Graphics Rendering Pipeline II

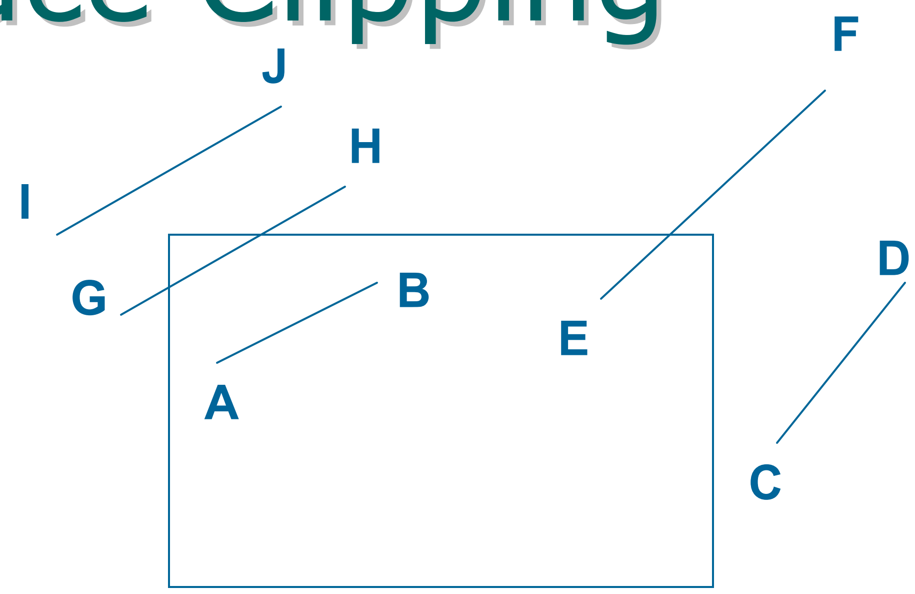
CS 211A

Clipping

- OpenGL does image space clipping
- Culling
  - Usually refers to object space
  - Done by the application programmer

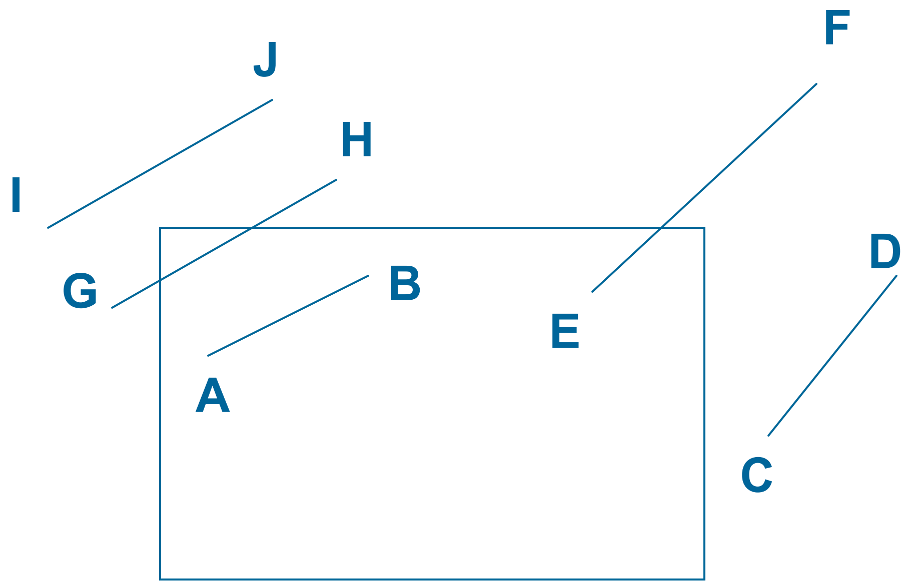
Image Space Clipping

- Accept AB
- Reject (Cull) CD
- Clip EF
  - One endpoint outside the window
- Clip GH
  - Both endpoints outside the window



