Computer graphics

Hidden Surfaces

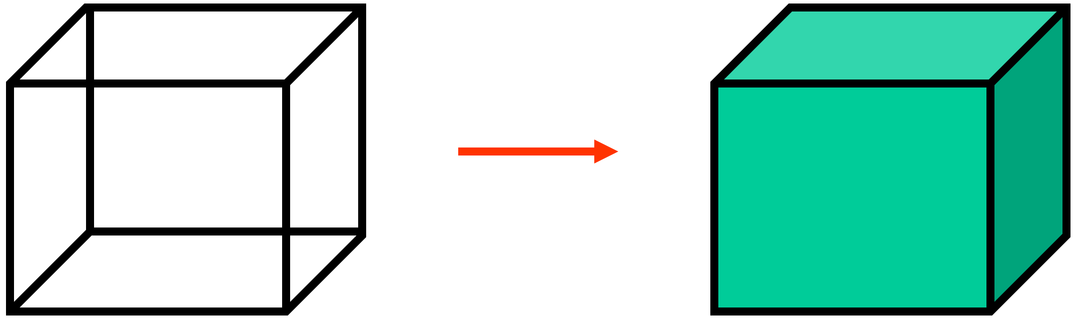
By: Suhani Chauhan

Visible-Surface Detection

Problem:

Given a scene and a projection,

what can we see?



Visible-Surface Detection

Terminology:

Visible-surface detection vs. *hidden-surface removal*

- **Characteristics of approaches:**

- Require large memory size?

- Require long processing time?

- Applicable to which types of objects?

- **Considerations:**

- Complexity of the scene

- Type of objects in the scene

- Available equipment

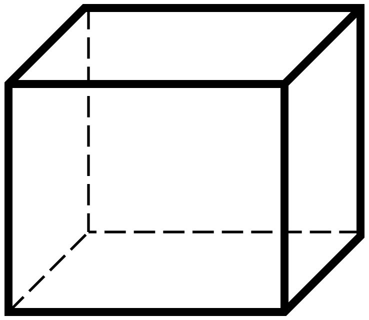
- Static or animated?

- Visible surface detection is the major concern for a realistic graphics for identifying those parts of scene which are visible from the chosen viewing position.
- Several algorithms have been developed. Some require more memory, some require more processing time & some apply only to the special types of objects. These methods are broadly classified according to how they deal with objects or with their projected images. These two approaches are:
- **Object-Space methods:** Compares objects & parts of objects to each other within scene definition to determine that surface, as whole, we should label as visible.
- **Image-Space methods:** Visibility is decided point-by-point at each pixel position on projection plane. Most visible surface detection algorithm uses this method but in some cases object space methods are also used for it.

Visible-Surface Detection

Two main types of algorithms:

- *Object space: Determine which part of the object are visible*
- *Image space: Determine per pixel which point of an object is visible*


Object space

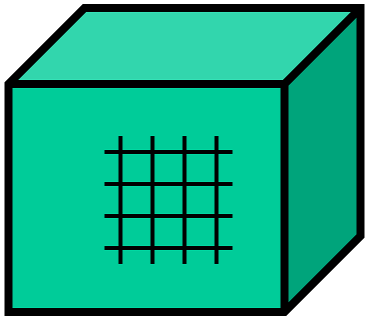


Image space

Visible-Surface Detection

Algorithms:

- Back-face elimination
- Depth-buffer
- Depth-sorting
- Scan line

And there are many other.

Back-face elimination

- In a solid object, there are surfaces which are facing the viewer (front faces) and there are surfaces which are opposite to the viewer (back faces).
- These back faces contribute to approximately half of the total number of surfaces. Since we cannot see these surfaces anyway, to save processing time, we can remove them before the clipping process with a simple test.
- Each surface has a normal vector. If this vector is pointing in the direction of the center of projection, it is a front face and can be seen by the viewer. If it is pointing away from the center of projection, it is a back face and cannot be seen by the viewer.

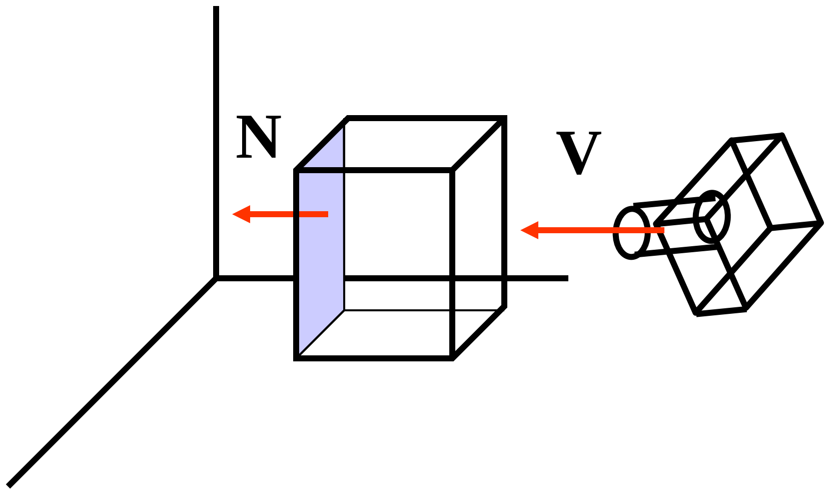
Back-face elimination

- A fast & simple object space method used to remove the hidden surface from 3D object drawing is known as "Plane equation method" or back-face detection method.
- A point (x,y,z) is inside the polygon surface if $Ax + By + Cz + D < 0$

Back-face elimination

We cannot see the back-face of solid objects:

Hence, these can be ignored



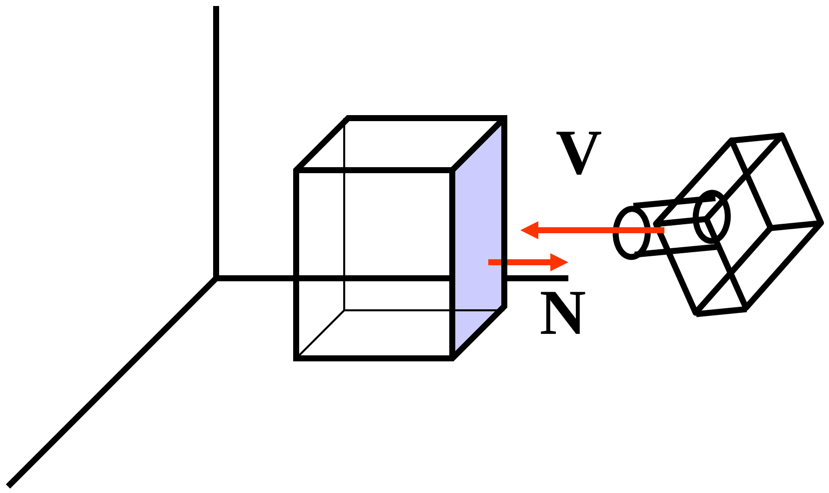
$$\mathbf{V} \cdot \mathbf{N} > 0 : \text{back face}$$

Back-face elimination

We cannot see the back-face of solid objects:

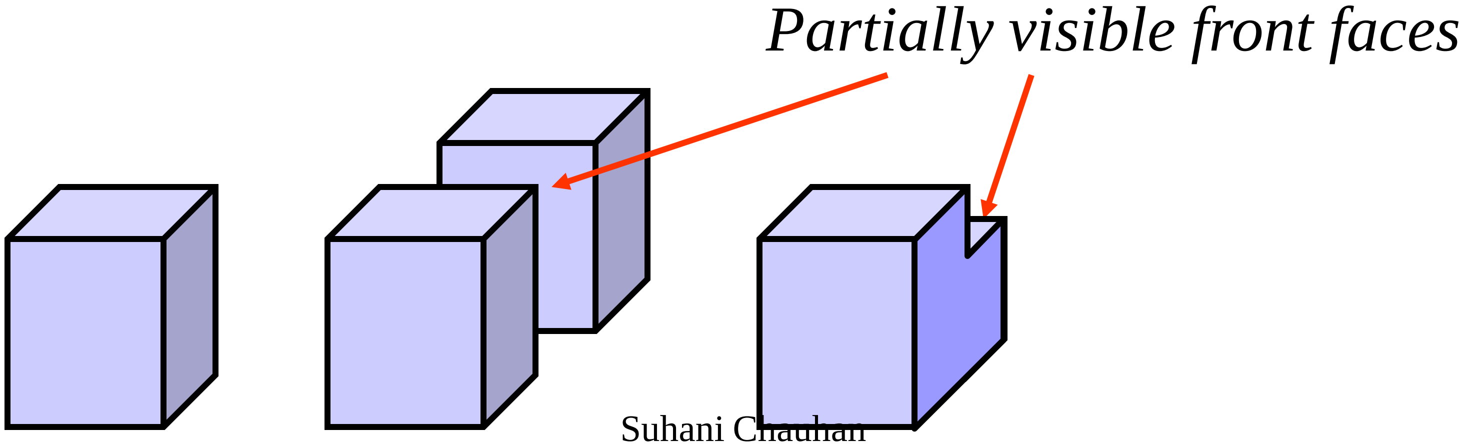
Hence, these can be ignored

$\mathbf{V} \cdot \mathbf{N} < 0 : \text{front face}$



Back-face elimination

- Object-space method
- Works fine for convex polyhedra: ±50% removed
- Concave or overlapping polyhedra: require additional processing
- Interior of objects can not be viewed

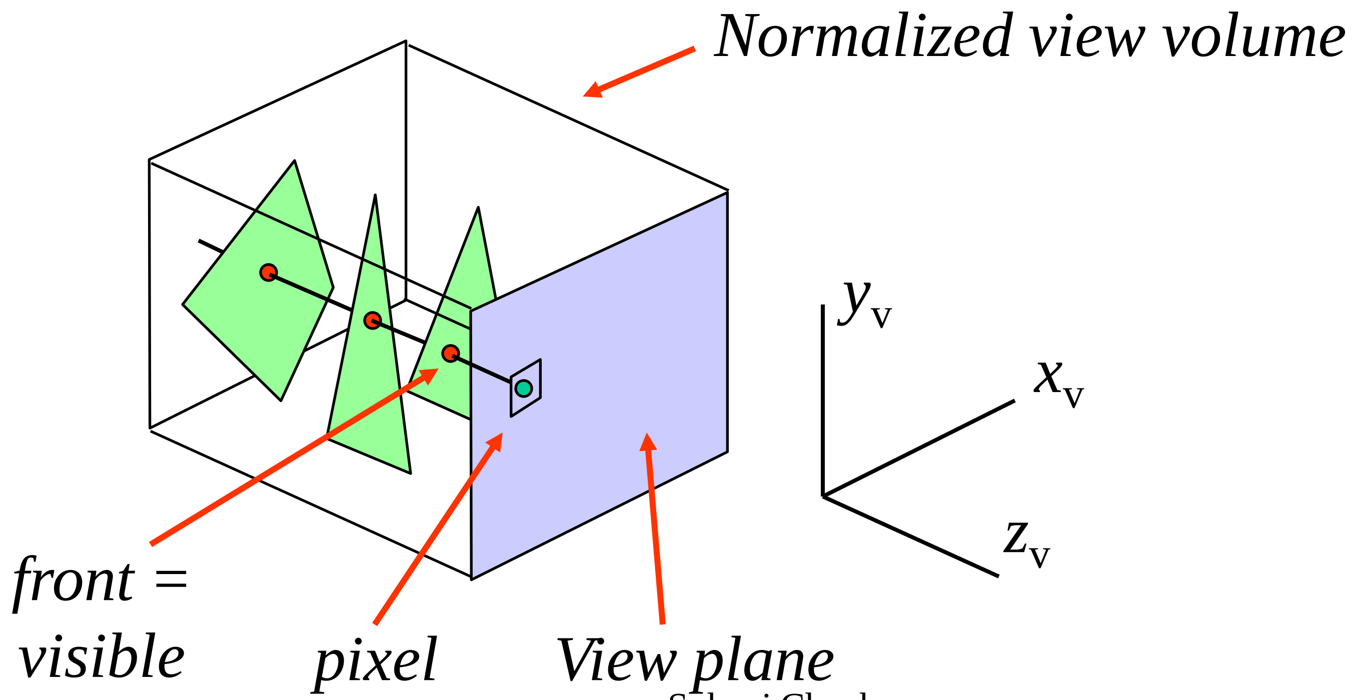


Depth-Buffer Algorithm (Z Buffer)

- This approach compare surface depths at each pixel position on the projection plane.
- Object depth is usually measured from the view plane along the z axis of a viewing system.
- This method requires 2 buffers: one is the image buffer and the other is called the z-buffer (or the depth buffer). Each of these buffers has the same resolution as the image to be captured.
- As surfaces are processed, the image buffer is used to store the color values of each pixel position and the z-buffer is used to store the depth values for each (x,y) position.

Depth-Buffer Algorithm

- Image-space method
- Aka *z-buffer algorithm*



Suhani Chauhan

Depth-Buffer Algorithm

Algorithm: Z-buffer

1. Initialize depth buffer & refresh buffer so that for all buffer position (x,y)

   depth (x,y) = -1, refresh (x,y) = Ibackground

2. For each position on each polygon surface, compare depth values to the previously stored value in depth buffer to the determine visibility.

   Calculate depth Z for each (x,y) position on polygon,

   If Z> depth (x,y) then

   depth (x,y) =Z

   refresh (x,y) = Isurface (x,y)

Depth-Buffer Algorithm

- Where Ibackground is an intensity value for background & Isurface (x,y) is an intensity value for surface at pixel position (x,y) on the projected plane.
- After all surfaces are processed, depth buffer contains depth value of the visible surface & refresh buffer contains corresponding intensity values for those surface.

Depth-Buffer Algorithm

+ Easy to implement
+ Hardware supported
+ Polygons can be processed in arbitrary order

- Costs memory
- Color calculation sometimes done multiple times
- Transparency is tricky

Depth-Sorting Algorithm

- This method uses both object-space & image-space method. In this method,

1. Sort all surfaces according to their distances from the view point.
2. Render the surfaces to the image buffer one at a time starting from the farthest surface.
3. Surfaces close to the view point will replace those which are far away.
4. After all surfaces have been processed, the image buffer stores the final image.

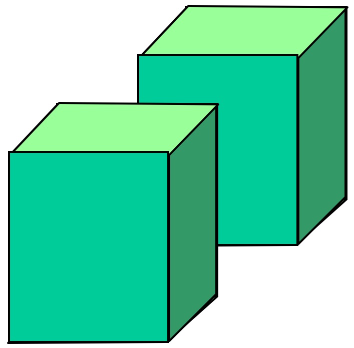
Depth-Sorting Algorithm

- The basic idea of this method is simple. When there are only a few objects in the scene, this method can be very fast. However, as the number of objects increases, the sorting process can become very complex and time consuming.
- Example: Assuming we are viewing along the z axis. Surface S with the greatest depth is then compared to other surfaces in the list to determine whether there are any overlaps in depth. If no depth overlaps occur, S can be scan converted. This process is repeated for the next surface in the list. However, if depth overlap is detected, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.

