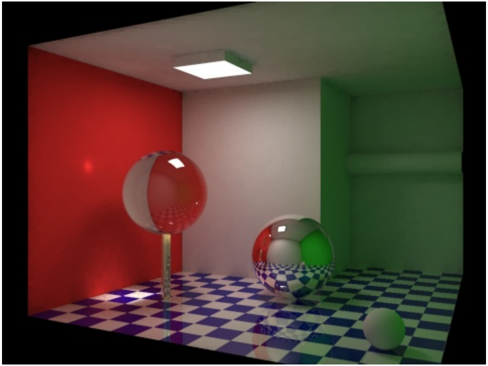
Basics for Enhanced Visualization: 3D/Data

Color, lighting and texture



Rodrigo Cabral

Polytech Nice - Data Science

cabral@unice.fr

Source: https://upload.wikimedia.org/wikipedia/commons/0/0d/Global_illumination.JPG

Outline

1. Introduction
2. Color
3. Lighting
   - Global lighting
   - Phong model
   - Diffusion, specular, ambient and emissive terms
   - OpenGL syntax
4. Texture
5. Conclusions

These slides were partially inspired from Lionel Fillatre's slides, from `http://web.eecs.umich.edu/~sugih/courses/eecs487/lectures/16-Phong+Shading.pdf` and `http://www.eng.utah.edu/~cs5600/slides/Wk20920Texture20mapping20OpenGL.pdf`

Introduction

What to do we need to set beyond geometry to have a realistic 3D rendering?

- A lot of things!
  - Color.
  - Illumination: reflection, refraction, emission...
  - Simulation of defocused objects.
  - Presence of fog.
  - Textured surface.
  - ...

Introduction

What to do we need to set beyond geometry to have a realistic 3D rendering?

- In this class:
  - Color.
  - Simple illumination sources, reflection and emission.
  - Flat texture patterns.

OpenGL primitive and vertices attributes

- Attributes are part of OpenGL state and determine the appearance of the objects:
  - Color (points, lines and surfaces).
  - Size and width (points and lines).
  - Stipple pattern (lines).
  - Polygon mode (show edges and vertices points).

- **glColor** sets the color state. OpenGL uses the set color until we reset it with **glColor**.
- Within OpenGL colors are not directly related to an object but are assigned to them when rendering occurs.

OpenGL primitive and vertices attributes

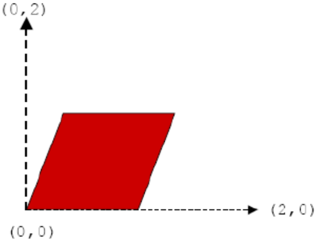
- We can create conceptual colored vertices by changing each time the state:

```
glBegin( primType )
for i in range( len( vertices ) )
glColor3f( red[i] , green[i] , blue[i] )
glVertex3fv( vertices[i] )
glEnd( )
```

Color

An example

```
def drawParallelogram( color ):
glBegin( GL_QUADS )
glColor3fv( color )
glVertex2f( 0.0 , 0.0 )
glVertex2f( 1.0 , 0.0 )
glVertex2f( 1.5 , 1.118 )
glVertex2f( 0.5 , 1.118 )
glEnd( )
```

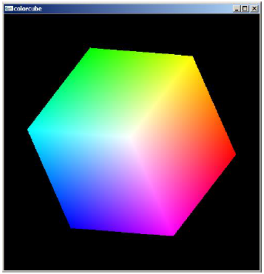


RGB color

- Each color component is stored separately in the frame buffer: a color is an array with three elements.
- Usually 8 bits per color component.
- Note that in **glColor3f** components are given in the range 0.0 to 1.0 whereas in **glColorub** in the range 0 to 255.

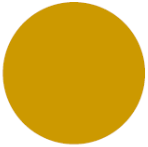
Smooth color

- Default is **smooth** shading.
  - OpenGL interpolates vertex colors across visible polygons.
- Alternative is **flat** shading.
  - Color of first vertex determines primitive color.
- **glShadeModel(GL_SMOOTH)** or **GL_FLAT**.



