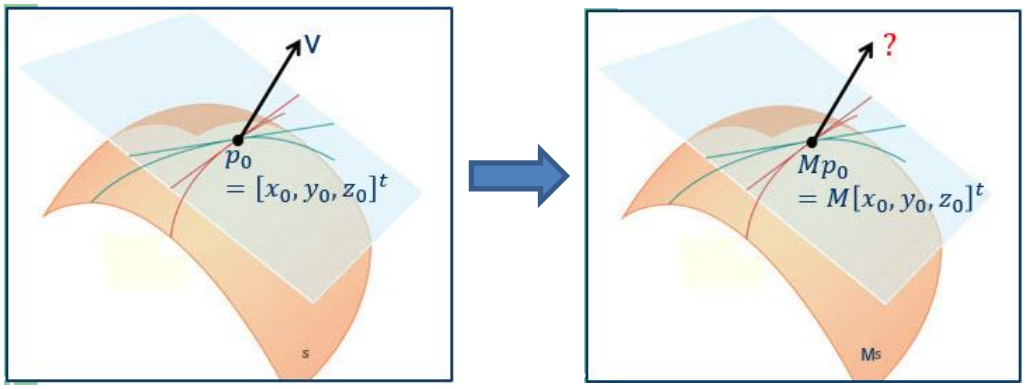
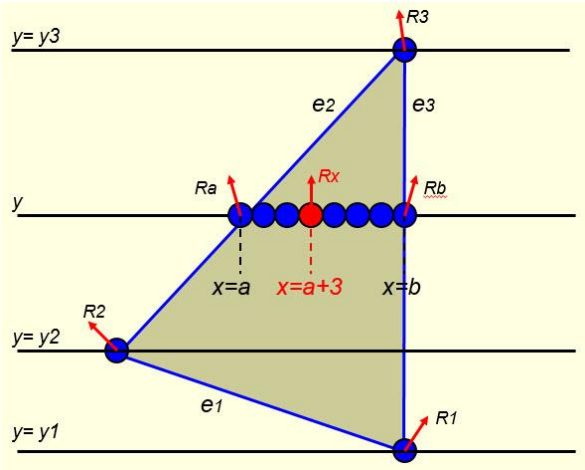
CS 535 Computer Graphics
Homework Assignment 5 Solution Set (40 points)
Due: 11/01/2024

1. Given a point p_0 of a surface *S* with normal vector *V* (see the left figure below). If the surface *S* is transformed by an MV matrix *M* to a new location MS, then the normal vector of p_0 at its now location (see the right figure below) would be $(M^{-1})^t V$ where V is represented as a column vector. Why? (5 points)



According to the Appendix of the note " Lighting and Shadows I", the normal vector at p_0 after the transformation is VM^{-1} where V is the homogeneous normal, a row vector. Therefore VM^{-1} is also a row vector. If V is represented as a column vector, then the normal vector would be $V^t M^{-1}$ and when written as a column vector we have $(V^t M^{-1})^t = (M^{-1})^t V$.

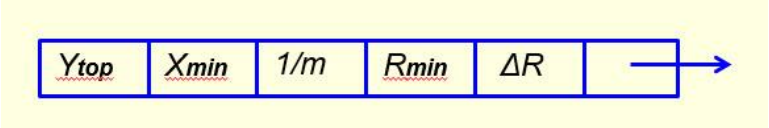
2. Given the normal of a surface at a given point *N* and an incident ray *L*, we need to compute the specular reflection ray *R* at that point to compute its shade. Develop an incremental method to compute that vector for points of a triangle, assuming *R1*, *R2* and *R3* at the three vertices of the triangle are already given. (5 points)



Sol:

The normal of a triangle is a constant. Therefore, to compute the intensity/color for a point of the triangle, we only need to know the specular reflection vector of that point. Computing the real specular reflection vector for each point of the triangle is a very expensive process, so one alternative (besides the Phong shading) is to estimate the specular reflection vector for each point of the triangle if we know the specular reflection vectors at the vertices of the triangle. This can be done as follows.

