Stanford CS248A: Computer Graphics: Rendering, Geometry, and Image Manipulation
Exercise 2

Projecting a 3D Point onto the Screen

Problem 1:

This problem is designed to make sure you understand the transformations needed to take a point in 3D world coordinates to a point on the screen. Let's define "camera space" to be the coordinate system where the camera is at the origin and looking down the -Z axis. A perspective projection matrix for this setup is given as $\mathbf{P}$ below.

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Consider a scene with a camera located at $\mathrm{P_c} = (0, 0, 4)$ and looking down the -Z axis. Also assume that after perspective projection (and [hint!] conversion from homogeneous coordinates back to Cartesian coordinates), the viewport is set so that the bottom left of the screen in normalized post-projection coordinates is (-1,-1) and the top right is (1,1).

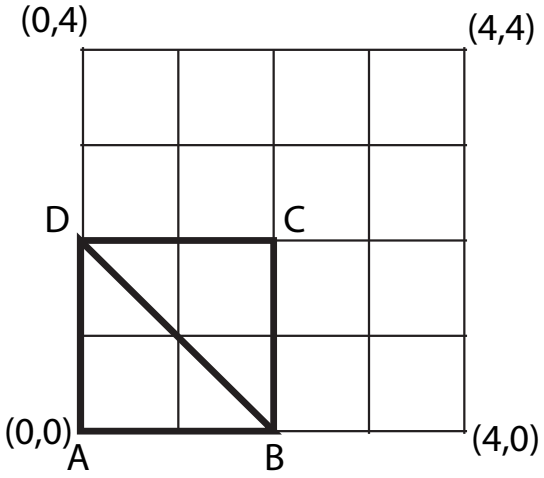
Finally assume that the scene is rendered to an output image that is (400,400) pixels in size. (0,0) in screen pixel space is the bottom-left corner of the screen (corresponding to the normlized post-projection coordinate (-1,1)) and (400,400) is the top-right corner of the screen (corresponding to normalized post-projection coordinate (1,1)). Given this setup, please compute the 2D screen pixel space coordinates (x,y) for a point X in world space located at (0,1,0).

We suggest that you show all your work converting input point X from its world space coordinates (1) to its camera space coordinates, (2) to its normalized view space coordinates, and then finally (3) to its screen space coordinates. Label these intermediates in your solution. Hint: steps 1 and 2 involve transmation of a 3D-H point.

Texture Minification and Magnification

Problem 2:

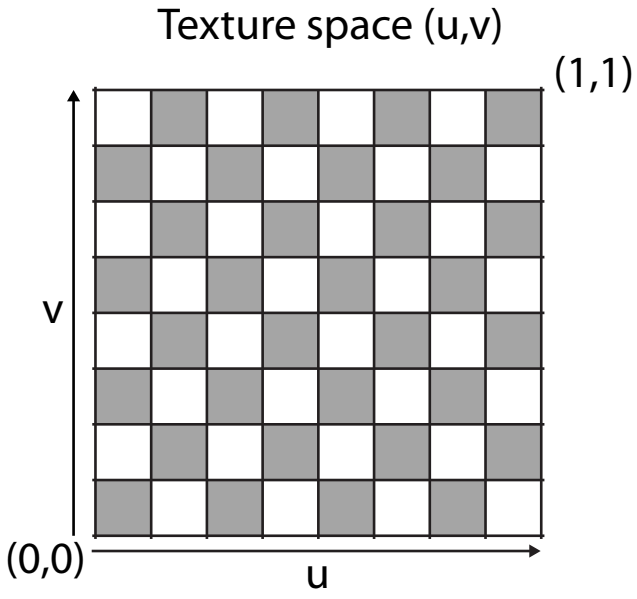
Consider rasterizing two texture-mapped triangles to a tiny 4×4 image. The triangles are positioned on screen as shown below, and vertices have the specified texture coordinates.