Why do we need to simulate lighting?

- A part of 3D perception comes also from lighting!
- Suppose we want to build a sphere in 3D, with **glColor** we get



- But in a environment with real light we get something like this

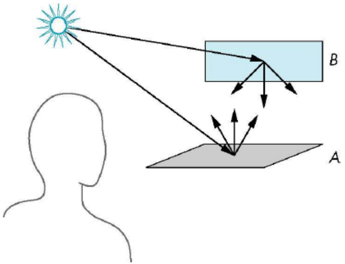
Real appearance with lighting

- Why does the image of a sphere look like this?

- Light-material interactions cause the image projection of each point in the object to have a different color shade.
- We need to consider
  - light sources,
  - material properties,
  - location of viewer,
  - surface orientation.

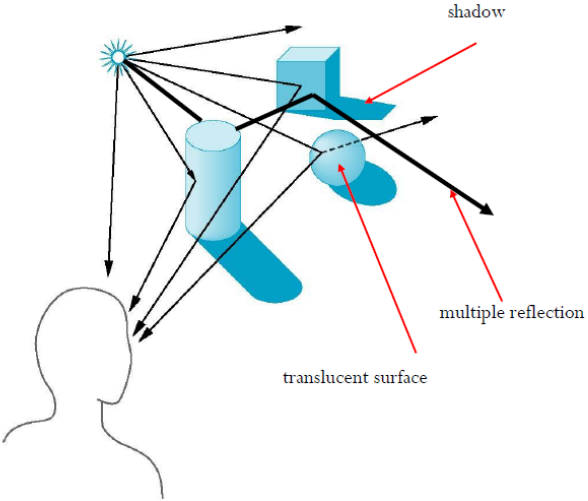
Ray scattering

- Light rays strike A
  - Some are absorbed.
  - Some are scattered.
- Scattered rays strike B
  - Some are absorbed.
  - Some are scattered.
- Scattered rays strike A
- and so on...



Lighting

Global effects



Global vs. local rendering

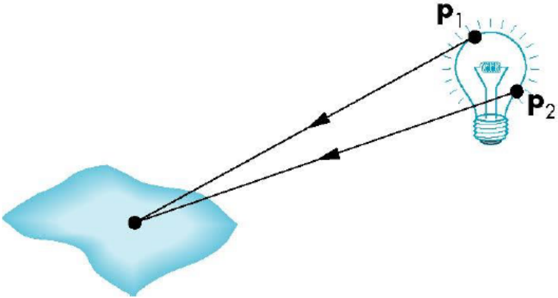
- Correct shading requires a global calculation involving all objects and light sources.
- Incompatible with pipeline model which shades each polygon independently (local rendering).

Global vs. local rendering

- Correct shading requires a global calculation involving all objects and light sources.
- Incompatible with pipeline model which shades each polygon independently (local rendering).
- However, in computer graphics, especially real time graphics, we are happy if things “look right".
- Many local techniques for approximating realistic lighting exist. We will see one of them in a few slides.

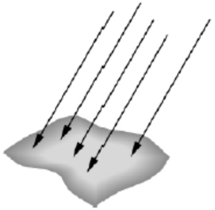
Light sources

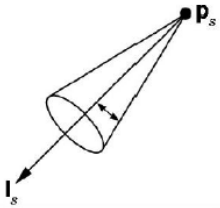
- General light sources are difficult to work with because we must integrate light coming from all points on the source.



Light sources

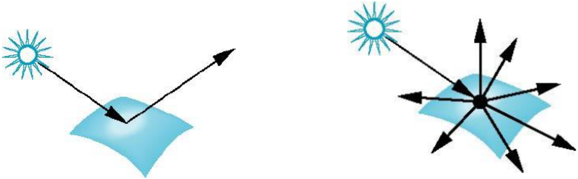
- Point source:
  - Model with position and color.
  - Distant source = infinite distance away (parallel).
- Spotlight:
  - Restrict light from ideal point source.
- Ambient light:
  - Same amount of light everywhere in the scene.
  - It models approximately all the scattering a local model cannot approximate.





