
  - Red control light had to be added to the test light
  - Value is subtracted from the control light



CIE 1931 Standard Colors



Color mixing curves

- Proportions of the three primary colors R, G, B
- Create color equivalence to a monochromatic stimulus of a certain wavelength
- under standard illumination

Imaginary color values instead of real primaries

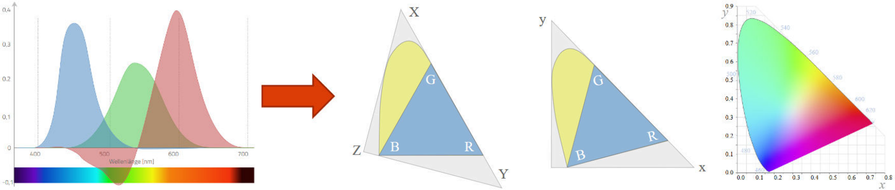
- X, Y, Z
- calculable from R, G, B
- Transformation matrix differs depending on color temperature/white balance

CIE Standard Color Table

- Dimensions: X, Y, Z projected to x, y

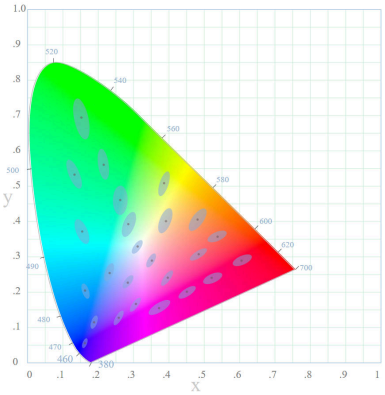
$$x = \frac{X}{X+Y+Z} \qquad y = \frac{Y}{X+Y+Z} \qquad z = \frac{Z}{X+Y+Z}$$

- $x + y + z = 1 \iff z = 1 - x - y$, i.e., the color space is describable by x and y.



Criticism of the CIE Color Space

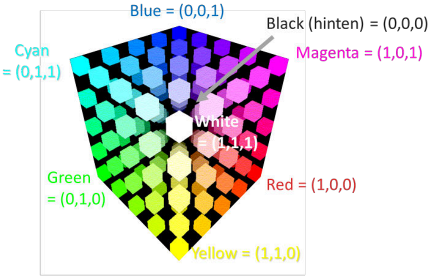
- Not all visible colors depicted in the projection
  - E.g., brown tones, no luminance components, it's all about chromaticity
  - Still mathematically included
- No linear color distances
  - Number of colors / distinguishability not comparable
  - McAdam ellipses
- Still based on measurements from 1931
  - error-prone
  - only 17 test subjects



McAdam Ellipses

RGB Color Space

- Basis for OpenGL
- *3D coordinate system* with axes for red, green, blue
- Colors can be described and calculated as vectors
  - 3D: RGB
  - 4D: RGBA
- A bit number is used as storage space for each color channel
  - e.g., 2 bits for 4 color levels, 8 bits for 256 color levels



Color model in OpenGL

Color model in OpenGL

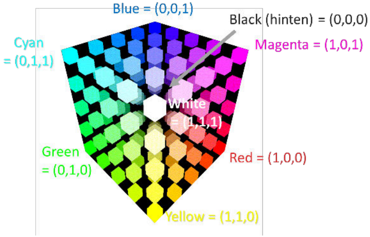
- RGB typical in computer graphics (*Red*, *Green* and *Blue*)
  - In OpenGL, however, a *fourth component A* is added
- RGBA is a 4-tuple consisting of
  - Red, Green, Blue RGB
  - and the *Alpha channel* A
- Alpha channel = transparent/opaque component
  - allows Alpha Blending
- *Alpha Blending*
  - Mixing pixels or areas like in a paint box
- *The range of all components is normalized*
  - Floating-point numbers *between 0.0 and 1.0*

Color model in OpenGL – Explanation of the components

- $R = 1.0$ is the *maximum intensity* of the red component
- $R = 0.0$ is in turn the *minimum intensity*
- The same applies to green and blue
- The component A is called *Alpha*
- Alpha is a measure of the opaqueness of a pixel
- $A = 0.0$ means 0% opaque (i.e., 100% transparent)
- *Example*: $RGBA = (1.0, 0.0, 0.0, 0.5)$ means red with half transparency

RGB Color Model

- Foundation for OpenGL
- *3D coordinate system* with axes for red, green, blue
- Colors can be described and calculated as vectors
- A bit number is used as storage space for each component
- *Brightness* of a color corresponds to the magnitude of the vector
  - Brightness$(r, g, b) = \sqrt{r^2 + g^2 + b^2}$
- *Gray values* occur when all color components are identical
- *Saturation* corresponds roughly to the distance to the gray value of a color
- *Hue* $=$ appearance of a color is determined by the largest components



