Stanford CS248A: Computer Graphics
Written Assignment 2

Transforms, But in 3D

Problem 1: (Graded for Correctness - 25 pts)

A. (8 pts) You are rendering a 3D scene with the camera located at the point (1,2,3), and oriented so that from the point of view of the camera, the camera is looking in the direction of the -Z axis, and the Y axis is "up". Please give a 4×4 matrix that transforms homogeneous 3D world-space points into a coordinate frame were the camera is at the origin, and still looking down the -Z axis with the Y axis as "up". Please call your matrix **C** in your answer.

B. (8 pts) Assume that you have a rendering system where after perspective projection the bottom left of the screen is coordinate (-1,-1) and the top right of the screen is coordinate (1,1). We'll call this **normalized 2D screen space.** You have the following transform **P** that is meant to implement perspective projection on points in the camera space defined in part A. Notice that one element of the matrix below is marked as X. What should the value of X be? (Hint: consider where a 3D camera-space point (-5, 0, -5) should end up on screen.) Please justify why your choice of value is positive or negative.

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

C. (9 pts) Assume the camera is at the location (1,2,3) just like in part A. Given the matrices $\mathbf{C}$ and $\mathbf{P}$ that you defined in parts A and B, please write down pseudocode for transforming a 3D **world-space** point $\mathbf{p}$ into normalized 2D screen coordinates. (Your answer need only refer to $\mathbf{C}$ and $\mathbf{P}$ symbolically, so there is no dependence on whether you correctly answered those subparts.) Hint: we want pseudocode for computing normalized screen space 2D (x,y) from 4×4 matrix $\mathbf{C}$, 4×4 matrix $\mathbf{P}$, and 3D point $\mathbf{p}$. Don't forget homogeneous divides.

Miscellaneous Short Problems

Problem 2: (Graded for Correctness - 25 pts)

A. (13 pts) Consider a triangle with 2D vertex positions v1=(-1,-1), v2=(3,-1), and v3=(-1,3) drawn onto a screen that is 1000×1000 pixels in size. Normalized screen space is defined as the bottom-left of the screen at (-1,-1) and the top-right of the screen at (1,1). The texture coordinates for the triangle's vertices are uv1=(.1, .8), uv2=(.2,.9), and uv3=(0,1). **Assume the texture map used for the triangle is also** 1000×1000 **in size, and that bilinearly filtered texture sampling with no mip-mapping is used.** Please describe whether rendering will look blurry, or whether it risks aliasing. Please justify your answer, but note that we do not expect calculations in your answer. (Hint: it might be helpful to draw the triangle's footprint in texture space as well as the triangle on screen. Is the triangle entirely on screen?)

B. (12 pts) In assignment 1, when rendering transparent surfaces, you composited a new primitive's color on top of existing sample values using the ″over″ operator. Assume that `C` is a premultiplied alpha representation of the current render target sample being modified, and `S` is the premultiplied alpha representation of the surface that is being drawn. Updating `C` to blend in the new surface is done by `C = S + (1-S.alpha) * C`.

Now imagine a case where supersampling using N samples per pixel is enabled. When drawing a semi-transparent surface you decide to do the following logic per sample.

```
// with propability given by alpha, overwrite (treat as opaque surface)
// alpha=0 = never overwrite, alpha=1 always overwrite

if (random1D() < S.alpha) // assume random1D() is a random float between 0..1
    C = S / S.alpha;  // convert to non-premultipled alpha and overwrite
```