For each edge of a polygon, we use a representation as follows:



Rmin is the specular reflection vector of the lower vertex of the edge. So, for edge e2, this entry would be R2 and for edge e3 this entry would be R1. **ΔR** is the step size for specular reflection vector in y direction. So for edge e2, **ΔR** is computed as follows:

$$\Delta R = (R3 - R2)/(y3 - y2)$$

For scan line y, once we have Ra and Rb from edge e2 and edge e3 (e2 and e3 are edges in the Active Edge List now), we compute a step size for the specular reflection vector $\Delta_x R$ as follows:

$$\Delta_x R = (Rb - Ra)/(b - a)$$

Then the specular reflection vector for each subsequent pixel in the span [a, b] would simply be the sum of the specular reflection vector of the previous pixel and $\Delta_x R$. So, the specular reflection vector for x=a+1 would be

$$R_{a+1} = R_a + \Delta_x R$$

And the specular reflection vector for x=a+2 would be

$$R_{a+2} = R_{a+1} + \Delta_x R.$$

3. Gouraud shading (intensity-interpolation shading) and Phong shading (normal-interpolation shading) can both be used to eliminate intensity discontinuities when rendering a polygonal mesh. However, Gouraud shading could generate the so-called ***Mach band effect*** and Phong shading would not. However, Phong shading has a

disadvantage over Gouraud shading. What is that disadvantage? Is there a way to overcome that disadvantage of Phong shading? (5 points)

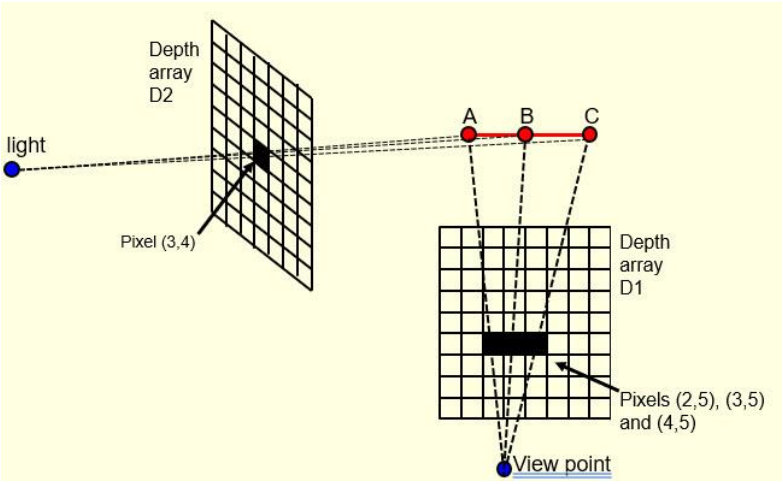
Phone shading is not as efficient as Gouaud shading because Phone shading has to compute the specular reflection vector R for each point of the polygon mesh. One option is to use the bisector the angle between L and V, H, to compute the color/intensity, the so called Blinn-Phone shading. This can only improve the performance to certain degree though. Another option is to use the above incremental method (Question #2) to estimate the specular reflection vector for each point of the triangle (also called **fast Phong shading**: https://people.eecs.berkeley.edu/~jfc/cs184f98/lec30/lec30.html)

4. The ***shadow volume based*** 'shadow generation' algorithm can be integrated with the scan-line hidden surface elimination process so that we can do hidden surface elimination and shadow generation at the same time. Can the ***shadow volume based*** 'shadow generation' algorithm be integrated with the Z-buffer method so we can also do hidden surface elimination and shadow generation at the same time, and how or why not? (10 points)

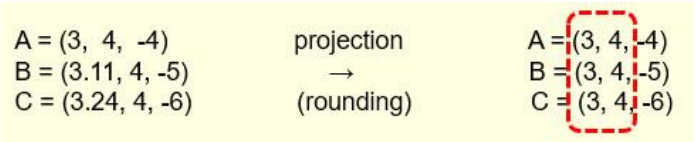
   See Section 9.6 of the notes “Lighting and Shadows II”