Color Model in OpenGL – Color Assignment of a 3D Object

- Assigning a color *for every vertex*
  - Assignable in the Vertex Shader
  - Interpolated in the Rasterizer
  - Contained as color in the Fragment Shader
- Assigning colors *for every pixel*
  - To be specified explicitly in the Fragment Shader
  - Mixing with a texture is also possible

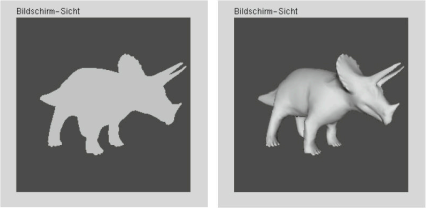
Models of Color Representation – RGBA Color Specification

- The background color is set with `glClearColor(R,G,B,A)`
  - The range of all components is in [0.0,1.0]
- Before each drawn image the color buffer must be cleared
- With `glClear(GL_COLOR_BUFFER_BIT)` the color values of the image memory are set to the background color
- *With the z-Buffer* both buffers can be initialized with one command (efficient)
  `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`

Light and Illumination

Illumination and Shading

- They are crucial elements for *three-dimensional perception*
  - Illumination creates noticeable *shadowing on the opposite side*
  - In technical jargon called “*shape from shading*”
- Figure (a): All polygons of the Triceratops have the same color values
  - Therefore, it appears *flat without illumination and shading*
- Figure (b): A three-dimensional impression is created through shading



(a) (b)

Nischwitz 2011 p.214

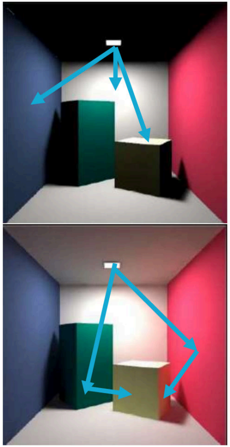
Illumination and Shading

- Light shines on the surface
  - enters the eye/camera
- Light factors
  - Ambient light
  - Light sources
  - Direction and spread
  - Color and brightness
- Surfaces
  - Absorption and transmission
  - Refraction, reflection, and gloss
  - Color and brightness of the surface

Zelda Breath of the Wild, Nintendo 2017, Source: PCGamesHardware

Local and Global Illumination Models

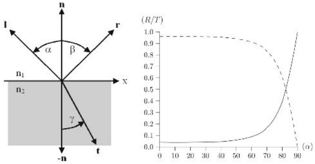
- Physically correct illumination is extremely complex
  - Constant flow of light rays partially absorbed by surfaces
  - Light rays refracted or reflected in different directions
- *Not only light sources* emit light, but *all surfaces of a scene*
  - All surfaces are potential light sources for all surfaces
- Local illumination models:
  - Only consider direct light from light sources
  - Calculate color at each point of a surface
- Global illumination models:
  - Consider indirect light that is emitted from a point on a surface
  - Include incident light from all spatial directions



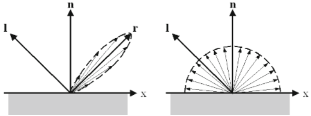
Local Illumination Models

Basic Considerations

- Calculation on the surface normal
  - Light direction l
  - Reflection direction r
  - Transmission direction t
- Fresnel effect
  - Reflection from flat angles also on transparent objects
- Scattering of the reflection direction
  - Uniform
    - Cigar shape for shiny surfaces
    - Hemisphere for matte surfaces (diffuse)
  - Real-life scenario is more complex
    - shape depends on wavelength per material



Ratio of reflection/transmission dependent on angle, Nischwitz 2011 p.219



Cigar-shaped reflection, diffuse reflection, Nischwitz 2011 p.217

Basic Considerations

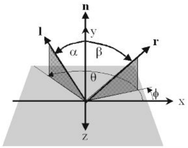
- Reflection in BRDF
  - Bidirectional Reflectance Distribution Function
  - Depending on wavelength, light/reflection direction

$$I_r(\beta, \phi, \lambda) = R(\alpha, \beta, \theta, \phi, \lambda) \cdot I_e(\alpha, \theta, \lambda) \cdot \cos(\alpha)$$

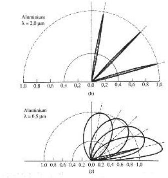
- For extended light sources in the surface integral under the hemisphere Ω

$$I_r(\beta, \phi, \lambda) = \iint_\Omega R(\alpha, \beta, \theta, \phi, \lambda) \cdot I_e(\alpha, \theta, \lambda) \cdot \cos(\alpha)\ dA(\alpha, \theta)$$

- Analogous transmission in BTDF
  - Bidirectional Transmission Distribution Function



BRDF in 3D space, Nischwitz 2011 p.222



Reflection of aluminium depending on wavelength, Watt 2002 according to He et al. 1991