Surface types

- The smoother a surface is, the more reflected light is concentrated in the direction a perfect mirror would reflect the light.
- A very rough surface scatters light in all directions.



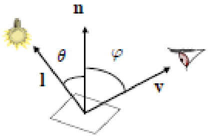
smooth surface

rough surface

Phong lighting model

- A local illumination model.
  - One ray bounce: light $\Longrightarrow$ surface $\Longrightarrow$ viewer.
- Lighting at a single point on a surface:

- **n**: surface normal (orientation of surface).
- **l**: light vector (surface to light).
- **v**: viewing vector (surface to eye).
- θ: light angle of incidence.
- ϕ: viewing angle.

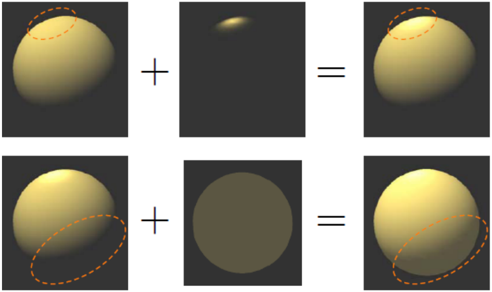


Phong lighting model

- Reflected light intensity is approximated by the sum of 3 components:

- An ideal diffuse component.
- A glossy/blurred specular component.
- An ambient component

3 light sources		
$\mathbf{s}_d$	diffuse	$\left[s_d^r\ s_d^g\ s_d^b\right]$
$\mathbf{s}_s$	specular	$\left[s_s^r\ s_s^g\ s_s^b\right]$
$\mathbf{s}_a$	ambient	$\left[s_a^r\ s_a^g\ s_a^b\right]$

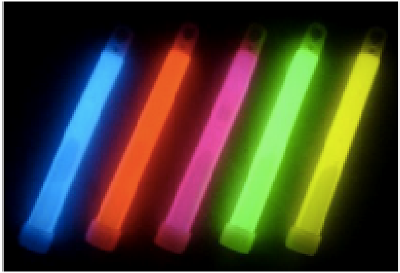


Phong lighting model + emissive term

- Reflected light intensity is approximated by the sum of 3 components.

- We add an emissive term for glowing objects.
- The emitted light does not interact with other objects.

$\mathbf{s}_e = \mathbf{m}_e$	emissive	$\left[m_e^r \; m_e^g \; m_e^b\right]$



Phong lighting model + emissive term

- Approximation is not physically based. But in most cases it “looks right".
- The perceived colors can be modified by changing the reflection coefficients **m** of the surface based on:

- Material type.
- Surface finish.
- What looks good...

$\mathbf{m}_d$	diffuse	$\left[m_d^r\ m_d^g\ m_d^b\right]$
$\mathbf{m}_s$	specular	$\left[m_s^r\ m_s^g\ m_s^b\right]$
$\mathbf{m}_a$	ambient	$\left[m_a^r\ m_a^g\ m_a^b\right]$
$\mathbf{m}_{shi}$	shininess	m_{shi}
$\mathbf{m}_e$	emissive	$\left[m_e^r\ m_e^g\ m_e^b\right]$

- All these coefficients are defined in the range $[0.0, 1.0]$, except m_{shi} which is defined in the interval $[0.0, 128.0]$.

Lighting: diffusion

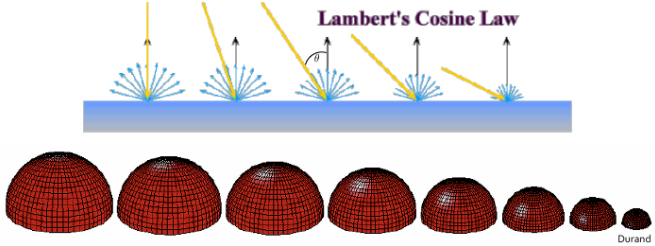
Diffusion example

Where is the light?

Where's the normal direction at the brightest point?