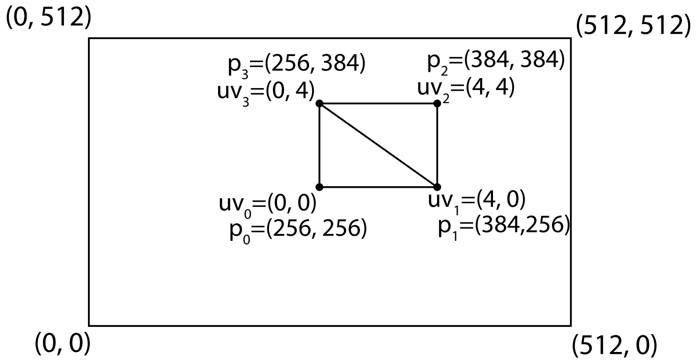
For simplicity, assume that your SVG file only has at most one transparent shape. Please describe why the final image, after performing a resolve to convert sample values into pixel values, the **expected value of each pixel is the same** as the actual pixel value produced by the first version of the code that always updates samples using the over operator. Then describe what the image using the randomized approach will look like compared to the original approach when N is small. (Hint: what is one desirable visual artifact of the second approach?)

Even More Texture Mapping Practice

Problem 3: (Graded on Effort Only - 25 pts)

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?

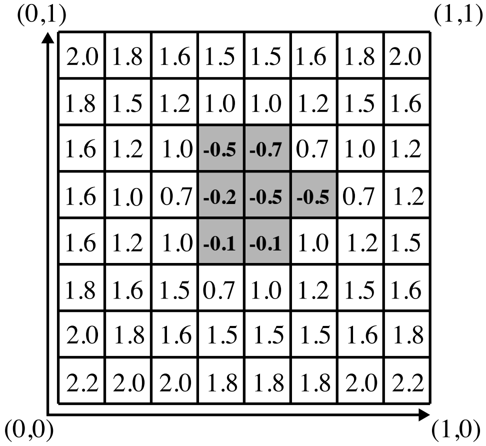
B. Consider the sample located at the center of pixel (256,256) in the image, its screen-space coordinate is (256.5, 256.5). What is the value of the texture coordinate at this location?

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?

Rasterizing a Level Set

Problem 4: (Graded on Effort Only - 25 pts)

In class we talked about a level-set surface representation where each cell in a grid stores the value of a function sampled at the center of each grid cell. The surface is given by the zero-crossing of this function when **it is reconstructed using bilinear interpolation**. For example, consider the surface defined on the following 2D $[0,1]^2$ domain, which is encoded as a 8×8 array of samples.



2.0	1.8	1.6	1.5	1.5	1.6	1.8	2.0
1.8	1.5	1.2	1.0	1.0	1.2	1.5	1.6
1.6	1.2	1.0	**-0.5**	**-0.7**	0.7	1.0	1.2
1.6	1.0	0.7	**-0.2**	**-0.5**	**-0.5**	0.7	1.2
1.6	1.2	1.0	**-0.1**	**-0.1**	1.0	1.2	1.5
1.8	1.6	1.5	0.7	1.0	1.2	1.5	1.6
2.0	1.8	1.6	1.5	1.5	1.5	1.6	1.8
2.2	2.0	2.0	1.8	1.8	1.8	2.0	2.2

Now imagine that you want to extend your SVG renderer implementation to also render level set primitives. Assume that all level set primitives are associated with a transform $\mathbf{T}$ that describes how to transform points in the domain of the level set to points in 2D canvas space, which is defined with (0,0) in the BOTTOM-LEFT of the screen and (W,H) in the TOP-RIGHT of the screen. You may assume that you also have the transform $\mathbf{T}^{-1}$.

A. Please describe an algorithm for rasterizing the level set. Color the screen black if it is INSIDE the level set (the function's value is less than zero), and white otherwise. You may assume that `getSamplePos(px,py)` returns the screen (canvas) sample point for pixel (x,y). You may also assume that you have access to a function `bilerp(s, t, i, j)`, which evaluates the value of the level set function via bilinear interpolation of the samples at level set cells (i,j), (i+1,j), (i,j+1), (i+1,j+1) according to coefficients s and t. You need not worry about algorithm efficiency, or edge-case behavior near the edges of the level-set.

   Hint: How do you transform the screen point (x,y) into the coordinate space of the level set? Then how do you convert this point to a index (i,j) in the level set matrix?

