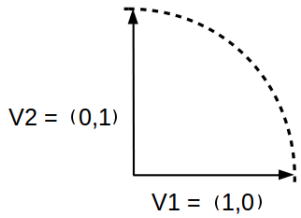
Last Name: _____ First Name: _____ Cornell NetID: _____

CS 4620 Final SOLUTION

1. [12 points] **Spherical Linear Interpolation**. In the animation process, we were often interested in interpolating unit quaternions over time. We discussed two plausible methods for this task. The first was the same method used to interpolate normal vectors for shading: linear interpolation followed by a normalization step (normalized LERP). The second was spherical linear interpolation (SLERP). In this problem, we will be examining the differences between these two methods by looking at the following example, where we are interpolating between a pair of 2D unit vectors, $\vec{v_1}$ and $\vec{v_2}$.



   (a) In 1-2 sentences, explain what the primary advantage of SLERP is for interpolation in this setting over normalized LERP.
   Solution: Normalized LERP will interpolate at a non-constant rate along the unit circle.

   4 points, on the usual 3 = mostly, 2 = almost, 1 = wrote something scale.
   "slerp prevents shrinkage" is 1/4 since both these methods will do that.
   "slerp is good for interpolating rotations" is 1/4 because it is not explaining the advantage.

   (b) Prove that SLERP has the property you described in part A by differentiating the interpolated vector with respect to t. You may find some of the formulas on the front of the exam to be helpful.

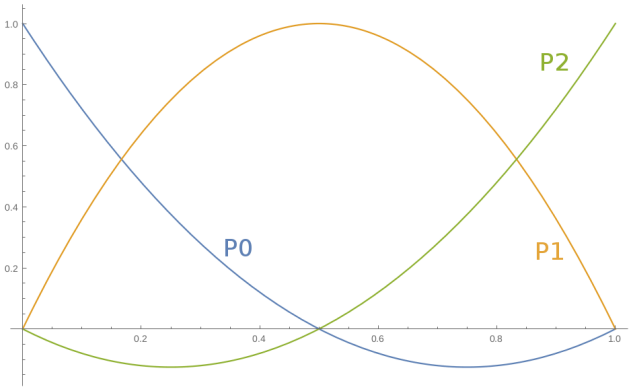
Solution: Note that $\Omega = \frac{\pi}{2}$. Then, $q(t)$ becomes

$$q(t) = \sin(\frac{\pi}{2}(1-t))\vec{v_1} + \sin(\frac{\pi}{2}t)\vec{v_2}$$
$$\frac{dq}{dt} = -\frac{\pi}{2}\cos(\frac{\pi}{2}(1-t))\vec{v_1} + \frac{\pi}{2}\cos(\frac{\pi}{2}t)\vec{v_2} = \frac{\pi}{2}\sin(\frac{\pi}{2}t)\vec{v_1} + \frac{\pi}{2}\cos(\frac{\pi}{2}t)\vec{v_2}$$

We see the magnitude of the resulting vector is $\frac{\pi}{2}$ which is independent of t, so, in this case, SLERP interpolates at a constant rate.
6 points: correct computation of derivative
2 points: correct interpretation and explanation (just computing the norm as a constant value is sufficient)

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

2. [16 points] **Splines: Blending Functions** Consider the following blending functions for a quadratic spline segment, which is controlled by three control points P_0, P_1, and P_2.



Their equations are given by

$$b_1(u) = 2u^2 - 3u + 1$$
$$b_2(u) = -4u^2 + 4u + 0$$
$$b_2(u) = 2u^2 - u + 0$$

(a) In 1-2 sentences, explain what blending functions are, in general. Solution: For a given u, the blending functions give you the weights in a weighted sum of control points
4 points, usual scale

(b) If the spline's parametric function is $f(u) = a_0 + a_1u + a_2u^2$, give a matrix X such that $(P_0, P_1, P_2)^T = X(a_0, a_1, a_2)^T$. You can answer this question without them, but if your solution includes functions such as matrix determinants/inversions/transposes, you may skip the computations and simply write your answer in terms of determinant/inversion/-transpose notation. Solution:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}^{-1}$$

It is also fine to directly derive the matrix by observing that the curve goes through the control points.
6 points: 4 for having a valid approach, 2 for getting the numbers right.

(c) Without doing all the simplification in part C, what features of the graph immediately indicate what the control points P_0, P_1, and P_2 represent? Solution: P_0 is the only nonzero blending function at $t = 0$ so $f(0) = P_0$
P_1 is the only nonzero blending function at $t = .5$ so $f(.5) = P_1$
P_2 is the only nonzero blending function at $t = 1$ so $f(1) = P_2$
6 points: 3 for correctly stating how the curve passes through the points, 3 for explaining how to tell.