Depth-Sorting Algorithm

Painter's algorithm

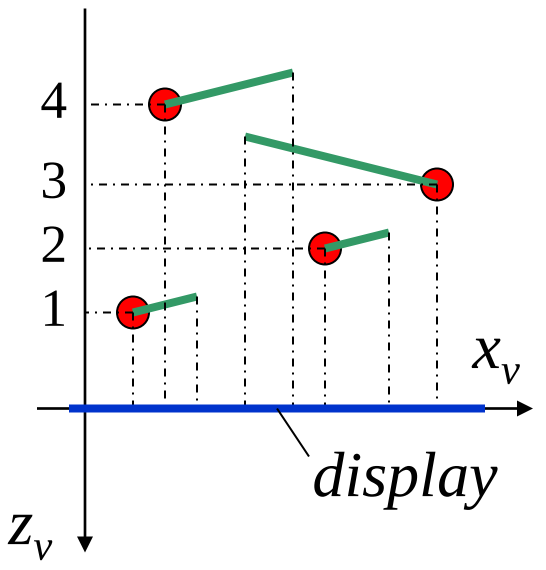
1. *Sort surfaces for depth*
2. *Draw them back to front*

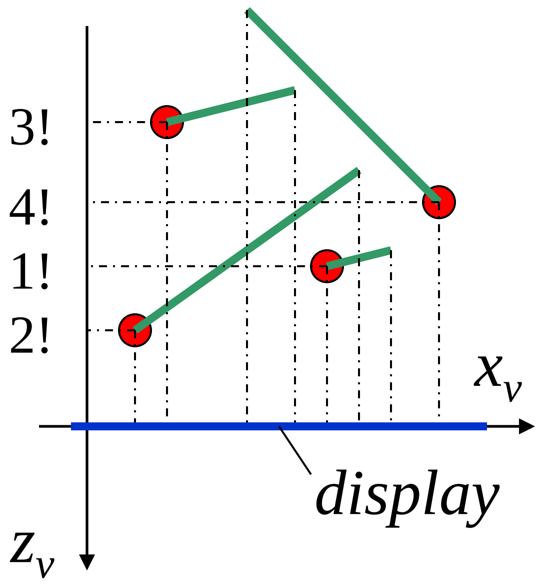


Depth-Sorting Algorithm

Simplistic version sorting:

- Sort polygons for (average/frontal) *z*-value





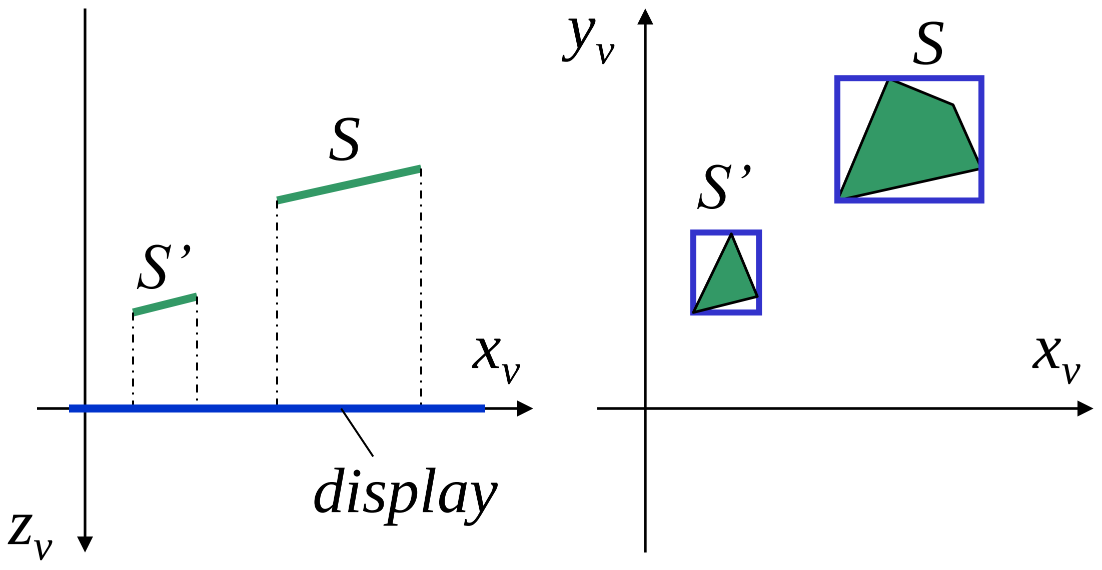
Depth-Sorting Algorithm

A polygon *S* can be drawn if all remaining polygons *S'* satisfy one of the following tests:

1. No overlap of *bounding rectangles* of *S* and *S'*
2. *S* is completely behind plane of *S'*
3. *S'* is completely in front of plane of *S*
4. Projections *S* and *S'* do not overlap

Depth-Sorting Algorithm

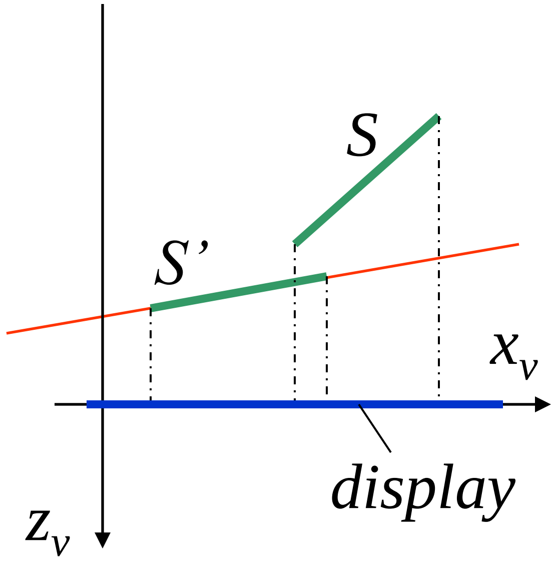
1. No overlap of *bounding rectangles* of *S* and *S'*

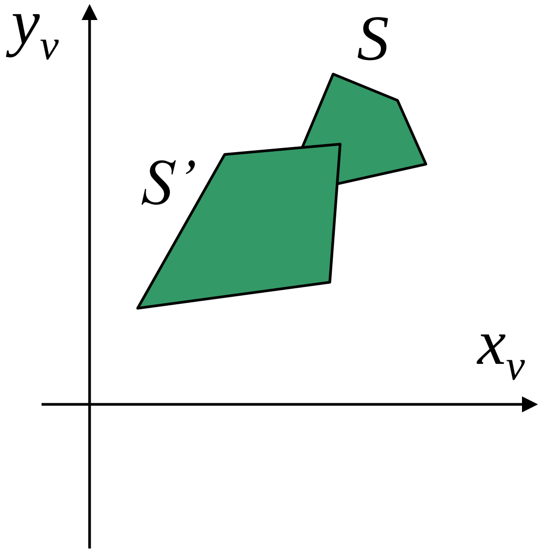


Depth-Sorting Algorithm

2. *S* is completely behind plane of *S'*

   Substitute all vertices of *S* in plane equation *S'*, and test if the result is always negative.

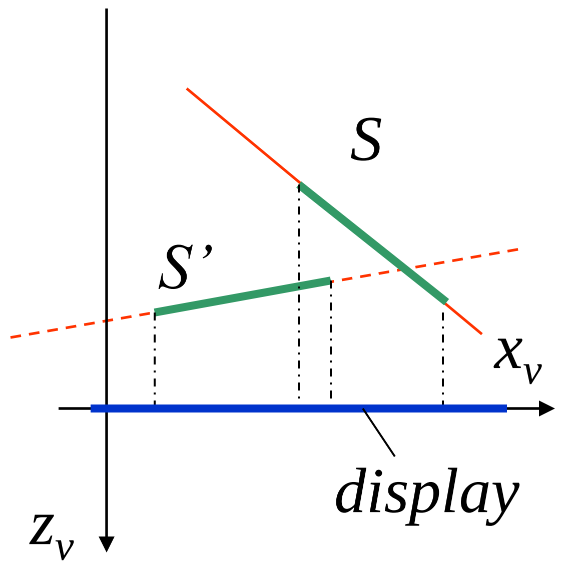


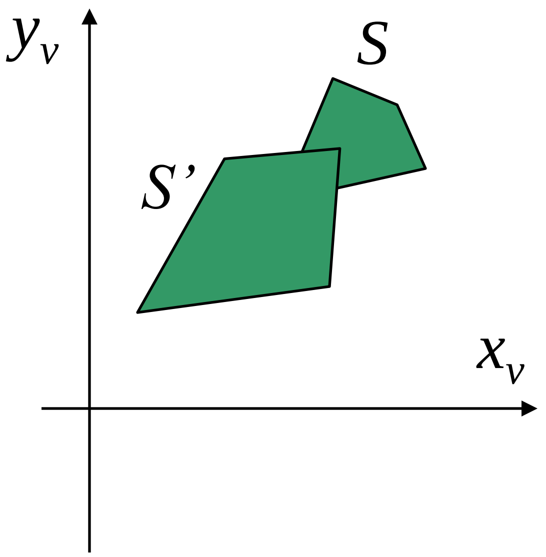


Depth-Sorting Algorithm

3. *S'* is completely in front of plane of *S*

   Substitute all vertices of *S'* in plane equation of *S*, and test if the result is always positive