Ideal diffuse reflection

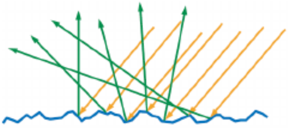
- Ideal diffuse surface reflects light equally in all directions, according to Lambert's cosine law:
  - Amount of light energy that falls on surface and gets reflected is proportional to the incidence angle θ.
  - Perceived brightness is view independent.



Lighting: diffusion

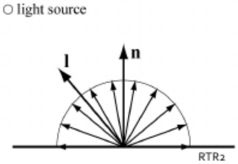
Ideal diffuse reflection and model

- At microscopic level an ideal diffuse surface is a rough surface.



- Energy that falls on surface depends on incident angle. For a given color channel:

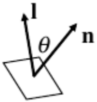
$$c_d = m_d\, s_d\, \cos(\theta)$$



- For normalized **n** and **l** we have:

$$c_d = m_d\, s_d\, \max(\mathbf{n}\cdot\mathbf{l}, 0)$$

- Why do we need $\max(\mathbf{n}\cdot\mathbf{l}, 0)$?

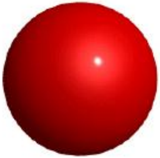


Lighting: specular

Specular/mirror reflection

Accounts for highlight seen on objects with smooth, shiny surfaces, such as:

- metal
- polished stone
- plastics
- apples
- skin



Curless,Zhang

Lighting: specular

Ideal specular/mirror reflection

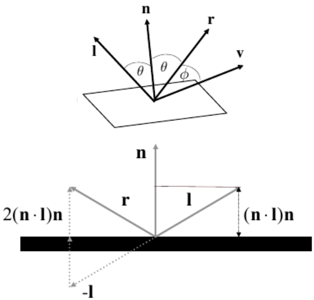
- Reflection only at mirror angle: highlight intensity depends on viewing direction.
- Model: all micro facets of mirror surface are oriented in the same direction as the surface itself.
- Examples: mirrors, highly polished metals.
- However, in Phong model there is no second bounce $\Longrightarrow$ no mirroring of the scene.

Durand

Phong specular reflection

- Simulates a highlight at reflection angle equal to incidence angle.
- Most intense specular reflection at $\mathbf{v} = \mathbf{r}$.
- Evaluation of r:

$$\mathbf{r} = -\mathbf{l} + 2(\mathbf{n} \cdot \mathbf{l})\mathbf{n}$$



Lighting: specular

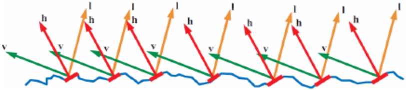
Glossy specular reflector

- Real materials tend to deviate significantly from ideal mirror reflectors (this is also true for diffusion).
- Consequence: highlight and reflections are blurry. This is also known as “rough specular", “directional diffuse" or “glossy" reflection.

Durand

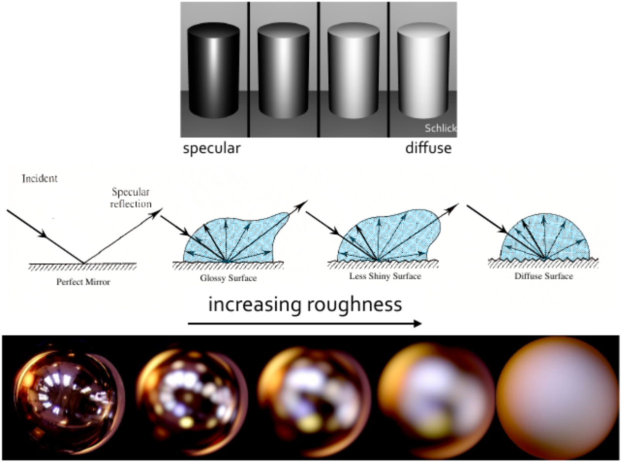
Glossy specular reflector

- Approximation model:
  - Most light is reflected on the ideal direction.
  - Small variations on micro facet orientations will reflect rays in slightly different angles.
  - As the angle variation increase from the ideal reflections angle, the reflected light intensity decreases.



