Last Name: _____ First Name: _____ Cornell NetID: _____

CS 4620 Midterm, March 20, 2018
SOLUTION

1. [20 points] **Meshes**

   Suppose we are given the geometry shown in the left side of Figure **??**, and we are tasked with collapsing the red edge. This means combining its neighboring vertices into a new vertex. The resulting geometry is given on the right side of the figure. The new vertex's position will be the average position of the two old vertices.

   Suppose also that each vertex in this mesh is associated with a normal vector. The new vertex's normal vector will be parallel to the average of the normals of the old vertices.

   Below is a buffer that is filled with the positions of each of the vertices of the original mesh:

```
-1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, 0.2, 0.0,
0.0, -0.2, 0.0,
0.0, -1.0, 0.0,
1.0, 0.0, 0.0
```

   We are also given the following normal buffer:

```
-5/13, 0.0, 12/13,
0.0, 5/13, 12/13,
0.0, 8/17, 15/17,
0.0, -8/17, 15/17,
0.0, -5/13, 12/13,
5/13, 0.0, 12/13
```

   Finally, we also have an index buffer. Each grouping of 3 entries describes a single triangle in the original mesh. An entry with value i selects the ith point in the position buffer above, and the ith vector in the normal buffer above.

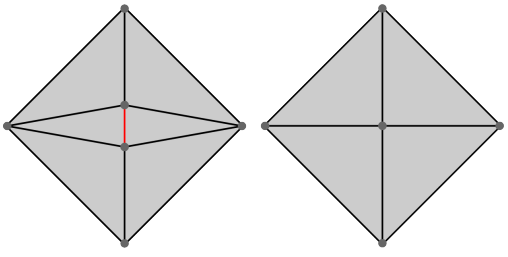


Figure 1: Left: the original mesh structure, with the edge to be removed highlighted in red. Note that this mesh is planar. Right: the resulting mesh structure.

```
0, 2, 1,
0, 3, 2,
0, 4, 3,
1, 2, 5,
2, 3, 5,
3, 4, 5
```

(a) Write new position, normal, and index buffers that correspond to the mesh shown on the right side of Figure **??**. For vertices that remain in the mesh, leave the data from the original mesh in the same order. When adding new data to the position and normal buffers, add it at the end of the buffer. When multiple orderings of indices are possible, chose the ordering that puts the lowest index first. The triangles should be oriented such that the front side is facing out of the page. Use 0-based indexing.

Position buffer:

Normal buffer:

Index buffer:

(b) Suppose that we are to perform similar edge-collapsing operations many times on a large mesh. Name a mesh storage scheme that would allow us to do this more efficiently than the indexed structure defined above.

Solution: (a): The new position buffer is:

```
-1.0, 0.0, 0.0,
0.0, 1.0, 0.0,
0.0, -1.0, 0.0,
1.0, 0.0, 0.0,
0.0, 0.0, 0.0
```

[7 points: 2 for keeping the right four points, 2 for removing the right two points, 2 for adding the correct point, 1 for having everything according to spec.]

The new normal buffer is:

```
-5/13, 0.0, 12/13,
0.0, 5/13, 12/13,
0.0, -5/13, 12/13,
5/13, 0.0, 12/13
0.0, 0.0, 1.0
```

[5 points: 2 for having the right set of old normals present, 2 for adding the correct new normal, 1 for having everything according to spec.]

The new index buffer is:

```
0, 4, 1,
0, 2, 4,
1, 4, 3,
2, 3, 4
```

[6 points: 1 for having the right indices in each triangle, 2 for having them all ordered correctly.]

[Any cycling of these indices within any group of 3 is technically correct; it is also correct if groups of 3 are rearranged. However, this is not what we asked for, so -1 for each for these errors.]

[-2 for listing at least one face clockwise.]

(b): Triangle neighbor structure, winged-edge structure, or half-edge structure are all valid answers. [2 points]

2. [15 points] **Cubemap**

Similar to the written question in A4 shaders, imagine we have a cube map with cross-like texture, shown in the figure below.

