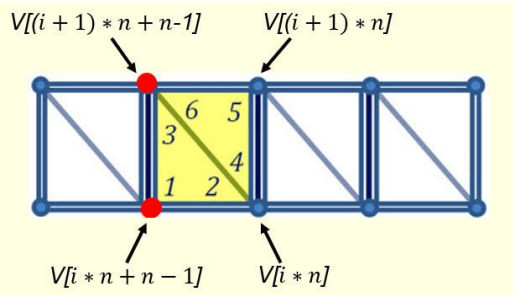
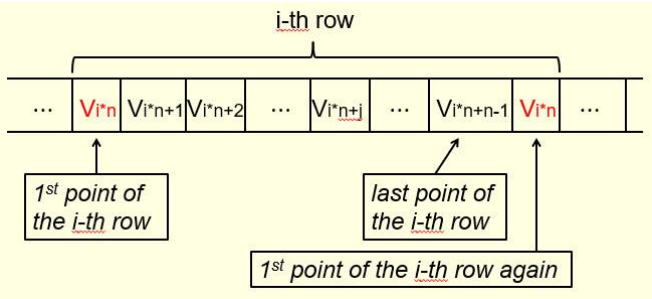
CS 535 Computer Graphics
Homework Assignment 4-2 Solution Set (20 points)
Due: 10/11/2024

1. In line 2 of slide 88 of the notes "3D Data Structures, 3D Data Management and 3D Models", number of vertices (*numVertices*) is set as $(prec + 1) * (prec + 1)$ even though the sphere is divided into $prec$ slices and each slice is divided into $prec$ segments only. Why? (2 points)

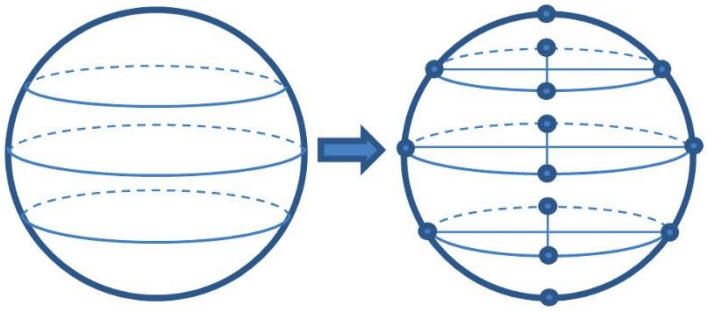
   Sol:
   Each sliced sphere region is a closed band and the texture map is supposed to wrap around the entire sphere, this means the last point of each row of vertices should be connected to the start point of each row so two triangles can be formed between the pair of last two points and the pair of first two points of each two adjacent rows of vertices as well. For instance, if $prec = n$ and the first point and last point of the i-th row of vertices are $V[i * n]$ and $V[i * n + n - 1]$, respectively, then these two points should be connected for each i (see the left figure below). This is implemented by putting $V[i * n]$ both as the start point and the end point for the i-th row in the vertex array (see the right figure below). So each sliced sphere region actually is represented by (n+1) vertices in the vertex array.





   On the other hand, the sphere should be sliced into even number of regions so slices in the upper half of the sphere would be symmetric to slices in the lower half of the sphere (see the left side of the following figure where the sphere is sliced into four symmetric regions). But most importantly, the number of rows of vertices that should be generated is 1 plus the number of sliced sphere regions because we generate two rows of vertices for each sliced region, one for each boundary of the region. If we only generate one row of vertices for each sliced region (by taking the central loop of the slice), we will not get rows of vertices for the north and south poles. By generating two rows of vertices for n slices, we get 2n rows of vertices. But (n-1) pairs of the vertex

rows coincide, (n-1) of them should be ignored. So totally we have 2n-(n-1)=n+1 distinct rows of vertices.

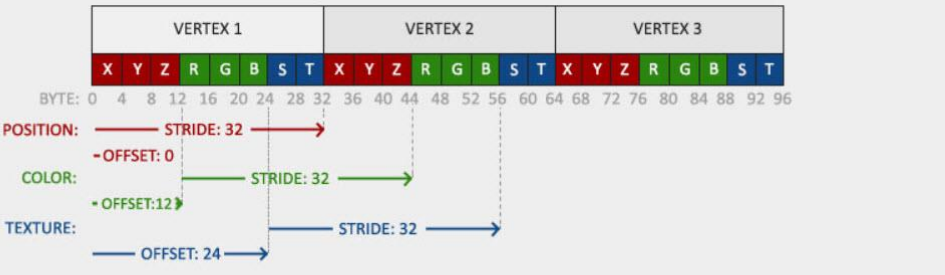


Combining the above results, we have n+1 rows of vertices and each row has n+1 vertices, so totally we have (n+1)*(n+1) vertices in the vertex array.