B. Consider the case where the output image size is 1024×1024 and the corners of the level set object map to screen coordinates (512,512), (1024, 512), (1024,1024), and (512,1024). Given your algorithm in part A, will the object described by the level set look blurry on screen, or will it have a sharp edge at the boundary of the object? Why or why not? (Hint: for every sample point, is there a definitive answer for whether you are inside or outside the shape given by the level set?)

C. Imagine you wanted to extend all shapes in your 2D SVG renderer to carry an additional value “depth” which is the distance of the shape from the “camera” (lower depths are closer objects). In this case all shapes are contained within a single Z plane. You decide to implement occlusion calculations with a depth-buffer as described in class. Your friend looks at you as says “hey, while that’s a correct implementation, that’s not necessary to correctly render pictures with correct occlusion in this case.” Given your renderer implementation in assignment 1 (which draws all objects in the order it is given), describe a method to get correct occlusion without using a depth-buffer.

D. Imagine that we extended the level set representation to also maintain a per-cell DEPTH value, so that the depth of the surface at a point in the domain was also determined by bilinear interpolation. Given your algorithm in part A, could a depth buffer be used to correctly handle occlusion in a scene with multiple level sets, as well as multiple triangles with different depths? Why or why not?

Projecting a 3D Point onto the Screen

PRACTICE 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).

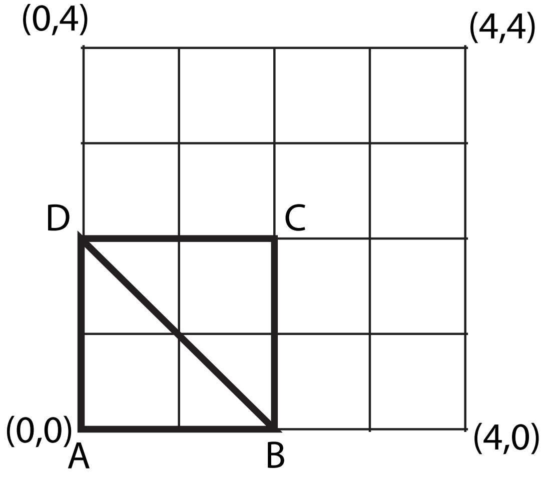
Finally assume that the scene is rendered to an output image that is (400,400) pixels in size. (0,0) in screen pixel space is the bottom-left corner of the screen (corresponding to the normlized post-projection coordinate (-1,1)) and (400,400) is the top-right corner of the screen (corresponding to normalized 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

PRACTICE 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.

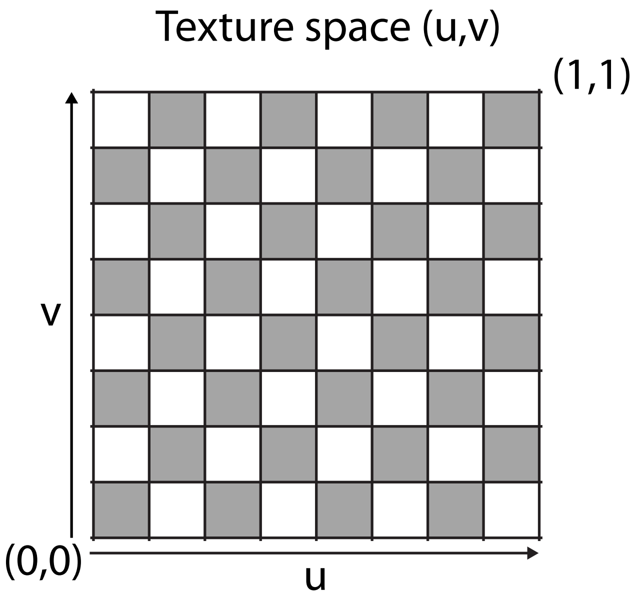


Vertex texture coordinates:

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?

