Problem 2 (3 marks): Back face culling.

Suppose we have a scene with a single, fully opaque, convex object, which is entirely inside the view volume. The convex object is specified by a mesh.

Under orthogonal projection, will back face culling alone be guaranteed to produce the correct set of visible polygons? Why or why not?

Under perspective projection, is it true that back face culling alone is guaranteed to produce the correct set of visible polygons? Why or why not?

For orthogonal projection:

Back face culling alone can't be guaranteed to produce the correct set of visible polygons.

∵ An orthogonal projection is a transformation of object descriptions to view a plane along lines that are all parallel to the view-plane normal vector N.

∴ After applying orthogonal projection, what we can get is Plan View, Front Elevation View as well as Side Elevation View. ⇒ Orthogonal projection could only give us correct information of lengths and angles.

∵ The object is fully opaque, and we want the correct set of polygon sets.

∴ With only these views generated by orthogonal projection, it is hardly possible that one can tell the correct set of polygon sets.

For perspective projection:

Back face culling alone may be guaranteed to produce the correct set of visible polygons.

∵ A perspective projection is that object positions are transformed to projection coordinates along lines that converge to a point behind the view plane.

∴ Perspetive views of scene are more realistic, because distant objects in the projected display are reduced in size.

∵ Back-face culling is perfect in applying opaque convex object.

∴ By this method, we may be able to tell the correct set of visible polygons!

Problem 3 (5 marks): BSP vs. depth-sort.

Show that the back-to-front display order determined by traversing a BSP tree is not necessarily the same as the back-to-front order determined by depth-sort, even when no polygons are split. To receive full marks on this problem, the number of polygons you use for your example must be the smallest possible and you also need to prove that the number of polygons you used is the smallest possible. **Hint: mostly you need an example. The rest is easy.**

∵ BSP enables easy traversal in any order relative to the observer.

∴ Suppose that we can use a set of polygons $P = \{P_1, P_2, \cdots, P_n\}$, to define the 3D environment.

∴ Let we choose a random polygon P_i in the P to be the root of the BSP tree.

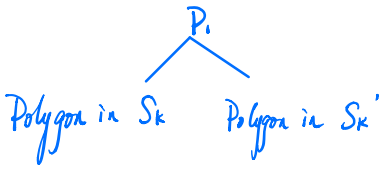
∵ The plane defined by the polygon P_i partitions the rest of three-space into 2 half-spaces.

Let S_k and S_k' be those 2 half-spaces.

∵ P_i has 2 subtrees (positive & negative)

∴ If a polygon in P_i lies in S_k, then it is placed in the positive subtree, otherwise it is in the negative one.

⇒



∴ If the partitioning plane cuts the polygon, then the polygon is split into 2, which one part placed in S_k and the others in S_k'.

∴ Most of the polygons will lie in just one of the two half-spaces, and such recursive process eventually generate a BSP tree.

∴ A new BSP tree is generated.

∴ It is easy to add new polygon into it.

∴ We can determine the visibility priorities of polygons in the BSP tree, as long as viewing position and orientation are given.

∴ The calculation of visibility priorities is a variant of an in-order traversal of a binary tree.

∴ It should be traverse one subtree, visit the root, and then traverse the other

subtree.

∵ We can easily classify the two sides of a node's polygon as the "near" side and the "far" side.

∵ The "near" side is the half-space containing the view point, while the "far" side is the other half-space.

∴ The traversal for a back-to-front order is the recursive implementation of the order:

① the far side
↓
② the node
↓
③ the near side