Lighting: specular

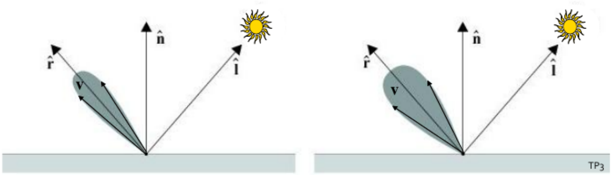
Surface roughness: from specular to diffusive reflection



Lighting: specular

Phong “glossy" specular reflection

- As **v** angles away from **r**, specular reflection falls off, simulating “glossy" reflection:



less glossy/
more specular

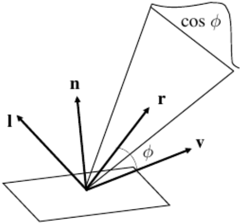
more glossy/blurry

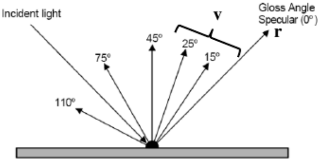
Lighting: specular

Phong “glossy" specular reflection

- Reflected light intensity for a given color channel is

$$c_s = m_s\, s_s \cos(\phi) = m_s\, s_s \max(\mathbf{r} \cdot \mathbf{v}, 0)$$





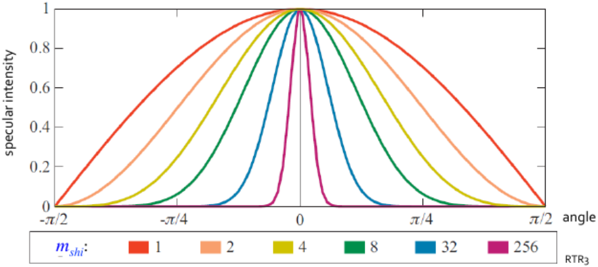
Lighting: specular

Phong “glossy" specular reflection

- To take into account the shininess of the material we introduce the coefficient m_{shi}:

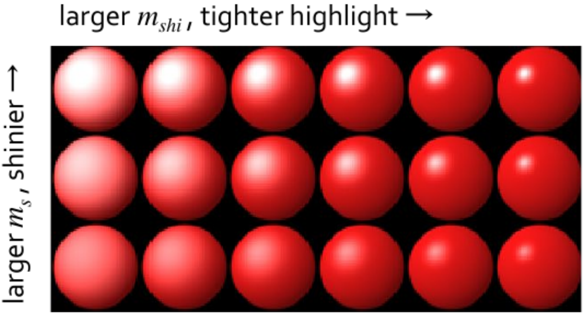
$$c_s = m_s \, s_s \, \max(\mathbf{r} \cdot \mathbf{v}, 0)^{m_{shi}}$$

- The larger m_{shi} is, the tighter and shinier is the highlight.



Lighting: specular

Phong "glossy" specular reflection



Lighting: specular

Highlight color

- For metals, highlight color is of the same color of the material: specular coefficients follow the material color.
- For non-metals, for example plastics, highlight color is the same as the source color: specular coefficients correspond to gray or white.

plastic

metal

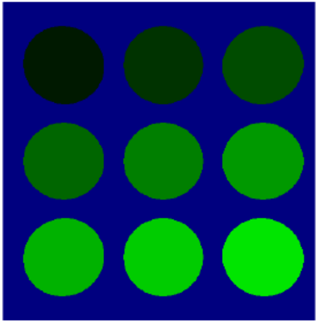
clay

Lighting: ambient

Phong ambient term

- This term approximated all the indirect reflections (all further ray bounces).

- Surfaces are uniformly lit.
- Areas with no direct illumination are not dark.
- Lighting is independent of light, surface normal and viewing directions



Light attenuation model

- Attenuation model simulates scattering effect: light falls off as get away from the source.
- Radiant energy attenuates $\propto \dfrac{1}{d^2}$, where d is the distance from the source (if the source is not at infinity).
- Attenuation function

$$f(d) = \frac{1}{a_0 + a_1 d + a_2 d^2}$$

  in most cases $a_2 = 0$ or no attenuation is considered.