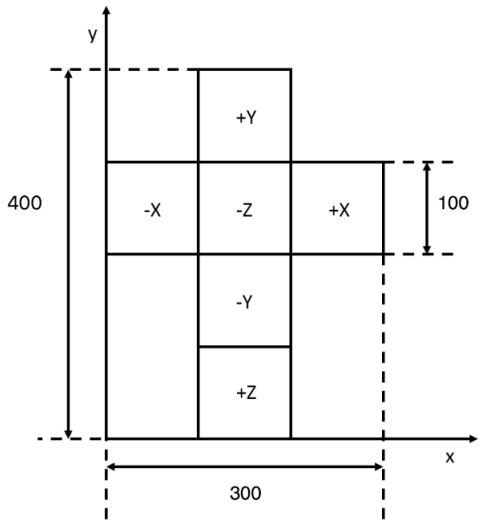


Figure 2: cube map with cross like texture.

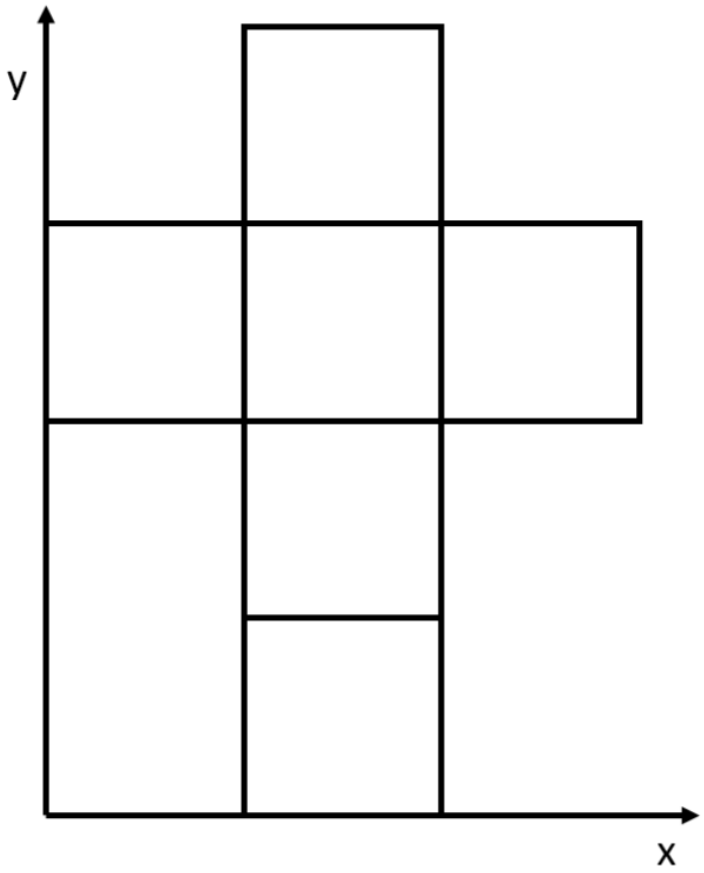
Suppose we have a perspective camera positioned at $(0, 0, -25)$ looking towards $(0, 0, 0)$. A square mirror surface 100 by 100 units in size is centered at $(0, 0, 0)$ with its normal pointing towards $-z$. The scene is rendered using the cube map as an environment map.

- (a) Point A $(25, 0, 0)$ lies on the specular reflecting square. Imagine a viewing ray that intersects the square at A and is specularly reflected. What is the 3D unit vector that will be used to look up in the cube map? At what pixel coordinates will the texture in Figure 2 be sampled?

- (b) Answer the same two questions for Point B $(25, 25, 0)$.

- (c) Answer the same two questions for Point C $(50, 0, 0)$ on the edge of the mirror.

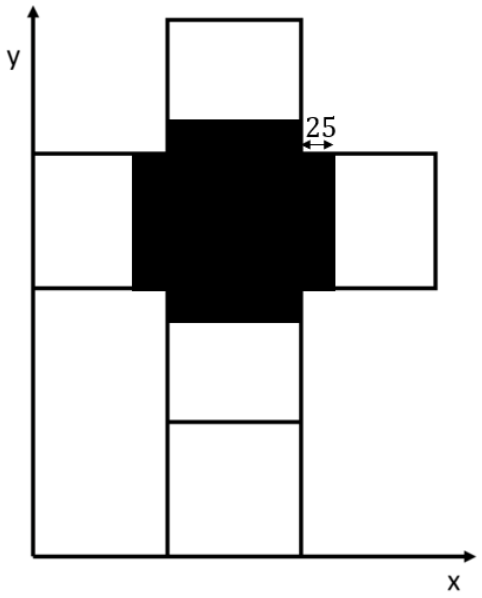
- (d) Which part of cube map is reflected by the specular square? Shade in the region in the figure below, marking dimensions so that we can tell exactly which part is shaded. Briefly explain the reasoning behind your answer.