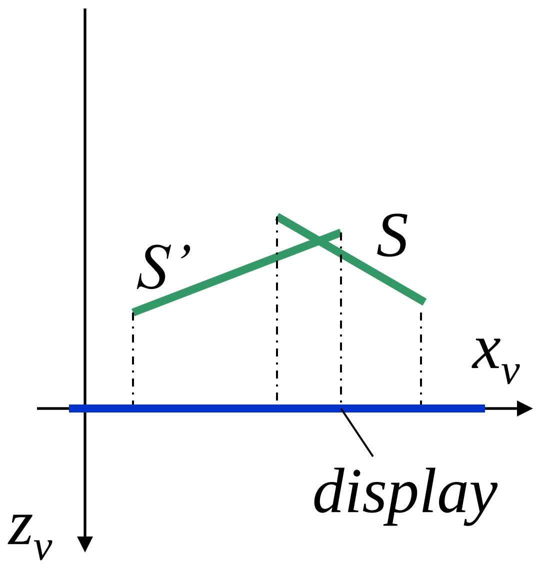


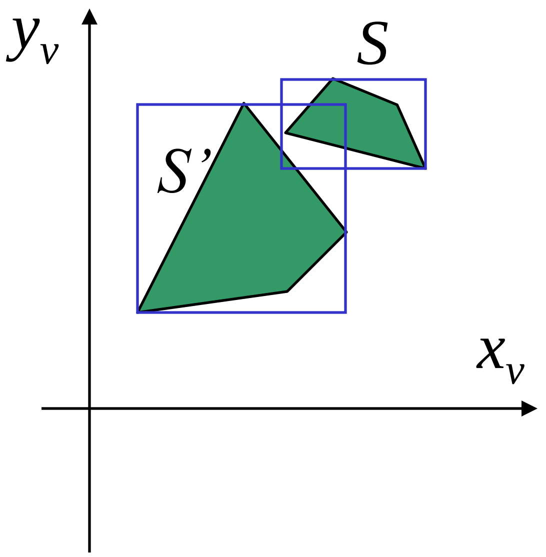


Suhani Chauhan

Depth-Sorting Algorithm

4. Projections *S* and *S'* do not overlap

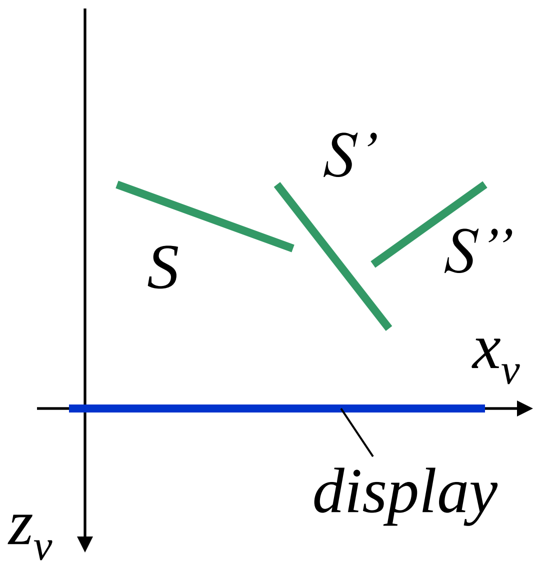




Depth-Sorting Algorithm

If all tests fail: Swap *S* and *S'*,

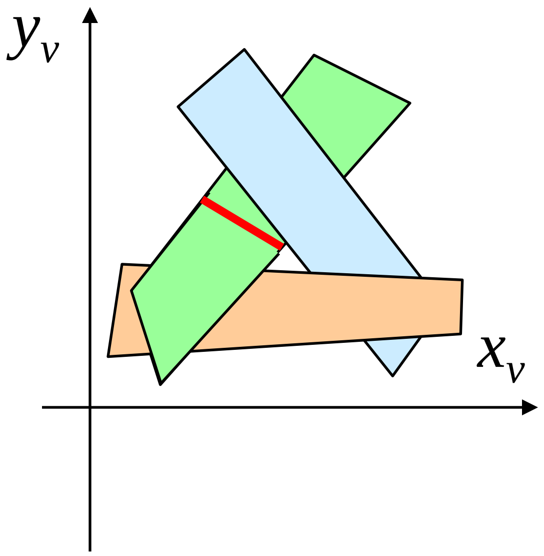
and restart with *S'*.

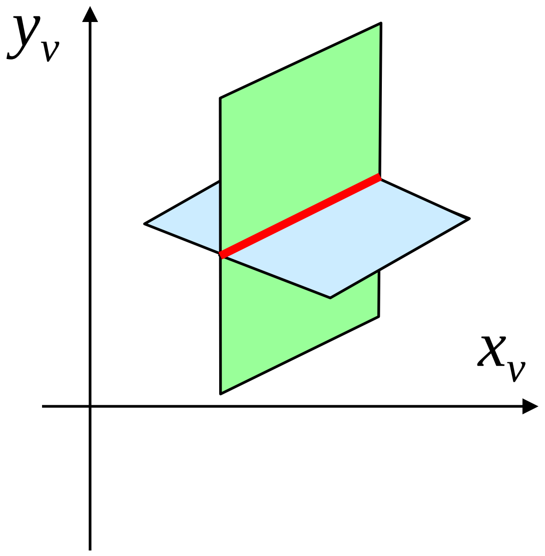


Depth-Sorting Algorithm

Problems: circularity and intersections

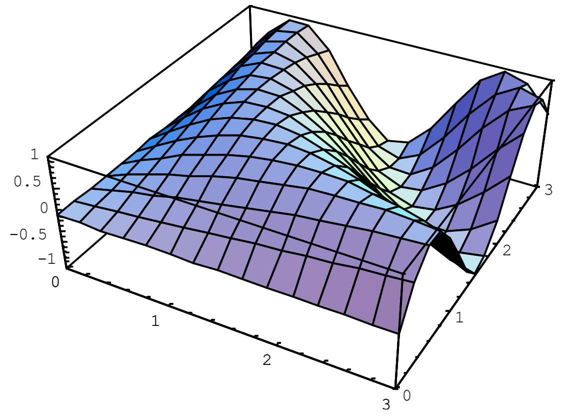
Solution: Cut up polygons.





Depth-Sorting Algorithm

- Tricky to implement
- Polygons have to be known from the start
- Slow: ~ #polygons



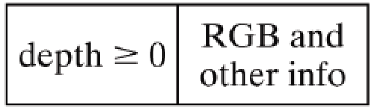
+ Fine for certain types of objects,
  such as plots of $z=f(x, y)$ or
  non-intersecting spheres
+ Produces exact boundaries polygons

A-Buffer Method

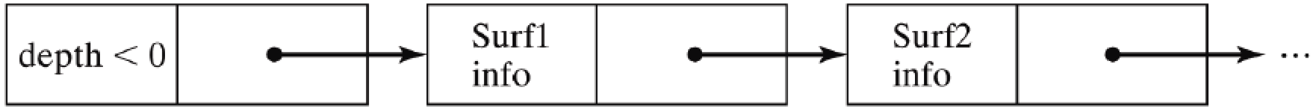
- Extends the depth-buffer algorithm so that each position in the buffer can reference a linked list of surfaces.
- More memory is required
- However, we can correctly compose different surface colors and handle transparent surfaces.