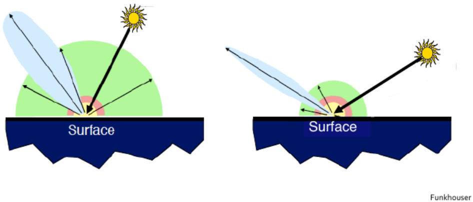
- Reflected light

$$c'(d) = c\, f(d)$$

Complete model

- Considering all terms for a source and a color channel

$$c = m_e + \left[m_a s_a + m_d s_d (\mathbf{n} \cdot \mathbf{l}) + m_s s_s (\mathbf{v} \cdot \mathbf{r})^{m_{shi}} \right] f(d)$$



- For multiple light sources

$$c = m_e + \sum_{i=1}^{K} \left[m_a s_a^i + m_d s_d^i (\mathbf{n} \cdot \mathbf{l}^i) + m_s s_s^i (\mathbf{v} \cdot \mathbf{r}^i)^{m_{shi}} \right] f(d^i)$$

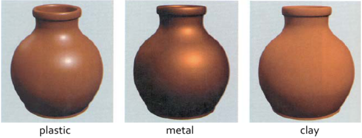
Complete model: choosing surface coefficients

- Try different sets of coefficients to get correct material appearance.

Suggestions:

- $m_d + m_s + m_a < 1$.
- Use a small m_a ($\approx$ 0.1).
- Try $m_{shi} \in [0, 100]$.

TP3



Apodaca&Gritz

Material	m_d	m_s	m_{shi}
Metal	small, color of metal	large, color of metal	large
Plastic	medium, color of surface	medium, color of light	medium
Clay	color of surface	0	0

Examples: `http://www.it.hiof.no/~borres/j3d/explain/light/p-materials.html`.

OpenGL syntax

- Steps in OpenGL:

  1. Enable lighting and select lighting model.
  2. Specify normals.
  3. Specify material properties.
  4. Specify lights.

OpenGL syntax: 1 - Enable lighting

- Lighting calculations are enabled by
  - **glEnable(GL_LIGHTING)**.
  - Once lighting is enable **glColor** is ignored.
- You must enable each light source individually.
  - **glEnable(GL_LIGHTi)**, $i = 1, \cdots, K$.
- You can choose light model parameters.
  - **glLightModeli(parameter,GL_TRUE)**
    - **GL_LIGHT_MODEL_LOCAL_VIEWER** does not use simplifying distant viewer assumption.
    - **GL_LIGHT_MODEL_TWO_SIDED** shades both sides of polygons independently.
  - You can also set a global ambient light.
    - **glLightModelfv(GL_LIGHT_MODEL_AMBIENT,global_ambient)**

OpenGL syntax: 2 - Specify normals

- In OpenGL the normal vector is part of the state and should be set by the application.
  - Set by **glNormal3f(x, y, z)**.
  - Or **glNormal3fv(n)**.
- Usually we want to set the normal to have unit length so cosine calculations are correct.
  - Length can be affected by non-rigid transformations.
  - Note that scaling does not preserved length.

OpenGL syntax: 3 - Material properties

- Material properties are also part of the OpenGL state and match the coefficients in the Phong model.
- Set by **glMaterialv(...)**:

$$
\begin{aligned}
&\text{ambient} = [0.2, 0.2, 0.2, 1.0]\\
&\text{diffuse} = [1.0, 0.8, 0.0, 1.0]\\
&\text{specular} = [1.0, 1.0, 1.0, 1.0]\\
&\text{shininess} = 100.0\\
&\text{emission} = [0.0, 0.8, 0.2, 1.0]
\end{aligned}
$$

```
glMaterialv(GL_FRONT, GL_AMBIENT, ambient)
glMaterialv(GL_FRONT, GL_DIFFUSE, diffuse)
glMaterialv(GL_FRONT, GL_SPECULAR, specular)
glMaterialv(GL_FRONT, GL_SHININESS, shininess)
glMaterialv(GL_FRONT, GL_EMISSION, emission)
```

OpenGL syntax: 4 - Specify lights

- Defining a **point source**: for each light source we can specify RGB values for the diffuse, specular and ambient sources. We can also set the light position.

