# CS 450/550 -- Fall Quarter 2025

# Project #3

### 100 Points

### Due: October 22

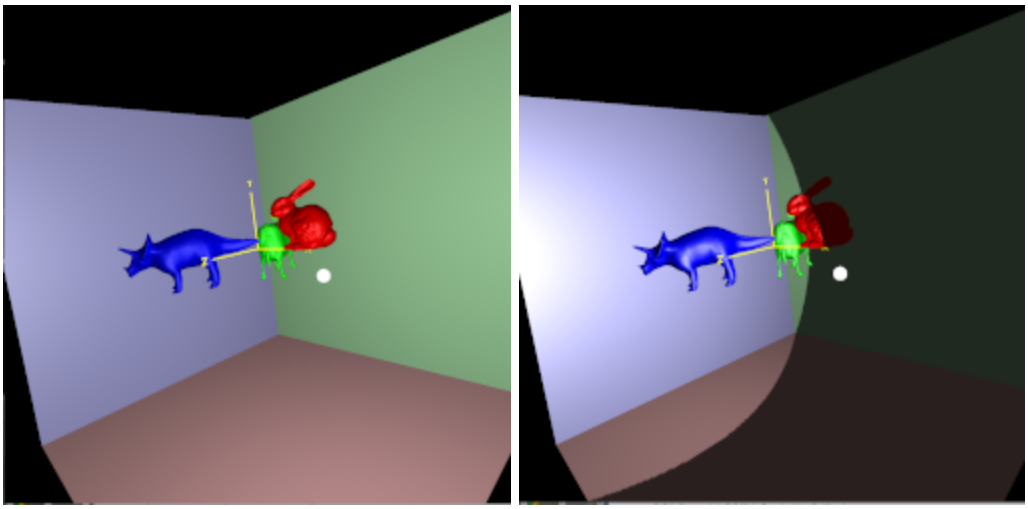
# Lighting

---

*This page was last updated: September 9, 2025*

---

## Introduction



White Point Light White Spot Light

The goal of this project is to create a 3D scene that demonstrates an animated OpenGL light source. (Notice in these images that these are not *shadows*. Shadows are where light is blocked. This shows light-source *shading*.)

## 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 a floor, 2 walls, and at least 3 3D objects.
    1. Use enough vertex detail on the floor and walls to show lighting. Code to do that is coming up. See below.
    2. Be sure the surface normals on the floor and walls are correct (that is, perpendicular to the surface and pointing into the area where the light is).
    3. For the floor and walls, use a close-to-neutral color. Joe Graphics likes to choose colors such as (0.6f, 0.8f, 0.6f) so that the surface is close to a dim gray but sightly colored so that you know which surface is which.
    4. At least one 3D object must come from an OBJ file (your choice). The other two can be anything you want (even other OBJ objects). Just be sure that all objects have surface normal vectors attached to them.
    5. At least one object must be dull and at least one must be shiny (GL_SHININESS).
    6. The 3 objects must each have a different material color.
    7. Since you are doing lighting, be sure to use the **SetMaterial** function to set the color, not glColor3f.
    8. Be sure to glEnable GL_LIGHTING and GL_LIGHT0 before drawing your floor, walls, and objects.
    9. Be sure to glDisable GL_LIGHTING before drawing the sphere and the axes.

2. Have the light source move through the scene. A path such as a circle around the origin works well.

3. The light source needs to be able to be toggled between a point light and a spot light, possibly by using the 'p' and 's' keyboard keys. When it is a spot light, it must be pointing sideways (i.e., horizontally) towards one of the walls. You will only use perspective in this project, not orthographic, so the 'p' key can be used enable the point light.

4. Have the light source's color be toggled between being white, red, orange, yellow, green, cyan, and magenta by using the 'w', 'r', 'o', 'y', 'g', 'c', and 'm' keyboard keys. A decent color orange can be created with (1.0, 0.5, 0.0).

