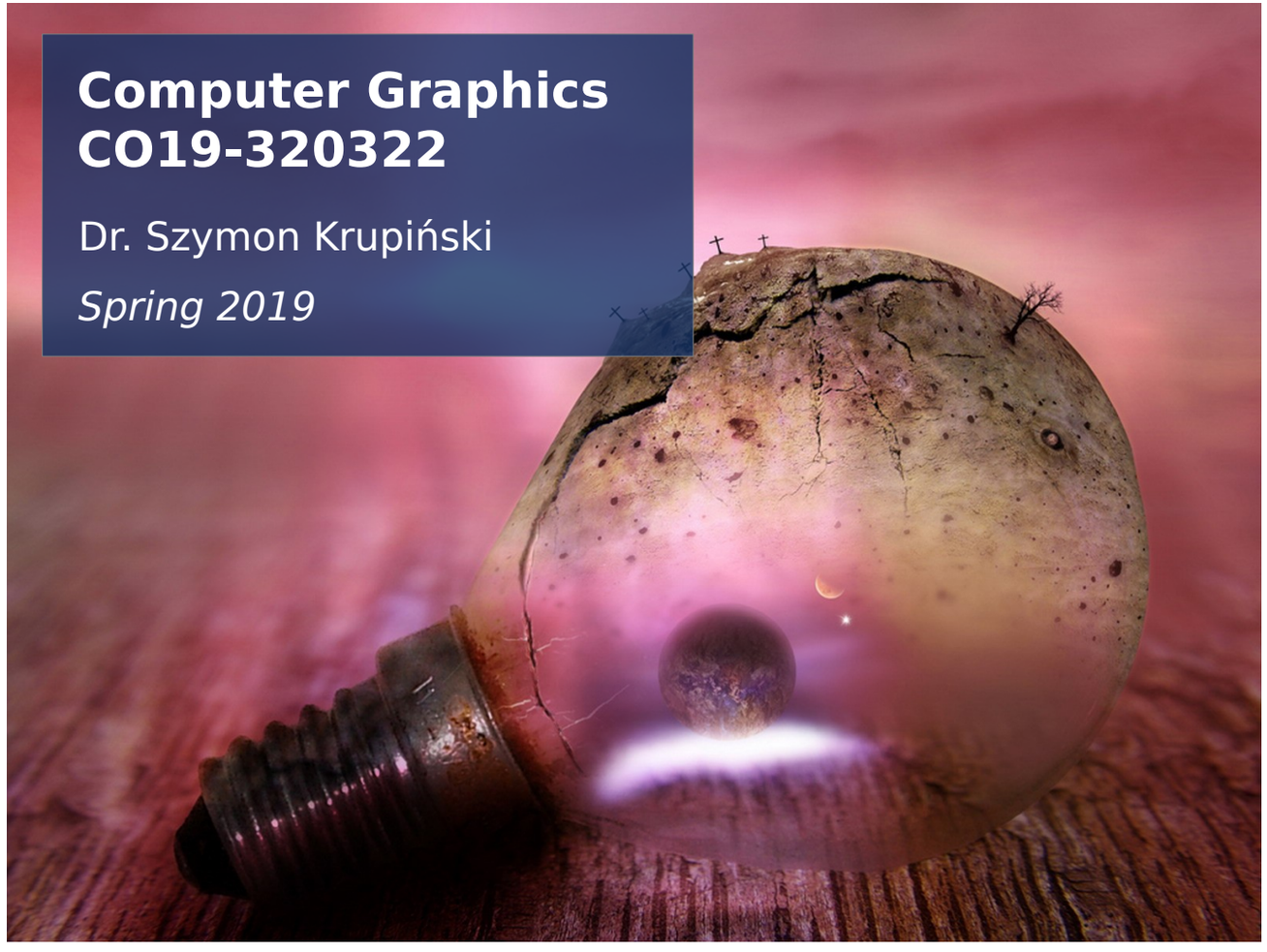
Lecture 15: OpenGL color, lighting and texturing 2

