CS 381 Computer Graphics, Fall 2012
Midterm Exam Solutions

The Midterm Exam was given in class on Tuesday, October 16, 2012.

1. [7 pts] **Synthetic-Camera Model.** Describe the "Synthetic-Camera Model": how does it work?

The Synthetic-Camera Model is a way of modeling a perspective projection of a 3-D scene. We place the viewer and the image to be rendered into the scene. To determine where an object appears in the image, draw a line segment from the object to the viewer; the point (if any) at which this line segment intersects the image is where the object appears.

2. [8 pts total] **Events & Redisplay.**

2a. [4 pts] In event-driven programming, window redisplays are generally *not* properly handled by simply redrawing the window contents when necessary. **Why not?**

First, because there may be multiple reasons why a window needs redisplaying. Redrawing immediately on meeting any of these conditions will mean multiple redisplays when only one is necessary.

Second, because the system (and the user) may consider other task to be higher priority.

2b. [4 pts] How *are* window redisplays properly handled?

To handle a window redisplay, post a redisplay event; deal with the event as usual.

3. [8 pts total] **Animation & Buffers.** When we do real-time animation, we usually do a certain trick involving more than one framebuffer.

3a. [5 pts] What is this trick called, and what exactly do we do? (*What **happens**, not what goes in the code.*)

It is called "double buffering". We use two framebuffers: front and back. We draw into the back buffer. Then, when a frame is completed, we swap the buffers.

3b. [3 pts] What bad thing happens if we do *not* do this buffer trick?

The user sees partially drawn frames, resulting in flickering.

4. [9 pts total] **Performance Issues.**

4a. [4 pts] What does it mean to say that a program is "fill-bound"?

A program is *fill-bound* if its rendering performance is limited by fragment/pixel operations.

4b. [5 pts] Suppose that a fill-bound program is running too slowly. How does the knowledge that the program is fill-bound affect the ways we might try to speed it up?

If a program is fill-bound, then we cannot increase the framerate by speeding up vertex & geometry-related operations. We would need to speed the fragment/pixel operations. One way to do this might be to move some of the work from fragment processing to vertex processing.

5. [14 pts total] **Internal Representations.** In the following questions about OpenGL's internal representations, we are talking about the way OpenGL *acts like* it represents things.

5a. [3 pts] How does OpenGL represent a point in 3-D space, internally?

A point in 3-D space is represented as a 4-D vector in homogeneous form. So <*x*, *y*, *z*> becomes <*kx*, *ky*, *kz*, *k*>.

5b. [4 pts] The vector <2, 3, 4> represents a point in 3-D space. Convert this vector into the form OpenGL uses internally.

I will use 1 for my "*k*".

<2, 3, 4> becomes <2, 3, 4, 1>.

5c. [4 pts] Do part b in reverse: demonstrate how to convert a vector from OpenGL's internal form to a 3-D vector. *Your initial vector should not have any component equal to* 1*.*

I will start with <2, 3, 4, 5>. Dividing each of the first three components by the fourth, we obtain <2/5, 3/5, 4/5>.

5d. [3 pts] How does OpenGL represent a transformation, internally?

A transformation is represented by a 4×4 matrix.

6. [8 pts total] **Viewing.**

6a. [3 pts] What is the "viewing frustum"?

The *viewing frustum* is the region in space in which visible objects lie, when we use a perspective projection. It is called a frustum, because it is shaped like the frustum of a pyramid.

6b. [5 pts] OpenGL includes a far clipping plane, limiting the distances of visible objects. This seems unnatural. Why is it done?

We use a far clipping plane to limit the range of depths of visible objects. This makes the depth test more accurate.

7. [5 pts] **GLSL Qualifiers.** GLSL has three qualifiers that specify communications paths between a shader and another shader, or the application: `attribute`, `uniform`, `varying`. Choose **one** of these three and describe what it means.

Qualifier: *See below.*

Meaning:

Here are all three.

`attribute`: Per-vertex data communicated from application to vertex shader.

`uniform`: Per-primitive data communicated from application to vertex and fragment shaders.

`varying`: Data communicated from vertex shader to fragment shader. Vertex shader sends value at each vertex. Fragment shader receives interpolated value.

8. [8 pts total] **Coordinates & Transformations.**

8a. [4 pts] Explain the difference between "object coordinates", "world coordinates", and "camera coordinates".

Object coordinates are coordinates in the object's frame of reference. This is what we give to a glVertex* command.

World coordinates are the coordinates of the world that our objects are placed in. These are distinct from camera coordinates, if we move the camera.

Camera coordinates are coordinates in the camera's frame of reference. This is the final coordinate system used before projecting onto the image.

8b. [4 pts] "Camera transformations are done backwards and in reverse order." Explain.

It is natural for a camera transformation to be the transformation that places the camera in the world. However, since rendering is done from the camera's point of view, we actually place the world into the camera's frame of reference. Thus, each camera transformation needs to be done in the opposite direction (move the world backward, to move the camera forward) , and the various transformations are performed in reverse order.

9. [8 pts total] **Lambert Cosine.**

9a. [3 pts] What is the Lambert cosine?

The Lambert cosine is the cosine of the angle between the surface normal vector and the light-source direction vector. Equivalently, it is the dot product of these two vectors, assuming they are normalized.

9b. [5 pts] How do we use the Lambert cosine? Be specific.

In the Lambertian illumination model, we multiply the Lambert cosine by the coordinatewise product of the light and paint colors; the result is the apparent color of the surface.