Local Illumination Models

- *Approximation* of physically correct BRDFs
- *Empirical model* for light calculation
- Only contributions from point light sources are included in the calculation
  - Indirect light is *completely ignored*
  - Scattered light from other objects is represented by a primitive *ambient* term
- It does not consider occlusion of light sources = *no casting of shadows*
- Transparency is completely ignored
- Uses material properties of the object and the properties of the light to calculate the colors

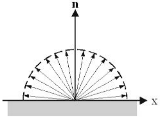
Illumination Components in OpenGL

- *Four components* that are calculated independently
  1. emitting/**emissive** component
     - *Self-luminous* color part of a surface
  2. **ambient** component
     - Part of the *indirect scattered light* and thus replacement for the global lighting components
  3. **diffuse** component (Lambert´s illumination model)
     - part of the *directed light from a point light source*, which is scattered equally in all directions from an *ideal diffuse surface*
     - *Unaffected by the position of the observer*
  4. **specular** component (Phong/Blinn illumination model)
     - Part of the *directed light from a point light source*, which is scattered mainly in the direction of the ideal reflection angle from a *real mirror-like surface*
     - *Depends on the position of the observer*
- The sum of all four lighting components gives the color
  Color = emissive + ambient + diffuse + specular

Emissive Component

- Easy to calculate as *no light source is required*
- Defined by the emitting / self-illuminating property of the material
- Like all other components, it's represented in the *RGBA model*
  - Alpha component is irrelevant for lighting (defined for consistency only)
- In contrast to light sources, it does not illuminate other surfaces

$$\text{emissiv} = \mathbf{e}_{mat} = \begin{pmatrix} R_{\mathbf{e_{mat}}} \\ G_{\mathbf{e_{mat}}} \\ B_{\mathbf{e_{mat}}} \\ A_{\mathbf{e_{mat}}} \end{pmatrix}$$



(a)

(b)

Ambient Component

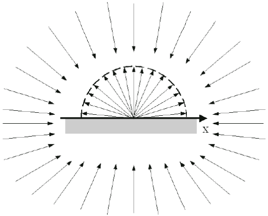
- Simplified *substitute for global lighting components*
  - Sum of the reflecting lights of all objects, basic light
- Easy to calculate.
  - Assumption: Room is ideally diffusely reflecting
- Has no direction and is therefore a simple constant
- Made up of ambient characteristics of the light and the material (*RGBA model*)

$$\text{ambient} = \mathbf{a}_{light} * \mathbf{a}_{mat} = \begin{pmatrix} R_{\mathbf{a}_{\mathbf{light}}} \\ G_{\mathbf{a}_{\mathbf{light}}} \\ B_{\mathbf{a}_{\mathbf{light}}} \\ A_{\mathbf{a}_{\mathbf{light}}} \end{pmatrix} * \begin{pmatrix} R_{\mathbf{a}_{\mathbf{mat}}} \\ G_{\mathbf{a}_{\mathbf{mat}}} \\ B_{\mathbf{a}_{\mathbf{mat}}} \\ A_{\mathbf{a}_{\mathbf{mat}}} \end{pmatrix} = \begin{pmatrix} R_{\mathbf{a}_{\mathbf{light}}} \cdot R_{\mathbf{a}_{\mathbf{mat}}} \\ G_{\mathbf{a}_{\mathbf{light}}} \cdot G_{\mathbf{a}_{\mathbf{mat}}} \\ B_{\mathbf{a}_{\mathbf{light}}} \cdot B_{\mathbf{a}_{\mathbf{mat}}} \\ A_{\mathbf{a}_{\mathbf{light}}} \cdot A_{\mathbf{a}_{\mathbf{mat}}} \end{pmatrix}$$

Nischwitz 2011 p.229

Ambient Component

- Uniform reflection of light
- No shading effects



Ungerichtetes Licht aus allen Richtungen

Nischwitz 2011 p.229

Diffuse Component (according to Lambert)

- Note: If light source is far away, like the sun, light rays arrive practically parallel
- Light intensity *depends on the orientation of the surface to the light source*
    - Lambert's law: The flatter the incidence angle, the lower the intensity
    - Perpendicular incident light = maximum intensity radiated on the surface
    - Is the *dominant component* (this is where 3D perception becomes clear)
    - Position of the camera does not influence

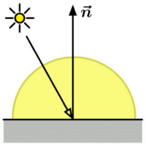
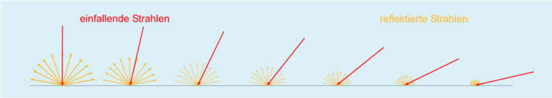


Abbildung 5.3.: Diffuse Reflexion: Licht verteilt sich gleichmäßig in alle Richtungen (Tramberend).

