Assignment 2

CSC 418/2504: Computer Graphics, Fall 2011

Part A due at 7:00 PM (in class) on Wednesday, November 2, 2011
Part B due electronically at 11:59 PM on Wednesday, November 2, 2011

Part A: Written (50 marks)

This section consists of 4 exercises covering 3D geometric primitives, the viewing and perspective transformations, and visibility. They require thought; you are advised to consult relevant sections of the textbook, slides, and your notes from lecture well in advance of the deadline. Proofs and derivations should be clearly written, mathematically correct, and concise. Show steps toward each solution; marks will be subtracted if your derivations or proofs do not include such steps. Partial credit will be awarded, including partial credit for solutions that are only partially complete; it is very important to show your partial answers and reasoning to get such credit.

1: (13 marks) To create a vase shape, the following parametric curve can be rotated about the z axis:

$$\mathbf{p}(t) = \begin{bmatrix} 0 \\ a\sqrt{t} + b\sin(t) \\ ct \end{bmatrix}, t \in [0, 2\pi]$$

a: (5 marks) Give a parametric equation and appropriate parameter bounds for the resulting surface of revolution $\mathbf{p}(u, v)$ in terms of the surface parameters u and v.

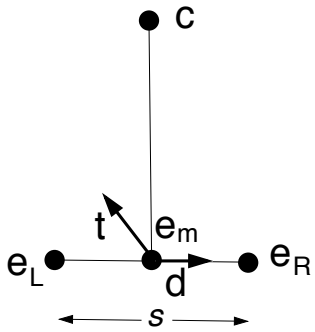
b: (4 marks) Give an equation for the tangent plane at a point $\mathbf{p}(u, v)$ in terms of the surface parameters u and v.

c: (4 marks) Give an equation for the outward–facing normal vector at a point $\mathbf{p}(u, v)$ in terms of the surface parameters u and v.

2: (21 marks) In stereo rendering, two cameras are needed, with slightly different vantage points and view directions; each is used to render an image for the corresponding eye. This can be specified with the following parameters:

- $\mathbf{c}$: The center of interest, a point in world space that lies along the optical axis of both cameras.
- $\mathbf{e}_m$: The midpoint between the eye positions of each camera.
- $\mathbf{t}$: An "up" vector that allows us to specify a tilt rotation about the axis passing through $\mathbf{c}$ and $\mathbf{e}_m$. The z axis is a special case.
- s: The distance between the two eyes.

All answers should be given in terms of the above stereo parameters, as well as any intermediate quantities you specify.



a: (3 marks) Given an expression for the unit vector $\mathbf{d}$ that is perpendicular to both $\mathbf{t}$ and $\mathbf{c} - \mathbf{e}_m$ such that $(\mathbf{c} - \mathbf{e}_m, \mathbf{t}, \mathbf{d})$ is a right–handed coordinate frame.

b: (3 marks) Give expressions for $\mathbf{e}_L$ and $\mathbf{e}_R$, the eye positions of each camera.

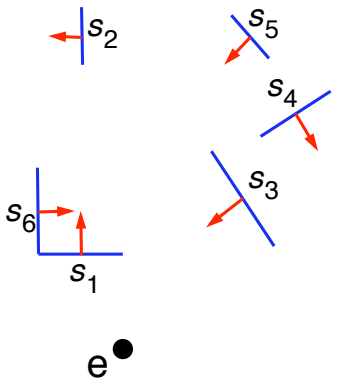
c: (5 marks) Give expressions for the basis vectors of that make up the two cameras' coordinate frames: $(\mathbf{u}_L, \mathbf{v}_L, \mathbf{w}_L)$, and $(\mathbf{u}_R, \mathbf{v}_R, \mathbf{w}_R)$.

d: (5 marks) Give an expression for the 4x4 homogeneous transformation matrix that transforms a point in the camera space of the left eye ($\mathbf{p}_L$) to the camera space of the right eye ($\mathbf{p}_R$). In other words, what is $\mathbf{M}_{LR}$ such that $\mathbf{p}_R = \mathbf{M}_{LR}\mathbf{p}_L$.