2. In the above case, how many VAOs and VBOs are needed for the sphere to be rendered? Remember the sphere is textured and shaded (that is, a normal vector is needed for each vertex) ? (2 points)

Sol:

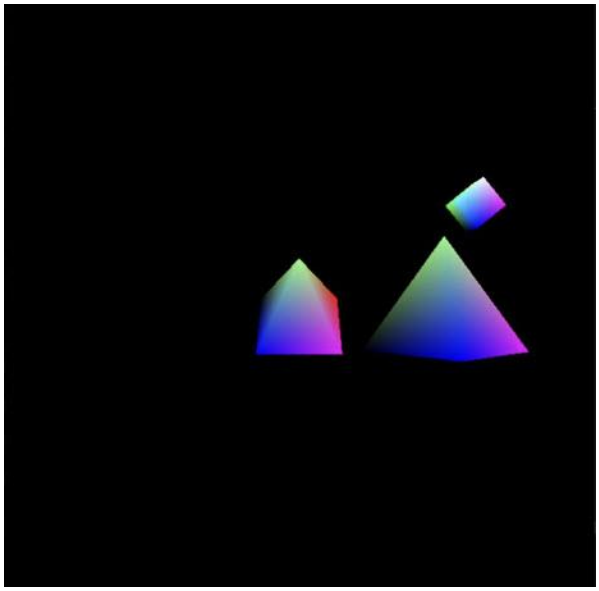
One option is to use one VAO and three VBO's, one VBO for vertex list, one for texture coordinates list, and one for normal list. Another option is to use one VAO and one VBO for a single vertex list. In this case, each vertex in the vertex list contains all the information about the vertex: vertex coordinates, texture coordinates, normal vector (or RGB colors of that vertex) as the example shown in the following figure. The shader needs to know what vertex attributes are included in each vertex and the byte stride to get from one attribute to the next.




```
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);
```

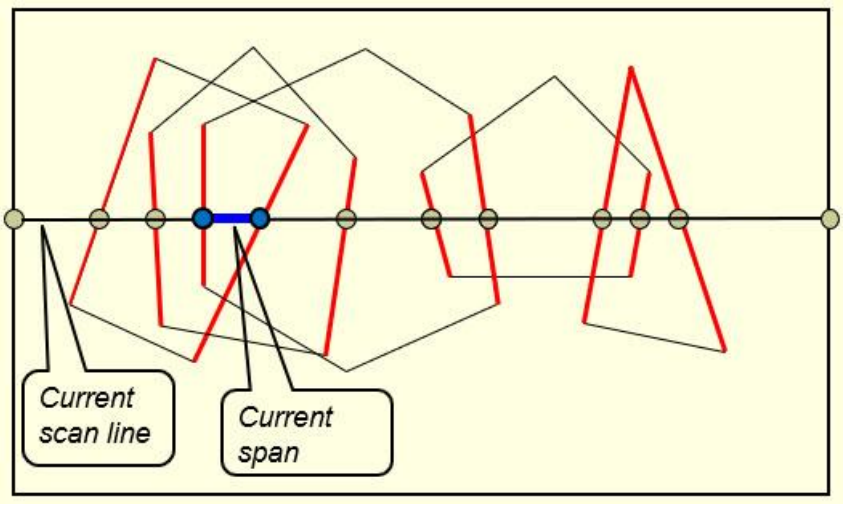
3. Test run example program 4-4 (to show the solar system using matrix stack), but use a pyramid, instead of a cube, to represent the sun. Attach a screen shot of your test run result. (6 points)

Sol:



4. The scan-line hidden surface elimination method is an extension of the 2D polygon scan conversion method. The 2D scan conversion method processes one polygon at a time while the scan-line hidden surface elimination method can process multiple polygons simultaneously. Using the scan-line method to eliminate hidden surfaces, one needs to build two tables: a *bucket-sorted edge table* (ET) and a *polygon table* (PT), and maintains two lists: an *active edge list* (AEL) and an *active polygon list* (APL). Given the following projected polygons, the current scan line and the current span, please answer the following questions:

   (i) how many edges are contained in the ET?
   (ii) how many polygons are contained in the PT?
   (iii) how many edges are contained in the current AEL?
   (iv) how many polygons are contained in the current APL? (4 points)



Sol:

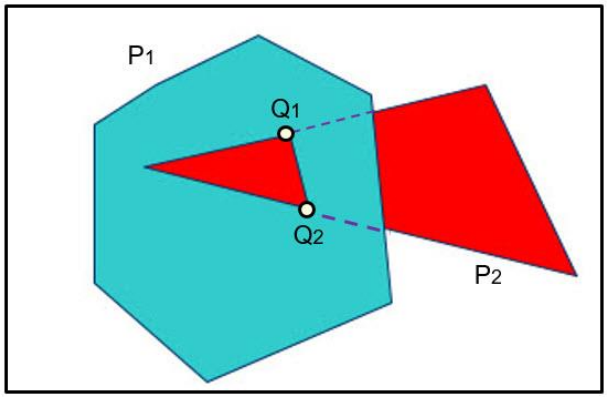
(i) There are 23 edges in the 3D scene. But one of the edges is a horizontal edge and will be ignored when we build the bucket-sorted edge table. So, totally, there are 22 edges in the bucket-sorted edge table (ET).

(ii) Five polygons in the polygon table (PT).

(iii) Ten edges are contained in the current AEL.

(iv) Three polygons are contained in the current APL.

5. Can Z-buffer method handle piercing 3D polygons like the case shown below? Justify your answer. Note that part of polygon P2 is blocked by polygon P1 and part of polygon P1 is blocked by polygon P2. (2 points)

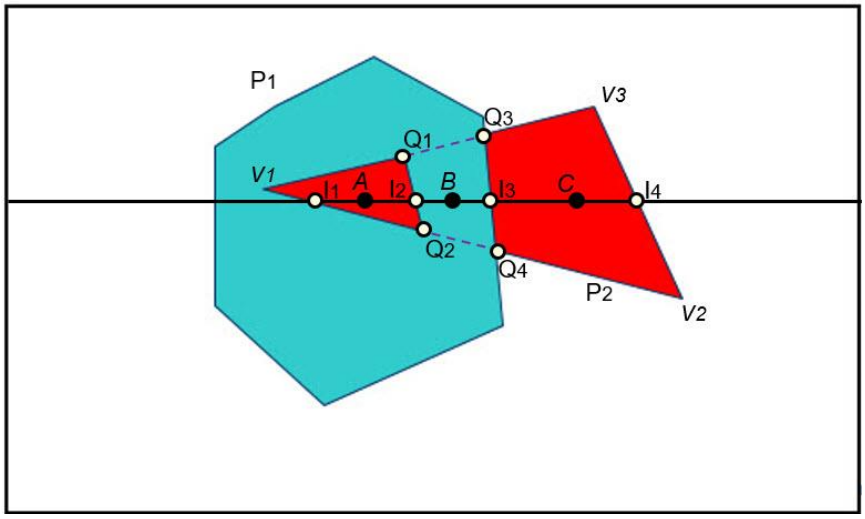


Sol:

YES. For simplicity, we shall assume there are no other polygons in the scene. We shall also assume that polygon P1 has already been processed. Therefore for each

pixel inside the projection of P1, the corresponding entry in the Depth Array contains the **Depth value** of the corresponding point on P1.

Now when P2 is processed, for each pixel inside the projection of the triangle V1Q2Q1 such as A in the following figure, since the corresponding point of A in the triangle V1Q2Q1 is closer to the viewer than the corresponding point on P1, depth value of the corresponding entry in the Depth Array will be replaced with the depth value of the corresponding point of A on the polygon P2.



