

Assignment 3

CS6533 Spring 2012

Due: 4/18 in class; Total Score: 165 points

Note: This assignment has 4 pages.

This assignment extends Assignment 2 by adding the option of producing the *shadow effect* (with respect to a fixed light source) while rolling a sphere, as well as the options of shading and lighting. You will practice implementing three main techniques: (1) producing a shadow, (2) drawing a *decal* on top of an object — in this case the shadow is a decal on top of the ground, and (3) shading and lighting. (Consult Lecture Notes for techniques of producing shadow and decal effects, and sample code A.12 in Textbook Chapter 6 and Appendix A for shading and lighting.)

General Instructions: same as Assignment 1.

(a) Consider the two unit-sphere files given in Assignment 2 (<http://cis.poly.edu/cs653/assg2/>). There are two more unit-sphere files given in <http://cis.poly.edu/cs653/assg3/>, with 256 and 1024 triangles, respectively. The file format is described in Assignment 2.

Draw the x-, y-, z-axes, the “ground” (the quadrilateral indicating the x-z plane), and roll the wire-frame sphere as in Assignment 2. Your task here is to add the corresponding shadow of the sphere on the ground given a fixed light source located at position $L = (-14.0, 12.0, -3.0, 1.0)$. Draw the shadow with color $(0.25, 0.25, 0.25, 0.65)$. Note that the shadow should be the wire-frame sphere projected onto the ground, projected from the light source. As the sphere rolls, the shadow should also “roll” accordingly.

As another option, draw the sphere as a *solid* sphere and roll it as before, by drawing the sphere triangles as *filled* triangles rather than just drawing the triangle edges. Produce the corresponding shadow effect for this rolling solid sphere as well, using the same shadow color and the same light source location L . (The options are put together by a menu; see **Part (b)**.)

Hint: Draw the shadow as a separate object. As discussed in class, the shadow object is obtained as follows. At each frame, from each generic point $p = (x, y, z, 1)$ on the sphere (the sphere is already **at its final position and orientation at this frame**), project p onto the ground using the ray going from L through p and intersecting the ground plane at point q ; q is then the shadow point of p . Since p can be any point on the sphere, any corresponding shadow point q can be obtained this way. Suppose $p = (x, y, z, 1)$ and $q = (x_q, y_q, z_q, 1)$. Derive x_q and z_q in terms of x, y , and z , using simple geometry and the fact that the ground plane is $y = 0$ and thus $y_q = 0$. Then, derive a 4×4 transformation matrix N to carry out this “shadow projection” that transforms p to q using the homogeneous-coordinate representation.

Note that in OpenGL, the homogeneous-coordinate representation (x, y, z, w) , $w \neq 0$, is equivalent to the representation $(x/w, y/w, z/w, 1)$, i.e., the *perspective division* to convert (x, y, z, w) to $(x/w, y/w, z/w, 1)$ is *automatically done* by OpenGL. Also, to produce the shadow, you first perform the sphere-rolling transformation (as in Assignment 2) on the input sphere, and then perform the shadow projection transformation N .

Programming Tips:

1. If you experienced a strange behavior of the z-buffer where some parts of the scene disappear randomly, check your `gluPerspective()` command; see **Useful Tips Item 2** on page 4 for the details.

2. In OpenGL, although the perspective division is automatically done to convert (x, y, z, w) to $(x/w, y/w, z/w, 1)$ for $w \neq 0$, the OpenGL implementation actually requires that $w > 0$. So if you tried $q = (\frac{f(x,y,z)}{h(x,y,z)}, 0, \frac{g(x,y,z)}{h(x,y,z)}, 1) \equiv (f(x, y, z), 0, g(x, y, z), h(x, y, z))$ and it did not work, try to use $q = (\frac{-f(x,y,z)}{-h(x,y,z)}, 0, \frac{-g(x,y,z)}{-h(x,y,z)}, 1) \equiv (-f(x, y, z), 0, -g(x, y, z), -h(x, y, z))$. **(60 points)**

(b) If you just draw the shadow as in **Part (a)**, then you will see a *broken* shadow. The undesirable effect is especially obvious for the solid sphere. This is because the shadow and the ground are on the same plane, but the z-buffer only has a limited numerical precision. As a result, some portion of the shadow may appear in front of the ground and some portion may appear *behind* the ground and thus *blocked* by the ground; the result is unpredictable. Note that what you want here is to make the shadow *always appear on top of* the ground, i.e., to make the shadow a *decal* on top of the ground. Use the technique of making a decal, as discussed in class, to achieve the desired effect.