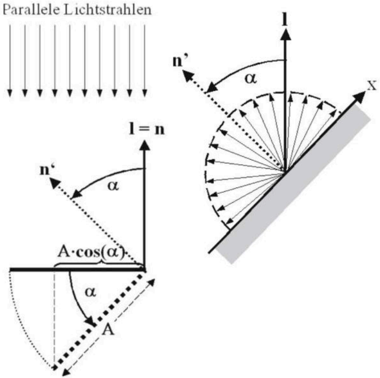


Diffuse Component (according to Lambert)

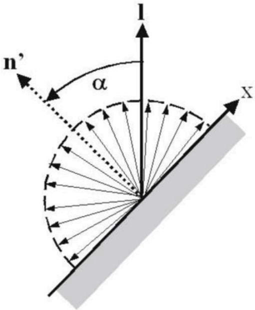
- Calculated from the *scalar product of light and the normal* of the surface
- Light is only reflected at angles of [-90°, +90°], otherwise *not* oriented towards light

$$\text{diffus} = \max(\mathbf{l}\cdot\mathbf{n}, 0)\cdot \mathbf{d}_{\text{light}} * \mathbf{d}_{\text{mat}}$$

$$= \max(\mathbf{l}\cdot\mathbf{n}, 0)\begin{pmatrix} R_{\mathbf{d}_{\text{light}}}\cdot R_{\mathbf{d}_{\text{mat}}} \\ G_{\mathbf{d}_{\text{light}}}\cdot G_{\mathbf{d}_{\text{mat}}} \\ B_{\mathbf{d}_{\text{light}}}\cdot B_{\mathbf{d}_{\text{mat}}} \\ A_{\mathbf{d}_{\text{light}}}\cdot A_{\mathbf{d}_{\text{mat}}} \end{pmatrix}$$



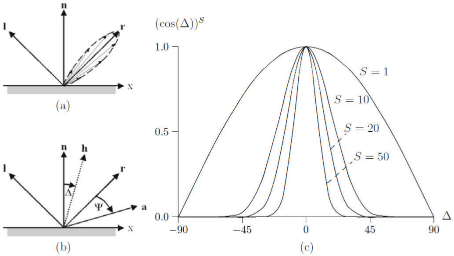
Diffuse Component (according to Lambert)



Nischwitz 2011 p.231

Specular Component (according to Phong)

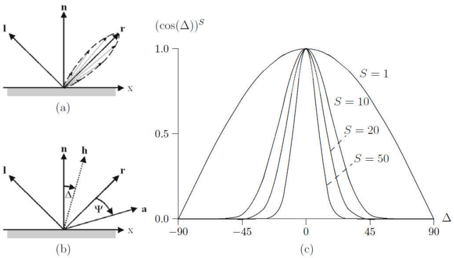
- Amount of directed light that comes from a point light source in a *single direction* and is *reflected in a preferred direction*
- The highest light intensity s is emitted in the reflection direction r
- The larger the angle Ψ between light reflection r and the eye point a, the lower is the reflecting light intensity



Nischwitz 2011 p.233

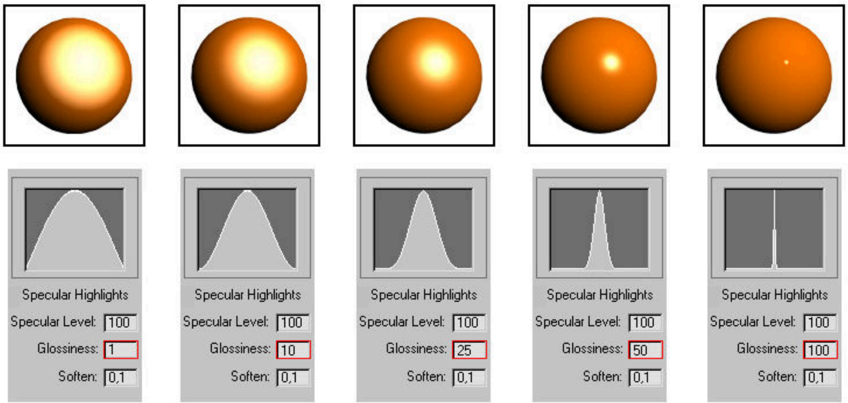
Specular Component (according to Phong)

- *Variation of the light intensity* through the angle between r and a is modeled by $(\cos(\Psi))^S$
- Exponent S describes the “shininess” factor
- Larger values (e.g., 50-100) make the surface appear very smooth
- Similarly for smaller values