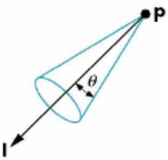
$$\text{ambient0} = [0.1, 0.1, 0.1, 1.0]$$
$$\text{diffuse0} = [1.0, 1.0, 1.0, 1.0]$$
$$\text{specular0} = [1.0, 1.0, 1.0, 1.0]$$
$$\text{light\_pos0} = [1.0, 2.0, 3.0, 1.0]$$

```
glEnable(GL_LIGHTING)
glEnable(GL_LIGHT0)
glLightv(GL_LIGHT0, GL_POS, light_pos0)
glLightv(GL_LIGHT0, GL_DIFFUSE, diffuse0)
glLightv(GL_LIGHT0, GL_SPECULAR, specular0)
glLightv(GL_LIGHT0, GL_AMBIENT, ambient0)
```

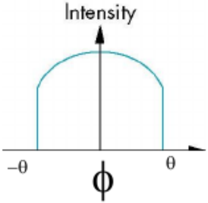
- The position is given in homogeneous coordinates. If $w = 0.0$ the source is at infinity in the specified directional vector.

OpenGL syntax: 4 - Specify lights

- Defining a **spotlight**: all previous parameters plus



- Direction: **d**,
  GL__DIRECTION.
- Angular attenuation exponent: α,
  GL__EXPONENT.
- Angular cutoff: θ,
  GL__CUTOFF.

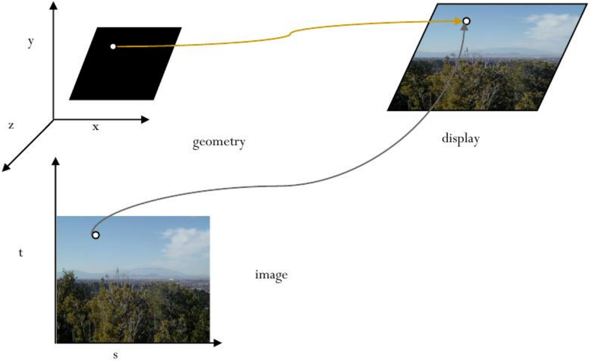


Texturing basic strategy

- Three steps to apply a texture:
  1. Specify the texture.
     - Read or generate image.
     - Assign to texture.
     - Enable texturing
  2. Specify texture parameters.
     - Wrapping and filtering.
  3. Assign texture coordinates to vertices.
     - Proper mapping function is left to application.

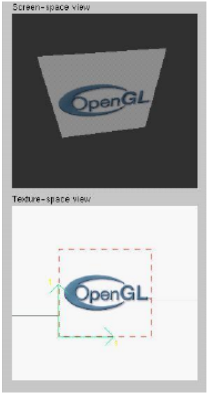
Texture

Texture mapping



Texture example

- The texture is a 256 $\times$ 256 image that has been mapped to a rectangular polygon which is viewed in perspective.



Texturing: 1 - Specify the texture

- Read the bitmap image and transform into bytes array:

```
global texture
from PIL import Image
image = Image.open("image.bmp")
ix, iy = image.size
ix = image.size[0]
iy = image.size[1]
image = image.tobytes("raw", "RGBX", 0, -1)
```

Texturing: 1 - Specify the texture

- Create texture and assign image to it:

```
global texture
glGenTextures(1, texture)
glBindTexture(GL_TEXTURE_2D, texture)
glPixelStorei(GL_UNPACK_ALIGNMENT,1)
glTexImage2D(GL_TEXTURE_2D, type of texture
             0, level used for mipmapping
             3, number of elements per texel
             ix, iy, width, height of texels in pixels
             0, width, height of texels in pixels
             GL_RGBA, OpenGL format
             GL_UNSIGNED_BYTE, texel type
             image texel array)
```