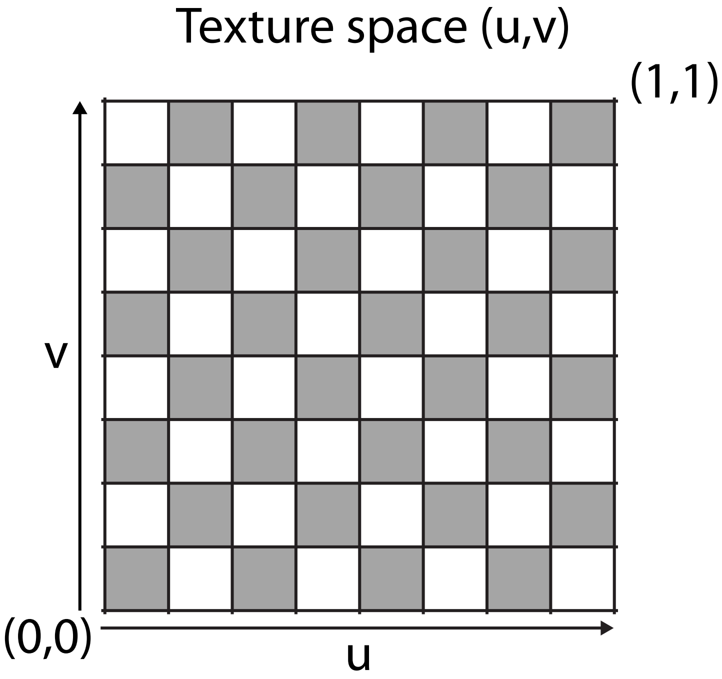


B. Using your answer to the previous problem, describe what the rasterized image looks like if texture filtering is performed using **bilinear filtering.** Please describe the color of key pixels in the output image. You can assume that that the gray in the texture map is the color [.5, .5, .5]. Keep 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.

PRACTICE 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 $\mathrm{I}_{\text{final}}(x, y)$, 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

PRACTICE PROBLEM 4:

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.

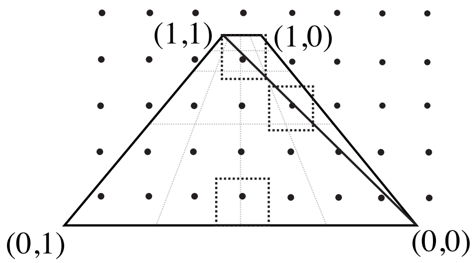
Also assume that screen and texture sample points are at pixel centers (as was the case in assignment 1), and that texture mapping uses **nearest neighbor filtering WITHOUT a MIPMAP.**

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.)

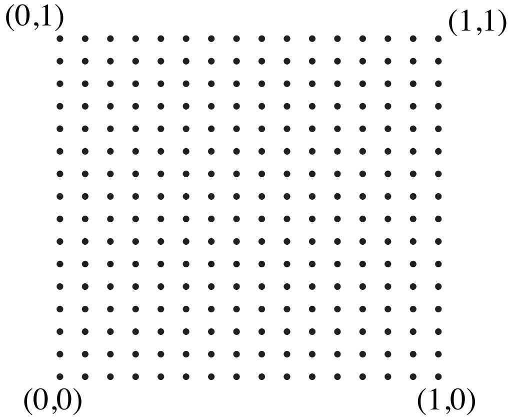
B. Now assume that the rendering mode is changed to **bilinear interpolation** and that the rendered image size is changed to 1024×1024. Describe how you might move the camera (aka pan the viewpoint) to make the triangle *disappear against the 50% gray background!*