A-Buffer Method

- Each position in the A-buffer has two fields:
  - a depth field
  - surface data field which can be either surface data or a pointer to a linked list of surfaces that contribute to that pixel position

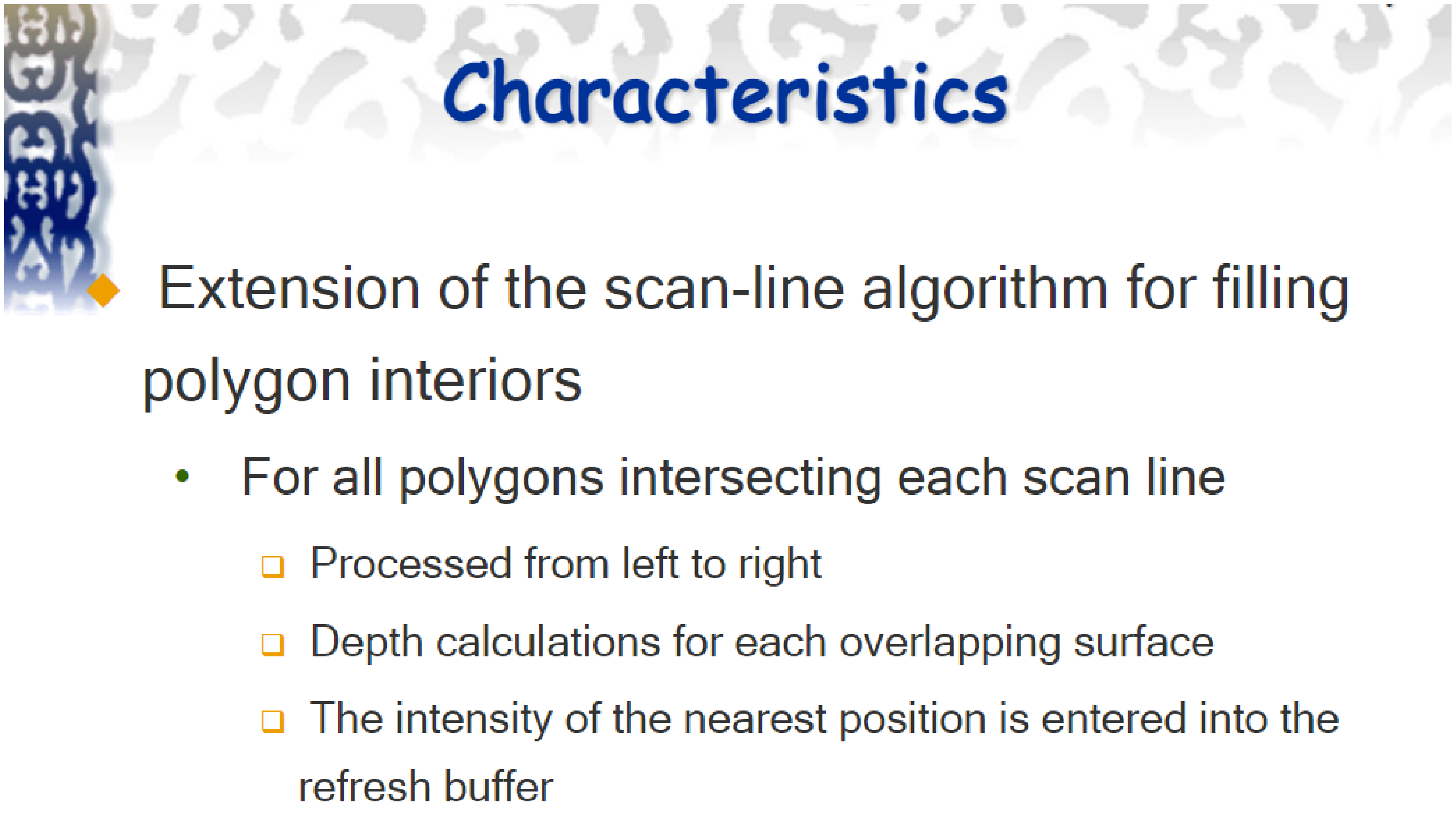


(a)



(b)

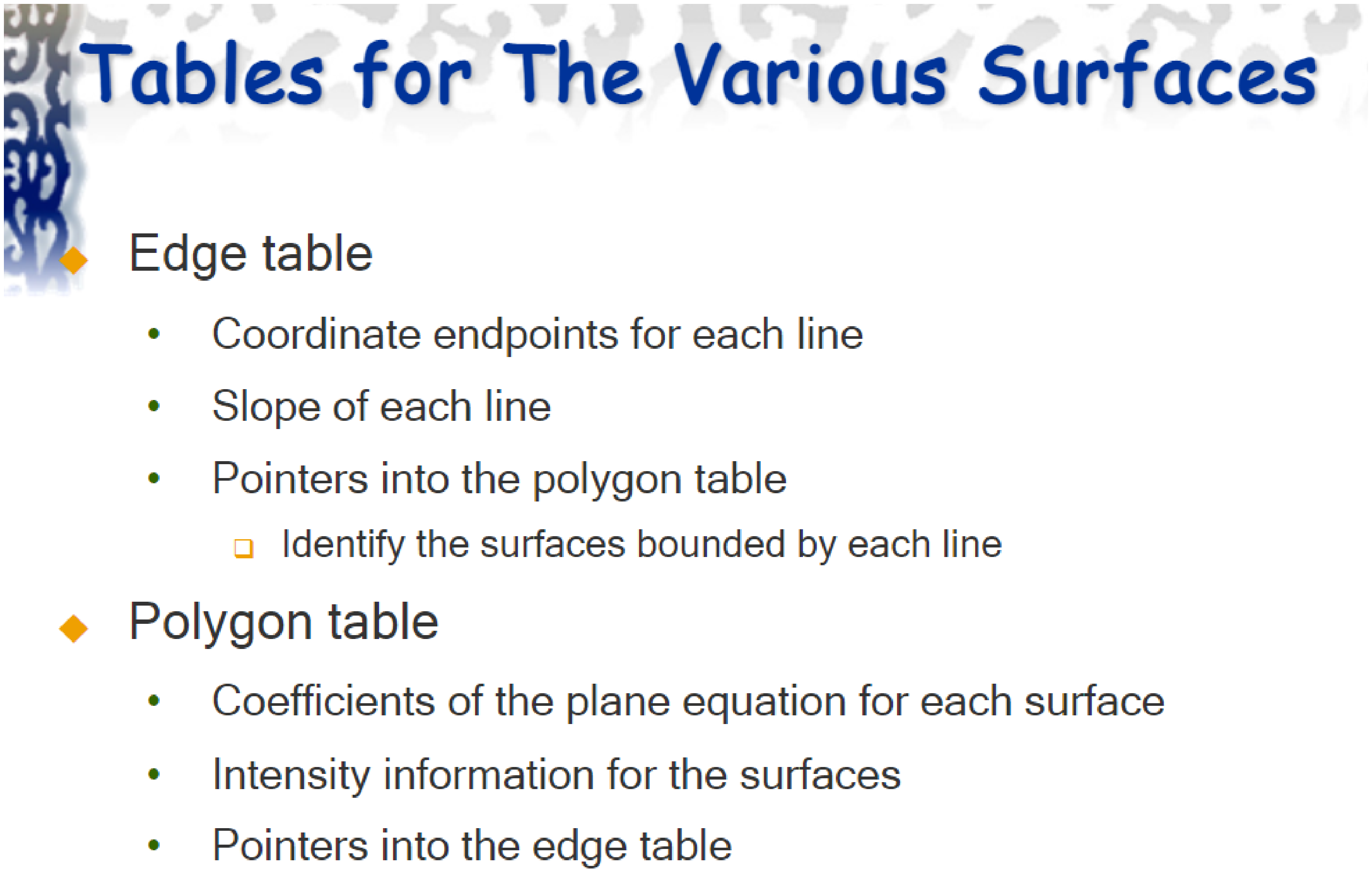
Scan line Algorithm



Characteristics

- Extension of the scan-line algorithm for filling polygon interiors
  - For all polygons intersecting each scan line
    - Processed from left to right
    - Depth calculations for each overlapping surface
    - The intensity of the nearest position is entered into the refresh buffer

