10. Visible-Surface Determination Revisited

10.1 Aliasing Problem of Z-Buffer Algorithm

If the following two triangles are scan converted, what would happen?

We should get a result similar to the one on the left. But sometimes we end up getting something like the one on the right. Why?
Why? Because of numerical inaccuracies in performing the \textit{z-interpolation} (and the \textit{x-interpolation})

Solution:
- When scan-converting a polygon, if an edge intersects the scan line \( y = j \) at a point between \( x = i \) and \( x = i + 0.5 \) then the pixel \((i,j)\) is skipped. The pixel \((i,j)\) will be plotted only if the intersection occurs between \( x = i - 0.5 \) and \( x = i \).

![Diagram showing scan-conversion process.](image)

- If adjacent triangles do not share the same vertices then extra vertices have to be inserted to ensure vertices occur at the same points along the common part of collinear lines.
10.2 Binary Space-Partition (BSP) Trees
(Fuchs, Kedem, Naylor)

- Efficient for calculating visibility relationships among 3D polygons (from any view point)
- Based on the following concept

A polygon on the same side of the plane as the eyepoint can not be obscured by polygons on the other side.
Given a set of 3D polygons (with assigned normal directions), a **BSP tree** can be constructed as follows:

"Choose an arbitrary polygon as the root polygon. Use the root polygon to partition the environment into two half spaces: **front** and **back** (relative to polygon normal). Any polygon lying on both sides of the root polygon’s plane is split. Then choose an arbitrary polygon on each side to divide the remaining polygons in its half-space in the same fashion. This process is recursively repeated until each region contains at most one polygon."
How to use a BSP tree to calculate visibility?

For a given view point, recursively display polygons of the tree in the following order:

- if the view point is in the root polygon’s front half-face, display polygons in the root’s rear half-space, the root polygon and then polygons in its front half-space.
- if the view point is in the root polygon’s rear half-space, display polygons in reverse order
- if the view point is on the plane that contains the root polygon then either way is okay.
10.3 Subpixel Area-Subdivision Algorithms

- to accomplish antialiasing

Catmull’s: for each pixel, determine amount of the pixel covered by the visible part of each polygon, then compute a weighted sum of the visible parts’ colors.

A-Buffer: for each pixel, clip the polygons against the pixel area to create a list of clipped polygon segments. Each fragment is associated with a 4x8 bit mask of those parts of the pixel it covers. The area-weighted average of the colors of the pixel’s visible surfaces is obtained by selecting fragments in depth-sorted order and using their bit masks to clip those of farther fragments.

Can be created by XORing together masks representing each of the fragment’s edges.
10.4 Algorithms for Curved Surfaces

Catmull’s recursive-subdivision algorithm (point sampling method)

```plaintext
for each patch do
    push patch onto stack;
while stack not empty do
    pop patch from stack;
    if patch covers ≤ 1 pixel then
        if patch’s pixel closer in z then
            determine shade and draw;
        else
            subdivide patch into 4 subpatches;
            push subpatches onto stack;
```

- Costly approach
- Does not use the build-in polygon scan conversion hardware
Tessellation Method

- Tessellate the surface into polygons and then scan convert the polygons

Lang-Carpenter Algorithm:

Add patches to patch table;
Initialize active-patch table;

for each scan line do
    update active-patch table;
    for each patch in active-patch table do
        if patch can be approximated by planar quadrilateral then
            add patch to polygon table;
        else
            split patch into subpatches;
            for each active subpatch do
                if subpatch intersects scan line then
                    add to active-patch table;
                else
                    add to patch table;
    process polygon table for current scan line;