Recall from Assignment 2 that associated with the left mouse button there is a menu entry “Shadow”. Implement it as a submenu with 2 submenu entries:

“No” — do not produce the shadow, as in Assignment 2, and

“Yes” — produce the *correct* shadow, as implemented in **Parts (a) and (b)**.

Recall that the left mouse button is also associated with menu entries “Enable Lighting” and “Shading”. Implement “Enable Lighting” as a submenu with 2 submenu entries:

“No” — the lighting is disabled, as is the case so far, and

“Yes” — the lighting is enabled, to be implemented in **Parts (c) and (d)**.

In addition, implement “Shading” as a submenu with 2 submenu entries: “flat shading” and “smooth shading”. When lighting is disabled (by choosing “Enable Lighting” and then “No”), *both* options should just draw the sphere as a solid sphere (see **Part (a)**) and give the same result. (Note that choosing both “flat shading” and “smooth shading” should draw a solid sphere, and choosing the menu entry “Wire Frame” should draw a wire-frame sphere (as in Assignment 2); you need to produce the corresponding *correct* shadow if “Shadow” is “Yes”.) When lighting is enabled, “flat shading” and “smooth shading” will be further implemented in **Part (c)**.

Hint:

1. You should always enable the z-buffer testing (by `glEnable(GL_DEPTH_TEST)`). Also, you need to draw the ground/shadow in multiple passes, enabling/disabling writing to the z-buffer or color buffer (color buffer is the frame buffer), as discussed in class.

2. Use `glDepthMask(GL_FALSE)` to disable writing to the z-buffer, `glDepthMask(GL_TRUE)` to enable writing to the z-buffer, `glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE)` to disable writing to the color buffer (in RGBA color mode), and `glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE)` to enable writing to the color buffer. **(40 points)**

In the remaining parts (**Parts (c) and (d)**), your task is to add shading and lighting effects when “Enable Lighting” is “Yes”. (You can turn on the lighting by calling `glEnable(GL_LIGHTING)`,

and turn off the lighting by calling `glDisable(GL_LIGHTING)`.)

(c) Provide a global ambient white (with color $(1.0, 1.0, 1.0, 1.0)$) light, and also provide a directional light source with black ambient color $(0.0, 0.0, 0.0, 1.0)$, white diffuse and specular color $(1.0, 1.0, 1.0, 1.0)$, and position $(0.0, 0.0, 1.0, 0.0)$ in the **eye coordinate system**. Note that $(0.0, 0.0, 1.0, 0.0)$ is in fact a vector rather than a point, and thus this light source is a distant (directional) light. The colors and the position described above are actually the default values of `GL_LIGHT0`, so you can set it up by just enabling `GL_LIGHT0` without specifying these values. If you do want to specify the values, notice that the position (vector) value is in the **eye coordinate system** (meaning that the vector is fixed relative to the viewer), and you have to specify it in an appropriate place in your program; see **Useful Tips Item 3** on page 4 for more details.

In addition, give your background and the x-, y-, and z-axes the same colors as in Assignment 2. Give your quadrilateral “ground” that indicates the x-z plane a green diffuse color $(0.0, 1.0, 0.0)$ (with default ambient and specular colors), and give your sphere a golden yellow diffuse and specular color $(1.0, 0.84, 0.0)$ (and a default ambient color) with a shininess coefficient of 125.0.

Consider the four unit-sphere files given. Actually, there is one more piece of information implicitly given in the file format: each triangle (given by the coordinates of its three vertices) has its three vertices ordered such that if the fingers of the right hand are curled along the direction of the vertex specification, the thumb will point towards the triangle’s *outward normal* (in this case this normal points towards the *outside* of the sphere). Augment your data structure from Assignment 2 so that associated with each triangle you also store its outward normal vector of length 1 (i.e., the *unit-length* outward normal vector).