3. [16 points] **Shader Programming** You will solve a shader programming problem like the ones you had in the Shaders Assignment.

   You will implement a toon shader that renders objects in the cartoon-like style shown in Figure 1. This is achieved by rendering with quantized colors (to imitate cel shading). The quantized shader uses a modified version of Lambertian shading in which the cosine of the angle between the surface normal and the light vector is quantized before using it to compute the final color. There is only one directional light source in the scene. To make sure the boundaries between the color regions form sharp, smooth curves, perform this quantization calculation for each pixel rather than interpolating across triangles. We ask you to quantize the cosine value according to the following pseudocode:

   ```
   intensity = floor(4 * cos_theta) / 4.0;
   ```

   You will have to define two files, one for the vertex shader and one for the fragment shader. We don't need you to write syntactically perfect GLSL code; pseudocode or GLSL-like code is fine, but it needs to be unambiguously clear what your shader is doing.

   Assume your Java code has access to vectors containing:

   - the camera position in world coordinates,
   - the direction toward the light source in world coordinates,
   - the diffuse surface color,

   as well as 4x4 matrices containing the usual transformations:

   - projection,
   - viewing (aka. camera),
   - modeling (aka. object-to-world).

   Your meshes have the following data available per vertex:

   - position,
   - normal,
   - texture coordinate.

   Here are some questions to get you started, you don't need to answer them, just include the needed variables in you shader code. Remember to unambiguously specify what is stored in the variable with comments, referring specifically to the data we've said is available. For instance, you might say that a particular variable contains the inverse of the modeling transformation.

   1. Which transformations should be passed to the vertex shader? Create uniform variables for them in your program and use them! Write comments that explain exactly what is to be stored into each one.
   2. Which vertex attributes should be passed to the vertex shader? Create vertex attribute variables in your program and use them! Write comments that explain exactly what data is passed into each one.
   3. Which variables should be passed between the vertex and fragment shader? Create varying variables for them in your program and use them! Write comments to indicate the function of each one.

Vertex shader:

```
// Put uniform variables here
// uniform mat4 apple;




// Put vertex attributes here
// attribute vec3 peach;




// Put varying variables here
// varying vec3 pear;




void main() {
   // Write vertex shader code here









    gl_Position = ...
}
```

Fragment shader:

```
// Put uniform variables here
// uniform float box;




// Put varying variables here
// varying vec3 pear;




void main() {
   // Write fragment shader code here









   gl_FragColor = ...
}
```

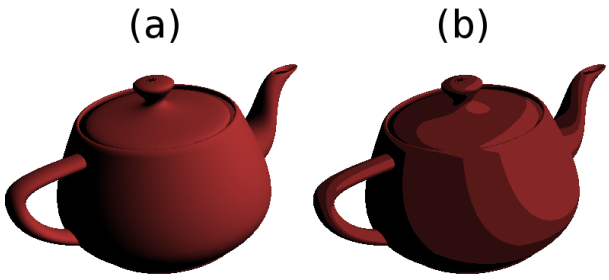


Figure 1: Building up the toon shader. Image (a) shows the teapot under diffuse shading, (b) is the final toon result.

Solution: **Vertex shader:** (9 pts)

```
uniform mat4 modelview; // (viewing matrix) * (modeling matrix)
  // [+2, one for decent explanation]
uniform mat4 modelview_invtranspose;  // inverse of transp of modelview [+1]
uniform mat4 projection; // projection matrix [+1]

attribute vec4 a_position; // vertex positions in object space [+1]
attribute vec4 a_normal; // vertex normals in object space [+1]

varying vec3 v_normal; // vertex normals in eye (aka camera) space [+1]

void main() {
   gl_Position = projection * modelview * a_position; [+1]
   v_normal = (modelview_invtranspose * a_normal).xyz; [+1]
}
```

Fragment shader: (7 pts)

```
uniform vec3 diffuse_color; // diffuse color of surface [+1]
// OK to assume there is a light intensity and pass it in a uniform
uniform vec3 light_dir; // direction towards the light [+1]

varying vec3 v_normal; // vertex normals in eye (aka camera) space [+1]

void main() {
   vec3 normal = normalize(v_normal); [+1]
   float intensity = floor(4 * dot(normal, light_dir)) / 4.0;   // [+2]
   gl_FragColor.rgb = intensity * diffuse_color;   // [+1]
   gl_FragColor.a = 1;  // OK to forget
}
```

For each variable, half credit if no explanation. -1 overall for not discussing what space the attributes and varying are in

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

4. [14 points] **Texture Mapping** This problem is about reflection and normal mapping.