Solution: [−1 for each normalization error.]

- (a)$(1, 0, -1)/\sqrt{2}$, $(200, 250)$. [3 points] The answer $(225, 250)$, arrived at by treating the cubemap as a finite-sized box (this might be called a "sky box"), is not right but we accepted it also because we realized we told some students a wrong way to do this.
- (b)$(1, 1, -1)/\sqrt{3}$, $(200, 300)$. [3 points] The answers $(225, 300)$ or $(200, 325)$ were also accepted for the same reason.
- (c)$(2, 0, -1)/\sqrt{5}$, $(225, 250)$. [4 points]
- (d) [5 points: 3 for having the idea of a face plus rectangular pieces, 2 for having the details right.]

Last Name: _____ First Name: _____ Cornell NetID: _____



Last Name: _____ First Name: _____ Cornell NetID: _____

3. [15 points] **Transformation Stack**

Consider how an object is transformed in the programming part of the Scene assignment. The object's canonical-to-world transformation is composed of 5 different transforms in a "stack." Here is a recap of the transformations.

- T A translation.
- R_x A rotation around the x-axis.
- R_y A rotation around the y-axis.
- R_z A rotation around the z-axis.
- S An axis-aligned scaling.

The stack of transformations is combined in order so that the final transformation is $M = TR_xR_yR_zS$.

The transformations are controlled by three kind of manipulators: translation, rotation, and scale. There are three manipulators of each kind, controlling transformations in the x, y, and z direction or about the x, y, and z axes.

For the following question, give your answers using the five transformations described above.

- (a) Consider a point **o** on the object, written in the coordinate system of the object. What is **o** written in world coordinates?

- (b) Consider a point **p** on the x-axis translation manipulator, written in the coordinate system of the manipulator itself. What is **p** written in world coordinates?

- (c) Consider a point **q** on the y-axis rotation manipulator, written in the coordinate system of the manipulator itself. What is **q** written in world coordinates? What is the axis of rotation in world coordinates?

Solution: (a): The full stack should be used. The answer is Mo. [4 points: 2 for something with a partly right explanation; 2 for correct answer even with unclear explanation.]
(b): The stack above and up to T should be used. The answer is Tp.[4 points: 2 for something with a partly right explanation; 2 for correct answer even with unclear explanation.]
(c): The stack above and up to R_y should be used. The answer is TR_xR_yq. The y direction in the local frame is $d_y = (0, 1, 0)$, thus the answer is $TR_xR_yd_y$ (or TR_xd_y, since R_y does not

change the axis). [7 points: 4 for the point and 3 for the axis. Only -0.5 if R_y is left out for the point.]

4. [15 points] **Normal interpolation**

(a) Explain why we want to interpolate vertex normals for shading.

(b) True or False?
For any point on a face ABC, the normal we use for shading is a convex linear combination of the vertex normal vectors $\mathbf{n}_a$, $\mathbf{n}_b$, and $\mathbf{n}_c$.[1]

(c) Suppose we have a triangle mesh with shared vertex normals, which contains two triangles PQR and RQS that share an edge. What are the barycentric coordinates of the midpoint of the edge QR in each of these two triangles?