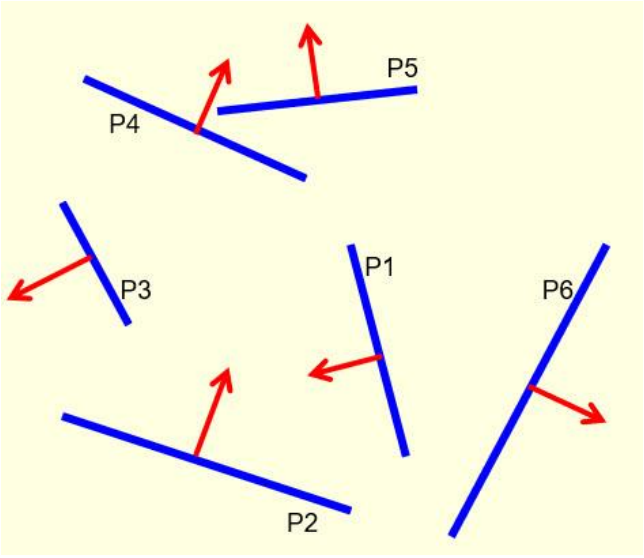
For each pixel inside the projection of the region $Q_1Q_2Q_4Q_3$ of P2 such as B in the above figure, since the corresponding point of B in the region $Q_1Q_2Q_4Q_3$ is farther away from the viewer than the corresponding point on P1, depth value of the corresponding entry in the Depth Array will not be replaced with the depth value of the corresponding point of B on the polygon P2.

For each pixel inside the projection of the region $Q_3Q_4V_2V_3$ of P2 such as C in the above figure, since the corresponding point on P2 is the only visible point to the viewer for this pixel, and the depth value of the corresponding entry in the Depth Array is the default value of -1, so this depth value will be replaced with the depth value of the corresponding point of C on P2.

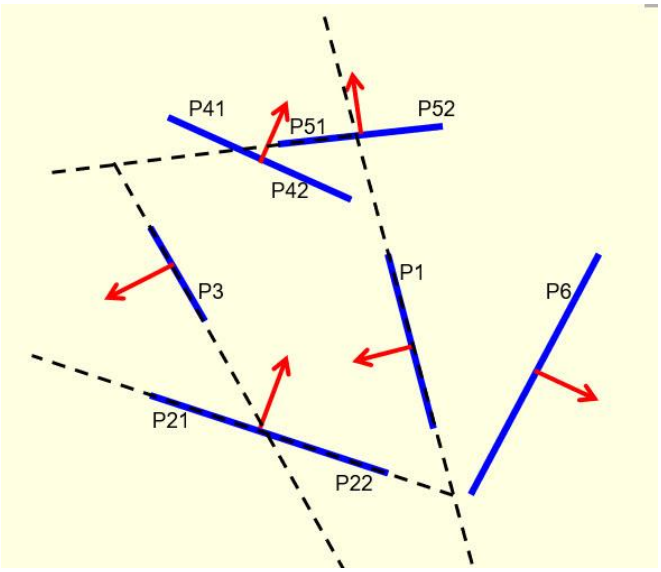
So, pixels within the projection of the polygon P2 are all properly processed.

6. The BSP tree method is basically a polygon sorting method. By sorting polygons using a BSP tree and then rendering polygons in the sorted order, one can eliminate hidden surfaces

through the "overwriting" process. Given the following 5 polygons, what is the minimum number of polygons that could be added to a BSP tree? What is the maximum number?

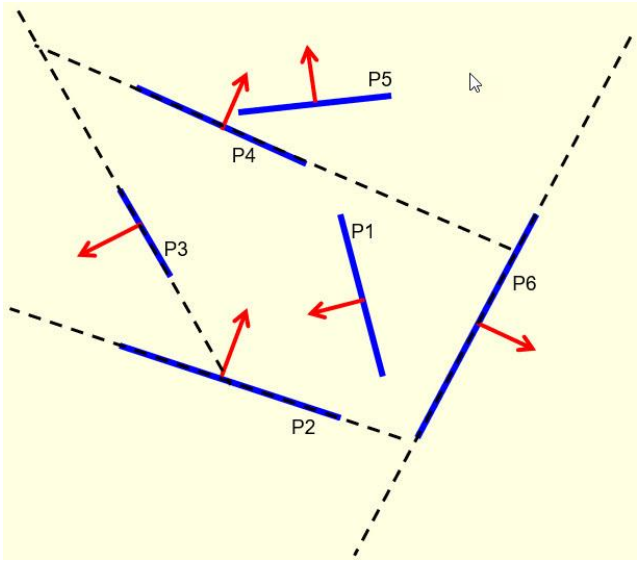


To help you figure out all possible BSP trees that you could get, the following figure is provided. (4 points)