   (a) You have a scene with a camera at (0, 0, 1), looking in the (0, 0, –1) direction, and a disc centered at the origin, with radius $r = 2$ and normal vector (0, 0, 1). The lighting comes from an environment map, and we use reflection mapping on the disc. There is a bright point in the environment map at the direction $(0, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$. What is the coordinate of the point on the surface of the disk where we see this bright point reflected?

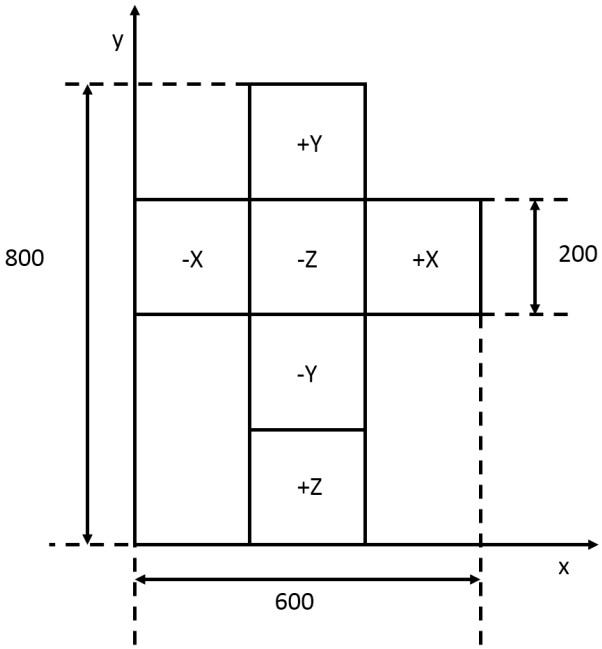
   Solution: First we have to notice that the problem is in the y-z plane (x = 0).
   Since the angle between environment map direction and the surface normal is 45, we have an isosceles triangle (0, 0, 1) [camera], (0, 0, 0), (0, y, 0) -¿ [reflection point]. Thus the reflection point is (0, 1, 0).
   6 pts: up to 4 points for valid work; last 2 points for getting the right answer.
   Just stating the answer is OK.

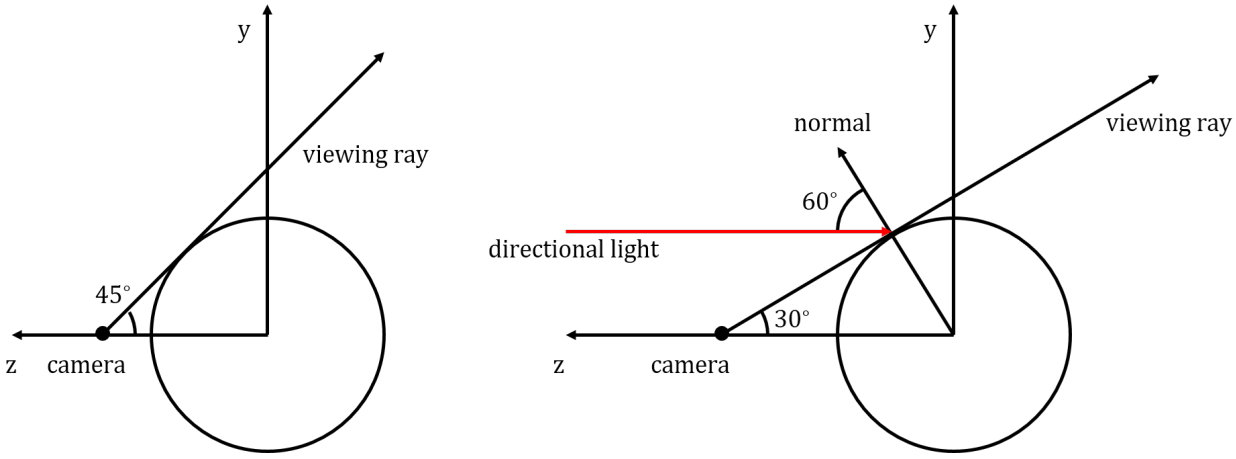
   (b) Now we want to make our disk fancier, so we add normal mapping to it. The normal map is determined in a way that the normal of the disk is the same as if we had a hemisphere with the same radius and center and projected it to the disk keeping the hemisphere's normals. Therefore a point at distance $r \sin\theta$ from the center has a surface normal that makes an angle θ with the disk's geometric normal. In this case we place the camera very far away, giving it the viewing direction $(0, \sin 15, -\cos 15)$, i.e. the camera is looking up by 15° from the original (0, 0, –1) direction. Now what is the coordinate of the point on the surface of the disk where we see this bright point reflected?

