December 10, 2012 Name _____

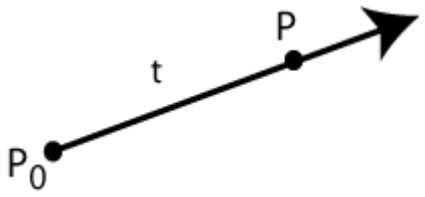
CS445 Final Solutions

Fall 2012

1.	(max = 8)	5.	(max = 8)
2.	(max = 12)	6.	(max = 14)
3.	(max = 20)	7.	(max = 17)
4. .	(max = 9)	8.	(max = 12)
Final Score:		(max=100)	

1) (8 pts) **Rays**
What is the parametric equation of a ray? Besides giving the formula, please explain in words what each of the terms in the formula represents. Include a picture.

Points P on a ray must satisfy: $P = P_0 + t\ dir$
Where
P_0 = the starting point of the ray
dir = vector pointing along the ray direction
t = positive scalar parameter indicating the distance P is along the ray



2) (6 pts each, 12 pts total) **Composition of 3D Transforms**: What is the sequence of transformations needed to achieve the operations given below. Also, include the corresponding inverse. You do not need to write out the 4x4 matrices. Instead, make use of the syntax:
Scale: $S(s_x,s_y,s_z)$
Translation: $T(t_x,t_y,t_z)$
Rotation: $R_x(\Theta)$, $R_y(\Theta)$, $R_z(\Theta)$.

a) A rotation of 35 degrees about the y direction, with fixed point (20,10,2).

The transforms:

$$T(20,10,2)\ R_y\ (35)\ T(-20,-10,-2)$$

The inverse:

$$T(20,10,2)\ R_y\ (-35)\ T(-20,-10,-2)$$

b) A scale by 2, with fixed point at (1,2,3), along the direction defined by the line from (0,0,0) to (1, 1, 0).

The transforms:

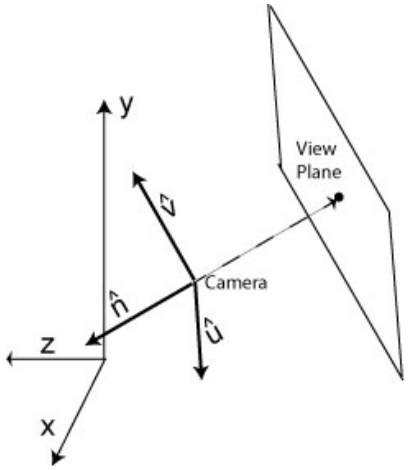
$$T(1,2,3)\ R_z\ (45)\ S(2,1,1)\ R_z\ (-45)\ T(-1,-2,-3)$$
or
$$T(1,2,3)\ R_z\ (-45)\ S(1,2,1)\ R_z\ (45)\ T(-1,-2,-3)$$

The inverse:

$$T(1,2,3)\ R_z\ (45)\ S(1/2,1,1)\ R_z\ (-45)\ T(-1,-2,-3)$$
or
$$T(1,2,3)\ R_z\ (-45)\ S(1,1/2,1)\ R_z\ (45)\ T(-1,-2,-3)$$

3) (20 pts total) **Camera Transforms**: Suppose you are given
- VPN = a vector that points in a direction *opposite* the way the camera looks
- VUP = the up direction vector
- Eye = location of camera in the World Coordinate System.

a) (6 pts) How does one calculate the normalized eye coordinate basis vectors: $\hat{u}, \hat{v}, \hat{n}$ (see picture). Assume you are using a ***right handed*** coordinate system as shown.



$$\hat{u} = \frac{VUP \times \hat{n}}{\|VUP \times \hat{n}\|}$$

$$\hat{v} = \hat{n} \times \hat{u}$$

$$\hat{n} = \frac{VPN}{\|VPN\|}$$

b) (6 pts) How do you use $\hat{u}, \hat{v}, \hat{n}$ and Eye to construct the View matrix, V, which transforms points from the World Coordinate System to the Camera Coordinate System?

ViewRotation = $\begin{pmatrix} -u- & 0 \\ -v- & 0 \\ -n- & 0 \\ 0\ \ 0\ \ 0 & 1 \end{pmatrix}$, and T_{eye} = Translate ($-Eye_x$, $-Eye_y$, $-Eye_z$)

Then View matrix V = ViewRotation * T_{eye}

c) (4 pts) When implementing fly-through navigation, how do you modify the View matrix so that the camera appears to turn to the right or left by some small angle Θ?

$V_{new} = R(\pm\Theta)\ V_{old}$

d) (4 pts) When implementing fly-through navigation, how do you modify the View matrix so that the camera moves forward by some small amount α?

