Lighting in OpenGL

Professor Raymond Zavodnik

Fachhochschule Regensburg, Germany, May 9, 2012

Color Models

- Color, as perceived, is light transmitted between 380 nm (Violet) and 720 nm (Red)
- The dominating wave length is the so-called *hue*
- Within a color one can speak of *brightness*, or the energy of the share of its black/white component
- The purity of the wave-length portion comprising the color is called the *saturation*

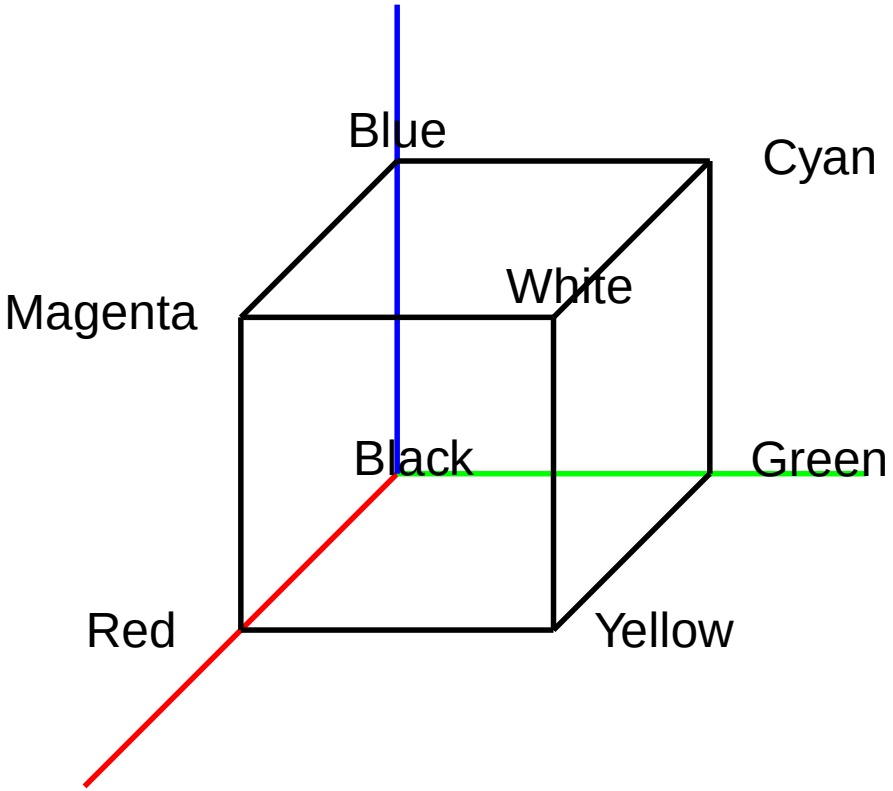
Human Perception of Color

- Perceived color usually contains many frequencies
- White contains all frequencies, black none
- The human eye has three basic cone cells that respond to blue, green and red
- Because of frequency mixing there are colors that humans perceive that do not exist in nature (i.e. color spectrum), e.g. magenta, which is red + blue

Computer Color Models

- Computer graphics concerns itself mostly with the RGB model
  - Take the basic colors Red, Green and Blue as orthogonal basis
  - Experiments have led to the conclusion that with at least three such components all perceivable colors can additively be generated
  - We will use the scale (0,0,0) to (1,1,1) to describe such a color
  - Later we willadd a fourth component A, or *alpha*, which denotes its transparency
  - Thus the RGBA model in OpenGL
- OpenGL employs a **Color Buffer**, that is, a dedicated memory segment that holds the four encoded color components for each pixel to be displayed
- Such a value is denoted by `glColor3f(0.3,0.7,0.2)` for instance

The Color Cube



Computer Color Models

- When calculating the color to be applied to a primitive such as a triangle there are two **Shading Models**:
  - `GL_FLAT`, which employs only one vertex's pixel value for the entire primitive
  - `GL_SMOOTH`, which interpolates the color inside the primitive from *all* of its vertices (Scan Conversion)
- Always clear the Color Buffer before you start
- Always enable the desired Shading Model