JACOBS UNIVERSITY



Reminder – your programming resources

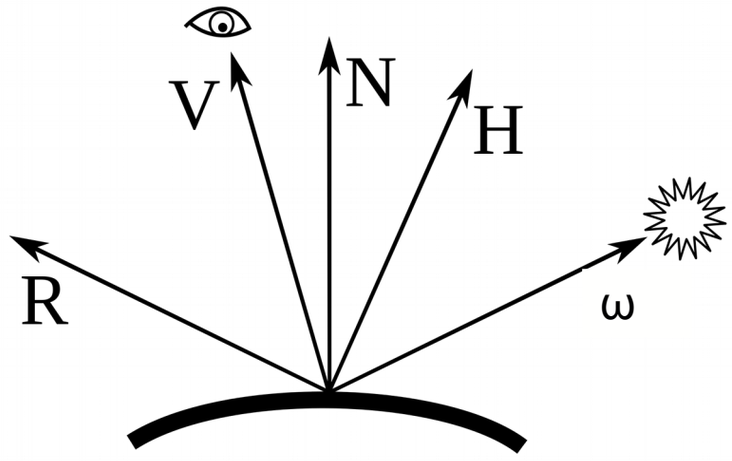
- Many interesting links to good quality explanations are posted on moodle
- ...including youtube tutorials
- ...and a series of programs which will allow you to do amusing things with relatively beginner CG skills:

  http://cs.lmu.edu/~ray/notes/openglexamples/
- Make use of them!

Phong – more details

- Lambertian – diffuse term

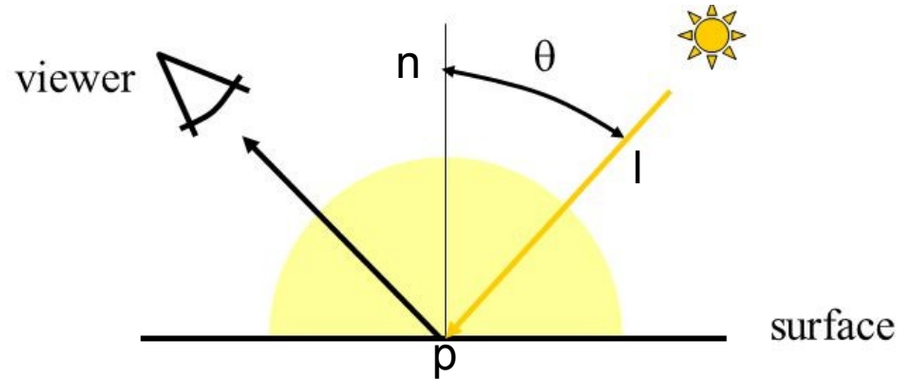
$$L_o = \sum_{j \in lights} \left(\underbrace{k_a \hat{I}^j_{i,a}}_{\text{Ambient}} + \underbrace{k_d \hat{I}^j_{i,d} \max(0, \omega_{i,d} \cdot \hat{\mathbf{N}})}_{\text{Diffuse}} + \underbrace{k_s \hat{I}^j_{i,s} \max(\mathrm{V} \cdot \mathrm{R}^j, 0)^s}_{\text{Specular}} \right)$$



  - Diffuse reflection denotes light scattering at a dull surface
  - The incoming directional light is reflected uniformly in each direction
  - The intensity of the reflected light is independent of the viewing angle
  - It depends on the direction of the incoming light ray

Phong – more details

- Lambert‘s law



The reflected light intensity I_d at a point p on a surface is proportional to $\cos(\theta)$, where θ is the angle between the vector l pointing from p to the light source and the surface normal n in point p

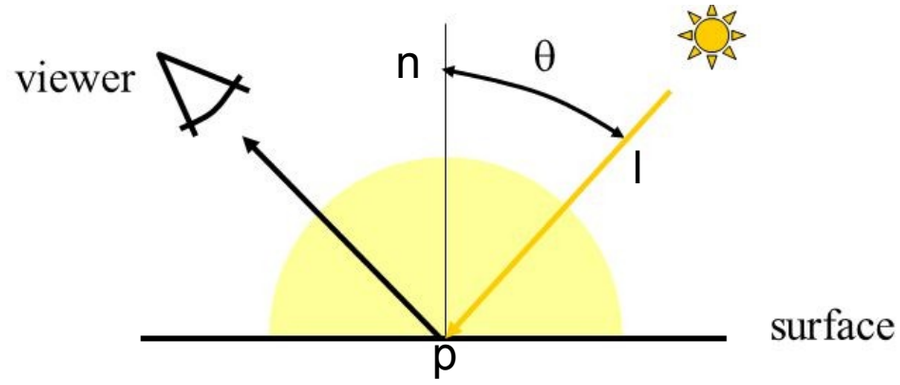
$$I_d = k_d I_L \cos(\theta)$$

More convenient to express as a dot product!

