Exam: Midterm
Course: CSE5542
Instructor: R. Machiraju
Duration: 11:00 AM 3/5/15 - Midnight, 3/8/15
Total: 150 Points
Submission: Please submit on Carmen. A dropbox will be available!
Allowed: Piazza Discussions, Open Books, Open Texts and Open Minds!
Not Allowed: Talking to any other human being inside and outside class!
Not Allowed: Googling answers!

How do I know you did not cheat?: I will not know. But you will know! And you need to answer yourself every day or at least some day! I believe in individual responsibility. I also believe that we should all trust each other and always act in good faith!

Please sign the following testimonial before you start the exam!

I promise that I will not betray the trust that the instructor, Prof. Machiraju, has bestowed upon us! I will only use the class slides, the text and my own inner resources and nothing else. I promise to be true to myself and follow the guidelines set by the instructor.

---------------------------------------------------------
1. **(30) Normals -**

1.1 (18) Consider the pyramid shown below. Find the unit, outward, normals of faces:
   a. (2) AOC (on YZ plane)
   b. (2) ABO (on XY plane)
   c. (2) OBC (on YZ plane)
   d. (12) ABC (inclined)

Please note that you do not have to compute anything really for first three faces. The last one requires you to find the cross product of two vectors.

1.2 (12) Write down the unit outward normals of all six principal planes of the cube as shown in the adjacent figure. The marked axes are the positive ones. Please use suitable notation to mark corners and faces.
2. **(30) Vectors & Dot Products** -

2.1. **(15)** Consider the scene to the left. A light source is at (0,2) over a square centered at the origin and of size 2x2. A viewer is at (2,1). Draw the $\mathbf{L}$ (light), $\mathbf{V}$ (view or camera vectors), and outward unit $\mathbf{N}$ (normal) for all four corners of the square (point P in figure to the right).

Further, you can ignore the corners or assume that the corner is consistently with one of the edges. Let us go with the latter scheme. Consider the top right corner at (1,1); I could assign it to the top edge (with arrow pointing to the right), while the left top corner at (-1,1) is assigned to vertical edge (with arrow pointing up). Now plot the variation of the quantity $\mathbf{N} \cdot \mathbf{L}$ (dot product of $\mathbf{N}$ and $\mathbf{L}$) on the four exterior edges in a single graph. Please normalize $\mathbf{N}$ and $\mathbf{L}$ first. You need not calculate the exact values. I need to see trends. The x-axis of the graph should include all points lying on the edges listed in clockwise order and starting at the top-left corner. In essence start at point (-1,1) and go around the square in a clockwise manner by following the arrows. Pick some major points on the square and mark them on the x-axis. The y-axis denotes the quantity $\mathbf{N} \cdot \mathbf{L}$ with a caveat; negative quantities should be clamped to zero. In essence, you will compute $\text{maximum}(0, \mathbf{N} \cdot \mathbf{L})$.

2.2. **(15)** Consider the adjacent scene: The unit circle (centered at origin and radius 1) is closed and perfectly reflecting. A light source and viewer are at the origin. All light is inside the circle. The normals at all points point inwards towards the origin and so do the light and view vectors. Draw the variation of the quantity $\mathbf{N} \cdot \mathbf{L}$ as above and $\mathbf{V} \cdot \mathbf{R}$ of all points in a single graph. $\mathbf{R}$ is the reflection of light from the surface and in this case will also be the same direction as $\mathbf{L}$. The x-axis of the graph should be composed of salient points (e.g., at quadrants, octants, etc.) lying on the inside of the circle in clockwise order. You can start at the point (0,1).
3. **(15) Camera Space.** For a camera located at origin and an object located at (0, 0, 10), find the (u, v, n) space when the up vector is the negative y-axis. The answer is really simple; use that to verify your steps.
4. **Matrices Galore (45)**

4.1. (15) What transformations will this model matrix achieve? Write down the two separate transforms? Will the order of the two transforms matter (or do they commute?).

\[
M = \begin{bmatrix}
  1 & 0 & 0 & 1.0 \\
  0 & 0.707 & 0.707 & 1.0 \\
  0 & -0.707 & 0.707 & 1.0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]

4.2 (15) Find the inverse of matrix M and determinant. Once again, you really do not have to compute. Just think :). Ask what kind of transform it is?

4.3 (15) Apply the appropriate transformation to the normals of the tetrahedron in **Question 1.1**. Please see your notes to learn which actual matrix should be applied to the normals.
5. (30) Answer these GLSL pertinent questions
   (10) 5.1 - What does this code do? Assume the objects are bunch of points.

   **Fragment Shader**
   ```glsl```
   void main(void){
       float alpha = 1.0 - length((gl_PointCoord-0.5)*2.0);
       gl_FragColor = vec4(1.0,1.0,1.0,alpha);
   }
   ```glsl```

   **Vertex Shader**
   ```glsl```
   attribute vec3 vertex;
   uniform mat4 _mvProj;
   uniform float pointSize;
   void main(void) {
       gl_Position = _mvProj * vec4(vertex, 1.0);
       gl_PointSize = pointSize / gl_Position.w;
   }
   ```glsl```

   (10) 5.2. What does these codes do when applied to a teapot model available as a soup of triangles. Note that we have pairs of vertex and fragment shaders!

   **5.2.1 Vertex Shader**
   ```glsl```
   void main(void) {
       vec4 v = vec4(gl_Vertex);
       v.z = sin(5.0*v.x)*0.25;
       gl_Position = gl_ModelViewProjectionMatrix * v;
   }
   ```glsl```

   **5.2.1 Fragment Shader**
   ```glsl```
   void main()
   {
       gl_FragColor = vec4(0.4,0.4,0.8,1.0);
   }
   ```glsl```

   **5.2.2 Vertex Shader**
   ```glsl```
   void main(void) {
       vec4 v = vec4(gl_Vertex);
       v.z = 0.0;
       gl_Position = gl_ModelViewProjectionMatrix * v;
   }
   ```glsl```

   **5.2.2 Fragment Shader**
   ```glsl```
   void main()
   {
       gl_FragColor = vec4(0.5,0.5,0.5,1.0);
   }
   ```glsl```
(10) 5.3 Things to Come:
What does this vertex shader code? Hint - go back to Question 1. To begin with this code is not modern, and latest-and-greatest GLSL code and let us be agnostic of WebGL. I wanted to give you a glimpse of how things were done before the current versions of GLSL. Please note that the code may not even be functional :). Think of it as a pseudo-code. Also, what does the fragment shader really have to do here?

/* Assume that there is a code in main memory which defines the variables as we saw earlier*/

void main() {
    vec3 normal, lightDir;
    vec4 diffuse;
    float NdotL;
    normal = normalize(gl_NormalMatrix * gl_Normal);
    /* NormalMatrix is specialized to work with normals */

    lightDir = normalize(vec3(gl_LightSource[0].position));
    /* LightSource is a 3D vector */

    NdotL = max(dot_product(normal, lightDir), 0);
    diffuse = gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse;

    gl_FrontColor = NdotL * diffuse;

    gl_Position = ftransform;

    /* ftransform is a function that provides the positions of vertices; the exact content of the function is not important */
}