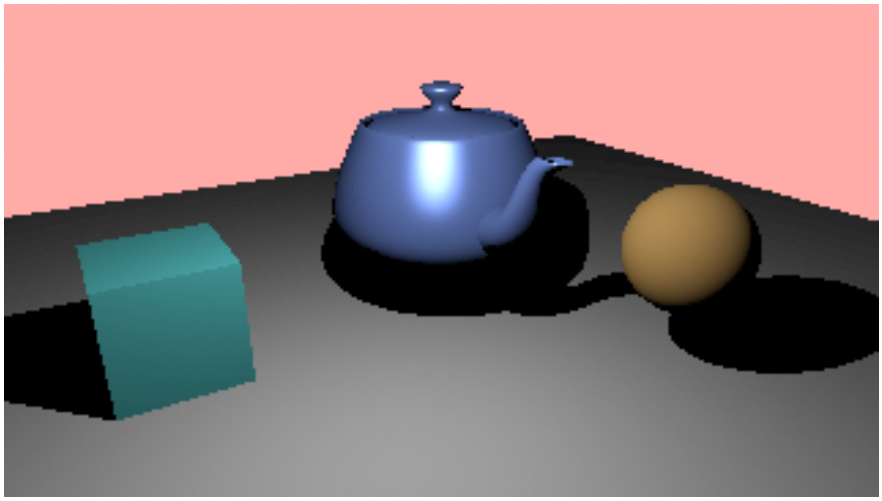
(d) Prove using barycentric coordinates that the interpolated normal at any point on the edge QR has the same value when calculated in triangle PQR using the vertex normals $\mathbf{n}_p$, $\mathbf{n}_q$, and $\mathbf{n}_r$ or in triangle RQS using the vertex normals $\mathbf{n}_r$, $\mathbf{n}_q$, and $\mathbf{n}_s$.

Solution:
1) Surface normal is generally not continuous across edge. If we interpolate vertex normal shading, normal is continuous across edge so we get more smooth rendering result. [5 pts: 3 for getting at the idea of continuity; 2 for a coherent explanation.]
2) False. Renormalizing the interpolated normal makes it no longer necessarily a convex linear comibnation of the vertex normals. [1 pt. We didn't ask for an explanation, so sensible or non-sensible explanations don't matter.]
3) The coordinates are $(0, 0.5, 0.5)$ and $(0.5, 0.5, 0)$. Putting the coordinates in a different order is fine, since we don't have a clear convention for how to order or name these coordinates in triangles that are not called "ABC." [5 pts: 3 for the pattern of one zero coordinate in each, 2 for having the right numbers. -1 for not actually writing down the coordinates.]

[1] Reminder: a convex linear combination is a linear combination in which the weights are positive and sum to 1.

4) For any point X on QR, $\frac{|QX|}{|QR|}$ and $\frac{|XR|}{|QR|}$ are equal to the two barycentric coordinate values, and the third is zero. This same holds in both triangles. So the normal at P is consistent when calculating using vertex normal of A, B, C or vertex normal of A, B, D. [4 pts: 2 for arguing (maybe implicitly) that only Q and R affect the interpolated value; 2 for arguing they will have the same weights in both triangles.]

5. [15 points] **Ray Tracing Bugs**



The image here is a correct ray traced rendering of a scene consisting of a single light, a box, a teapot mesh, a sphere, and a plane (approximated by a box with a small height). The teapot has Microfacet Beckmann shading, and all other surfaces have Lambertian. We have modified the ray tracer to introduce several single-line bugs, one for each image you see on the next page. For each image, choose one of the three possible explanations for the bug:

- i. The error is caused by a problem with ray generation.
- ii. The error is caused by a problem with ray intersection.
- iii. The error is caused by a problem with shading computations.

For each choice, back it up with an example of an error that would cause the observed symptoms. There is no right or wrong explanation; only plausible and implausible ones. But when there is a clearly plausible cause, very far-fetched explanations will not make full credit. Shadow computations and texture operations count as part of shading. Computing surface normals counts as part of ray intersection. The first one is done as an example.

(a) Example: Bug ii: Ray intersection is not properly determining the first object hit.

(b)

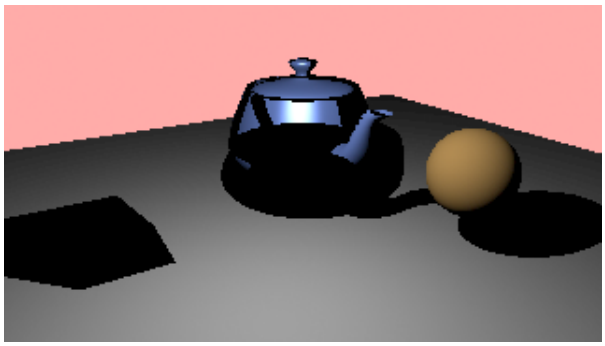
(c)

(d)