   Solution: The viewing direction is 15 degrees up from the (0, 0, -1) direction which is $(0, \frac{\sqrt{3}-1}{2\sqrt{2}}, -\frac{\sqrt{3}+1}{2\sqrt{2}}) = (0, \sin 15, -\cos 15)$. So the reflection angle is (180 - 45 - (90 - 15))/2 = (180 - 45 - 75)/2 = 60 / 2 = 30. The angle between the shading normal and the geometry normal is 90 - 75 = 15. So r * sin 15 = y, and the coordinate of the point on the surface of the disk where we see this bright point reflected is $(0, y, 0) = (0, \frac{\sqrt{3}-1}{\sqrt{2}}, 0) = (0, 0.5176, 0)$.
   8 pts: up to 6 for valid work, last 2 for correct final answer.
   Just stating the answer is OK.

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

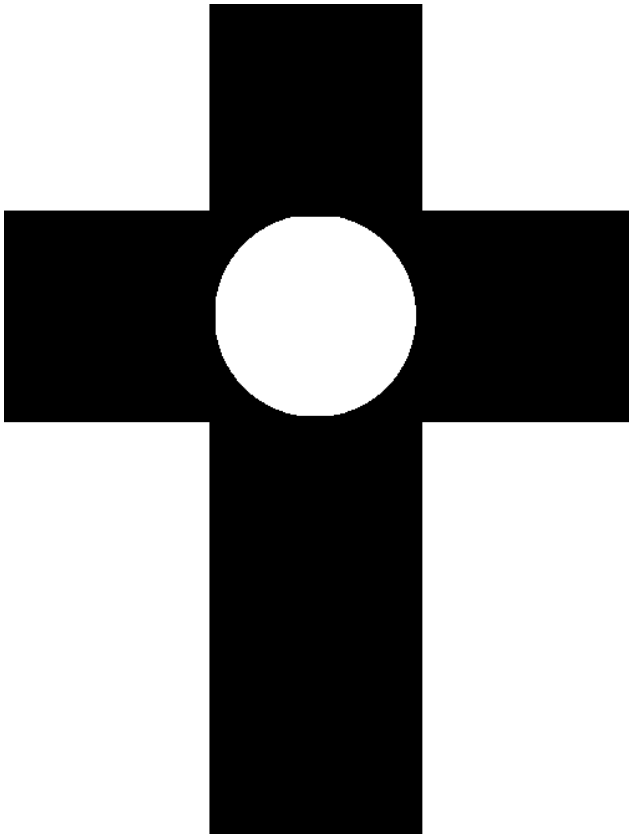
5. [14 points] **Reflection and cube maps**. Imagine that we would like to render a glazed diffuse sphere under environment lighting. The sphere (diffuse reflectance 0.2, index of refraction 1.5) is centered at (0, 0, 0) and its radius is 1. The cube map is a cross-like texture (just like the ones in Ray2 assignment), with width 600 and height 800 (See the figure below). A directional light is from direction (0, 0, 1).





(a) A perspective camera is positioned at (0, 0, $\sqrt{2}$), viewing at (0, 0, 0). Which part of the cube map will be reflected by the sphere? Draw the region on the cross in the next page and mark your sketch with enough positions and dimensions (in texel units) so that it's clear exactly what region is shaded. The left part in the above figure is an orthogonal projection of the scene onto the y-z plane and it shows the viewing ray tangent to the sphere.

Solution: See the shadowed region in the left figure. The center of the circle is (300, 500), radius is 100 (tangent to the square).



8 pts: 4 for understanding basically what is going on; 2 for work indicating some understanding; last 2 for details.

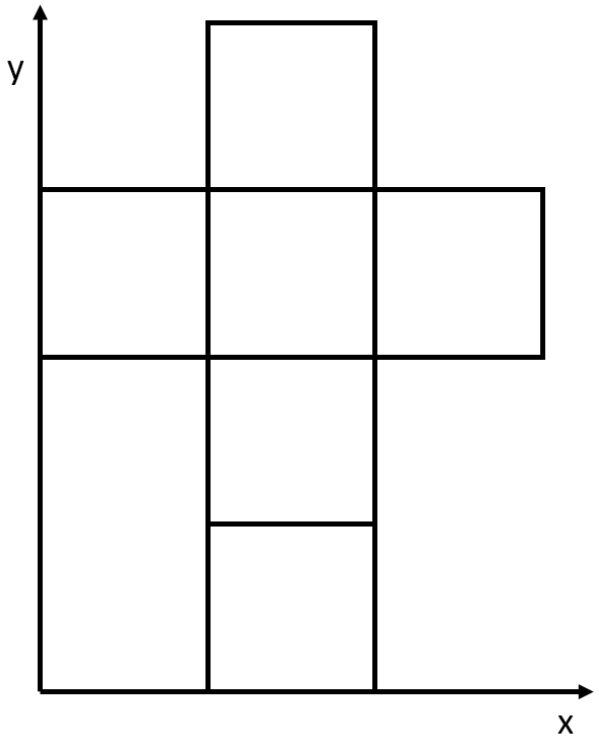
(b) If the camera is positioned at (0, 0, 2) and the cube map has a constant value 1.0, how bright should the directional light be, in order to make the border and center of the sphere look equally bright in the rendered image? You don't need to scale the diffuse term by $(1 - R)$ here; the shading is simply the sum of the Fresnel reflection from the surface and a Lambertian diffuse component. The right part of the figure in the previous page is an orthogonal projection of the scene onto the y-z plane. It shows the viewing ray, directional light and normal of a point on the border. Solution: The brightness of pixel is $L = k_d I_d max(n \cdot l, 0) + RI_s$

