CSCD18 Computer Graphics, Fall 2007
Assignment 2

Part A Written: Due at the drop-box on Wed, Oct. 31 2007 at 11:59PM [50 marks]
Part B Programming: Due at the drop-box and online on Wed, Nov. 7, 2007 at 11:59PM [50 marks]

New collaboration policy: You must work alone and submit your own work. Feel free to discuss the programming assignment with others, but you must write and submit your own code and report.

Part A [50 marks in total]

Below are 4 exercises covering different topics from the last few weeks of class. You are advised to consult the relevant sections of the textbook, the online lecture notes, and your notes from class. Your proofs and derivations should be clearly written, correct, and concise. Please show your work.

1. *[5 marks]* Describe in words what happens to light reflected from the object point that is f-distance away (along the optical axis) from the thin lense camera with focal length f. (Hint: Think about how far one needs to put the film to ensure the point is in focus?) This explanation need not be longer then one or two sentences.

2. *[10 marks]* Let there be two cameras, described in world coordinates by their respective frames of reference, $(\bar{e}_1,\ \vec{u}_1,\ \vec{v}_1,\ \vec{w}_1)$ and $(\bar{e}_2,\ \vec{u}_2,\ \vec{v}_2,\ \vec{w}_2)$. Show mathematically how, given a point $\bar{p}^1$ in the local camera-centered frame of reference of camera 1, you would compute the local camera-centered coordinates of the point in the local camera-centered frame of reference of camera 2.

3. *[20 marks]* We know that, under perspective projection, points map to points, and linearity is preserved. This is an important property for rendering. Nevertheless, perspective projection is not a linear transformation. One example of this is that the midpoint of a 3D line segment will not project to the midpoint of the projected 2D line segment.

   (a) *[10 marks]* Prove that a line passing through two 3D points $\bar{p}_1$ and $\bar{p}_2$ is still a line in the viewplane under perspective projection.

   (b) *[10 marks]* Show that the midpoint of a line segment $\overline{p_1 p_2}$ between two 3D points $\bar{p}_1$ and $\bar{p}_2$ does not project to the midpoint of the projected 2D line segment $\overline{x_1 x_2}$ in the viewplane, where $\bar{x}_1$ is the projection of $\bar{p}_1$, and $\bar{x}_2$ is the projection of $\bar{p}_2$.

4. *[15 marks]* The tangent plane of a surface at a point is defined so that it contains all tangent vectors. In this exercise, you will verify that a specific tangent vector is contained in the tangent plane. Let S be a torus in 3D defined by the implicit equation $f(x, y, z) = (R - \sqrt{x^2 + y^2})^2 + z^2 - r^2 = 0$, where $R > r$.

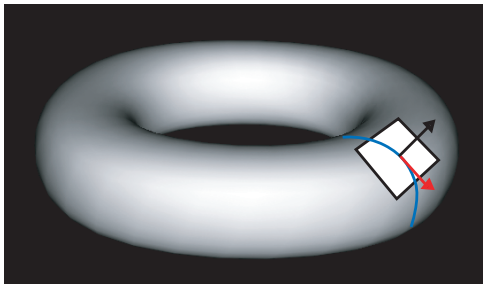
   (a) *[3 marks]* Give a surface normal at a point $\bar{p} = (x, y, z)$, using the implicit equation for the surface.

   (b) *[3 marks]* Give an implicit equation for the tangent plane at $\bar{p}$.

   (c) *[3 marks]* Show that the parametric curve $\bar{q}(\lambda) = (R\cos\lambda, R\sin\lambda, r)$ lies on the surface.

   (d) *[3 marks]* Find a tangent vector of $\bar{q}(\lambda)$ as a function of λ.

   (e) *[3 marks]* Verify that the tangent vector at $\bar{q}(\lambda)$ lies on the tangent plane derived from the implicit form.

(a) 

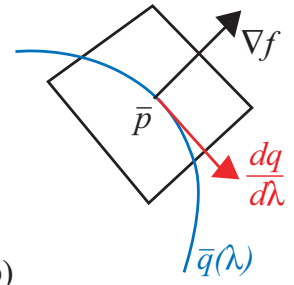
(b) 

Figure 1: *Left:* Torus, showing a tangent plane and normal and 3D curve at a point. *Right:* Close-up view of the tangent plane and 3D curve.

Part B [50 marks in total]

Write a 3D viewer that displays a textured, lit, 3D mesh. First, define a mesh as a 2D array of vertices (i, j), where i and j each range from 0 to 99. The mesh is defined by tessellating a **surface-of-revolution (SOR)** as discussed in the tutorial and the online notes. Neighbouring vertices in the array define 4-point patches, (i.e., (i, j), $(i + 1, j)$, $(i, j + 1)$, and $(i + 1, j + 1)$) which can then be divided to form triangles. Each vertex has a 3D position $\bar{p}_{i,j}$ and a normal $\vec{n}_{i,j}$.

Your job is to create meshes for, and render, several types of SOR. The renderer should have the following features. You may need to refer to the OpenGL reference books in order to find the right calls to make (and their arguments) to set up all the different rendering modes. Starter code is provided to get you up and going.

1. **Mesh rendering.** (15 marks)
   The program always renders the 3D mesh, which can be rotated in the user interface. The user may select different types of meshes using the radio buttons in the user interface. The types of meshes to render are: Sphere, Torus, and Surface From File (described below). (A parametric definition for the torus is given in Question 4 of Part A.) The parametric form for a sphere can be found online (e.g., at mathworld.wolfram.com or Wikipedia). The mesh should be set up so that it is easily visible when the program first loads, and rotates about the world origin when you rotate the camera. In the default rendering style, the mesh is drawn in white without shading or lighting.