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.**

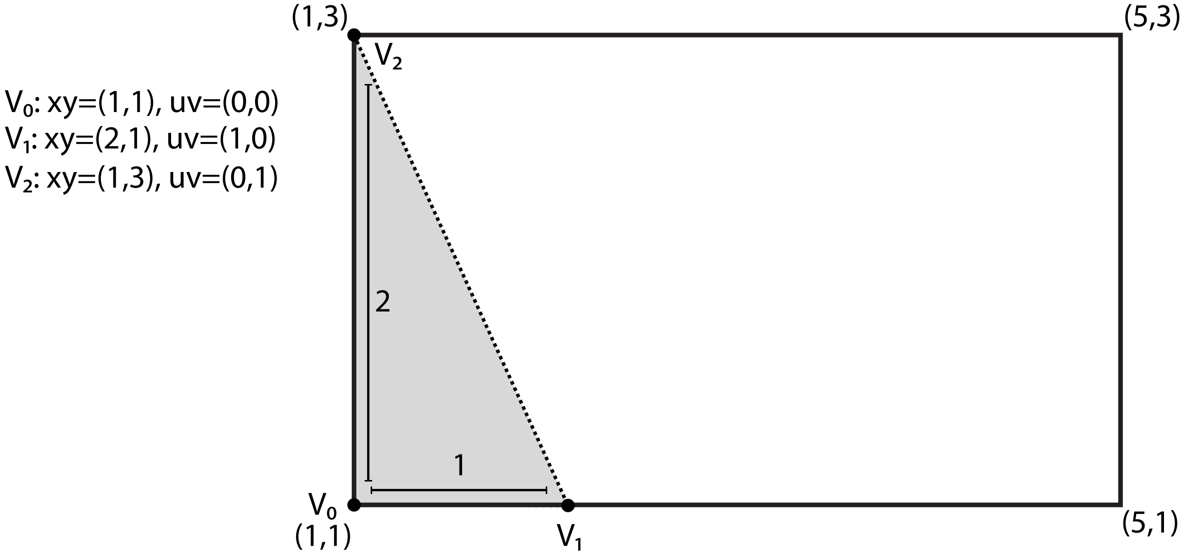
D. Assume that texture mapping is performed using *bilinear filtering with a mipmap*. Will texture mapping operations to compute the color of the triangles near the top of the rendered image access higher levels of the mipmap (lower resolution textures) or lower levels of the mipmap? Why?

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?

Another Texture Question: A Skinny Triangle

PRACTICE PROBLEM 5:

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?

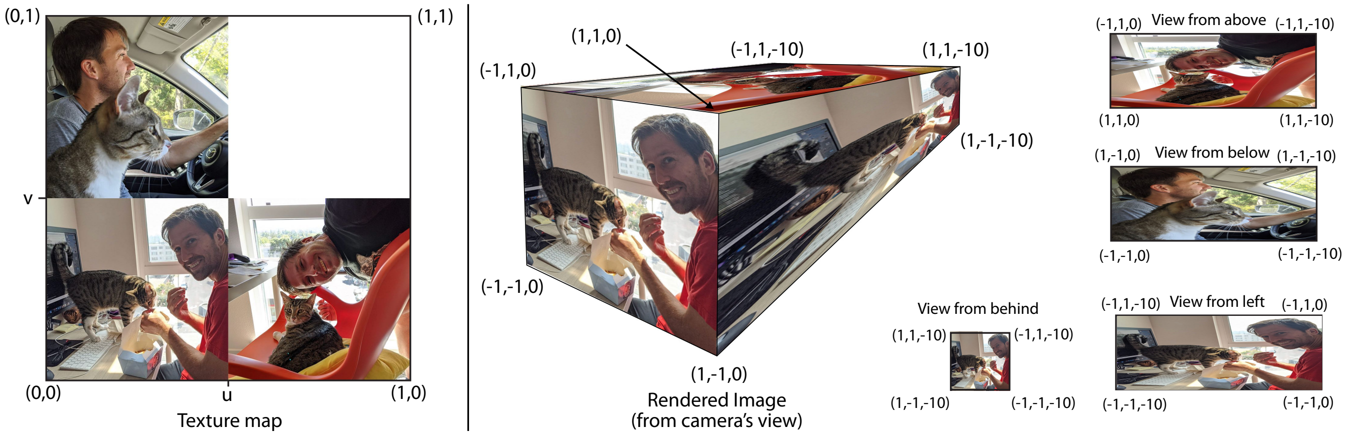
F. Now assume that the rendered **DOES NOT** support supersampling, but does support trilinear texture sampling using a mipmap. Describe how use of trilinear filtering can significantly reduce aliasing in this example. **Advanced question: In your answer describe why filtering using a mipmap will result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