On the border, the incident angle of environment light is $\theta = 90°$ and Fresnel term $R = 1$. The incident angle of directional light is $\theta = 60°$. $L_b = 0.2 * I_d * 0.5 + I_s$.

At the center, the incident angle of environment light is $\theta = 0°$ and Fresnel term $R = 0.04$. The incident angle of directional light is $\theta = 0°$. $L_c = 0.2 * I_d + 0.04 * I_s$.

Let $L_b = L_c$ and we get $I_d = 9.6$.

6 pts: 2 for working out the two Fresnel reflections; 2 for working out the diffuse reflections; 2 for final conclusion.



6. [12 points] **Bounding Volume Hierarchies**.

Suppose we have a number of objects in a 2D scene that we want to ray trace, as shown on the left sides of Figures 2 and 3 (the scenes are the same in both figures). We can set up an acceleration structure using axis-aligned bounding boxes to try to avoid costly intersection operations. Suppose that surface A has a glazed shader, and we want to calculate the first intersection of the reflecting rays shown (assuming the rays start some small distance epsilon away from the surface). The right sides of Figures 2 and 3 give two possible configurations for a bounding volume hierarchy containing the surfaces. For each ray and for each tree, please list the objects (including bounding volumes) on which intersection operations are performed (the objects can be listed in any order—no need to take tree traversal order into account). How many total intersection operations are performed in each case? What about in the naïve case, i.e. without any acceleration structure?

Ray 1, Tree 1:

Ray 2, Tree 1:

Ray 1, Tree 2:

Ray 2, Tree 2:

Solution:

Ray 1, Tree 1: AABB1, AABB2, AABB3, AABB4, AABB5, AABB6, AABB7, A, B, E, F, G, H (or any permutation) : 13 total intersection operations

Ray 1, Tree 2: AABB1, AABB2, AABB3, AABB4, AABB5, A, B, E, F, G: 10 total intersections

Ray 2, Tree 1: AABB1, AABB2, AABB3, AABB4, AABB5, AABB6, AABB7, A, B, F, H : 11 total intersection operations

Ray 2, Tree 2: AABB1, AABB2, AABB3, AABB4, AABB5, A, B: 7 total intersection operations

For each: -0.5 per missing object; -0.5 per extra object (assume they counted correctly). Clamped at 0 for each part.

Naive: 8 total intersection operations for both rays. [+1]

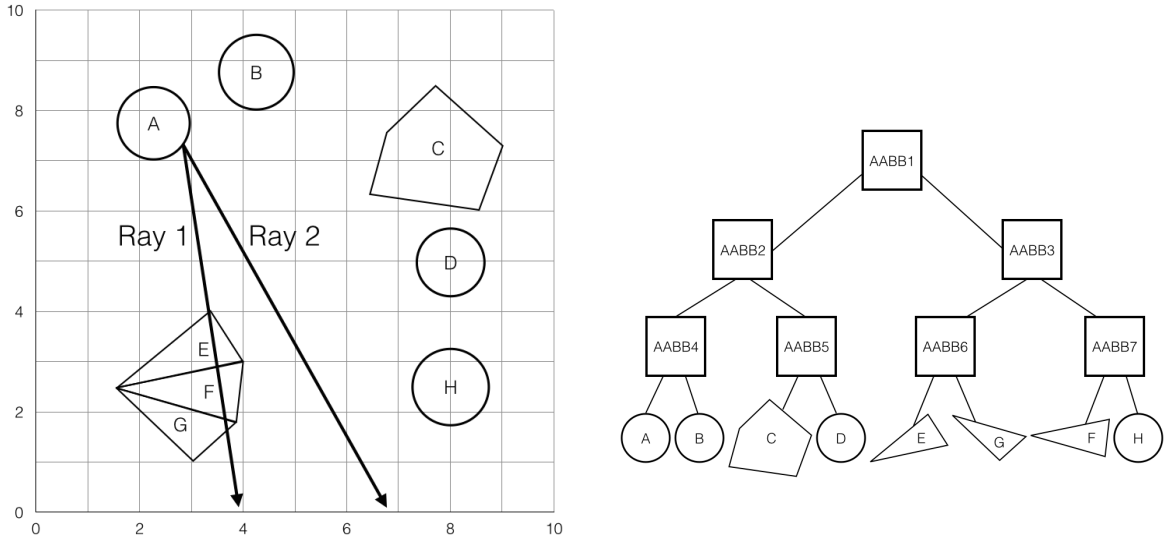


Figure 2: The 2D scene to ray trace, along with BVH tree 1.