Vertex texture coordinates:
A: (0,0)
B: (1,0)
C: (1,1)
D: (0,1)

(Diagram also shows screen coordinates of image corners.)

A. Assume that the texture map used is the 8×8 pixel image below. Please draw a dot on the figure for all texture space locations where the texture is sampled during rendering. Please assume that when rasterizing the triangles texture color is sampled at the texture locations corresponding to pixel centers in screen space. Hint: how many dots should there be?

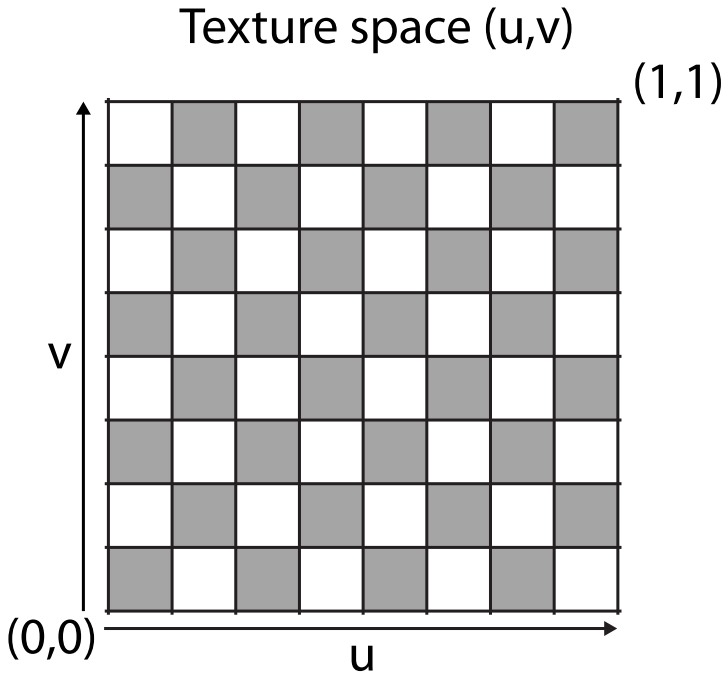


B. Using your answer to the previous problem, describe what the rasterized image looks like if texture filtering is performed using **bilinear filtering.** Please describe the color of key pixels in the output image. You can assume that that the gray in the texture map is the color [.5, .5, .5]. Keep in mind that the texture image is 8×8 pixels, and that although the diagram visualizes the color of texture map "pixels", remember the texture map really represents a set of 8×8 samples of the continuous texture function `texture(u,v)`. Assume sample positions in texture space are located at the texture pixel centers.

C. Now assume the triangles are enlarged on screen so that the the vertices have the following screen space locations. Yes, we increased the size of the triangles by $8\times$. But the output image is still 4×4 pixels.

```
A: (0,0)
B: (16,0)
C: (16,16)
D: (0,16)
```

Please draw a dot on the figure for all texture space locations where the texture is sampled during rendering. Please assume that when rasterizing the triangles texture color is sampled at the texture locations corresponding to pixel centers in screen space. Hint: how many dots should there be?



D. Assume the same setup as in part C (the quadrilateral formed by the two triangles is still $4\times$ wider and taller than the screen), but now assume that output image resolution is increased to 1000×1000. Concretely, in this setup screen space ranges from bottom-left=(0,0) to top-right=(1000,1000), and vertex position coordinates range from (0,0), (4000,0), (4000,4000), and (0,4000). Assuming bilinear filtering is still used during texture sampling, please describe why the rendered image will look "blurry". Keep in mind the texture is an 8×8 image.

Alpha Compositing + a Two Phase Rasterization Algorithm

Problem 3:

A. You are given three surfaces, S1, S2, and S3 which have the following RGBA values. (Note that alpha **is not** premultiplied into RGB in the representations below.):

- S1: $[1, 1, 1, 0.5]$
- S2: $[1, 0.5, 0.5, 0.5]$
- S3: $[1, 1, 1, 1]$

What is the **premultiplied alpha representation** of the color that results from **compositing S1 over S2 over S3**? It is sufficient to give an expression for each component (R,G,B,A) of the final output, or you can reduce that expression to a final value. Remember, we want answers in pre-multiplied alpha form although the inputs S1, S2, S3 are not in premultiplied alpha form.

