This practice exam has been made a little longer than the actual midterm is expected to be. Also, some questions are expected to be answered using references, whereas on the actual midterm all needed information will be provided.
Problem 1: A scene contains a light at position $L$ and a vertex at position $V$. The scene contains a plane which includes point $P_0$ and has normal $\hat{n}$. The scene also contains a circle of radius $r$ with the center at $C$, and with normal $\hat{m}$.

(a) Let $P$ be the point on the plane where the vertex casts a shadow. Find an expression for $P$.

(b) For what value of $r$ will shadow cast by $V$ be exactly on the circle perimeter. *Hint: Simple once the previous part solved.*
Problem 2: A scene contains a moving sphere with center at \( P(t) = P_0 + tV \), where \( t \) is the time, \( P_0 \) is the position at \( t = 0 \), and \( V \) is the velocity.

The scene also contains a cube with which the sphere may collide. The cube center is at \( C \), a side of the cube has length \( a \), and the face normals are \( \pm \hat{n}_1 \), \( \pm \hat{n}_2 \), and \( \pm \hat{n}_3 \). \textit{Hint: From that information we can determine that a point on one of the faces is} \( C + \frac{a}{2} \hat{n}_1 \), \textit{a point on one of the edges is} \( C + \frac{a}{2} \hat{n}_1 + \frac{a}{2} \hat{n}_2 \) \textit{and one of the corners is} \( C + \frac{a}{2} \hat{n}_1 + \frac{a}{2} \hat{n}_2 + \frac{a}{2} \hat{n}_3 \).

\( (a) \) Describe the three ways the sphere can collide with the cube. Each will require solving a different kind of geometric problem.

\( (b) \) Write expressions for collision time with each type of element (assuming they do collide). \textit{Hint: One of these is a solution to a prior problem.}

\( (c) \) It’s possible the sphere never collides with the cube. Show a computationally inexpensive way to test for this possibility.
Problem 3: A scene contains a tray upon which are cups. On the next page is code that renders the scene using a Tray class and a Cup class. A tray object can hold multiple cups.

Each object provides an array of vertices that describes its appearance (but not contained objects). So, for example, `tray->vtx_array_pointer` returns vertices that form a tray, but not the cups upon it. The vertices are in the object’s own coordinate space. The location member gives the location of an object in its parent’s coordinate space. So `cup->location` is a coordinate in the tray’s coordinate space. See the code on the next page.

(a) The code sets the correct modelview matrix for rendering the tray, but code for setting the modelview matrix for the cup is not finished. Finish it.

(b) Changing a transformation is relatively time consuming, especially when it’s only done for a few vertices, as might be the case with the cup at the center of the loop nest.

Show how to compute cup vertex coordinates for which the modelview matrix set for the tray is sufficient. That is, find a transformation matrix for the code below that will make the `glLoadTransposeMatrixf(mcup)` call unnecessary (but the `glLoadTransposeMatrixf(mtray)` will still be needed).

```c
for ( int i=0; i<cup->vtx_cnt; i++ )
  cup->vtx_array_pointer_new[i] = matrix * cup->vtx_array_pointer[i];
```

(c) If the loop from the previous part were executed each time `show_tray` was called, execution would probably be slower despite the fact that the transformation matrix is not changed as often. Explain why.
Problem 3, continued: The code for rendering tray and cups:

```cpp
void show_tray()
{
    // Location of tray in world coordinate space.
    pCoor tray_location = tray->location();

    // Matrix that rotates tray coordinates (in tray space) to correct
    // position in world coordinate space.
    pMatrix tray_orientation = tray->orientation();

    // Note: modelview is the current modelview matrix, which maps
    // world coordinates to eye coordinates.
    glMatrixMode(GL_MODELVIEW);
    pMatrix mtray = modelview * pTranslate(-tray_location) * tray_orientation;
    glLoadTransposeMatrixf(mtray);
    glVertexPointer(3, GL_FLOAT, 0, tray->vtx_array_pointer);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_TRIANGLES, 0, tray->vtx_cnt());

    while ( Cup* cup = tray->contents_iterate() )
    {
        // Location of cup in tray coordinate space.
        pCoor location = cup->location;

        // A rotation matrix that will rotate the cup into the correct
        // position in tray coordinate space.
        pMatrix orientation = cup->orientation;

        glMatrixMode(GL_MODELVIEW);

        pMatrix mcup = ; // SOLUTION HERE.

        glLoadTransposeMatrixf(mcup);
        glVertexPointer(3, GL_FLOAT, 0, cup->vtx_array_pointer);
        glEnableClientState(GL_VERTEX_ARRAY);
        glDrawArrays(GL_TRIANGLES, 0, cup->vtx_cnt());
    }
}
```
Problem 4: Write OpenGL code that renders a simple cup (a hollow cylinder with one end covered).

The cup center is at the origin. The radius of the cup is 11 (in world space coordinates). The cup is upright (up is (0, 1, 0)), and has a height of 22.

- Include vertices and normals.
- Ignore colors and lighting.
- The code should be reasonably efficient.
Problem 5: A scene contains a wall that has letters (forming some text) cut out of it. The wall itself is dark gray and behind the wall are light-colored objects, which you can see through the holes forming the letters.

You are given a texture with the text (say, the EE 4702 syllabus) where the letters (the ink) are black and opaque ($\alpha = 1$) and the background is transparent (any color, $\alpha = 0$). The command `glBindTexture(GL_TEXTURE_2D, texid_text);` will bind this texture to a texture unit.

Describe how to achieve this effect using a texture, the stencil buffer, and an alpha test. See section 4.1 of the OpenGL 3.0 specification for details on how the alpha and stencil test relate. *If this weren’t a practice exam more details would be given.*

Assume the code `wall->render()` will send wall vertices to OpenGL and `other->render()` will send vertices of other parts of scene to OpenGL.
Problem 6: Determine the best lighting type (emissive, ambient, diffuse, specular) to use for each of the following situations, and explain your reason.

(a) A triangle out in space illuminated only by a nearby star.

(b) A computer monitor screen (turned on, working properly).

(c) A glossy sphere.

(d) A triangle in a room with white walls and many light sources.
Problem 7: Answer each question below.

(a) OpenGL defines several ways to combine texels with the lighted color to obtain the final color to write. Why does one need to combine a texel with a lighted color, why not just write the texel?

(b) What is the advantage of using TRIANGLE STRIPS over TRIANGLES?