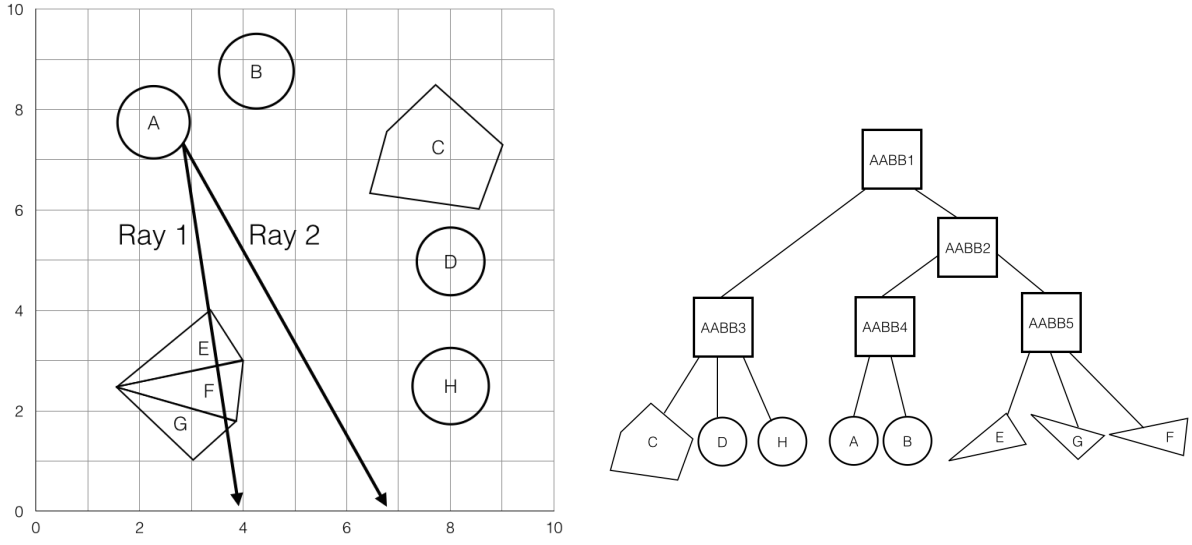
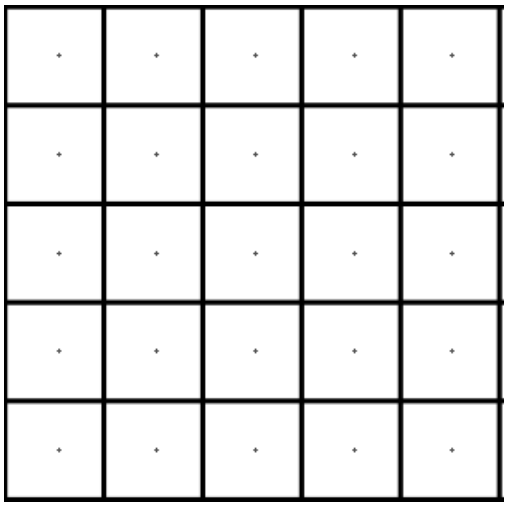
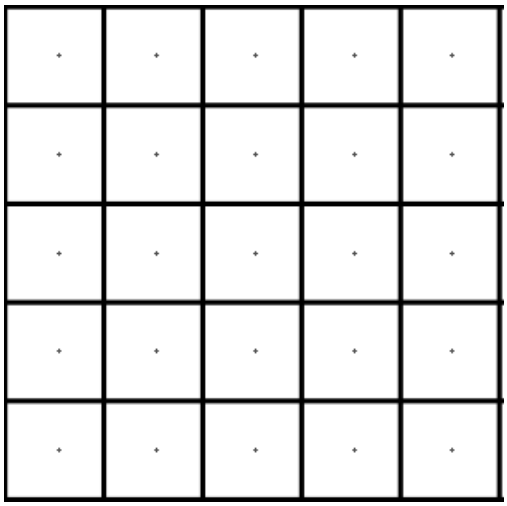


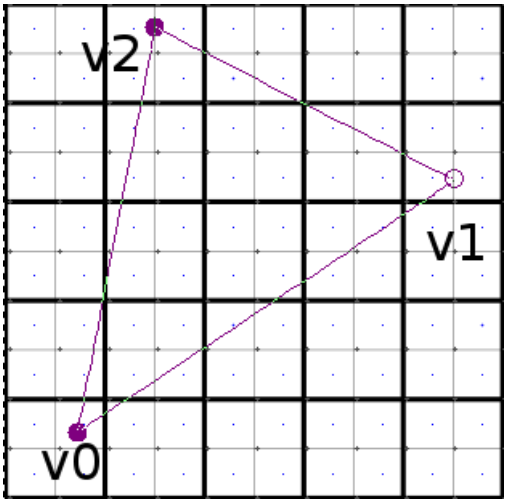
Figure 3: The 2D scene to ray trace, along with BHV Tree 2.