B. You want to rasterize a scene containing N triangles. **Unfortunately, your rasterizer can only render scenes containing at most $N/2$ triangles.** Imagine that you first rasterize the first $N/2$ triangles from the scene to produce output RGB image $\mathrm{I}_1(x, y)$ and a depth buffer $\mathrm{D}_1(x, y)$. Next, you reset your renderer (you clear the rasterizer's output color and depth buffers) and rasterize the remaining $N/2$ triangles to produce a new RGB image $\mathrm{I}_2(x, y)$ and new depth map $\mathrm{D}_2(x, y)$.

Assuming that all triangles in the scene are opaque (no transparency), give pseudocode for an algorithm that uses $\mathrm{I}_1(x, y)$, $\mathrm{I}_2(x, y)$, $\mathrm{D}_1(x, y)$, and $\mathrm{D}_2(x, y)$ to produce the output image $\mathrm{I}_{\mathrm{final}}(x, y)$, which is the image you would have obtained if you could rasterize all N triangles in a single step using a rasterizer that supports larger scenes. **Hint: how do you tell what would be visible at pixel (x,y) if you "combined" the results at pixel (x,y) from step 1 and step 2?**

More Texture Mapping Practice

OPTIONAL PRACTICE PROBLEM 1:

Consider a 1024×1024 texture map whose value at pixel (x,y) is white if x mod 2 = 0, and black otherwise. This texture is used to texture a single triangle with vertices p0=(0,0), p1=(1,0), and p2=(0,1) and uv texture coordinates uv0=(0,0), uv1=(1,0), and uv2=(0,1)

Consider rendering this triangle to a 512×512 image, where the **background color is 50% gray**. The scene viewport is set up so that scene coordinate (0,0) is in the bottom left of the image, and (1,1) in the top-right corner.

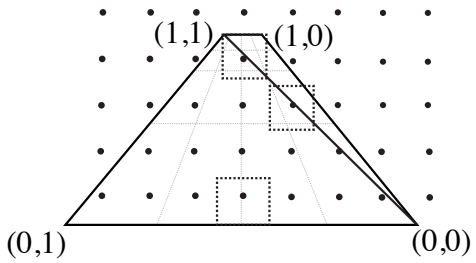
Also assume that screen and texture sample points are at pixel centers (as was the case in assignment 1), and that texture mapping uses **nearest neighbor filtering WITHOUT a MIPMAP.**

A. Describe the image that you will see when you render the scene. Describe both the position of the triangle on screen and what the triangle looks like. (A simple sketch would suffice.)

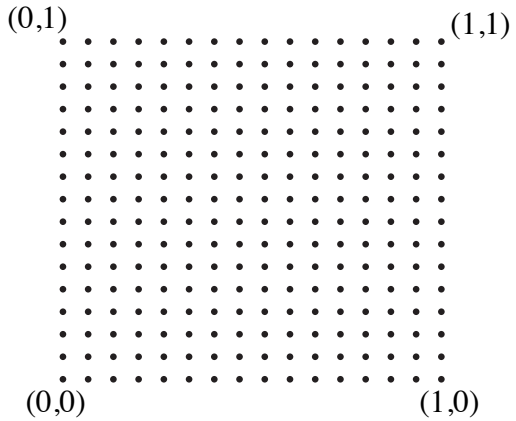
B. Now assume that the rendering mode is changed to **bilinear interpolation** and that the rendered image size is changed to 1024×1024. Describe how you might move the camera (aka pan the viewpoint) to make the triangle *disappear against the 50% gray background!*

C. **This problem is unrelated to parts A and B.**

Consider rendering the two triangles under perspective projection shown at left in the figure below. Per-vertex texture coordinates for the four vertices are given, and the dots indicate the position of screen sample points during rasterization. Now consider the computation to compute the color of the scene at the highlighted screen sample point, which requires a texture lookup into the 16×16 texture shown at right.



