|  |  |
| --- | --- |
|  | |
| **DEBRE MARKOS UNIVERSTY**  **Department Software Engineering** | |

**Computer Graphics Group Assignment**

**Lighting and shading in OpenGL**

Name ID

1. Mesenbet Tenaw…………………………………………1273
2. Biruk Begashaw……………………………………….. 1251
3. Melkam Ayenaw………………………………………….1272
4. Yonass Telayneh……………………………………………..1292
5. Biruk Shewaferahu…………………………………………..1238
6. Tigist abay ……………………………………………………1285

Introdution:-

What is Lighting?

**Computer graphics lighting** is the collection of techniques used to simulate light in [computer graphics](https://en.wikipedia.org/wiki/Computer_graphics) scenes. While lighting techniques offer flexibility in the level of detail and functionality available, they also operate at different levels of computational demand and [complexity](https://en.wikipedia.org/wiki/Computational_complexity). Graphics artists can choose from a variety of light sources, models, shading techniques, and effects to suit the needs of each application.

# Types of lighting implemented in OpenGL:

* **Ambient:** No source point; affects all polys independent of position, orientation, and viewing angle; used as a ‘fudge’ to approximate 2nd order and higher reflections.
* **Diffuse:** Light scattered in all directions after it hits a poly; dependent upon incident angle
* **Specular:** ‘Shininess’; dependent upon incident and viewing angles
* **Emissive:** Makes a poly appear as though it is glowing; does not give off light

# Implementation Specifics

**Two kinds of parameters properties:-**

* Material properties: state variables, can be changed as you draw different polys in a scene
* Light properties: parameters indexed to light numbers; Opengl can use up to 8 lights; light positions are affected by the model-view matrix stack

# Shading models:

Shading is referred to as the implementation of the illumination model at the pixel points or polygon surfaces of the graphics objects. It is used to compute the intensities and colors to display the surface.

Shading models determine the shade of a point on the surface of an object in terms of a number of attributes. The shading Mode can be decomposed into three parts, a contribution from diffuse illumination, the contribution for one or more specific light sources, and a transparency effect.

Shading refers to how the lighting equations are applied to a rasterized poly.

**OpenGL supports two shading models:**

* **GL\_FLAT:** Lighting is evaluated once per poly, and the resulting color value is used for the whole thing.
* **GL\_SMOOTH:** Lighting is evaluated at each vertex, and pixel colors are linearly interpolated across polys. This is more expensive, but it looks much better.

OpenGL uses the Phong lighting model at vertices but has no built-in support for Phong shading. Modern programmable lighting hardware can implement full Phong shading (and much more!), but you have to do this yourself.

# Normals:

The lighting equations depend upon normal, so we have to provide them. The current normal is specified with a call to a normal function and will be applied to every subsequent vertex. Normal should be of unit length, or the lighting equations will not work correctly. This can be a problem because normal is affected by any scaling done in the matrix stack. You must either re-normalize the normal as a pre-processing step or enable GL\_NORMALIZE (which is computationally expensive).

**In OpenGL a polygon can have the following material properties:**

* ambientColor (R, G, B) - Contribution due to ambient light
* diffuseColor (R, G, B) - Contribution due to directed, point, and spot lights
* specularColor (R, G, B) - Contribution due to specularity, usually white
* emissiveColor (R, G, B) - For objects that glow of their own accord, independent of light
* transparency 0.0 - 1.0 - Alpha channel, for combining with pixels behind the object.
* shininess 0.0 - 1.0 - Controls the contribution of the specular component

# Programming Example:

This code would typically be placed with your opengl initialization code

//set the global lighting / shading params glShadeModel(GL\_SMOOTH); // or GL\_FLAT glEnable(GL\_NORMALIZE); //or not glEnable(GL\_LIGHTING);

//set the global ambient light

GLfloat ambient = {.2,.2,.2,1};

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, globalAmb);

This code sets up a light and enables it

GLfloat diffuse[] = {1,0,0,1};

GLfloat ambient[] = {.5,0,0,1};

GLfloat specular[] = {1,1,1,1};

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuse); glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambient); glLightfv(GL\_LIGHT0, GL\_SPECULAR, specular); glEnable(GL\_LIGHT0); //enable the light

//set light position (see discussion later for details)

// set last term to 0 for a spotlight (see chp 5 in ogl prog guide)

Glfloat lightpos[] = {1,1,1,1};

glLightfv(GL\_LIGHT0,GLPOSITION, lightpos);

This code sets a simple material property

GLfloat ambient[] = {.5,0,0,1};

GLfloat specular[] = {1,1,1,1};

//can set params for front and back separately (GL\_BACK, GL\_FRONT\_AND\_BACK) glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, ambient); glMaterialfv(GL\_FRONT, GL\_SPECULAR, ambient);

Shaded polygons can now be drawn using the usual method, so long as normals are specified correctly.

**Implementing Shading groups**

When using smooth shading, the usual way to calculate a vertex normal is to average the normal of all of the faces that it is a part of. This normal is then used for all polygons that use the vertex. This looks good for smooth, but terrible for things like cubes. To make a hard edge in a smooth shading model, specify the normal for each vertex along the edge separately for each polygon, and set them equal to the face normal.

**Controlling Light Positions**

How can we make our light position stay fixed relative to our eye position? How do we make a headlight?

You need to specify your light in eye coordinate space. To do so, set the Model View matrix to the identity, then specify your light position. To make a headlight (a light that appears to be positioned at or near the eye and shining along the line of sight), set the Model View to the identity, set the light position at (or near) the origin, and set the direction to the negatives-axis.

**How can we make our light stay fixed relative to our scene?**

As your view changes, your Model View matrix also changes. This means you'll need to specify the light position, usually at the start of every frame. A typical application will display a frame with the following pseudo-code

* Set the view transform.
* Set the light position
* Send down the scene or model geometry. Swap buffers.

**How can we make a light that moves around in a scene?**

Again, you'll need to specify this light position every time the view changes. Additionally, this light has a dynamic modeling transform that also needs to be in the Model View matrix before you specify the light position. In pseudo-code, you need to do something like:

* Set the view transform
* Push the matrix stack
* Set the model transform to update the light’s position
* Set the light position
* Pop the matrix stack
* Send down the scene or model geometry Swap buffers.