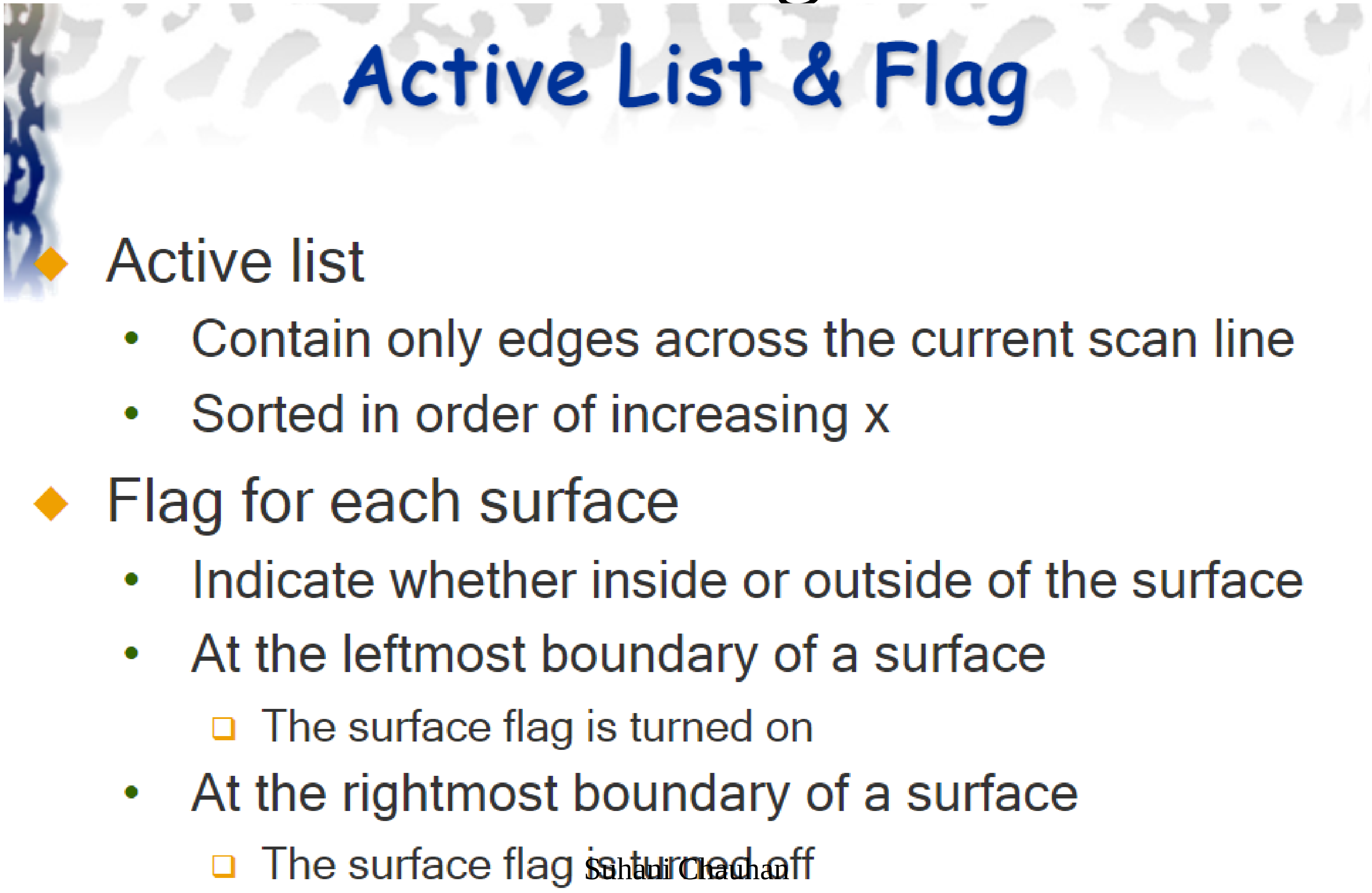
Scan line Algorithm



Tables for The Various Surfaces

- Edge table
  - Coordinate endpoints for each line
  - Slope of each line
  - Pointers into the polygon table
    - Identify the surfaces bounded by each line
- Polygon table
  - Coefficients of the plane equation for each surface
  - Intensity information for the surfaces
  - Pointers into the edge table

Scan line Algorithm



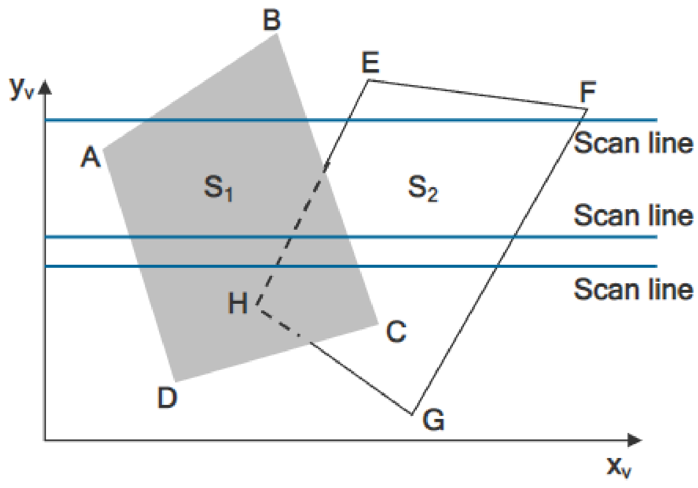
Active List & Flag

- Active list
  - Contain only edges across the current scan line
  - Sorted in order of increasing x
- Flag for each surface
  - Indicate whether inside or outside of the surface
  - At the leftmost boundary of a surface
    - The surface flag is turned on
  - At the rightmost boundary of a surface
    - The surface flag is turned off

