CS 450/550 -- Fall Quarter 2023

Project #3

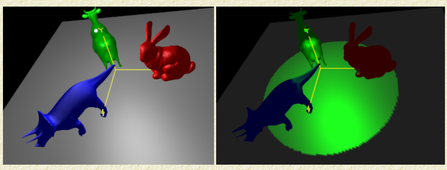
100 Points

Due: October 25

Lighting

This page was last updated: October 10, 2023

Introduction



The goal of this project is to create a 3D scene that demonstrates an animated OpenGL light source.

Learning Objective:

When you are done with this assignment, you will understand how to add dynamic light sources to your programs so that the viewer can better see the three-dimensionality of the scene. This is also great to know because oftentimes a major part of the storyline in games and movies is controlled by what the user can see and how well they can see it.

Requirements:

1. Create a 3D scene consisting of at least 3 objects.
    1. At least one object must come from an OBJ file (your choice). The other two can be anything you want (even other, different, OBJ objects). Just be sure that all objects have surface normal vectors attached to them.
    2. At least one object must be dull and at least one must be shiny (GL_SHININESS).
    3. The objects must each have a different material color.
2. Place a grid under the scene. I will give you the code to do this.
3. Have a light source that travels above the scene in some motion pattern. The exact path is up to you. (A circle works well.) The light source needs to be able to be toggled between a point light and a spot light by using the 'p' and 's' keyboard keys. When it is a spot light, it must be pointing down at the scene. You will only use perspective in this project, not orthographic, so the 'p' key can be used for a point light.
4. Have the light source's color be toggled between being white, red, green, blue, and yellow by using the 'w', 'r', 'g', 'b', and 'y' keyboard keys.
5. Put a small sphere at the light source location so that we can see where it is. Make the sphere the same color as the light source it is representing. *Don't use lighting on the light-source sphere*. Just make it a **glColor3f** blob with lighting disabled.
6. ***I recommend you use all OsuSphere objects at first, just to get the lighting working, then try with other objects.*** Keep it simple to begin with!
7. All OBJ files will need to have per-vertex surface normal vectors for the lighting to work. Edit the .obj file and look for lines that begin with the letters **vn** to see if it does.
8. Warning: not all of the GLUT solids have surface normal vectors. However, the OSU sphere, cone, and torus all have them.
9. Remember that OpenGL light positions automatically get transformed by the GL_MODELVIEW transformation *as it exists at the moment you specify the light position*. You can't prevent this from happening. You can only control the GL_MODELVIEW matrix to be what you need it to be at the time and control where in your code you specify the light source positions.
10. Feel free to use the SetLight functions from the Lighting notes. This file can be enabled in your sample.cpp file by uncommenting its #include.
11. Leave the atteniation set to "no attenuation", like is in the notes. You can do anything with the attenuation that you'd like. I recommend you set it to "no attenuation" (C=1., L=0., Q=0.), at least at first, so that you can see everything. The SetLight functions already do that.
12. Help us get the grading right! Be sure your video clearly shows the effect of the light on the objects.

Drawing the Grid

The purpose of the grid is to have somthing planar for the light to shine on. This makes it easier to see what the effect of the light is. Exactly how you do this is up to you. Something like this works well:

```
#define XSIDE   ?????                   // length of the x side of the grid
#define X0      (-XSIDE/2.)             // where one side starts
#define NX      ?????                   // how many points in x
#define DX      ( XSIDE/(float)NX )     // change in x between the points

#define YGRID   0.f

#define ZSIDE   ?????                   // length of the z side of the grid
#define Z0      (-ZSIDE/2.)             // where one side starts
#define NZ      ?????                   // how many points in z
#define DZ      ( ZSIDE/(float)NZ )     // change in z between the points

GridDL = glGenLists( 1 );
glNewList( GridDL, GL_COMPILE );
        SetMaterial( 0.6f, 0.6f, 0.6f, 30.f );
        glNormal3f( 0., 1., 0. );
        for( int i = 0; i < NZ; i++ )
        {
                glBegin( GL_QUAD_STRIP );
                for( int j = 0; j < NX; j++ )
                {
                        glVertex3f( X0 + DX*(float)j, YGRID, Z0 + DZ*(float)(i+0) );
                        glVertex3f( X0 + DX*(float)j, YGRID, Z0 + DZ*(float)(i+1) );
                }
                glEnd( );
        }
glEndList( );
```

Since you are using OpenGL's built-in *per-vertex* lighting, it is important that the grid have lots of vertices!