Rendered Image



16 x 16 texture (color samples stored at dots)

Three sample points are highlighted in the left side of the above figure, along with dotted boxes showing the extent of the corresponding pixel. In the figure at right, draw the corresponding polygons that correspond to the texture space extent of these screen regions. **BE CAREFUL! Pay attention to the texture coordinate values.**

D. Assume that texture mapping is performed using *bilinear filtering with a mipmap*. Will texture mapping operations to compute the color of the triangles near the top of the rendered image access higher levels of the mipmap (lower resolution textures) or lower levels of the mipmap? Why?

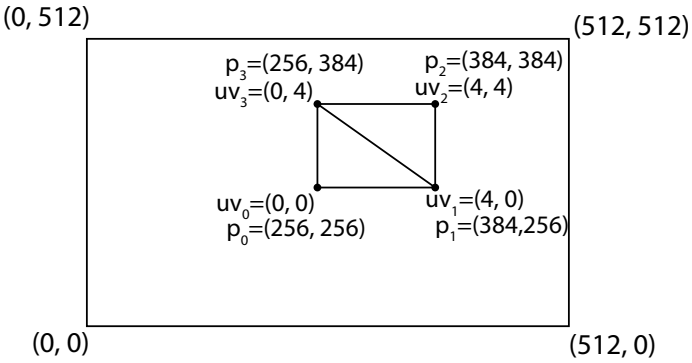
E. Consider the compute cost of texture mapping operations (using mipmapping and bilinear filtering as in part D) for samples at the top of the image or the bottom of the rendered image. Is the cost of texture mapping higher at the top or bottom, or the same? Why?

Even More Texture Mapping Practice

OPTIONAL PRACTICE PROBLEM 2:

Consider rendering a texture-mapped quadrilateral formed by the two triangles shown below. The quadrilateral is rendered onto a 512×512 pixel screen, and the screen coordinates of the quadrilateral's vertices are given below. (The quad is 128×128 on screen.)

512x512 texture



Notice that the texture coordinates (uv) associated with each vertex are not constrained to be between 0.0 and 1.0. In assignment 1 you implemented texture border behavior of "clamp to edge" but another common behavior when texture mapping is to have texture coordinates "wrap". *In other words, texture coordinates are still interpolated over the surface of a triangle as normal, but if the value of the texture coordinate at a sample point is a, then texture mapping would use the fractional part of a, in other words, $a - floor(a)$, for use in the texture lookup*. (Yes, this means that when bilinear filtering is enabled, a bilerp operation might blend between pixels on the right column of the image and the left column (similarly for the top and bottom row)).

A. Describe the image that you will see when the scene is rendered. In particular how many ice cream cones will you see on the quad?