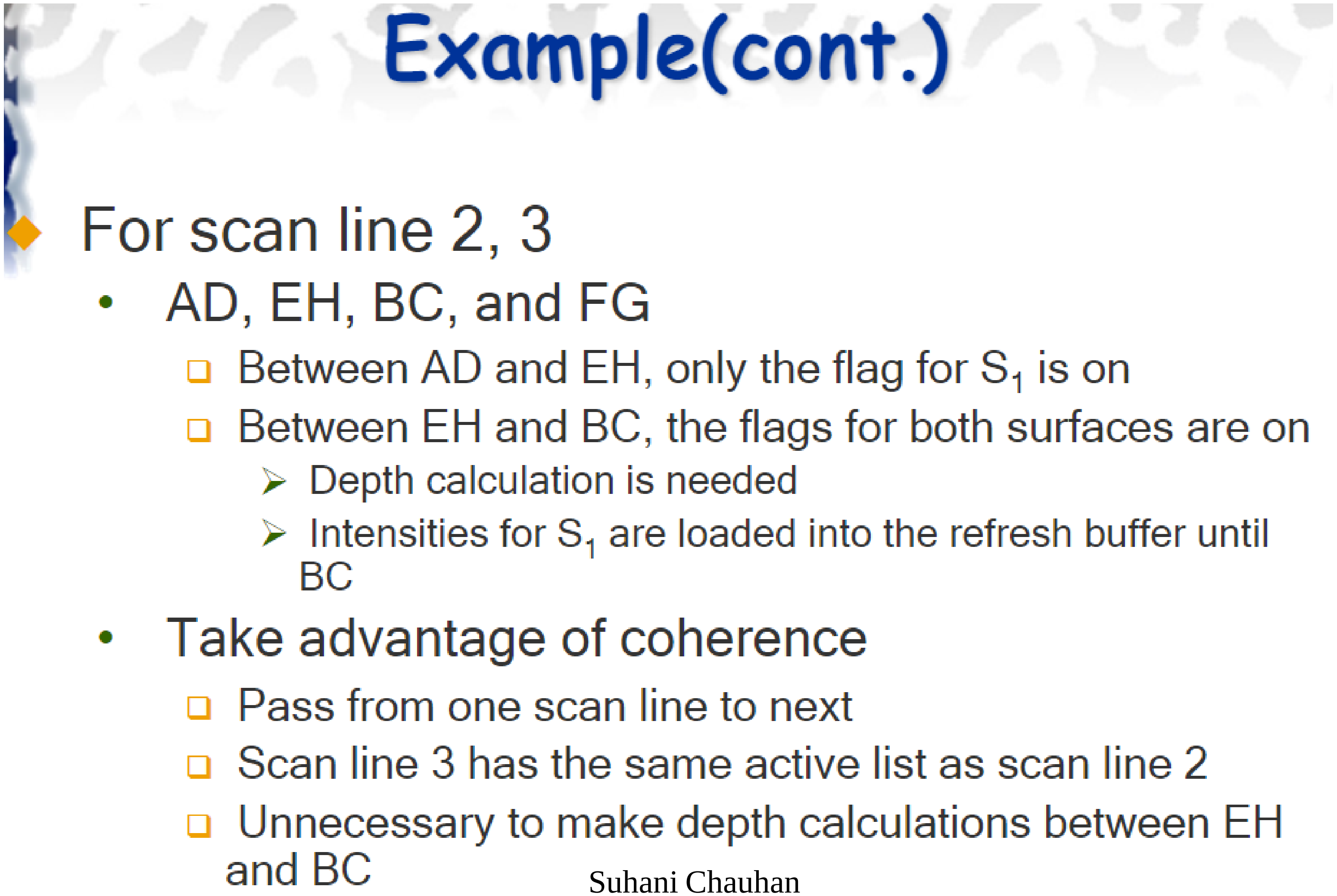
Scan line Algorithm

Example

- Active list for scan line 1
  - Edge table
    - AB, BC, EH, and FG
    - Between AB and BC, only the flag for surface S_1 is on
      - No depth calculations are necessary
      - Intensity for surface S_1 is entered into the refresh buffer
    - Similarly, between EH and FG, only the flag for S_2 is on



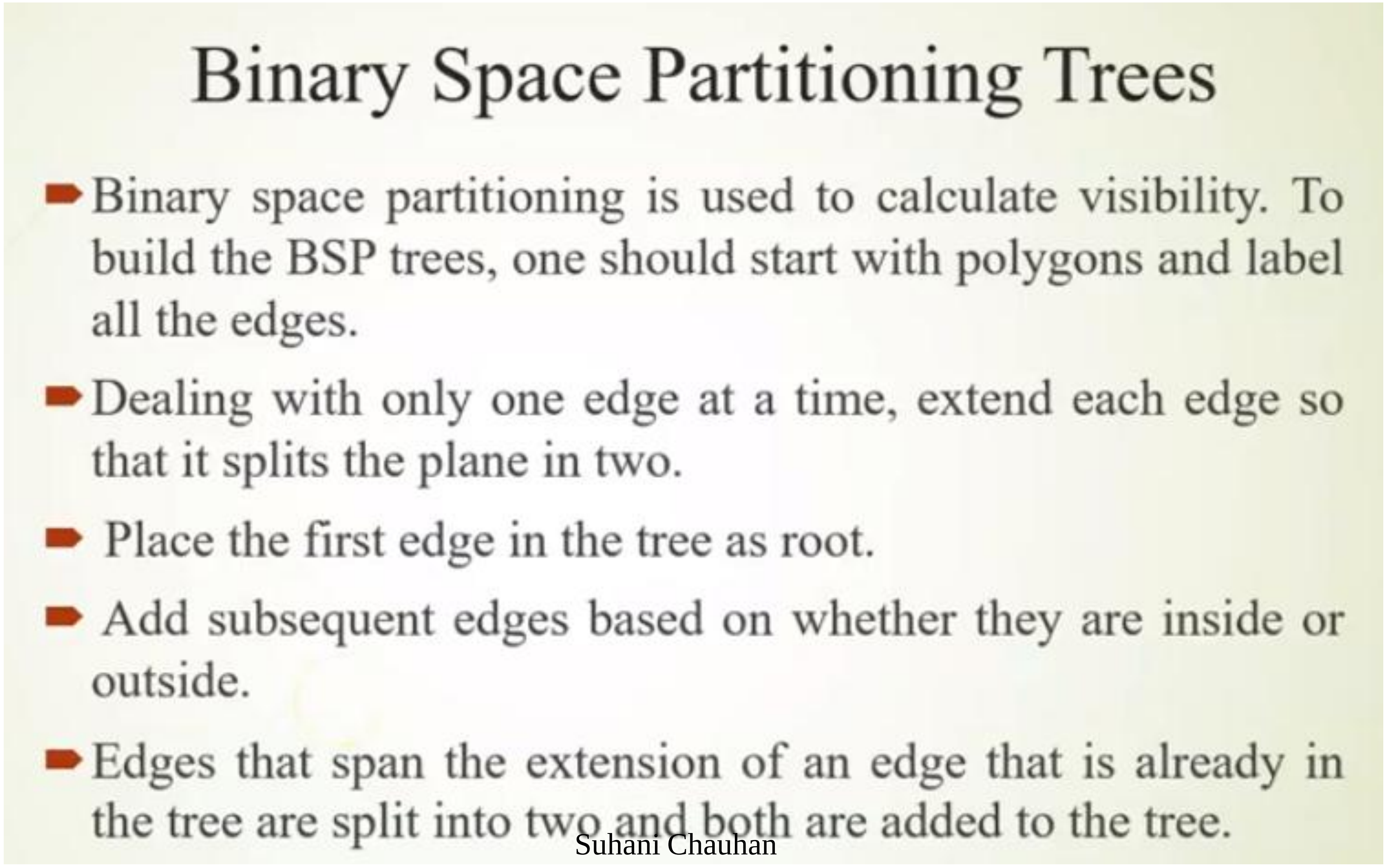
Scan line Algorithm



Example(cont.)

- For scan line 2, 3
  - AD, EH, BC, and FG
    - Between AD and EH, only the flag for S_1 is on
    - Between EH and BC, the flags for both surfaces are on
      - Depth calculation is needed
      - Intensities for S_1 are loaded into the refresh buffer until BC
  - Take advantage of coherence
    - Pass from one scan line to next
    - Scan line 3 has the same active list as scan line 2
    - Unnecessary to make depth calculations between EH and BC