Update the eye position: $Eye_{new} = Eye_{old} - \alpha\ \hat{n}$
Then
V = ViewRotation * T_{eye}

4) (3 pts each, 9 pts total) **Model-View Transformations:** In class, we discussed how points are transformed from Object to Eye coordinates:

$$p_{eye} = V\ M\ p_{obj}$$

where V is the view matrix and M is the modeling matrix. Now suppose we have a standard x-axis rotation matrix $R_x(\Theta)$ for some angle Θ. Describe the effect on either the geometry and/or the camera's view when R_x is inserted as follows:

a) $p_{eye} = V\ M\ R_x\ p_{obj}$

Rotates the point around the x-axis in object coordinates.

b) $p_{eye} = V\ R_x\ M\ p_{obj}$

Rotates the point around the x-axis in world coordinates.

c) $p_{eye} = R_x\ V\ M\ p_{obj}$

Rotates the point around the x-axis in eye coordinates.

5) (8 pts) **Shading Algorithms**: What is the difference between Gouraud and Phong Shading? Which one is preferred and why?

In Gouraud shading, the phong lighting is evaluated only at the vertices. The resulting light intensity at the vertices is then interpolated over the fragments.

In Phong shading, the normals of the vertices are interpolated over the fragments. These normals are then used to compute the phong lighting at each fragment.

Since the lighting can be very sensitive to the direction of the normal (especially the specular lighting), phong shading produces much more accurate results.

6) (14 pts total) **Z-Buffer:**

a) (6 pts) What is a z-buffer? Why is it used? How does it work?

The framebuffer is a block of memory with dimensions nxm, where each element is a pixel color. The associated z-buffer also a block of memory which has dimensions nxm, but each element contains a distance. Each distance corresponds to the distance from the camera of the closest object that has been encountered, so far, at that pixel.

It is used take into account occlusions, i.e. which objects are blocked by other objects.

When a fragment is to be drawn, its distance from the camera is first compared with the value in the z-buffer. If the fragment's distance is smaller than what is in the z-buffer, then the fragment is written to the framebuffer and the corresponding z buffer is updated. If the fragment's distance is greater than the value in the z-buffer then the fragment is not written to the framebuffer.

b) (4 pts) Why can it cause problems when implementing transparency?

Suppose a transparent object is in front of a second object. If transparent object is drawn first, then the transparent object cause the z-buffer to be updated. The second object, which is farther away, will then not be drawn at all. If the second object is not drawn then it won't be seen through the transparent object.

c) (4 pts) How might the values in a z buffer be used to identify the edge boundaries between objects and the background?

After an image has been rendered, one can take a look at where there are discontinuities in the z-buffer. That is, if two adjacent elements in the z-buffer are very different, then those two elements must have been at very different distances and, thus, were most likely different objects.

7) (17 pts) **Shaders:**

a) (9 pts) What is the difference between a vertex and fragment shader? Include as part of your answer: Which shader is executed first? What is a fragment? What are the minimal inputs/outputs of each shader?

A vertex shader performs calculations on the vertices and a fragment shader performs calculations on fragments.

A fragment is a preliminary pixel.

The vertex shader is executed first. Rasterization converts the vertex primitives (e.g. triangles) to fragments which are then passed to the fragment shader.

Vertex positions are passed into the vertex shader. The transformed position must be an output of the vertex shader.

The position (from rasterization) of the fragment is sent to the Fragment shader. The output of the fragment shader is the color of the pixel.

b) (4 pts) What is the difference between an attribute variable and a uniform variable?

An attribute variable is a variable associated with a vertex. That is, each vertex will have a value for a given attribute varilable, e.g. position, color, normal

The value of a uniform variable is constant across an entire primitive, and thus all vertices and fragments of that primitive (e.g. triangle) will have the same value. An example of a uniform variable is the model view matrix.

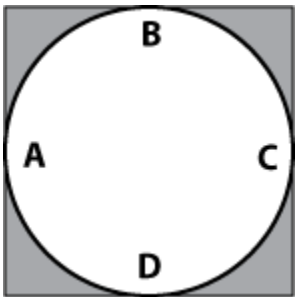
Attribute variables can be accessed in the vertex shader but not the fragment shader. Uniform variables can be accessed from either the vertex or fragment shader.

c) (4 pts) Vertex object coordinates are sent to the shader and transformed to eye coordinates using the modelview matrix. However, the position of a light is sent to the shader in eye coordinates. Why is it necessary to treat the light differently than the vertex coordinates?

The vertex shader is executed for each vertex. The modelview matrix sent to the shader are the transformations needed to convert the vertex from object to eye coordinates. However, these are not the same transformations needed to transform the light to the eye coordinates, therefore, these transformations must be done in the application.

8) (3 pts each, 12 pts total) **Texture Coordinates**: Suppose you want to use the image below as a texture on a square (e.g. a quad modeled as 2 triangles, although this doesn't matter for this problem).

image to be used as a texture object:



For each of the following possible texture coordinates, draw how the texture will look on the square. *Assume that the texture wrap parameter is set to repeat in both dimensions*.