Nischwitz 2011 p.233

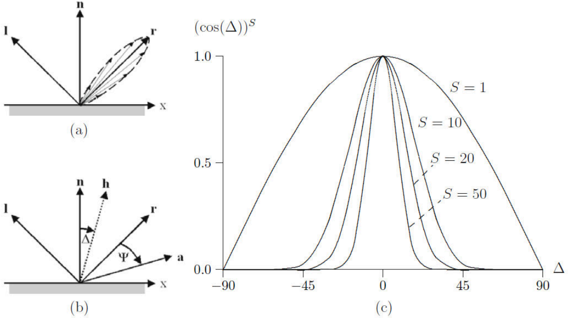
Specular Component (according to Phong)



© Hagler

Specular Component (according to Phong/Blinn)

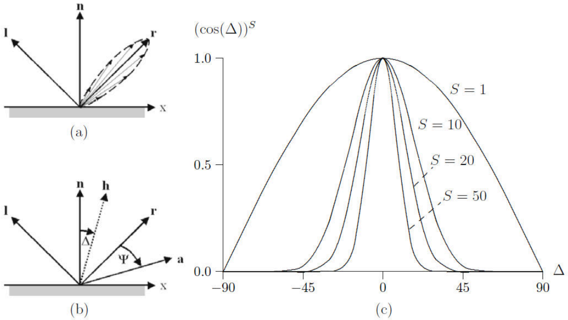
- Calculation of the reflecting vector r relatively complex: $r = 2(l \cdot n) \cdot n - l$
- Therefore, a *halfway vector* can be used for the calculation (*according to Blinn*) $h = (l + a)/|l + a|$



Nischwitz 2011 p.233

Specular Component (according to Blinn)

- By using the halfway vector, the angle Δ between n and h is considered instead of the angle between r and a
- Consequently, exponent S must be chosen *about four times as large* to achieve the same results as through Phong (approximated)



Nischwitz 2011 p.233

Specular Component (according to Blinn)

- Calculation similar to the Diffuse Component

$$\text{spekular} = (\max(\mathbf{h}\cdot\mathbf{n}, 0))^{\mathbf{S}} \cdot \mathbf{s}_{\mathbf{light}} * \mathbf{s}_{\mathbf{mat}} = (\max(\mathbf{h}\cdot\mathbf{n}, 0))^{\mathbf{S}} \begin{pmatrix} R_{\mathbf{s_{light}}} \cdot R_{\mathbf{s_{mat}}} \\ G_{\mathbf{s_{light}}} \cdot G_{\mathbf{s_{mat}}} \\ B_{\mathbf{s_{light}}} \cdot B_{\mathbf{s_{mat}}} \\ A_{\mathbf{s_{light}}} \cdot A_{\mathbf{s_{mat}}} \end{pmatrix}$$

- With the halfway vector $h = (l + a)/|l + a|$

(d1), $S = 10$ (d2), $S = 20$ (d3), $S = 50$

Nischwitz 2011 p.233

Lighting = Light Source · Material

- Setting all colors and brightness for
  - Light source
  - Material
  - Per light contribution: ambient, diffuse, specular, shininess
  - Per color: R, G, B
- Certain materials reflect certain light contributions in certain colors
  - e.g. Colored plastic shines White

Material	R_a, G_a, B_a, A_a	R_d, G_d, B_d, A_d	R_s, G_s, B_s, A_s	**S**
Schwarzes Plastik	.00, .00, .00, 1.0	.01, .01, .01, 1.0	.50, .50, .50, 1.0	32.0
Schwarzer Gummi	.02, .02, .02, 1.0	.01, .01, .01, 1.0	.40, .40, .40, 1.0	10.0
Messing	.33, .22, .03, 1.0	.78, .57, .11, 1.0	.99, .94, .81, 1.0	27.9
Bronze	.21, .13, .05, 1.0	.71, .43, .18, 1.0	.39, .27, .17, 1.0	25.6
Poliertes Bronze	.25, .15, .06, 1.0	.40, .24, .10, 1.0	.77, .46, .20, 1.0	76.8
Chrom	.25, .25, .25, 1.0	.40, .40, .40, 1.0	.77, .77, .77, 1.0	76.8
Kupfer	.19, .07, .02, 1.0	.70, .27, .08, 1.0	.26, .14, .09, 1.0	12.8
Poliertes Kupfer	.23, .09, .03, 1.0	.55, .21, .07, 1.0	.58, .22, .07, 1.0	51.2
Gold	.25, .20, .07, 1.0	.75, .61, .23, 1.0	.63, .56, .37, 1.0	51.2
Poliertes Gold	.25, .22, .06, 1.0	.35, .31, .09, 1.0	.80, .72, .21, 1.0	83.2
Zinn	.11, .06, .11, 1.0	.43, .47, .54, 1.0	.33, .33, .52, 1.0	9.8
Silber	.19, .19, .19, 1.0	.51, .51, .51, 1.0	.51, .51, .51, 1.0	51.2
Poliertes Silber	.23, .23, .23, 1.0	.28, .28, .28, 1.0	.77, .77, .77, 1.0	89.6
Smaragdgrün	.02, .17, .02, 0.5	.08, .61, .08, 0.5	.63, .73, .63, 0.5	76.8
Jade	.14, .22, .16, 0.9	.54, .89, .63, 0.9	.32, .32, .32, 0.9	12.8
Obsidian	.05, .05, .07, 0.8	.18, .17, .23, 0.8	.33, .33, .35, 0.8	38.4
Perle	.25, .21, .21, 0.9	.99, .83, .83, 0.9	.30, .30, .30, 0.9	11.3
Rubin	.17, .01, .01, 0.5	.61, .04, .04, 0.5	.73, .63, .63, 0.5	76.8
Türkis	.10, .19, .17, 0.8	.40, .74, .69, 0.8	.30, .31, .31, 0.8	12.8