e: (5 marks) Give a test that determines whether or not a polygon face with normal $\mathbf{n}$ is a backface that can be culled *when rendering from both cameras.*

3: *Exercise removed. You may skip this item.*

4: (16 marks) This illustration shows a top–down view of a 3D scene, where each blue edge corresponds to a planar square perpendicular to the image plane (i.e. coming out of the page) and the eye of an alternative viewpoint lies in the image plane. The short vectors are normal vectors.



a: (4 marks) Assuming the particular scene and camera placement shown above, is it possible to exclude any polygons from rendering? Explain your answer.

b: (7 marks) Draw a BSP tree for the above scene by adding the polygons to the tree in the order $(S_1, S_2, S_3, S_4, S_5, S_6)$.

c: (5 marks) Describe how your tree will be traversed when rendering the scene from the eye location specified.

Part B: Programming (50 marks)

Your programing task for this assignment is to implement an application for interactive viewing of 3D triangular meshes. Triangular mesh data will be specified in an ASCII file, along with initial camera parameters for the eye position and the gaze and up vectors. You will need to control the camera view using OpenGL, and then implement four different display styles. For the first three styles, you will explore OpenGL's 3D rendering options and learn how to specify material and lighting parameters. The fourth display style requires you to implement a custom view that emphasizes certain mesh edges using criteria based on the current view and local geometric properties of the mesh. Finally, you will implement interactive navigation controls that allow you to track the camera, rotate about the origin, and move toward and away from the object.

Technical Requirements

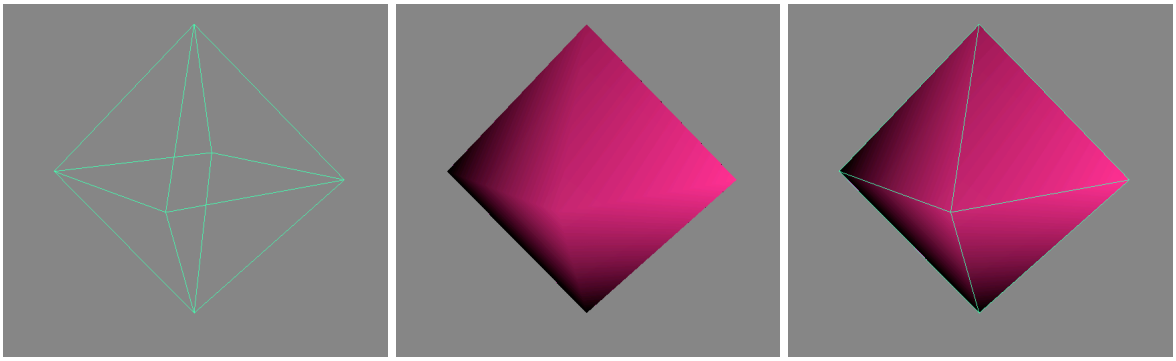
For this assignment, complete the following tasks:

a: (10 marks) Load and display a mesh. The name of the mesh file will be specified as the first command line argument; i.e. we want to run your program by typing *meshViewer something.obj*. The file format is based on the wavefront *obj* format—many commercial applications will export this format. An example of our extended format is as follows:

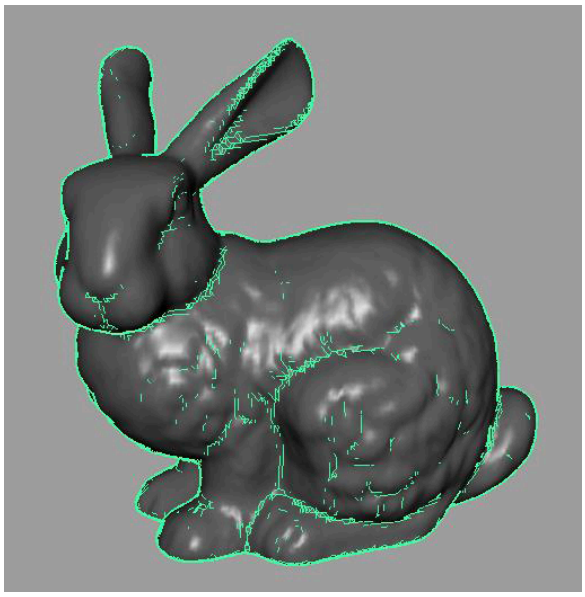
```
v 5.0 0.0 0.0
v -5.0 0.0 0.0
v 0.0 5.0 0.0
v 0.0 -5.0 0.0
v 0.0 0.0 5.0
v 0.0 0.0 -5.0
f 1 3 5
f 3 2 5
f 2 4 5
f 4 1 5
f 6 3 1
f 6 2 3
f 6 4 2
f 6 1 4
e 3.0 4.0 20.0
g -3.0 -4.0 -20.0
t 0.0 1.0 0.0
```

Each line beginning with a *v* specifies the location of a 3D vertex in object space. Each line beginning with an *f* specifies a face via indices into the sequence of vertices. Faces in the *obj* format use 1–based indexing and our files will use counter-clockwise ordering of the vertex indices. The last three lines are optional and specify the initial values for the eye, gaze, and up vector of the camera. All vertex information will be

specified before all face information, and all face information will be specified before all camera information. If the camera information is present, it will all be present in the order shown in the example data. If no mesh file is passed in, display the shape defined by this example data (shown in the figure below). If a file is given and does not have values for **e**, **g**, or **t**, use the values above as defaults. Example files are included with the starter code. Choose good values for the perspective transformation and document them in your report. To get credit for this part, you must have at least some kind of display style working (the sample code includes basic drawing–that is acceptable).



b: (10 marks) Implement three display options based on OpenGL's functionality. These display styles are wireframe, shaded, and shaded with wireframe, and are illustrated above using the sample data given in part **a**. Use *glPolygonMode* and *glPolygonOffset* to control these options. Users should be able to toggle between these display modes by pressing the 1, 2, and 3 keys. For the latter two options, specify an *interesting* color, *interesting* material properties, and one or more point lights. By interesting, we mean that the color should *not* be white, black, or grey, and the material properties must include nonzero values for each lighting component. The third option must show the wireframe in a significantly different color than the surface. Use *glMaterial* and *glLight* to specify the material and lighting properties. Document your choices in your report. Set solid as your default display style when your program starts.



c: (12 marks) Implement a fourth display option that draws *silhouette lines* and *feature lines* on top of a solid shaded view. These types of lines are illustrated above. Silhouette lines are defined as edges that

connect to both a front face and a back face. Use the same test described in lecture for the back face culling problem to determine which edges connect front faces and back faces. These should be drawn thicker than wireframe lines or feature lines. For feature lines, calculate the outward-facing normal of each face. Then measure the angle between these normals. If the angle is above some tolerance, draw the feature edge. A good default value for the tolerance is 85 degrees. Pre–calculate and store your angle values for each edge the first time this display style is selected, or, if you prefer, when you load the mesh. Users should be able select this display style by pressing the 4 key. Add UI for specifying the silhouette line thickness and the tolerance value for the feature lines.

d: (4 marks) Implement interactive tracking of the camera. If the left mouse button is held down and the user drags left and right, move the eye along the **u** basis vector of the camera. Similarly, measure up and down motion and use that to control movement of the eye along the **v** basis vector of the camera. Appropriately scale motion along each axis to provide meaningful control–don't allow small movements to quickly move objects off screen and don't allow large movements to have little effect on the view.

e: (2 marks) Implement interactive dollying of the camera. Dollying is motion of the eye along the **w** axis of the camera. The user should control this by right-clicking and dragging left and right. Mouse motion to the right should move the camera forward (along -**w**) and mouse motion to the left should move the camera back. Again, scale your movement to provide meaningful control.