Phong – more details

- This model assumes that the light source sends out directed light and is of infinitely small extent (point light source)
- I_d must be 0 if $n \bullet l < 0$
  - To take this into account, we use the max function

$$I_d = I_P \cdot k_d \cdot \max(n \bullet l, 0)$$



OpenGL lighting model

- Define the surface properties of a primitive

```
glMaterialfv( face, property, value );
```

- separate materials can be applied to front and back of a face
  - `GL_FRONT, GL_BACK or GL_FRONT_AND_BACK`
- Scalar values for parametrising the B-P reflection model
  - `GL_DIFFUSE` Diffuse refl. coeff.
  - `GL_SPECULAR` Specular refl. coeff.
  - `GL_AMBIENT` Ambient refl. coeff.
  - `GL_AMBIENT_AND_DIFFUSE` Two coeff's at the same time
  - `GL_EMISSION` Emission
  - `GL_SHININESS` Specular reflection exponent
- e.x.

```
gLfloat mat_specular[] = {1, 1, 1, 1};
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular); // During specular reflection, how much
                                                   // Red, Green and Blue will be scattered

GLfloat mat_shininess[] = {50};
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess); //Specular coefficient

GLfloat black[] = {0, 0, 0};
glMaterialfv(GL_FRONT,GL_AMBIENT, black); //Basically - Material color
```

From official documentation:
GL_AMBIENT
"params contains four integer or floating-point values that specify the ambient RGBA reflectance of the material. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient reflectance for both front- and back-facing materials is (0.2, 0.2, 0.2, 1.0)"

Per-color channel coefficients!

OpenGL lighting model

- `glLightfv( light, property, value );`

  – light specifies which light is affected

- multiple lights, starting with GL_LIGHT0
- Can be restricted to as little as 8 light sources. This limit can be verified using

  ```
  glGetIntegerv( GL_MAX_LIGHTS, &n );
  ```

- properties
  - colors
  - position and type
  - attenuation
- Light color properties
  - `GL_AMBIENT`
  - `GL_DIFFUSE`
  - `GL_SPECULAR`

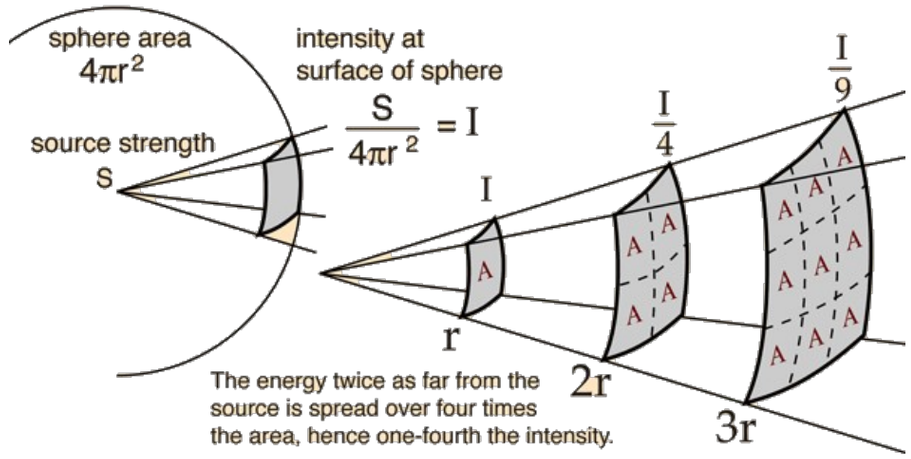
Don’t forget to...
Flip each light’s switch
`glEnable( GL_LIGHTn );`
Turn on the power
`glEnable( GL_LIGHTING );`

OpenGL lighting model

- Attenuation - decrease light intensity with distance
- Beer law is not applied since the light transmission will be mostly happening in the air and not in just one ray
- Instead, the effect of the decreasing intensity due to the light front spreading over bigger and bigger sphere is included (approximately):