Recall that the menu entry “Shading” has two submenu entries “flat shading” and “smooth shading”. Implement these entries with the following shading effects when “Enable Lighting” is “Yes”:

- (1) “flat shading” — flat shade the *filled* triangles of the unit sphere, where the unit-length normal vector associated with each triangle is the normal vector that you just computed and stored.
- (2) “smooth shading” — smooth shade the *filled* triangles of the unit sphere. To assign the unit-length *vertex normals*, use the fact that if the unit sphere is centered at the origin o and v is a point on the surface of the unit sphere, then \vec{ov} is the unit-length normal vector at v .

Notice that the options for the wire-frame sphere and the shadow effects should still work, as you worked out before.

Hint: When lighting is enabled, everything you draw needs normal vector(s) for the shading computation (otherwise you will get strange colors). In particular, you need to specify the ground normal for the ground, and the normals for the sphere as described in (1) and (2) above.

However, when you draw the 3 axes and/or the wireframe sphere, you are actually only drawing edges, for which it would not make sense to assign vertex normals since there is no plane associated with the edge. So you should *turn off* lighting (by calling `glDisable(GL_LIGHTING)`) when you draw edges or points. Similarly, it would not make sense to perform shading on the shadow, and thus you should also *turn off* lighting when you draw the shadow. You can turn on the lighting by calling `glEnable(GL_LIGHTING)`. **(40 points)**

(d) Provide another light source which is positional, white diffuse and specular color $(1.0, 1.0, 1.0, 1.0)$ and default ambient color (black $(0.0, 0.0, 0.0, 1.0)$), with position at $L = (-14.0, 12.0, -3.0, 1.0)$ (that is in the usual **world coordinate system**). This is the light source that produces the shadow.

Set up its constant, linear, and quadratic attenuation values to be 2.0, 0.01, and 0.001, respectively. Give two choices for the type of this light:

- (1) spot light, whose direction is from its position toward the point $(-6.0, 0.0, -4.5, 1.0)$ (again in the usual **world coordinate system**), with the exponent value 5.0 and the cutoff angle 20.0° , and
- (2) point source.

Recall from Assignment 2 that there is a menu entry called “Lighting”. Implement it as a submenu with two submenu entries: “spot light” (corresponding to (1) above) and “point source” (corresponding to (2) above). Your program should allow the user to switch between (1) and (2).

Hint: In OpenGL, you can set the cutoff angle to be 180° , which disables the spot light feature and the light source becomes a point source. **(25 points)**

Useful Tips:

1. Consult the sample code A.12 in Textbook Chapter 6 and Appendix A for **Parts (c) and (d)**.
2. If you experienced a problem of random black spots coloring the sphere surface or a strange behavior of the z-buffer where some parts of the scene disappear randomly, check your `gluPerspective()` command, in which no argument should be given a value too close to 0. Check in particular your value of the Z_{min} parameter.
3. In OpenGL the position and direction of a light are treated as geometric primitives, and can be transformed by the ModelView matrix (the Projection matrix has no effect on them).

There are 3 places in the program to specify the light position/direction, with different effects:

- (1) Before any viewing or modeling transformation, e.g.,

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

```
glLightfv(GL_LIGHT1, GL_POSITION, ...);
```

```
gluLookAt(); /* viewing transformation */
```

```
....
```

```
/* modeling transformation */
```

Then the light position is treated as in the *eye coordinate system*, i.e., it is fixed relative to the viewer, and will move with the viewer.

- (2) After the viewing transformation but before the modeling transformation, e.g.,

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

```
gluLookAt(); /* viewing transformation */
```

```
glLightfv(GL_LIGHT1, GL_POSITION, ...);
```

```
....
```

```
/* modeling transformation */
```

Then the light position is fixed relative to the origin of the *world coordinate system*.

- (3) After both viewing and modeling transformations. Then you can animate the light independent of the viewer or the world coordinate system.