2. **Show normals.** (5 marks)
   When the "Show Normals" checkbox is selected in the UI, the surface normals for each surface point should be visualized in red. The surface normals should be clearly visible and correct (you might need to scale them and/or reverse their directions to make them easier to see). Make sure that the normals are not rendered with Phong shading or texturing (i.e., when the buttons "Diffuse Shading", "Specular Shading", or "Texture" are turned on). An example for an ellipse is shown in Figure 2.

3. **Different rendering styles.** (10 marks)
   If the "Wireframe" button is checked, then the surface should be drawn as a wireframe, *without* shading. If either the "Diffuse Shading" checkbox or the "Specular Shading" checkbox is checked, then the surface should be rendered with Phong shading. The surface should be rendered as a white surface. If "Diffuse Shading" is checked, then the surface should have a nonzero diffuse term; otherwise, the diffuse term should be zero. Similarly, the specular term should be nonzero only if "Specular Shading" is checked. The specular shading parameters should be set up such that they

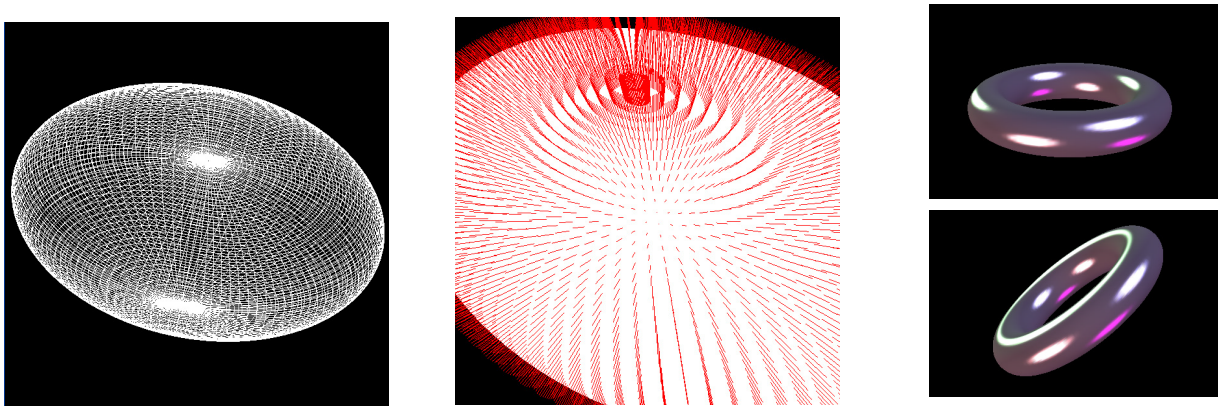


Figure 2: *Left:* Wireframe rendering of an ellipse. *Middle:* Viewing the surface normals in the default rendering mode. *Right:* Torus with specular highlights. Note that, as the torus rotates, the highlights "travel" on the surface, due to the fixed light sources.

create visible highlights on the surface. You should have at least two light sources, and *no light sources should rotate* when the surface is rotated.[1]

4. **Generate a surface from a file.** (10 marks)
   When the "Surface From File" radio button is selected, the program should open a file called "sor.txt". This file will contain 100 numbers separated by spaces, where the i-th entry is r_i. This list of numbers specifies the *profile curve* $r(\alpha)$, such that $r(\alpha) = r_i$, for $\alpha = [0, 1, ...99]$. The formula of the SOR is:
   $$\bar{p}(\alpha, \beta) = (r(\alpha)\cos\beta, r(\alpha)\sin\beta, c\alpha + d) \tag{1}$$
   where c and d are constants that you can adjust to make the surface visible in the display. Moreover, we'll define the curve so that
   $$\left.\frac{dr}{d\alpha}\right|_{\alpha=i} = \frac{r_{i+1} - r_{i-1}}{2}$$
   This formula will be useful for when you need to compute the surface normal. (For $i = 0$, you can use $dr/d\alpha|_{i=0} = r_1 - r_0$, and a similar formula for $i = 99$).

5. **Texture.** (5 marks)
   Pressing the "Texture" radio button should texture-map the mesh. The texture should be loaded from a file in the same directory called "earthmap2.ppm." The texture should exactly cover the mesh (so that all of the texture is visible, there is no wrapping or tiling, etc.). When mapped to a sphere, the surface should look like a globe of the earth. The texturing should be compatible with lighting, i.e., the mesh should be Phong shaded if and only if one of the shading options is selected.

[1] In OpenGL, the lights are normally affected by the current modelview matrix. In order to keep the lights stationary, you'll need to ensure that the lights are always rendered with the same modelview matrix, whereas the modelview matrix might change for other purposes. This can be achieved by pushing the current matrix right before specifying the lights, replacing it with the identity matrix, and then popping the matrix stack.

Marking:

The work you do on this assignment should be your own. The course policy concerning extensions and late assignments are given on the course web site. You may use code snippets from the OpenGL Programming Guide, but not from any other source. All consulted references should be properly acknowledged in your report.

Part A: Hand in your written solutions for Part A on paper by the due date and time in the drop box.

Part B: Hand in a paper listing for the program along with a concise report in the drop box. The report should be a well-structured clearly written explanation of your design, and should contain a description of the parts of the program that you implemented: how is the mesh generated, what happens in the display loop, and what happens outside of it. In addition to correctness, you *will* also be marked on the clarity and quality of your writing. The code should be submitted using the following command:

```
submit -N a2 cscd18 A2.zip
```

Your should submit all the code required to compile and run your program.

Your assignment must run on `fissure` at UTSC. Marks will be deducted if it will not compile and run without modification. If the TA cannot easily figure out how to compile and execute the code, it will receive zero marks. To avoid this unpleasant scenario you may want to submit screen shots of your program running with various options as part of your report (this is not a requirement!).