f: (7 marks) Implement interactive rotation of the camera. This is controlled by the user holding down the shift key, either left or right clicking, and dragging. If the mouse is left-clicked and dragged right and left, rotate the camera about the up vector **t**, with the world origin as the pivot. If the mouse is right clicked, up and down dragging should cause the camera to rotate about an axis along the current **u** vector passing through the world origin. Rotate both the eye position and the camera basis vectors to keep the world origin in place in the camera's view. Appropriately scale the angle of rotation to provide meaningful control. Use *glutGetModifers* to determine if the shift key is held down.

g: (5 marks) Explain everything you did in a written report (described below).

Starter Code

As with Assignment 1, we are providing starter code. The starter code draws a 3D cube and sets the camera view using hard coded values—you must extend the design to read in the file and handle mouse and keyboard input. Empty glut callbacks for keyboard and mouse input are present to get you started with the interactive navigation part. Document all your choices in your report. We recommend that you use the <iostream> and <string> portions of the C++ standard library for file parsing, although feel free to use older–style approaches if you have experience with them. We will again include a template Makefile for compilation and linking of your programs. To unpack, compile, and run this demo on CDF, download the sample code from the Assignments section of the course website and use the following commands:

```
tar xvfz a2.tgz
cd a2/meshViewer
make -f Makefile.CDF
meshViewer
```

Compilation

Your program must compile on CDF Linux with a Makefile, as in Assignment 1. We are providing a similar Makefile.

Turning in your Solution to Part B

In addition to your code, you must complete the report and name it `report.txt`. The report should be a well-structured, written explanation of your design. In particular, describe 1) how you meet each technical requirement and 2) where in your code your implementation of each requirement is. The report does not need to be long or overly detailed, but we want to be able to quickly understand how you did things and where to find the implementation of the requirements. In addition to correctness, you will also be marked on the clarity and quality of your writing. Note that this file should not be thought of as a substitute for putting detailed comments in your code. Your code should be well-commented if you want to receive full (or even partial) credit for it.

Submission

Use the following commands to submit your project. If you are an undergraduate, use:

```
submit -c csc418h -f -a a2 meshViewer.cpp
submit -c csc418h -f -a a2 report.txt
```

If you are a graduate student, use:

```
submit -c csc2504h -f -a a2 meshViewer.cpp
submit -c csc2504h -f -a a2 report.txt
```

The submission deadline for the programming portion is 11:59pm. However, late submissions will be allowed (with the corresponding penalty). Make sure to **not** submit after 11:59pm **unless** you intend submit late and to incur a late submission penalty. We will use the CDF timestamp to determine time of submission. Also, if you submit late, please email me at `psimari@cs.toronto.edu` to inform me and to make sure your assignment is collected.

Compatibility

All of your assignments must run on CDF Linux. You are welcome, however, to develop on other platforms and then port to CDF for the final submission. The sample code is designed to work on Linux and Mac OS X machines, but should be portable to other platforms (including Windows with Visual C++). If you are running your own machine, it almost certainly has OpenGL installed; if not, you can search for an rpm or go to http://www.opengl.org/ (see their getting started FAQ). Mac OS X users might need to install the glui library. If you develop on a platform other than CDF's Linux machines, be sure you know how to compile and test your code on CDF well before the deadline. *Your assignment must run on the CDF Linux machines to receive any credit.* Several marks will be deducted if your code does not compile and/or run without modification. If the marker cannot easily figure out how to compile and execute the code, it will most likely receive zero marks. If you choose to develop the assignment on Mac OS X, test the porting of your code to CDF Linux long before the deadline, perhaps even before you have finished the assignment. Often bugs that are "hidden" when compiling on one platform make their presence known by crashing the application on a different platform. There will not be any special conditions allowed for problems encountered while porting at the last minute.