Hidden Surface Elimination

- It is obviously important not to output pixel values of vertices and primmitives that will be covered in the final scene rendition
- OpenGL uses the so-called *Depth Buffer*
  - Each point of a scene's surface is converted to a pixel which, in addition to its color, contains a *depth* or z-value ($0 \leq z \leq 1$).
  - If this pixel's z-value z' is smaller than the one already written in the Depth Buffer, then write the pixel's RGB value into the Color Buffer and overwrite the value in the Depth Buffer with z'.
  - Result: At the end of the rendering process only the visible parts of the scene will be placed in the Color Buffer and hence on the monitor
- Remember to clear the Depth Buffer at the onset: `glClear(GL_DEPTH_BUFFER_BIT);`
- Remember to enable the Depth Buffer:

OpenGL Lighting

- The light reflected from an object of a scen depends on:
  - The light source
  - The material properties of the object
  - The **Ambient Light** present
- In general there are three important kinds of light
  - **Ambient** or background light. This is position independent and is found throughout the scene. Example: sunlight
  - **Diffuse** light emanating from a light source that strikes a surface and is scattered in all directions. The surface absorbs part of the light, the rest is reflected as the perceived color. If the light source is placed at infinity, only its direction is important. Use Lambert's Cosine law to calculate the reflectance, depending on the surface's orientation (**Normal Vector**) and the direction of incoming light
  - **Specular** light, also eminating from a light source, but reflected in a preferred direction. For example a shiny plastic sphere will reflect the light source as a small circle

Material Colors

- This is calculated as the percentage of incoming RGB that an object reflects
- This depends on the light source–a blue sphere viewed in a green light will appear black
- As with lights, a scene object's material properties have ambient, diffuse and specular reflectances
- Thus the ambient reflectance is combined with the incoming ambient percentage
- Objects can have other material properties, such as roughness or uneveness (Bump Mapping)

Lighting Models in OpenGL

- Lights can be positioned anywhere, even at infinity (with a direction)
- There are many types of light, thus not only **Point Sources** but also **Spot Lights** with a well-defined light cone
- Lights can be turned off and on

Selecting a Lighting Model

- Use `glLightModel*()` to describe the lighting parameters:
  - Position of the viewer
  - Ambient light
  - How to apply to front and/or back faces
  - Decide whether to separate the diffuse and specular components
  - Watch out for texturing!
- Note the parameters: Always in pairs–the lighting model property followed by its value

Light Model Parameters

Parameter	**Default Value**	Meaning
GL_LIGHT_MODEL_AMBIENT	(0.2,0.2,0.2)	Scene ambient RGBA
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 or GL_FALSE	Reflection angle computation
GL_LIGHT_MODEL_TWO_SIDE	0.0 or GL_FALSE	One or two sided lighting
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR	Specular separate

Defining The Material Properties

- You must describe how an object reflects light
- Use `glMaterial*()`
  - Declare what faces to apply light to (usually front face)
  - Describe ambient, diffuse, specular and shininess components
- This is an empirical art

Material Properties

Parameter	**Default Value**	Meaning
`GL_AMBIENT`	(0.2,0.2,0.2)	Material ambient color
`GL_DIFFUSE`	(0.8,0.8,0.8)	Material diffuse color
`GL_AMBIENT_AND_DIFFUSE`		Material ambient and diffuse
`GL_SPECULAR`	(0.0,0.0,0.0,1.0)	Material specular color
`GL_SHININESS`	0.0	Specular exponent
`GL_EMISSION`	(0.0,0.0,0.0,1.0)	Material emissive color

Light Position and Direction

- These are subject to the same matrix transofrmations as primitives
- When specifying `glLight*()`, the position is modified by the current modelview matrix. The result is in eye coordinates
- The position or direction of a light can be changed by using a different modelview matrix
- There are various possibilities for placing and aiming a light:
  - Keep the light stationary: set its position after modelview
  - Move the light relative to a stationary object: set its position *after* modelview, which is itself changed to determine the new light position
  - Move the light together with the viewpoint: set its position *before* modelview. Here modelview affects both light and viewpoint