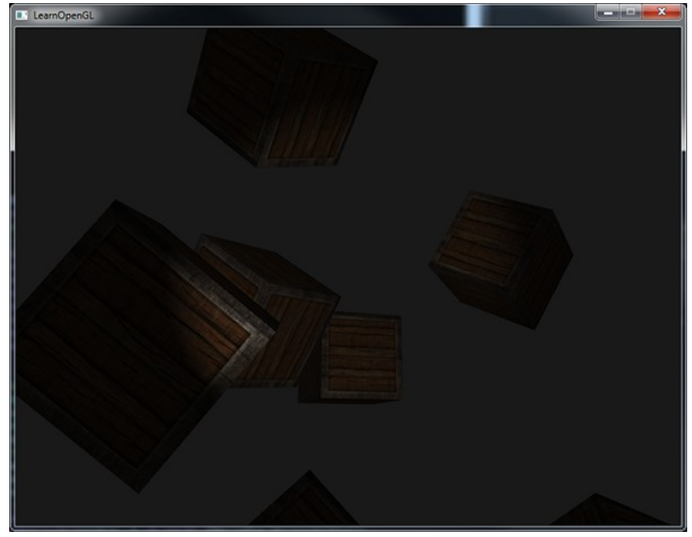
$$f_i = \frac{1}{k_c + k_l d + k_q d^2}$$

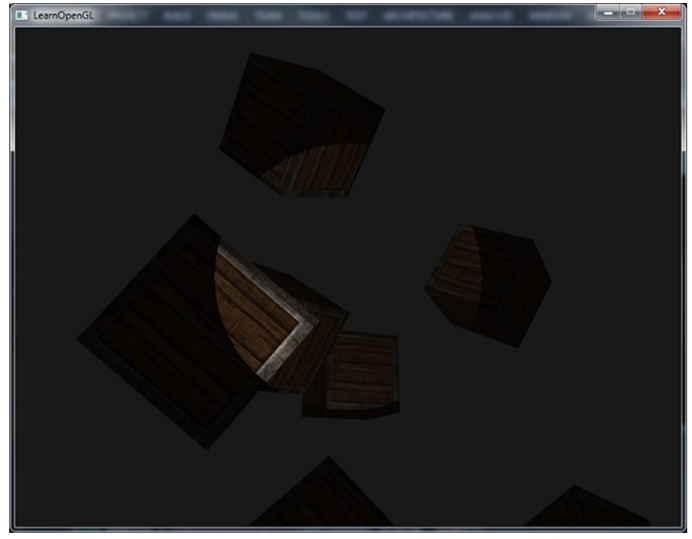
- The parametres:
  - `GL_CONSTANT_ATTENUATION` - k_c
  - `GL_LINEAR_ATTENUATION` - k_l
  - `GL_QUADRATIC_ATTENUATION` - k_q

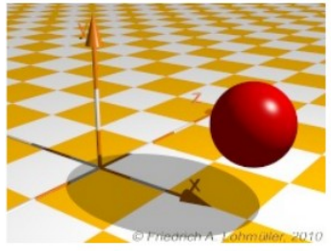


OpenGL lighting model

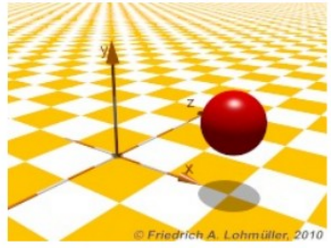
- OpenGL does not come with many light source models
- Spotlights – localize lighting affects
  - GL_SPOT_DIRECTION
  - GL_SPOT_CUTOFF
  - GL_SPOT_EXPONENT