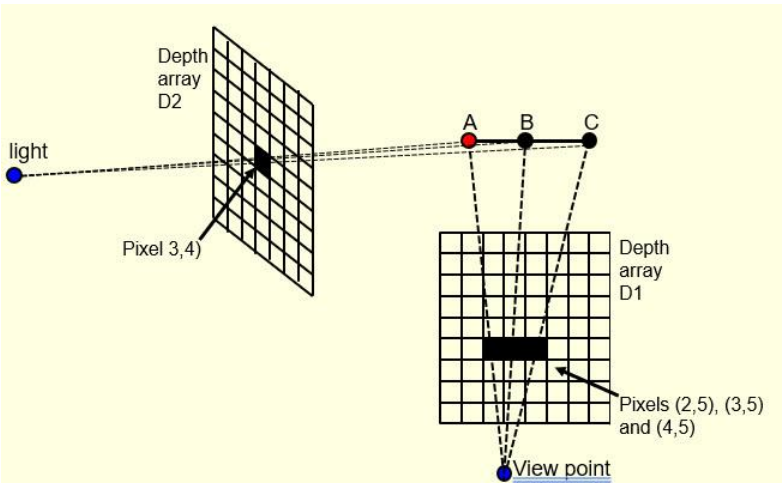
5. The ***shadow map based*** 'shadow generation' algorithm is easy to implement. But it has a potential problem. What is it? What is the reason for getting this potential problem? Is there a way to overcome this potential problem? (10 points)

   Sol: Consider the case shown in the following figure. The line segment AC is visible both to the view point and the light source. Therefore, theoretically, none of the points of the line segment should be in shadow.



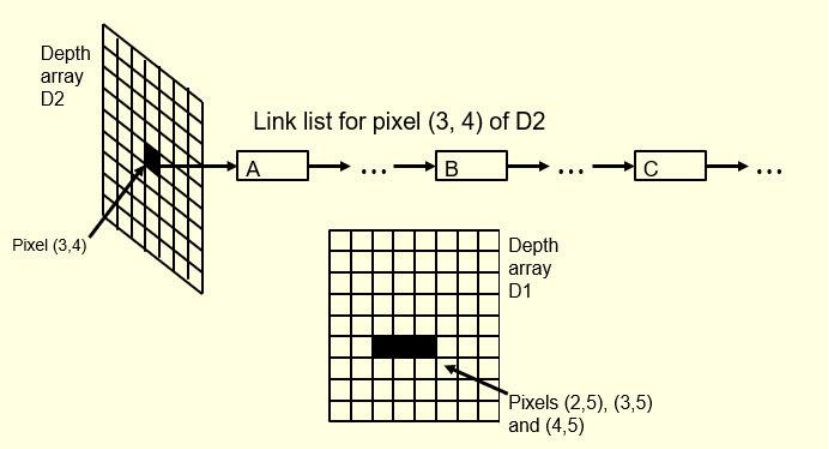
However, if the coordinates of points A, B and C in the light source coordinate system are (3, 4, -4), (3.11, 4, -5) and (3.14, 4, -6), respectively (see the left side of the following figure), then after projection, they will all have (3, 4) as the x- and y-coordinates (see the right side of the following figure). Since A has the largest depth (z-coordinate) value, entry (3, 4) of the depth array D2 will store the depth of A, -4. The depth value of B is -5 and the depth value of C is -6, both are smaller than the depth of A, content of entry (3, 4) of D2. Therefore, both points (actually the entire line segment except A) will be considered not visible to the light source and consequently will be shown as points in shadow (see the next figure).





To overcome this problem, one way is to increase the resolution of the display surface (this is not really a solution, but a way to reduce the seriousness of the problem). A real solution to this problem is to build a link list for each entry of the depth array D2 (see the following

figure).



When the Z-buffer method is performed with respect to the light source, each point projected into pixel (m, n) will not only be comparing its depth (z-coordinate) value with the current content of entry (m, n) of the depth array D2, the point (with coordinates before projection) will also be inserted into a link list for that entry. The link list is sorted by depth. Therefore for entry (3, 4) of D2, we would have A, B, and C in the link list (in that order; see the above figure).

For each point that is visible to the view point, say point B, we first compute coordinates of B in the light source coordinate system, (3.11, 4, -5). After rounding, we get (3, 4, -5). Then we compare the depth of B, -5, with the content of entry (3, 4) of D2. If -5 equals the content of entry (3, 4), B is obviously not in shadow. If -5 is smaller than the content of entry (3, 4) of D2, we then search through the link list of entry (3, 4) to find B. If none of the points in the list before B blocks B, then B is not in shadow. Otherwise, B is classified as in shadow. Question: how do you tell if a point in the list before B blocks B? (if light source location, that point and B are collinear)