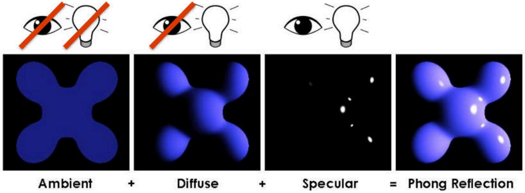
Nischwitz 2011 p.240

Conclusion: Local Lighting Model in OpenGL

- *Combination of all four light contributions forms the final color value* in RGBA mode

Vertexfarbe	3D-Anordnung		Lichtquelle		Material		Komponente
$\mathbf{g}_{Vertex} =$					$\mathbf{e}_{mat}$	$+$	emissiv
			$\mathbf{a}_{light}$	$*$	$\mathbf{a}_{mat}$	$+$	ambient
	$\max(\mathbf{l} \cdot \mathbf{n}, 0)$	$\cdot$	$\mathbf{d}_{light}$	$*$	$\mathbf{d}_{mat}$	$+$	diffus
	$(\max(\mathbf{h} \cdot \mathbf{n}, 0))^{\mathbf{S}}$	$\cdot$	$\mathbf{s}_{light}$	$*$	$\mathbf{s}_{mat}$		spekular

Nischwitz 2011 p.234



© Wikimedia Commons

Shading

Shading

- With lighting models, it is possible to determine the color of any point on a surface

Vertexfarbe	3D-Anordnung		Lichtquelle		Material		Komponente
$\mathbf{g}_{Vertex} =$					$\mathbf{e}_{mat}$	$+$	emissiv
			$\mathbf{a}_{light}$	$*$	$\mathbf{a}_{mat}$	$+$	ambient
	$\max(\mathbf{l} \cdot \mathbf{n}, \mathbf{0})$	$\cdot$	$\mathbf{d}_{light}$	$*$	$\mathbf{d}_{mat}$	$+$	diffus
	$(\max(\mathbf{h} \cdot \mathbf{n}, \mathbf{0}))^{\mathrm{S}}$	$\cdot$	$\mathbf{s}_{light}$	$*$	$\mathbf{s}_{mat}$		spekular

Nischwitz 2011 p.234

- However, points on a surface can have any number
  - Therefore, we need *explicit points* for calculation
- There are three different shading methods
  - Flat-Shading
  - Smooth-Shading (Gouraud-Shading)
  - Phong-Shading (not to confuse with the Phong lighting model)

Smooth-/Gouraud-Shading

- Eliminates jumps in color gradient
- Evaluation of lighting models *at each vertex* of an object
- *Interpolates color values between the calculated color values* (hence smooth transitions)
- The evaluation at each vertex is done using the vertices' normals

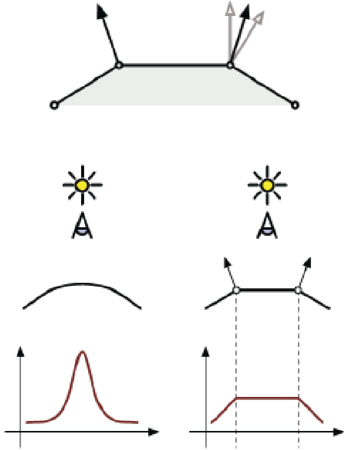
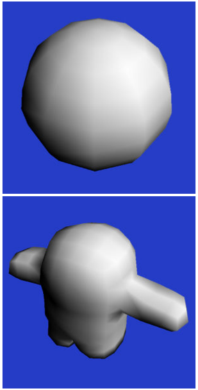


Abbildung 6.8.: Gouraud Shading (Tramberend).