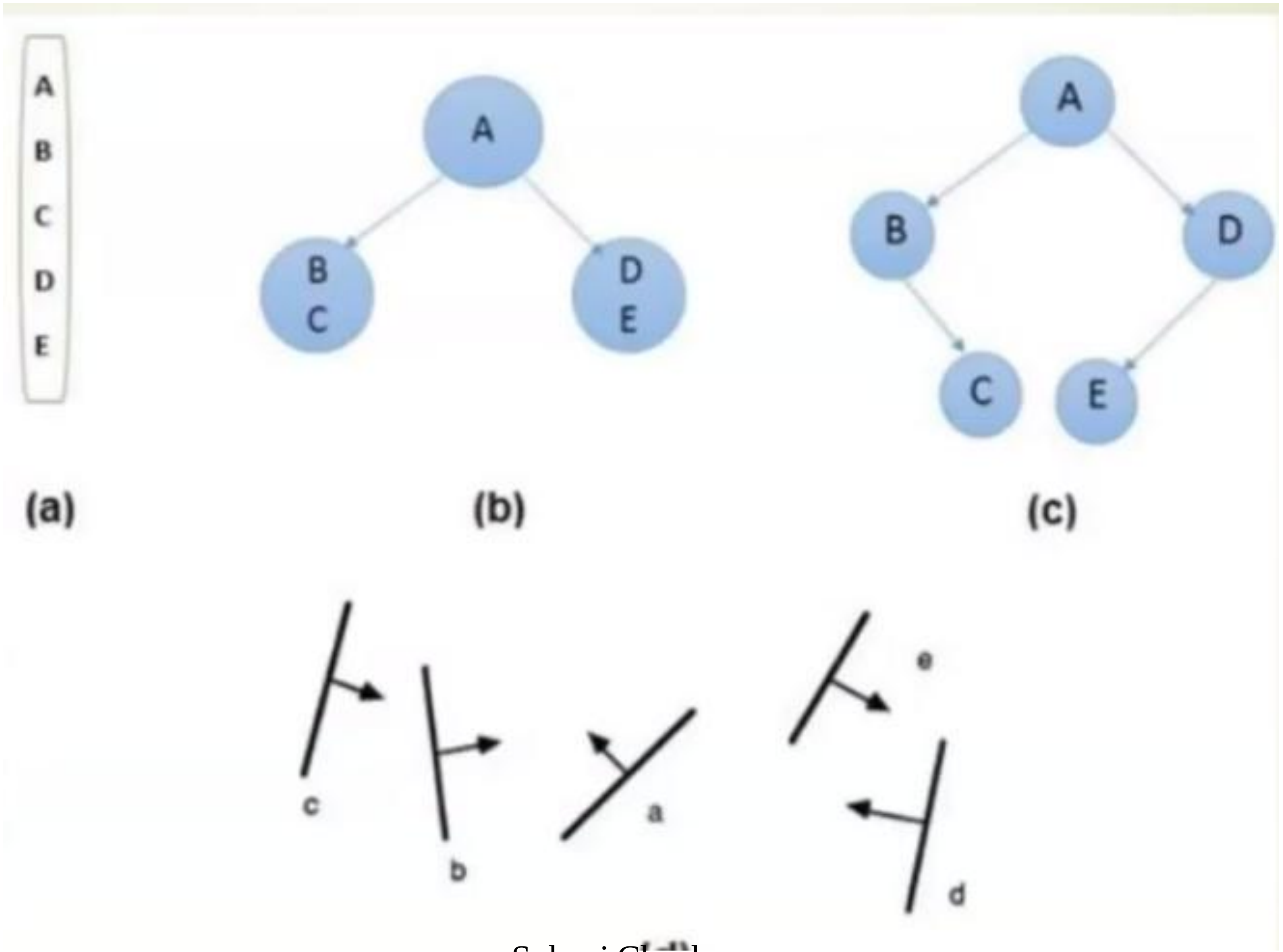
BSP Method



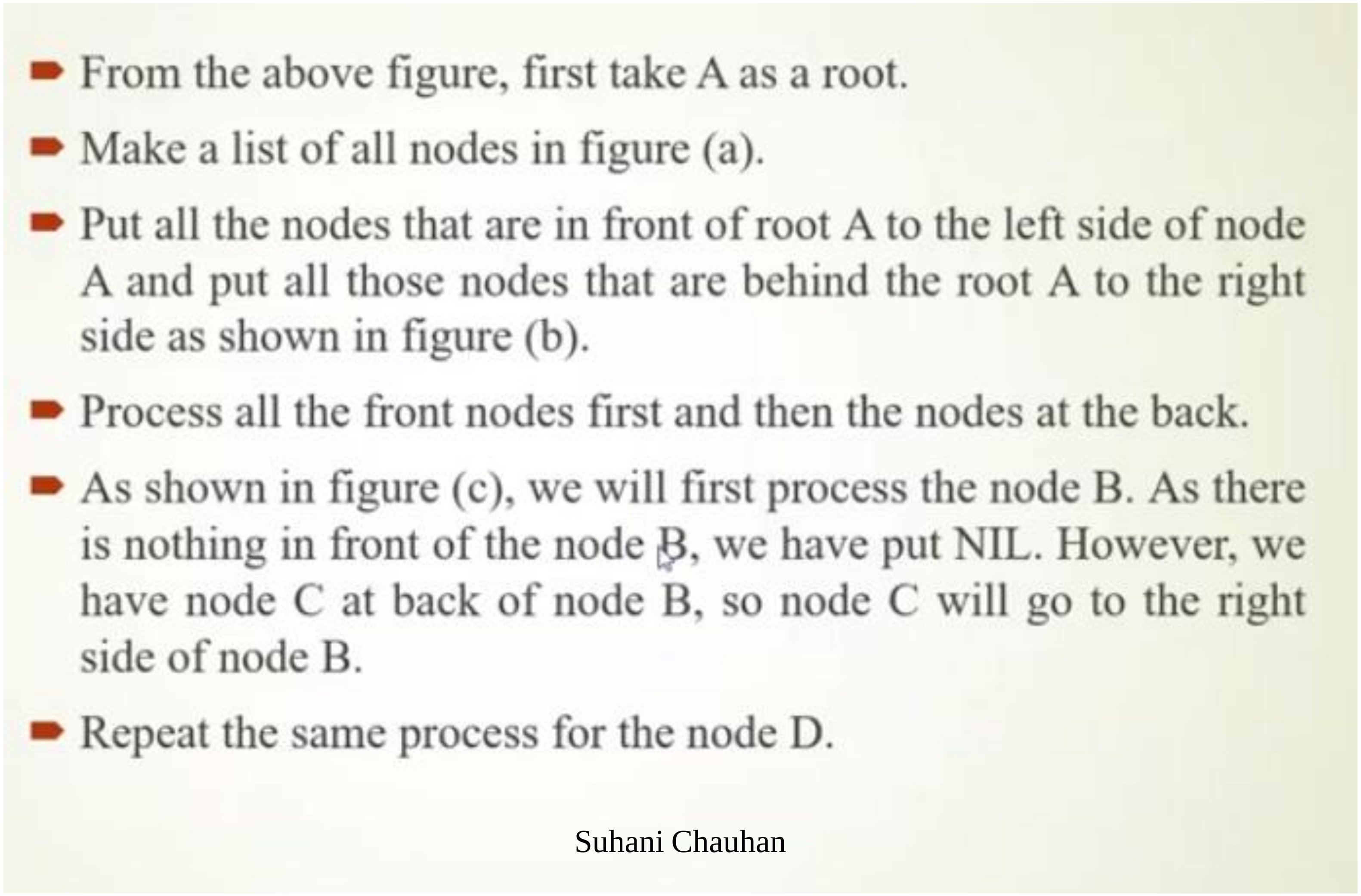
Binary Space Partitioning Trees

- Binary space partitioning is used to calculate visibility. To build the BSP trees, one should start with polygons and label all the edges.
- Dealing with only one edge at a time, extend each edge so that it splits the plane in two.
- Place the first edge in the tree as root.
- Add subsequent edges based on whether they are inside or outside.
- Edges that span the extension of an edge that is already in the tree are split into two and both are added to the tree.

BSP Example



BSP Example



- From the above figure, first take A as a root.
- Make a list of all nodes in figure (a).
- Put all the nodes that are in front of root A to the left side of node A and put all those nodes that are behind the root A to the right side as shown in figure (b).
- Process all the front nodes first and then the nodes at the back.
- As shown in figure (c), we will first process the node B. As there is nothing in front of the node B, we have put NIL. However, we have node C at back of node B, so node C will go to the right side of node B.
- Repeat the same process for the node D.