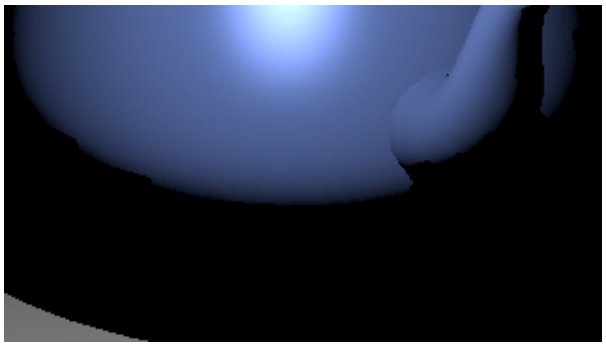
(e)

(f)

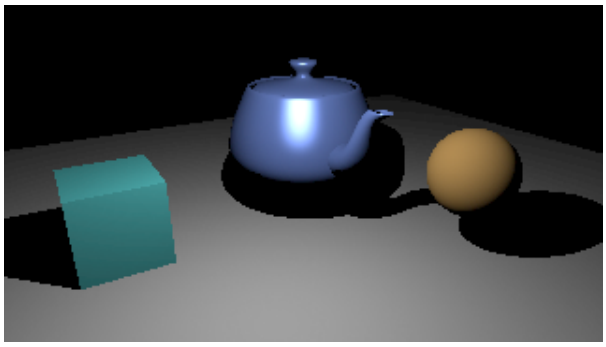
Solution:
(a) Already solved.
(b) Bug i: Incorrect camera ray generation.
(c) Bug iii: Rays that don't intersect any object are not handled properly. Should set to background color.
(d) Bug ii: Incorrect triangle intersection calculation.



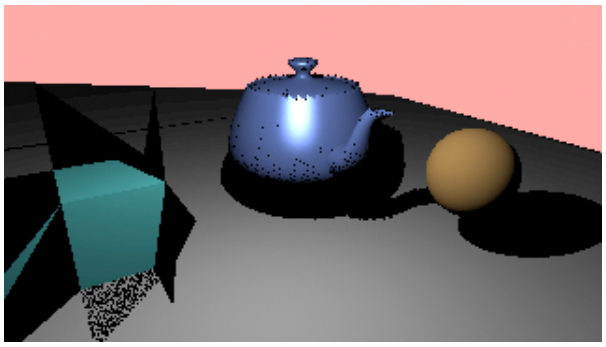
(a)



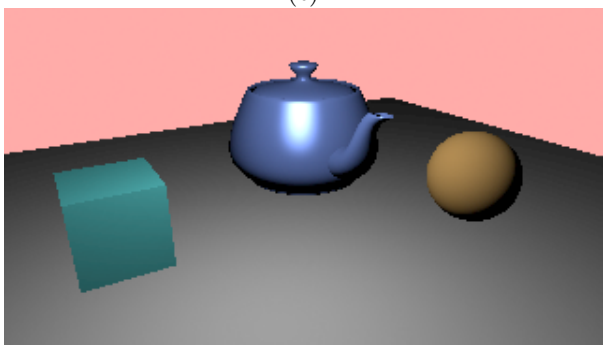
(b)



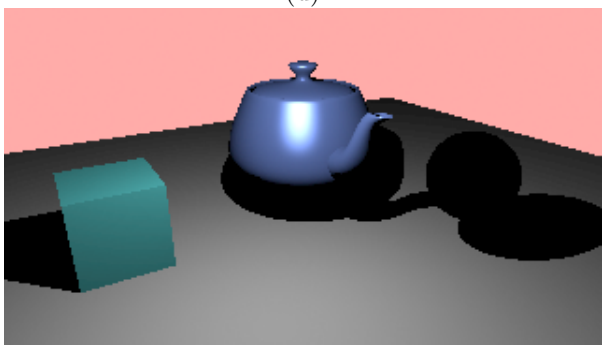
(c)



(d)



(e)



(f)

(e) Bug iii: Didn't properly check if is shadowed.
(f) Bug ii: One pausible error: incorrect sphere intersection, returning the larger root all the time; another pausible error: incorrect surface normal calculation.

[each is 3 points: 2 for a story that makes some kind of sense, 3 for a fully plausible explanation.]