A Textured Cube (Putting Texturing and Geometry Together)

PRACTICE PROBLEM 6:

In this problem you are rendering the textured box shown in the center of the figure below. The box is 2 units in width and height, and 10 units in depth. 3D world space vertex positions are shown on the figure. On the left of the figure is the texture image used. The front, right, back, and right sides of the box all display the bottom-left region of the texture. The top of the box is the bottom-right quadrant (see top view). The bottom of the box maps to the top-left quadrant of the texture (see bottom view).



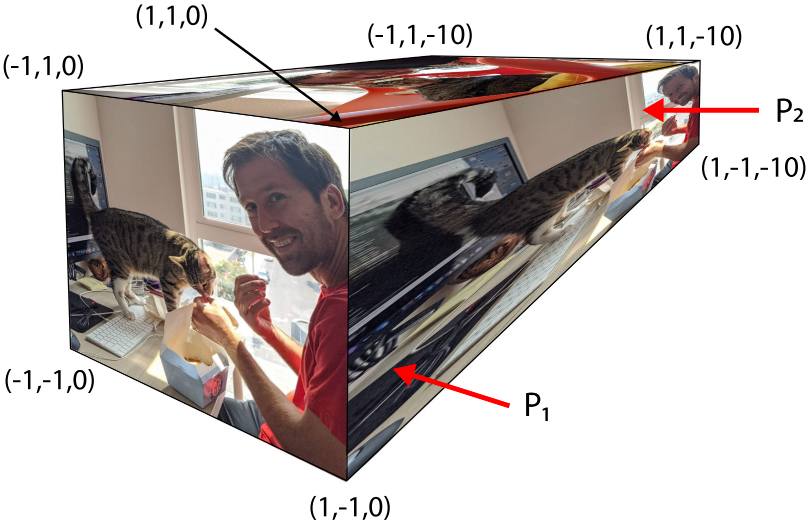
A. In class we talked about indexed mesh representations, where the vertices of each triangle are specified by an index into an array of 3D vertex positions. Below is a partial definition of an indexed mesh. **Please complete the specification of the mesh by filling in the missing indices for the triangles on the box's back and bottom faces. (three triangles are missing.)** Be careful to ensure your triangle "windings" are correct! (They should be consistent with the windings of the other faces and yield a normal that points away from the inside of the box.)

```
Vec3d positions[8] =
    { Vec3D(-1,-1,0),   Vec3D(1,-1,0),   Vec3D(1,1,0),   Vec3D(-1,1,0),
      Vec3D(-1,-1,-10), Vec3D(1,-1,-10), Vec3D(1,1,-10), Vec3D(-1,1,-10) };

int NUM_TRIANGLES = 12;
int posIndices[3 * NUM_TRIANGLES] =
       { 0,1,2,  0,2,3,       // triangles 0 and 1: front face
         1,5,6,  1,6,2,       // triangles 2 and 3: right face


                              // triangles 4 and 5: back face



         0,3,7,  0,7,4,       // triangles 6 and 7: left face
         3,2,6,  3,6,7,       // triangles 8 and 9: top face


         5,1,0,               // triangles 10 and 11: bottom face


       };
}
```