Texturing: 1 - Specify the texture

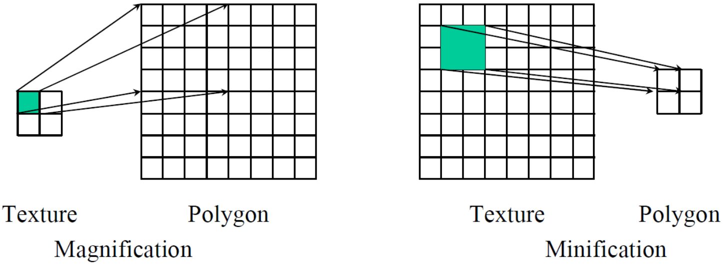
- OpenGL requires texture dimensions to be powers of 2.
- If dimensions are not powers of 2, you can rescale the image:

```
gluScaleImage(format, w_in, h_in, type_in,
              data_in, image to be rescaled
              w_out, h_out, type_out,
              data_out, output image)
```

- Image is interpolated and filtered during rescaling.
- Enable texturing with **glEnable(GL_TEXTURE_2D)**.

Texturing: 2 - Specify texturing parameters

- Texture needs to be interpolated if magnification (zoom in) occurs and down sampled if minification (zoom out) occurs.



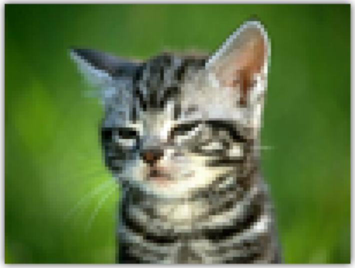
Texturing: 2 - Specify texturing parameters

- Two interpolation/down sampling methods can be chosen: nearest neighbor and linear interpolation.
- Examples:
  - **glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST)**: for magnification with nearest neighbor interpolation.
  - **glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_LINEAR)**: for minification with linear interpolation.

Texture

Texturing: 2 - Specify texturing parameters

- Example of a texture rasterized on a 16 times larger GL_QUAD.



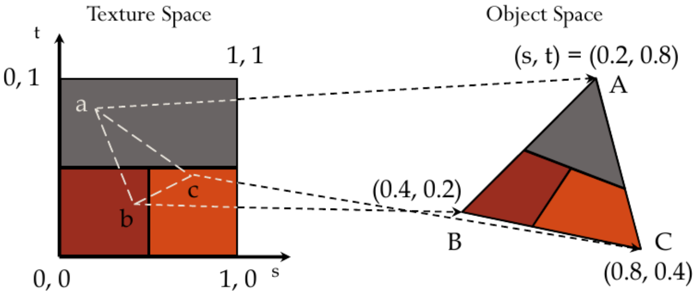
GL_NEAREST

GL_LINEAR

Source: `https://open.gl/textures`

Texturing: 3 - Assign texture coordinates

- To map part of a texture to a primitive surface we use texture space coordinates (s, t). Note that $(s, t) \in [0, 1]^2$.
- **glTexCoordf(...)** specified for each vertex.



Texture

Texturing: 3 - Assign texture coordinates

- Typical code:

```
def drawTexturedPrimitive( color ):
glBegin( PRIMITIVE_TYPE )
glColor3fv( color_0 ) # If no lighting.
glNormal3fv( normal_0 ) # If lighting is used.
glTexCoord2f( s_0 , t_0 ) # Coordinates in texture space
glVertex3fv( vertex_0 ) # Vertex coordinates
⋮
glEnd( )
```

Conclusions

- Only color attributes may generate too simple objects $\Longrightarrow$ we can add lighting and texturing to have more realistic 3D rendering.
- Global lighting is in general very difficult to code and to render in real-time $\Longrightarrow$ Phong lighting model is often used with real-time constraints.
- Phong reflection model is local and non physical. But is often sufficient for real-time applications: augmented/virtual reality and simple games.

Conclusions

- Other effects can be added, with ray tracing for example, but they are not embedded in OpenGL: shadows, mirroring reflection and refraction.
- Texture adds a further level of realism. Some texture types can be used to simulate a background layer: **GL_TEXTURE_CUBE_MAP**.