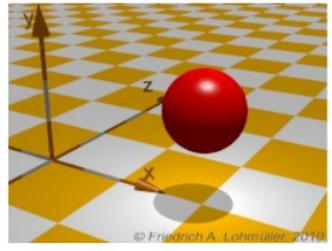




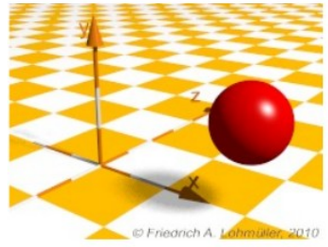
Point Light



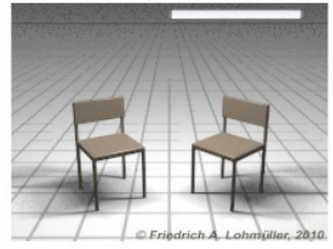
Directional Light



Spot Light



Area Light



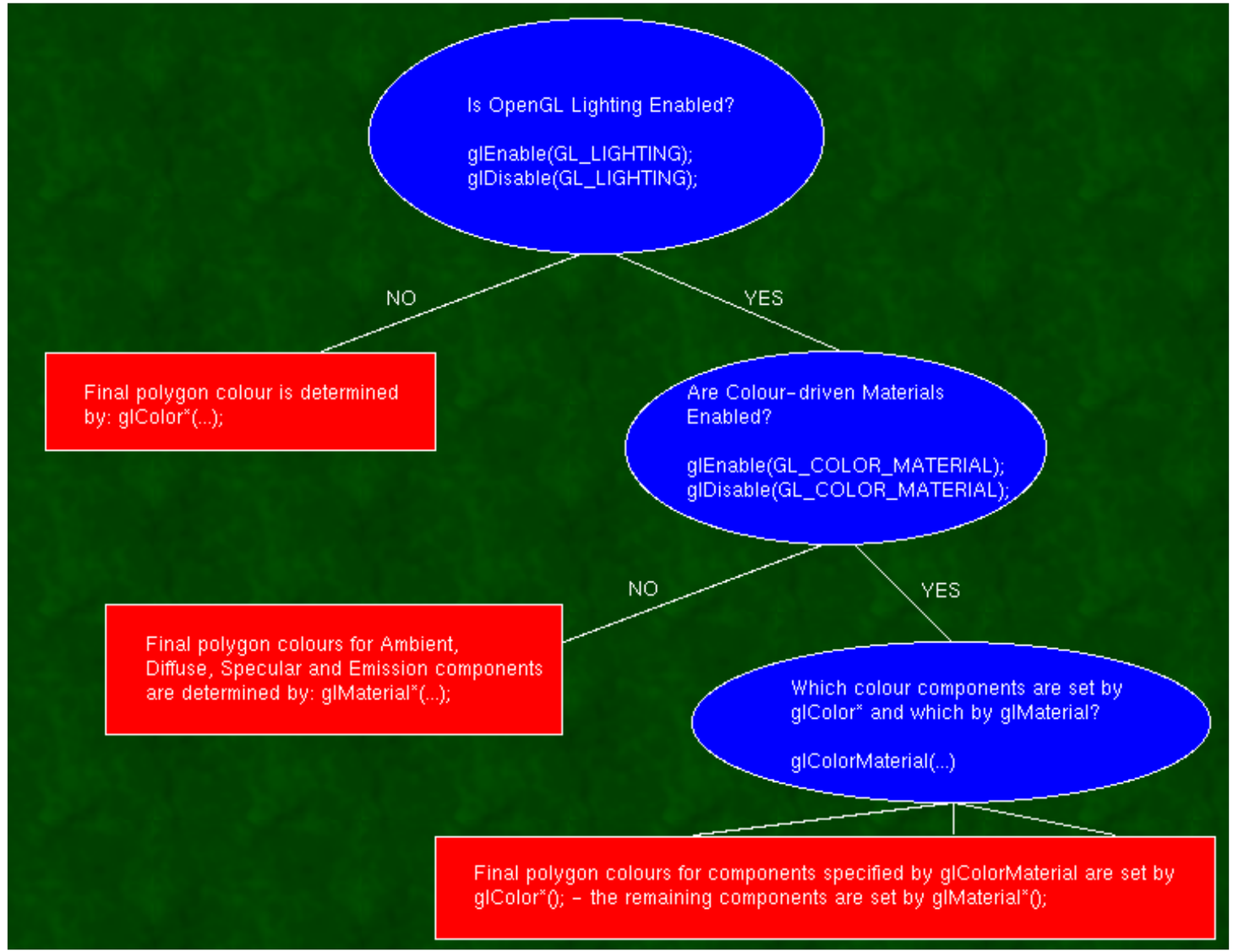
Area Light from a light tube

Volume light

OpenGL lighting model

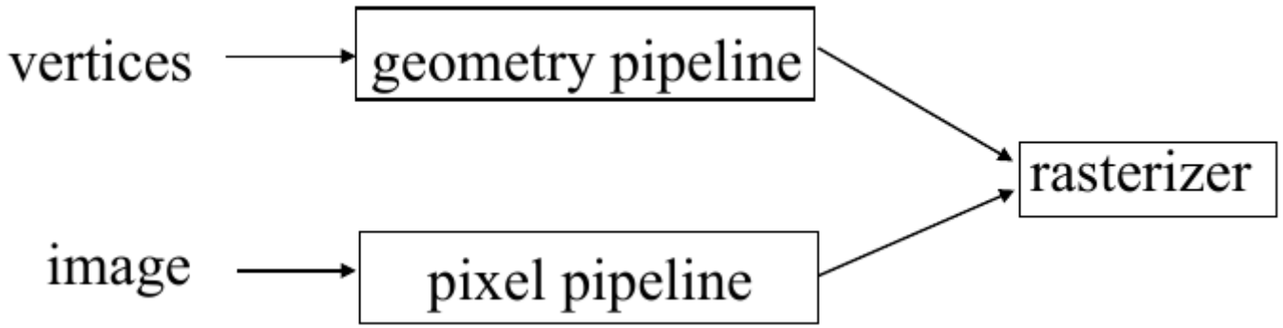
- Many parametres affecting the lighting! How are they processed?
- Great explanation:

  https://www.khronos.org/opengl/wiki/How_lighting_works



OpenGL textures

- Images and geometry flow through separate pipelines that join at the rasterizer
  - ➔ even a “complex” texture will not affect geometric complexity
  - But the OpenGL has to track the relationships between the two – hence “binding”



OpenGL textures

- Three steps are necessary in OpenGL to render a texture on an object

1) specify texture
   - read or generate image
   - assign to texture slot
   - enable texturing
2) assign texture coordinates to vertices
3) specify texture parameters
   - wrapping, filtering

OpenGL textures

- one image per texture object
- Generate texture names