B. The same indexed representation can also apply to per-vertex texture coordinates as well. **Please complete the specification of the mesh texture coordinates by filling in the eight missing texture coordinate values.** Then provide texture coordinate indices corresponding to the vertices of triangle 0 (front face), triangle 2 (right face), triangle 8 (top face), and triangle 10 (bottom face). The result of rendering the box using these texture coordinates should be the image shown in the figure. *Hint: unlike with positions, the same vertex on the box may be different texture coordinates in different triangles!*

```
Vec2D uvCoords[8] =
    { Vec2D(    ,     ), Vec2D(    ,     ), Vec2D(    ,     ), Vec2D(    ,     ),
      Vec2D(    ,     ), Vec2D(    ,     ), Vec2D(    ,     ), Vec2D(    ,     ) };

int uvIndices[3 * NUM_TRIANGLES] =
       {
          // you don’t need to fill out indices for all 12 triangles, but please
          // give the texture coordinate indices for triangles 0, 2, 8, and 10
          // (for the grader label them clearly, this doesn’t have to be valid C code)










       };
```

C. Imagine that you render the image from the camera viewpoint shown below (it's the same figure copied from the figure on the previous page). Consider shading sample points located at P_1 and P_2 shown in the figure. You implement texture mapping using a mip-map. Which point will require sampling from a **HIGHER** mipmap level? **Please describe why.** (Recall that level 0 is the "bottom" of the mipmap, which corresponds to the full resolution texture.)



D. Consider the appearance of the rendered box when sampling texture color from the mipmap using TRILINEAR FILTERING vs. BILINEAR FILTERING. **In your description, describe what undesirable artifacts we might see on the right face of the box if only bilinear filtering is used.** Remember, in both the bilerp and trilerp cases the shader computes a mipmap level and uses the texture mipmap for sampling.

PRACTICE PROBLEM 7:

In class we talked about the limitations of rendering transparent triangles using rasterization. First, to get correct output, the triangles need to be drawn in front-to-back (or back-to-front) order. Second, if two triangles interpenetrate, it's actually impossible to order drawing so that the ordering of the triangles is the same for all sample points.

Now consider a modified rendering algorithm where instead of there being a single RGBA and depth value stored at each sample point, there is an array of up to 16 values. The frame buffer also stores the number of fragments stored in the frame buffer at each sample point, as shown below.

```
struct Sample {
  float r,g,b,a,z;
};

Sample frame_buffer[WIDTH][HEIGHT][16];  // all samples initialized to (0,0,0,0,INFINITY)
int    num_values[WIDTH][HEIGHT];        // initialized to 0
```

Now imagine you have the following two functions:

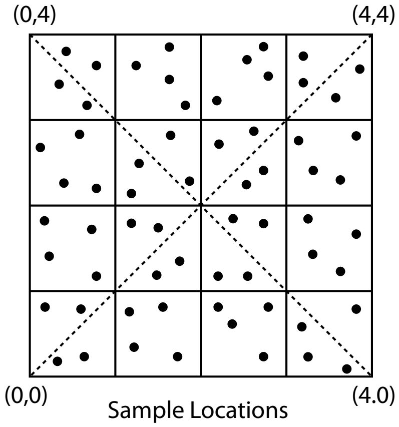
```
void process_fragment(Sample new_frag, int x, int y)
void done_rendering(Sample result[WIDTH][HEIGHT])
```

Recall that a "fragment" is the name given to a sample of a triangle. `process_fragment` is called for each fragment generated by each rasterized triangle. It can modify `frame_buffer` and `num_values` as needed. `done_rendering()` is called after all triangles in the scene have been processed. When `done_rendering` returns, the final image pixel values should be written to the buffer `result`. **Assume that the scene has at most 16 triangles**, all triangles are semi-transparent, and that you can make no assumptions about the depth order of the triangles when rendering. In rough pseudocode, describe an implementation of `process_fragment` and `done_rendering` that results in a correct alpha composited image. You may assume that you have handy helper functions that sort an array, and composite two samples on top of each other and return the result (`Sample OVER(Sample s1, Sample s2)`).

