

Lighting, GLSL

Shaoting Zhang

May 11, 2008

- 1 for Project 2
- 2 Lighting
- 3 GLSL
- 4 Q/A

javax.vecmath

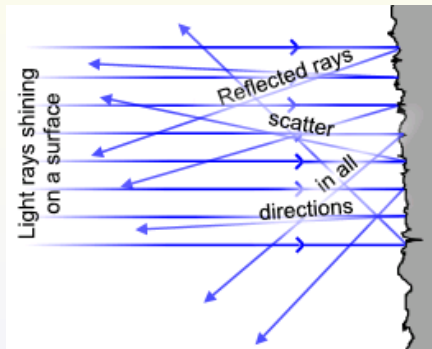
- vecmath.txt
- cross
- normalize
- dot

- 1 for Project 2
- 2 Lighting
- 3 GLSL
- 4 Q/A

Different lights

- 1 **Emitted:** Originates from an object and is unaffected by any light sources.
- 2 **Ambient:** From that source that's been scattered so much by the environment that its direction is impossible to determine.
- 3 **Diffuse:** Once it hits a surface, it's scattered equally in all directions, so it appears equally bright.
- 4 **Specular:** Comes from a particular direction, and it tends to bounce off the surface in a preferred direction.

Diffuse and Specular



Materials

- In OpenGL, a perfectly red ball reflects all the incoming red light and absorbs all the green and blue light that strikes it.
- Red ball with white light?
- Red ball with pure green light?
- if an OpenGL light has components (LR, LG, LB), and a material has corresponding components (MR, MG, MB), then, ignoring all other reflectivity effects, the light that arrives at the eye is given by (LR*MR, LG*MG, LB*MB).
- Two lights, which send (R1, G1, B1) and (R2, G2, B2) to the eye, OpenGL gives (R1+R2, G1+G2, B1+B2). If any of the sums are greater than 1, the component is clamped to 1.

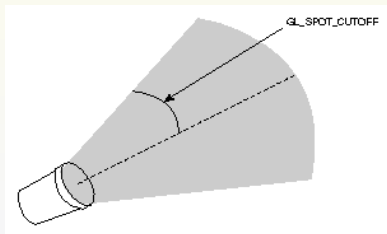
Lights and materials, Demo: lit sphere

- Demo from red book examples using JOGL.
<http://ak.kiet.le.googlepages.com/theredbookinjava.html>

Control lights, Demo: move light, lit sphere

- `void glLight{if}[v](GLenum light, GLenum pname, TYPEparam)`
- It takes three arguments: to identify the light whose property is being specified, the property, and the desired value for that property.
- **pname:** `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`
- **pname:** `GL_POSITION`, `GL_SPOT_DIRECTION`, `GL_SPOT_CUTOFF`
- **pname:** `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION`, `GL_QUADRATIC_ATTENUATION`

Control lights, spotlights



Control lights, attenuation

$$\textit{attenuationfactor} = \frac{1}{k_c + k_l * d + k_q * d^2}$$

where

d = distance between the light's position and the vertex

k_c = GL_CONSTANT_ATTENUATION

k_l = GL_LINEAR_ATTENUATION

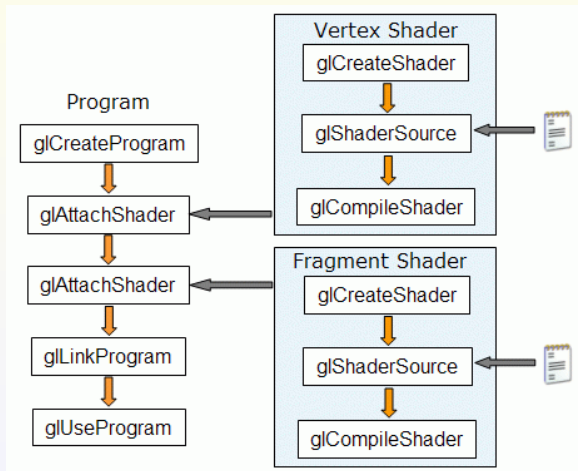
k_q = GL_QUADRATIC_ATTENUATION

Define materials, Demo: material

- `void glMaterial{if}[v](GLenum face, GLenum pname, TYPEparam)`
- **face:** GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
- **pname:** GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_SHININESS, GL_EMISSION

- 1 for Project 2
- 2 Lighting
- 3 GLSL
- 4 Q/A

Overview



from www.lighthouse3d.com

Vertex and fragment shader

- **Vertex shader:** A vertex shader operates on every vertex. if you use a vertex shader ALL Per-Vertex operations of the fixed function OpenGL pipeline are replaced.
- **Fragment shader:** A fragment shader operates on every fragment which is produced by rasterization. Just like a vertex shader, a fragment shader replaces ALL Per-Fragment operations of the fixed function OpenGL pipeline.

Attributes, Uniforms and Varyings

- **Uniforms** are values which do not change during a rendering, for example the **light position** or the **light color**. Uniforms are available in **vertex and fragment** shaders. Uniforms are **read-only**.
- **Attributes** are only available in **vertex** shader and they are input values which change every vertex, for example the **vertex position or normals**.
- **Varyings** are used for passing data from a vertex shader to a fragment shader. Varyings are read-only in fragment shader but are read and writeable in vertex shader.

Build-in attributes

- 1 `gl_Vertex`: 4D vector representing the vertex position
- 2 `gl_Normal`: 3D vector representing the vertex normal
- 3 `gl_Color`: 4D vector representing the vertex color

Build-in uniforms

- 1 `gl_ModelViewMatrix`: 4x4 Matrix representing the model-view matrix.
- 2 `gl_ModelViewProjectionMatrix`: 4x4 Matrix representing the model-view-projection matrix.
- 3 `gl_NormalMatrix`: This matrix is used for normal transformation.

Build-in types for shader output

- 1 `gl_Position`: 4D vector representing the final processed vertex position. Only available in vertex shader.
- 2 `gl_FragColor`: 4D vector representing the final color which is written in the frame buffer. Only available in fragment shader.
- 3 `gl_FragDepth`: float representing the depth which is written in the depth buffer. Only available in fragment shader.

Build-in functions

- ① dot: a simple dot product
- ② cross: a simple cross product
- ③ normalize: normalize a vector
- ④ clamp: clamping a vector to a minimum and a maximum

Demo: ambient shader

From NeHe production and lighthouse3D.

The ambient shader surely is the simplest shader available. Each rendered pixel has one specific color.

Demo: diffuse shader

varyings and build-in functions.

Demo: pass Uniforms to GL

Uniforms can be passed to the GL using
`void glUniform1fARB(GLint location, TYPE val)`
where
location is the location of the uniform.
val is a value from TYPE.

Demo: pass Uniforms to GL

Getting the uniform location can easily be done with
`GLint glGetUniformLocationARB(GLhandleARB program, const
GLcharARB * name)`

where

program is our program object.

name is the name of the uniform which location we want to get.

- 1 for Project 2
- 2 Lighting
- 3 GLSL
- 4 Q/A

Any questions?