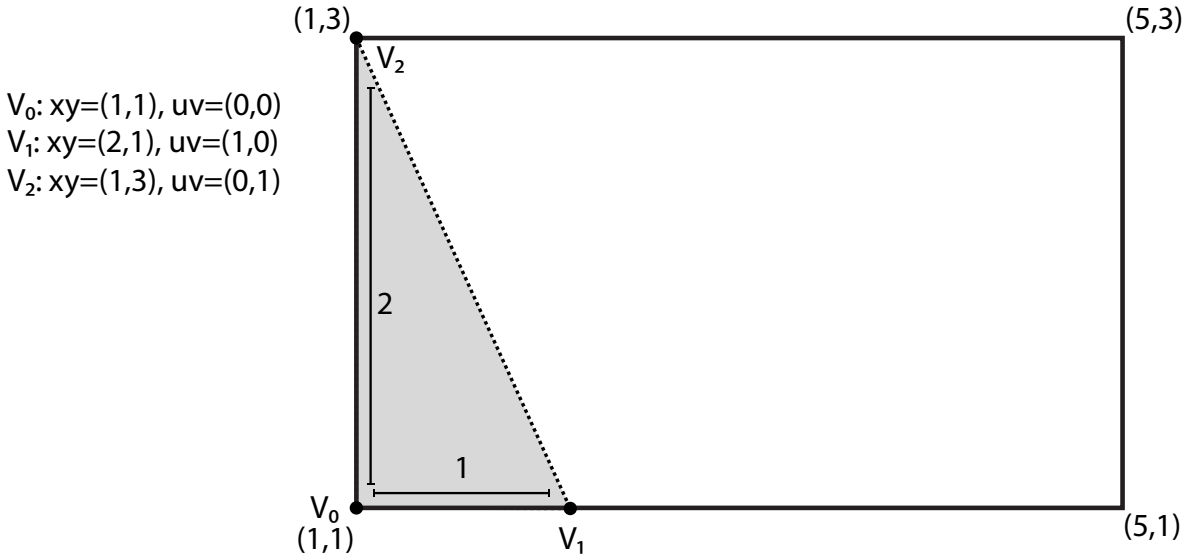
B. Consider the sample located at the center of pixel (256,256) in the image, its screen-space coordinate is (256.5, 256.5). What is the value of the texture coordinate at this location?

C. Given the geometry and texture coordinates in the scene, what is the size of a pixel in screen space when projected into texture space?

Another Texture Question: A Skinny Triangle

OPTIONAL PRACTICE PROBLEM 3:

Consider rendering the the triangle below onto a screen. In the figure below we show the location triangle vertices (in world coordinates), the texture coordinates of triangle vertices, and the world space coordinates of the corners of the image viewport. (e.g., the point (1,1) maps to the bottom left of the region that is visible on screen.)



A. Assume that the output image is rendered at 720p HD resolution (1280×720 pixels). Please give the image-space coordinates of vertex V_1 of the triangle. **In this problem assume that image space is defined as follows: the bottom-left corner of the visible image is at image-space coordinate (0,0) and the top-right corner is at coordinate (1280,720)**. This means that the center of pixel (i,j) in image space coordinates is at $(i + 0.5, j + 0.5)$

Note: throughout this problem you can express your answers as fractions. The math is not meant to reduce nicely to integers.

B. Assuming that point-in-triangle coverage sampling is performed at pixel centers in image space, please give the texture coordinate (uv) of the sample associated with pixel (0,0). (First confirm this sample covers the triangle. *It does!* Then compute the value of the texture coordinates at this screen sample location.)

C. Now assume that the texture map is a very high resolution 4096×4096 image. Please describe the region of texture space that corresponds to the image-space region spanned by the pixel (0,0). Make sure your answer describes the number of texture pixels in width and height. *(Hint: if you do have a calulator handy, it might be useful to take fractional answer to a real number to get a sense of the number of pixels spanned.)*

D. Consider a texture map that contains high-frequency detail, such as lines a few pixels in width. Describe why aliasing may be visible in this example.

E. Imagine that this rendering system **DID NOT** support any form of mip-mapping, but does support **SUPERSAMPLING** of triangle coverage. (Supersampling = sampling triangle coverage and triangle's color many times per pixel.) Will supersampling reduce aliasing in the rendered image? Describe why or why not?

F. Now assume that the rendered **DOES NOT** support supersampling, but does support trilinear texture sampling using a mipmap. Describe how use of trilinear filtering can significantly reduce aliasing in this example. **Advanced question: In your answer describe why filtering using a mipmap will result in overblurring in the vertical direction.**

G. There's one type of aliasing in the resulting image that mip-mapped texture sampling WILL NOT remove in this example (hint: think about aliasing during triangle/sample coverage testing). Even with proper texture pre-filtering, can you describe one aliasing artifact that will be noticeable when sampling coverage once per pixel?