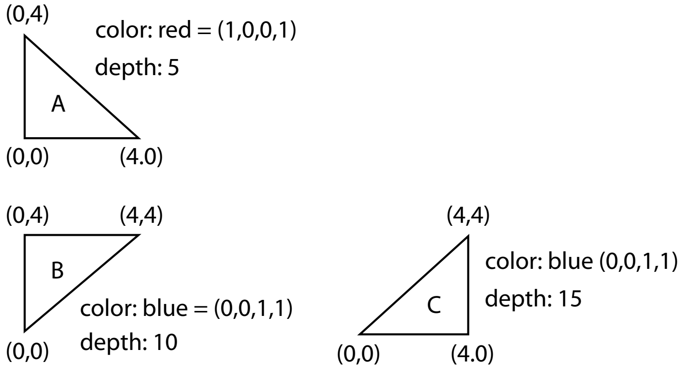
Rasterizing Triangles

PRACTICE PROBLEM 8:

Consider rasterizing the three triangles (A, B, C) given below (at right) to a 4×4 pixel image with 4× supersampling shown at left.. The coordinates of image space are given on the figure (We're using the convention that (0,0) is in the *bottom left*.) Note that unlike assignment 1, the four sample positions per pixel are now placed at **random locations** in each pixel.

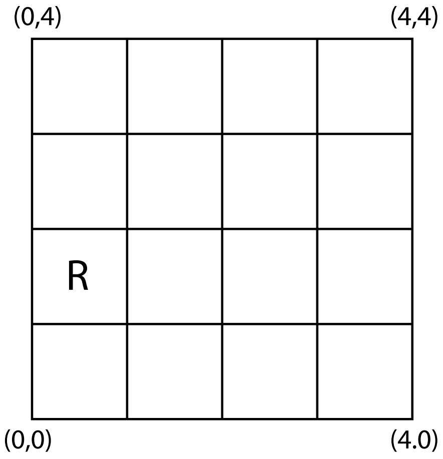


Sample Locations
(also shows pixel grid)



Triangles to Render

A. On the grid below, draw the final rendered output assuming that coverage and depth testing are performed at the provided sample locations, and that the supersample buffer is resampled to a final image by means of **convolving the supersample buffer with a 1-pixel wide box filter**. For simplicity, define the values of several color RGBA variables and just write the variable name in each pixel. (e.g., let R = (1,0,0,1), and one red pixel has been marked for you in the figure.)



B. How would the results in part A change if the resampling filter in part A was replaced with a 3-pixel wide box filter? You only need to answer in words – you do not need to illustrate a result. (You many ignore boundary conditions as well.)

C. Now assume triangle A's color is changed to be 75% opaque red. Recall that in a **non-multiplied alpha representation** this is $C = (1.0, 0, 0, 0.75)$.

Now, assume the renderer is changed to work in the following way... Instead of updating ALL SAMPLES COVERED BY A TRIANGLE that pass the coverage and depth tests, and doing so using the alpha blending equation $C_{\text{new}} = \alpha C_{\text{tri}} + (1 - \alpha) C_{\text{old}}$, the renderer discards a fraction of the triangle's covered samples according to α. Specifically, for a triangle with opacity 75%, the renderer **randomly discards** 1-.75=25% of the samples covered by the triangle, and for all other covered samples, treats the triangle as if it is fully opaque (e.g., has color (1.0, 0, 0, 1.0)).

Assuming the supersample buffer is resolved to a single sample per pixel using a 1-pixel box filter (as was done in Part A), **describe why the new rendering scheme results in the same answer as if alpha blending was used on all covered samples.**

Hint: To keep things simple, your answer need only consider the case where the transparent triangle covers a entire pixel. A clear answer will describe why proposed algorithm gives the same result as the equation above, showing that the math works out the same.

**** For extra credit, give (1) a clear explanation why, in expectation, this approach can rendering transparent surfaces in any order using regular depth testing and no alpha blending. (2) consider why it might not work so well with only a small number of samples per pixel.*