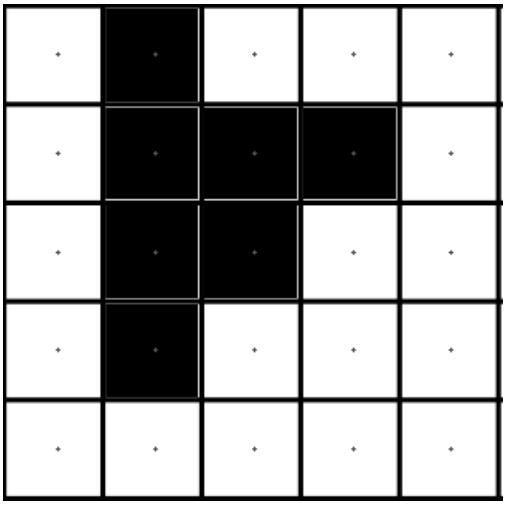
7. [16 points] **Rasterization, antialiasing, and the z-buffer**

(a) Suppose we have the screen-space coordinates for the three vertices that make up a triangle, where $v_0 = (0.2, 0.2, 1), v_1 = (4, 2.75, 7), v_2 = (1, 4.25, 7)$ and the origin is the center of bottom-left pixel in the grid. Draw and label the vertices for this triangle in the left grid below.









Solution:
5 points: -1 per wrongly marked pixel; clamped at 0.

(b) Use the right-hand side grid to shade-in the fragments that will end up being emitted. What rule did you follow to rasterize the triangle?

Solution: Emit a fragment for each pixel whose center falls within the area of the triangle. The problem ensures that all points are clearly on one side or the other of the edges.
2 points.

(c) The boundaries of the triangle above will have a fairly jaggy appearance. We wish to antialias this using regular supersampling with 4 samples per pixel. Use the following grid to compute the fractional intensities for the rasterized pixels at the positions given. You can assume that all of the previously drawn fragments had an intensity of 1.

(0,0): _____

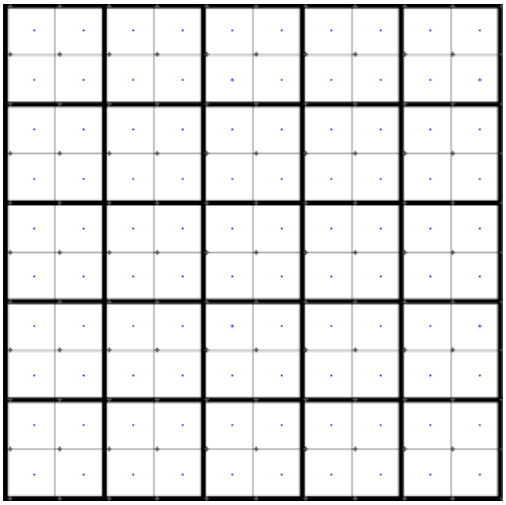
(1,2): _____

(2,2): _____

(2,3): _____

(3,1): _____

(3,3): _____



Solution:

(0,0): 1/4

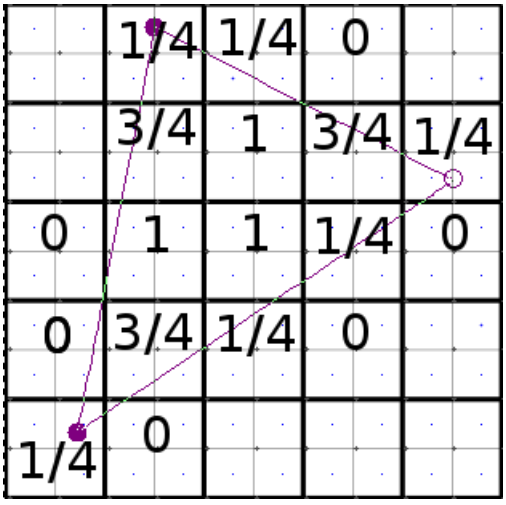
(1,2): 1

(2,2): 1

(2,3): 1

(3,1): 0

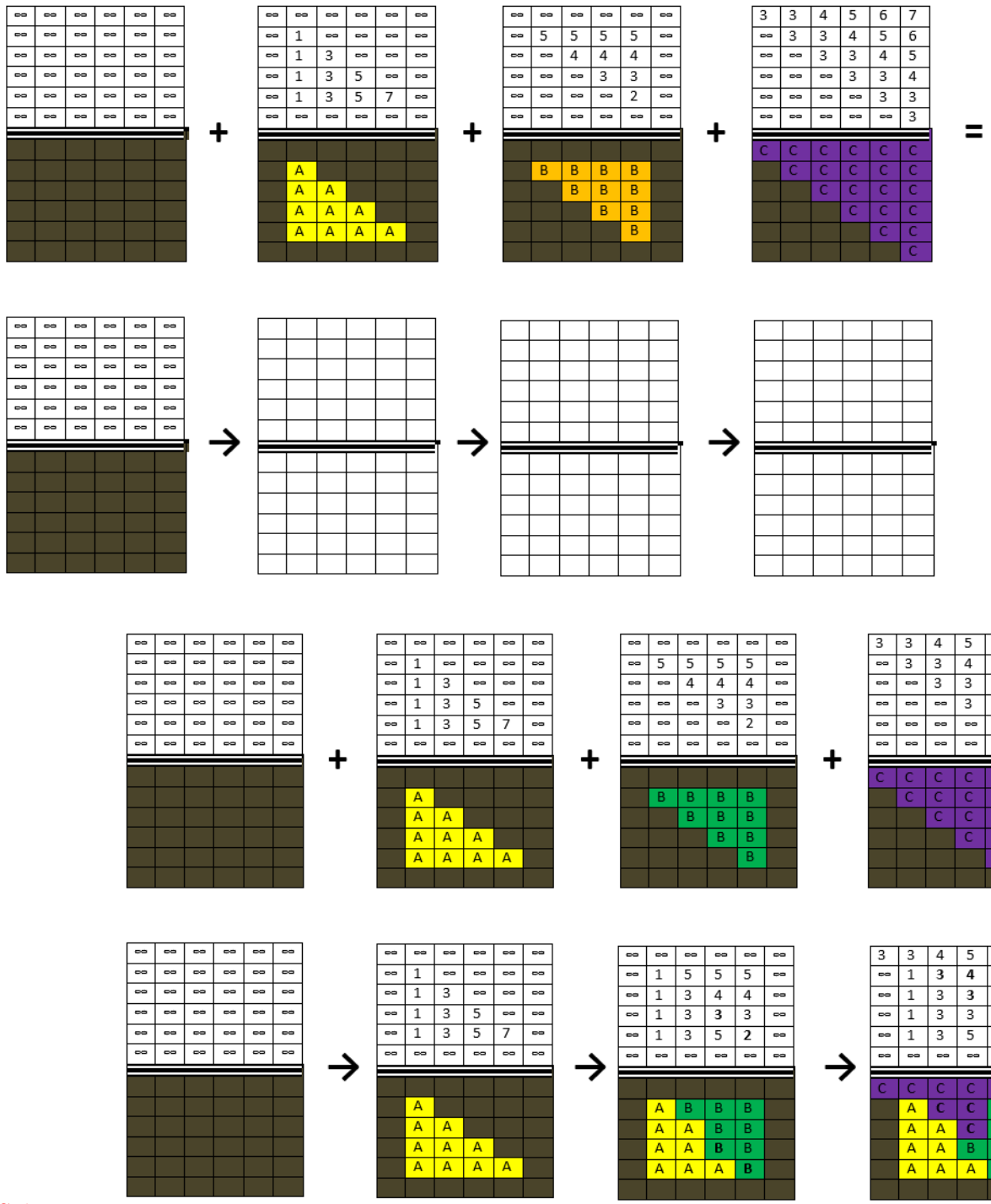
(3,3): 3/4



3 points: 1/2 for each value.

(d) The figure below illustrates the depth values and emitted fragments for three separate polygons in the top row. Complete the bottom row to show how the zbuffer is updated as the rasterized polygons are considered in the given sequence.

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



Solution:

4 pts: 2 for updating depths and 2 for updating colors. How depth ties are broken does not matter.

(e) Briefly describe, in one or two sentences, the difference between a fragment and a pixel?

Solution: A fragment is a value that may contribute to a pixel, and it is the thing processed by a fragment shader.
A pixel is the thing in the framebuffer that is the final result (some other answers OK).
Not all fragments will contribute; many do not because they fail the depth test or their values are overwritten by later fragments.
2 points: +1 for anything sensible, +1 for correct explanation

Question:	1	2	3	4	5	6	7	Total
Points:	12	16	16	14	14	12	16	100
Score:								