Computer Graphics I. Frauke Sprengel, 2013, p.107

Smooth-/Gouraud-Shading

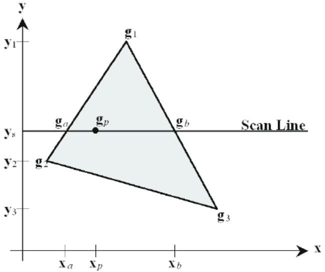
- Advantages
  - Appears *more realistic than Flat-Shading*. If only diffuse components occur, Smooth-Shading is sufficient
  - Not very computationally intensive
- Disadvantages
  - *Similar problems as Flat-Shading with large polygons*
  - Reflective reflections *not satisfactorily taken into account*



Linear interpolation between color values

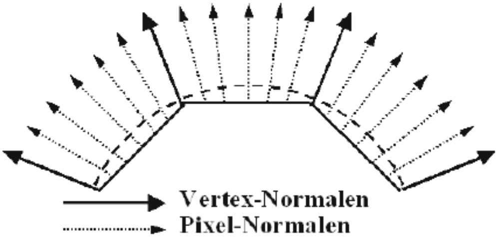
- Smooth-Shading
  - = Linear Interpolation, to determine the area color values
  - Takes place on projected faces
- Scan Line Algorithm
  - Determination of color values of individual pixels g_p through *linear interpolation of color values*
  - Between vertices along the surface edges g_1, g_2 and g_3
  - Then between edges along the *Scan Lines* concerning the *intersection points* g_a and g_b
- This task is *done by the Rasterizer*

$$
\begin{aligned}
g_a &= g_1 - (g_1 - g_2) \cdot \frac{y_1 - y_s}{y_1 - y_2} \\
g_b &= g_1 - (g_1 - g_3) \cdot \frac{y_1 - y_s}{y_1 - y_3} \\
g_p &= g_b - (g_b - g_a) \cdot \frac{x_b - x_p}{x_b - x_a}
\end{aligned}
$$



Phong-Shading

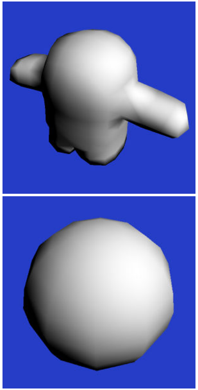
- Evaluation of lighting models *at each pixel of the faces*
- Interpolates *from the vertex normal the normal vector* at the pixel (here, too, with the *Scan Line*)
- Evaluation of lighting models, therefore, takes place *not in the Vertex-Shader but in the Fragment-Shader*



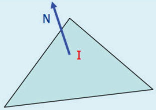
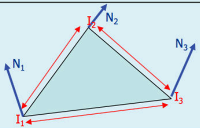
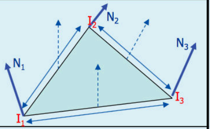
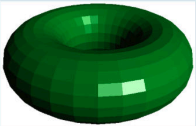
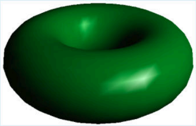
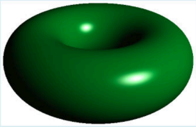
Nischwitz 2011 p.257

Phong-Shading

- Advantages
  - *Eliminates the problems of Smooth-Shading*
  - For many procedures, it provides *very good results* (e.g., Bump-Mapping)
- Disadvantages
  - *More computationally intensive than all other shading procedures* (since evaluations take place for every pixel)
  - Often has almost identical results as Smooth-Shading for Objects with a sufficiently high number of vertices (without reflective reflection)



Comparison of Shading Techniques

Flat-Shading	Gouraud-Shading	Phong-Shading
		
Ein Helligkeitswert für das gesamte Polygon. *Anwendung:* Entwurfs-ansichten in CAD- und Modellierungsprogrammen.	Bestimme Helligkeit an den Eckpunkten. Interpoliere im Inneren.	Bestimme Normalen an den Eckpunkten und interpoliere sie im Inneren. Bestimme Helligkeit mit Hilfe der interpolierten Normalen.
		

Lighting in OpenGL

Smooth-Shading in OpenGL

- Evaluation of lighting in the Vertex-Shader
- Transfer of color values with `out vec4 vColor;`
  - Rasterizer performs interpolation
- Only assignment of color $\rightarrow$ pixel in Fragment-Shader

Phong-Shading in OpenGL

- Transfer of the *transformed* normals and the transformed vertices' positions from Vertex-Shader to Fragment-Shader (*usually in View coordinates*)
  ```
  out vec3 Normal;
  out vec3 Position;
  ```
- Evaluation of lighting in Fragment-Shader

Example Phong-Shading with Blinn-Lighting

In the *Vertex-Shader*

1. Calculation of gl_Position
2. Calculation of the parameters *necessary for the lighting in FragmentShader*:
   - Position – Transforms each vertex *in View-Space*
   - Normal – Transforms normals with the Normal-Matrix (*inverse, transposed 3x3 ModelView-Matrix*)