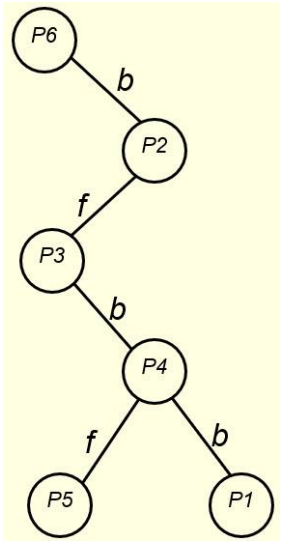


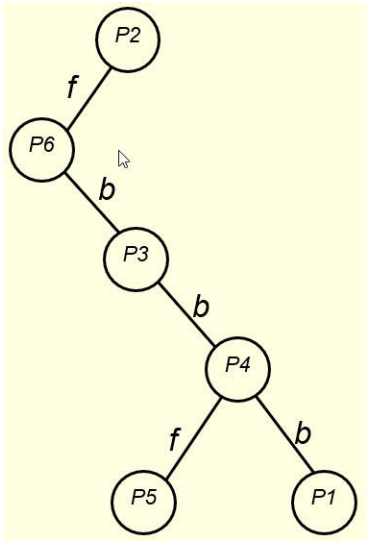
Sol:

The answer for the **minimum case** is 6. Consider the case shown in the following figure where we first extend **P6** then **P2** then **P3** and then P4.



In this case we get a BSP tree with only 6 nodes, as the one shown on the left below.

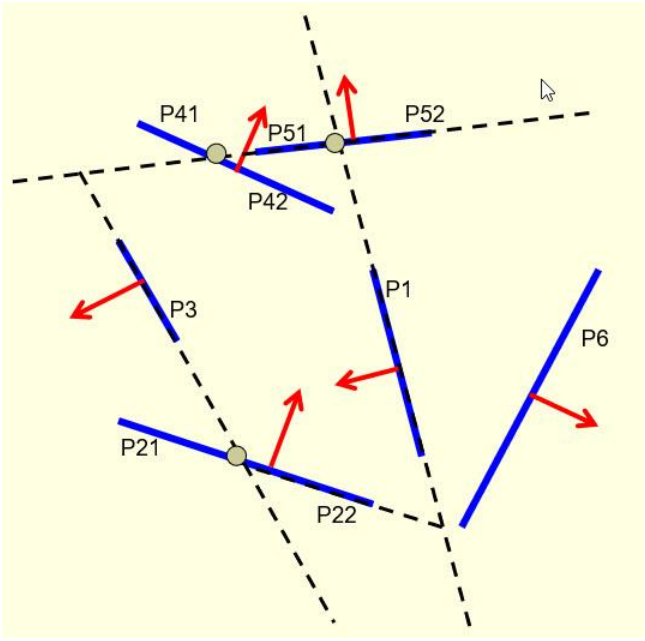




Note that this BSP tree with minimum number of nodes is not unique. It is possible to get a different BSP tree with only 6 nodes too such as the one shown on the right above.

The answer for the **maximum case** is 9 . Consider the case shown in the following figure where we extend P1 first. The line that passes through P1 intersects P5. Hence P5 has to be split into P51 and P52. Back side of P1 contains two segments P51 and P6. No matter which segment we extend next, the partition will stop after the extension. We choose to extend P52 in this case. The front side of P1 contains four segments P51, P4, P3 and P2. If we extend P51, the line that contains P51 intersects P4, hence P4 has to be split into P41 and P42. The front side of P51 contains only one segment P41. So no

partition is required for the front side of P51. The back side of P51 contains three segments P42, P2 and P3. If we extend P3, the line that contains P3 intersects polygon P2, so P2 will be split into P21 and P22. The front side of P3 contains on polygon P21 only, so no partition is required. The back side of P3 contains two polygons P42 and P23. No matter which polygon we extend, the partition will stop after the extension. We choose to extend P22 in this case. After this point, each subspace is either empty or contains at most one segment, so the partition stops.



In this case we get a BSP tree with 9 nodes, as follows. Any other partition of the space would get us a BSP tree with less nodes.