  `glGenTextures( n, *texIds );`

- Create texture objects with texture data and state
- Bind textures before using

  `glBindTexture( target, id );`

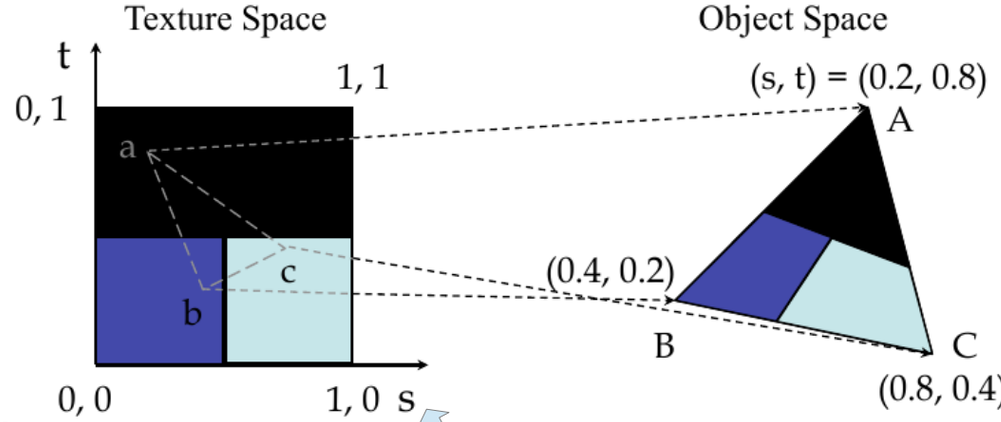
- Define a texture image from an array of texels in CPU memory

  `glTexImage2D( target, level, components, w, h, border, format, type, *texels );`

- Mapping a texture – in the primitives, based on (u,v) texture coordinates

  `glTexCoord()` specified at each vertex

Number of texture IDs to be generated – how complex will your program be?



(s,t) convention is often used interchangeably with (u,v)

OpenGL textures

- Define image:

```
glTexImage2D( target, level, components,
    w, h, border, format, type, texels );
```

`target`: type of texture, e.g. `GL_TEXTURE_2D`
`level`: used for mipmapping
`components`: elements per texel
`w, h`: width and height of `texels` in pixels
`border`: used for smoothing
`format and type`: describe texels
`texels`: pointer to texel array

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0,
 GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

Thank you!

- Questions?