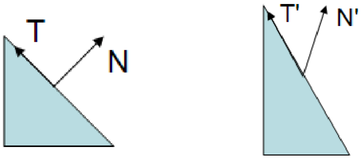
Calculation of the *Normal Matrix outside the shader*:

```
#include <glm/gtx/inverse_transpose.hpp>
gl_NormalMatrix =
  glm::inverseTranspose(
    glm::mat3(gl_ModelViewMatrix));
```

```
// Eingabe-Werte pro Vertex
in vec4 vVertex;    // Vertex-Position in Objektkoordinaten
in vec3 vNormal;    // Normalen-Vektor in Objektkoordinaten

// Uniform-Eingabe-Werte
uniform mat4 MV;          // ModelView-Matrix
uniform mat4 MVP;         // ModelViewProjection-Matrix
uniform mat3 NormalM;     // Normalen-Matrix

// Ausgabe-Werte
smooth out vec3 Position;  // Vertex-Position in Augenpunktskoordinaten
smooth out vec3 Normal;    // Normalen-Vektor in Augenpunktskoordinaten

void main(void)
{
    // Vertex aus Objekt- in Projektionskoordinaten
    gl_Position = MVP * vVertex;
    // Vertex aus Objekt- in Augenpunktskoordinaten
    vec4 Pos = MV * vVertex;
    Position = Pos.xyz / Pos.w;
    // Vertex-Normale aus Objekt- in Augenpunktskoordinaten
    Normal = normalize(NormalM * vNormal);
}
```

Normal-Matrix

- Normals cannot be transformed like positions
  - *Rotations* are not a problem
  - *Translations* are not a problem, if $w = 0$ (then the 4th column of the matrix has no effect)
  - All other transformations are a problem
- Instead of matrix M, we use $N = (M^{-1})^T$
  - For rotations, $M^{-1} = M^T$



http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/the-normal-matrix/

Example Phong-Shading with Blinn-Lighting

In the *Fragment-Shader*

- Accept *vertices* with position and normal
- Accept *parameters of lighting components*
  - *Material* properties
  - *Light source* properties
- Calculation of the Fragment-Color
- In older GLSL versions Avoid uniform-struct
  - Pass individual uniforms

```
// linear interpolierte Eingabe-Werte pro Fragment
in vec3 Position;   // Fragment-Position in 3D
in vec3 Normal;     // Normalen-Vektor

// Uniform-Block für Material-Eigenschaften
uniform MaterialParams{
    vec4 emission;
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    float shininess; } Material;

// Uniform-Block für Lichtquellen-Eigenschaften
uniform LightParams{
    vec4 position;
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    vec3 halfVector } LightSource;
```

Example Phong-Shading with Blinn-Lighting

In the *Fragment-Shader*

- *Calculation of the Fragment-Color* by applied lighting

Vertexfarbe	3D-Anordnung		Lichtquelle		Material		Komponente
$\mathbf{g}_{Vertex} =$					$\mathbf{e}_{mat}$	$+$	emissiv
			$\mathbf{a}_{light}$	$*$	$\mathbf{a}_{mat}$	$+$	ambient
	$\max(\mathbf{l}\cdot\mathbf{n},0)$	$\cdot$	$\mathbf{d}_{light}$	$*$	$\mathbf{d}_{mat}$	$+$	diffus
	$(\max(\mathbf{h}\cdot\mathbf{n},0))^{\mathbf{s}}$	$\cdot$	$\mathbf{s}_{light}$	$*$	$\mathbf{s}_{mat}$		spekular

- GLSL offers corresponding mathematical functions for this

```
void main(void)
{
    // Berechnung des Phong-Blinn-Beleuchtungsmodells
    vec3 N = normalize(Normal);
    vec4 emissiv = Material.emission;
    vec4 ambient = Material.ambient * LightSource.ambient;
    vec3 L = vec3(0.0);     // alle drei Komponenten werden auf 0.0 gesetzt
    vec3 H = vec3(0.0);
    if(LightSource.position.w == 0){
        L = normalize(vec3( LightSource.position));
        H = normalize( LightSource.halfVector);
    } else {
        L = normalize(vec3( LightSource.position) - Position);
        // Annahme eines infiniten Augenpunkts:
        // somit zeigt der Vektor A zum Augenpunkt immer in z-Richtung
        vec4 Pos_eye = vec4(0.0, 0.0, 1.0, 0.0);
        vec3 A = Pos_eye.xyz;
        H = normalize(L + A);
    }
    vec4 diffuse = vec4(0.0, 0.0, 0.0, 1.0);
    vec4 specular = vec4(0.0, 0.0, 0.0, 1.0);
    float diffuseLight = max(dot(N, L), 0.0);
    if (diffuseLight > 0) {
        diffuse = diffuseLight * Material.diffuse * LightSource.diffuse;
        float specLight = pow(max(dot(H, N), 0), Material.shininess);
        specular = specLight * Material.specular * LightSource.specular;
    }
    FragColor = emissiv + ambient + diffuse + specular;
}
```