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, cube, cylinder, 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 Set*Light functions from the Lighting notes. This file can be enabled in your sample.cpp file by uncommenting the setlight.cpp #include.

11. I recommend you leave the light atteniation set to "no attenuation", like is in the notes and like it is in the Set*Light functions (C=1., L=0., Q=0.). This lets you see everything. Feel free to experiment with this later after everything else if working.

12. Be sure you are setting
    glShadeModel( GL_SMOOTH );
    for the floor, walls, and each 3D object.

13. Help us get the grading right! Be sure your video clearly shows the effect of the light on the objects.

## Drawing the Floor

The purpose of the floor and walls 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 create the floor and 2 walls is up to you. Something like might work well for the floor:

```
// in the globals:
GLuint GridDL;

. . .

        // in InitLists( ):
        #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                     // y-height of the grid

        #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, 0.8f, 0.6f, 30.f );  // or whatever else you want
                glNormal3f( 0., 1., 0. );               // for each floor vertex, pointing straight up
                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!

## Drawing the Walls

Each wall can be drawn in a similar manner to the floor. Just be sure you get the orientation of the grid and the direction of the surface normal correct!

## Freezing the Animation

In any animating program, it is sometimes helpful to be able to freeze the animation. For example, you could use the 'f' key to toggle the Idle Function on and off.

```
// in the globals:
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 |
| 'o' | Turn the light source color to orange |
| 'y' | Turn the light source color to yellow |
| 'g' | Turn the light source color to green |
| 'c' | Turn the light source color to cyan |
| 'm' | Turn the light source color to magenta |
| 'p' | Make the light source a point light |
| 's' | Make the light source a spot light |
| 'f' | Freeze/Unfreeze the animation (optional) |

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.

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 SPEED         3.
```

```
#define LIGHTRADIUS   ????

. . .

// in the globals:
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 theta = SPEED * F_2_PI * Time;
float r = LIGHTRADIUS;
float xlight = -r * cosf(theta);
float zlight =  r * sinf(theta);
float ylight =  0.f;
```

## Using OBJ Files

There are a lot of free .obj files on the web.

Here is a folder with some of my favorite OBJ files.

Here are some sites you can also look through:

- TurboSquid: http://www.turbosquid.com/Search/3D-Models/free/obj
- Free3D: https://free3d.com/3d-models/
- SketchFab: https://sketchfab.com/features/free-3d-models

You can also just Google the phrase "free obj files".

When you want to bring in a 3D object as a .obj file, use our LoadObjMtlFiles( ) function. This file can be enabled in your sample.cpp file by uncommenting the loadobjmtlfiles.cpp #include. (You will also need to uncomment the #include for bmptotexture.cpp .)

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

```
// in the globals:
GLuint DinoDL;

. . .

// in InitLists( ):
        DinoDL = LoadObjMtlFiles( (char *)"dino.obj" );

. . .

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

Warning! Not all obj files have normal vectors! 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 Canvas to turn in:

1. Your .cpp file
2. A short PDF report containing:
    - Your name
    - Your email address
    - Project number and title
    - A description of what you did to get the display you got
    - A cool-looking screen shot from your program
    - The link to the Kaltura 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.
3. To see how to turn these files in to Canvas, go to our Project Notes noteset, and go the the slide labeled **How to Turn In a Project on Canvas.**
4. **Be sure that your video's permissions are set to *unlisted.***.
   The best place to set this is on the OSU Media Server.
5. A good way to test your video's permissions is to ask a friend to try to open the same video link that you are giving us.
6. The video doesn't have to be made with Kaltura. Any similar tool will do.

## 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 |
|---|---|
| The floor and 2 walls exist and are GL_SMOOTH lighted | 10 |
| There are at least three 3D objects, colored differently and lit | 20 |
| The point light works properly by shining in all directions | 20 |
| The spot light works properly by shining horizontally towards a wall | 20 |
| The light color can be changed to any of the 7 required colors | 20 |
| There is a small unlit sphere to show where the light source is | 10 |
| Potential Total | 100 |