Freezng the Animation

In any animating program, it is helpful for debugging to be able to freeze the animation. The 'f' key is implemented by turning the Idle Function on and off.

```
// a global:
bool Frozen;


// in Reset( ):
        Frozen = false;


// in Keyboard( ):
        case 'f':
        case 'F':
                Frozen = ! Frozen;
                if( Frozen )
                        glutIdleFunc( NULL );
                else
                        glutIdleFunc( Animate );
                break;
```

Keyboard Keys

'w'	Turn the light source color to white
'r'	Turn the light source color to red
'g'	Turn the light source color to green
'b'	Turn the light source color to blue
'y'	Turn the light source color to yellow
'p'	Make the light source a point light
's'	Make the light source a spot light
'f'	Freeze/Unfreeze the animation

In reality, you can use any user interface that you want. We won't know how you are demonstrating the features of your program, just that it works. The keyboard table above is just a suggestion.

Timing Your Scene Animation

Deliberately time the animation like we've seen before. Here is a good way to do that. Set a constant called something like MS_PER_CYCLE that specifies the number of milliseconds per animation cycle. Then, in your Idle Function, query the number of milliseconds since your program started and turn that into a floating point number between 0. and 1. that indicates how far through the animation cycle you are. Then in Display, you might use that 0.-1. number to position the light, something like this: If you wanted to make the light move in a planar circle, you might do this:

```
// in the defined constants:
#define MS_PER_CYCLE  10000
#define LIGHTRADIUS = ????


// a global:
float Time;


// in Animate( ):
        int ms = glutGet( GLUT_ELAPSED_TIME );
        ms %= MS_PER_CYCLE;
        Time = (float)ms / (float)MS_PER_CYCLE;         // [0.,1.)


// in Display( ):
        float xlight = << some function of Time >>
        float ylight = << some function of Time >>
        float zlight = << some function of Time >>
        . . .
```

Using OBJ Files

There are a lot of free .obj files on the web. There are some links at the end of the Class Resources Page. You can also find a lot of files by Googling "free obj file".

When you want to bring in a 3D object as a .obj file, use our LoadObjFile( ) function. This file can be enabled in your sample.cpp file by uncommenting its #include.

Incorporate the .obj file into your own code by placing the .obj object into a display list, like this:

```
// a global:
int DinoDL;


// in InitLists( ):
        DinoDL = glGenLists( 1 );
        glNewList( DinoDL, GL_COMPILE );
                LoadObjFile( (char *)"dino.obj" );
        glEndList( );


// in Display( ):
        glCallList( DinoDL );
```

Warning! Not all obj files have normals. Take a look at the lines in the obj file (it is ascii-editable).
If you see lines of text beginning with **vn**, it has normals.

Turn-in:

Use the Teach system to turn in your:

1. .cpp file
2. A PDF report containing:
    - Project number and title
    - Your name
    - Your email address
    - A description of what you did to get the display you got
    - A cool-looking screen shot from your program
    - The link to your video demonstrating that your project does what the requirements ask for. If you can, we'd appreciate it if you'd narrate your video so that you can tell us what it is doing.

Grading:

Note: you don't get credit for these things by just having done them. You get credit by convincing us that your program's lighting behavior is correct. That is, you only get credit if you have made a video that demonstrates the correct visual lighting behavior.

Item	Points
At least three objects	10
Point light works	20
Spot light works	20
Light in some pattern above the scene	20
Light changes color correctly	20
There is a small unlit sphere to show where the light source is	10
Potential Total	100