CS535 Computer Graphics
Homework Assignment 8 Solution Set (40 points)

Due date: 12/09/2024

1. Can ray tracing reproduce texture of a surface? Justify your answer. (10 points)

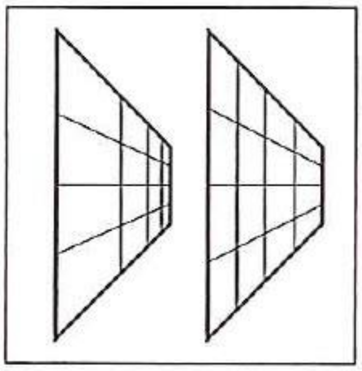
   Sol.
   Yes, ray tracing can reproduce the texture of a surface if the texture of the surface is created by a given texture map. To do this, we need to incorporate UV mapping into the ray tracing process so that a texel (or, a set of texels) in the texture image can be identified and used in the rendering process of the intersected point of the surface, instead of the standard approach.

   Specifically, first, we generate a ray and find the first intersection point on the object as we did in a normal ray tracing process. Then we derive the u and v coordinates of the intersection point on the surface. Instead of setting the color of the intersected point using the standard approach, we use the UV texture coordinates to find the corresponding entry (texel) in the texture image and set the initial color of the intersected point to be that entry of the texture image. After that, we can calculate diffuse light, specular light and refraction as we did in normal ray tracing process. With such modification to a normal ray tracing process, proper texture information can be applied to each intersection point.

   If the projection type is perspective, we also need to apply the perspective correction on the UV mapping.

2. Why perspective correction is necessary when doing texture mapping? How should it be done? Your answer should address two issues here: (1) Theoretically, how can it be done? (2) Practically, how should it be done efficiently? The term "efficiently" means the process is so efficient that it can be integrated with the triangle rasterization process without changing its performance much at all. (20 points)

   Sol.
   Perspective correction is necessary when doing texture mapping because when we do scan conversion, the UV coordinates are calculated by linearly interpolating points in screen space, so the linearly interpolated UV texture coordinates do not give the perspective effect. Consider the following graph:



We get the image on the right if we do interpolation in screen space, which is not correct. The correct one is shown on the left which shows the perspective effect.

To do texture mapping with perspective correction, after we have derived the incorrect UV texture coordinates, u and v, we need to divide them by the depth value Z, which is also interpolated. So we can use the new u and v coordinates to find the proper texel value from the texture image which will correctly represent the perspective effect when mapped to the object.

At each pixel, instead of interpolating the texture coordinates directly, the coordinates are divided by their depth Z (relative to the viewer), and the reciprocal of the depth value 1/Z is also interpolated. Then, we take the new U and V values, index into the texture map and find the appropriate entry for the screen pixel. A pseudo code for this process is shown below.

```
Pseudo-code:
  su = Screen-U = U/Z
  sv = Screen-V = V/Z
  sz = Screen-Z = 1/Z
  for x = startx to endx
        u = su/sz
        v = sv/sz
        PutPixel(x, y, texture[v][u])
        su+ = Δsu
        sv+ = Δsv
        sz+ = Δsz
end
```

Note that u=U and v=V only for the start pixel, not for the remaining pixels. Why?

3. Answer the question "Are correct_s and correct_t integers?" on slide 104 of the notes (Texture mapping II). Justify your answer. (10 points)

Sol.

'correct_s' and 'correct_t' are not necessarily to be integers. In the following, I will show why this is so for 'correct_s' only. The situation for 'correct_t' can be explained similarly.

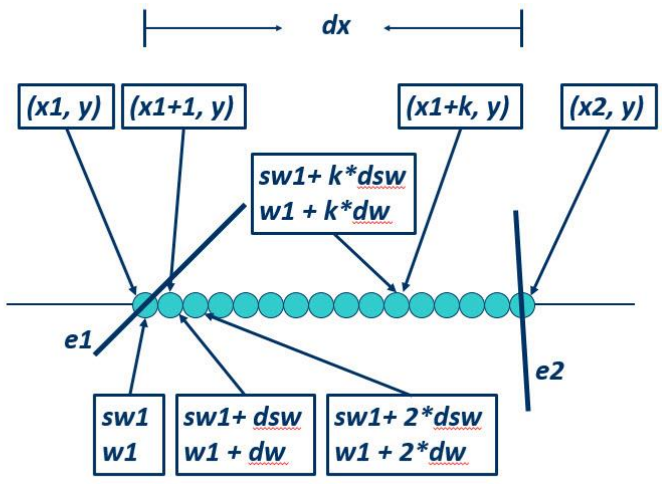
According to the code on slide 104, correct_s is defined as:

correct_s = sw * denom = sw / w (1)

sw and w are defined as follows. Initially,

sw = sw1 // sw value of the start pixel (x1, y) of the current span

w = w1 // w value of the start pixel (x1, y) of the current span



For the following pixels, we have

sw((x1+1, y)) = sw1 + dsw = sw1 + (1/dx)(sw2 – sw1)

w((x1+1, y)) = w1 + dw = w1 + (1/dx)(w2 – w1)

⋮

sw((x1+k, y)) = sw1 +k*dsw = sw1 + (k/dx)(sw2 - sw1) (2)

w((x1+k, y)) = w1 + k*dw = w1 + (k/dx)(w2 – w1) (3)

Note that sw2 = s2*w2 and sw1= s1*w1, hence (2) can be expressed as

sw((x1+k, y)) = sw1 +k*dsw = s1*w1 + (k/dx)(s2*w2 – s1*w1) (4)

Then from (1), (4) and (3), the value of correct_s for pixel (x1+k, y) has the following form:

$$\text{correct\_s}\ ((x1+k,\ y)) = \frac{s1*w1+\left(\frac{k}{dx}\right)*(s2*w2-s1*w1)}{w1+\left(\frac{k}{dx}\right)*(w2-w1)} \qquad (5)$$

If we replace (k/dx) with t (note that k is a variable), then (5) is nothing but the equation at the bottom of slide 101 (remember that we use s to represent u here). The numerator is the linear interpolation of $s1 * w1$ and $s2 * w2$ and the denominator is the linear interpolation of w1 and w2. The denominator is not necessarily a factor of the numerator. Hence, correct_s is not necessarily an integer.

Note that if you interpret (sw/w) as (s∗w/w), the value of correct_s would be just s, a contradiction because this is not an interpolation in 3-space, but in the image space.