Efficiency

- How fast you can accept and reject?
- Do intersection computations minimally

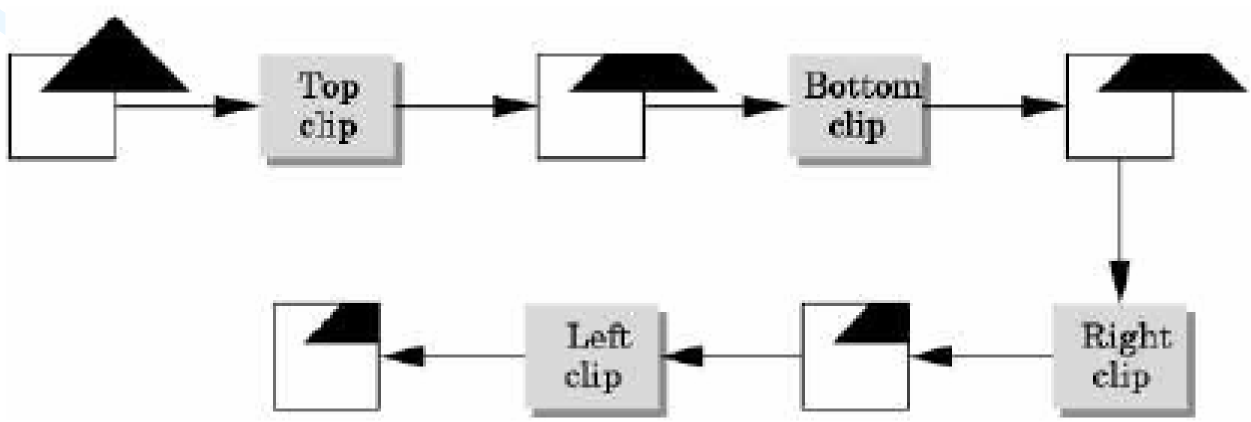


Many methods

- Cohen-Sutherland
  - Accept or reject based on binary codes associated with each vertex
- Liang-Barsky
  - Clip based on ordering of parametric intersections
  - Done only by integer comparisons

Polygon Clipping

- Concave polygon
  - Change it to convex
- Sutherland Hodgeman
  - Pipeline based method
  - Each stage is clipping against one of the sides

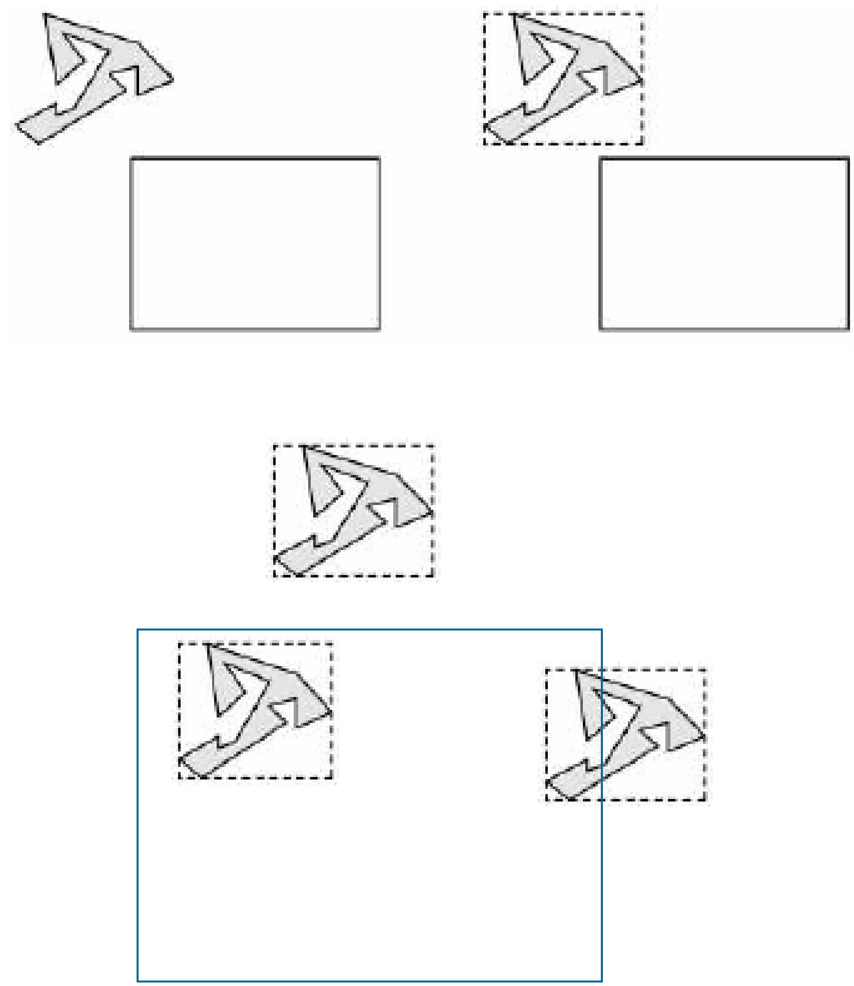


Spatial Subdivision

- Can be used for both image space and object space culling
- Based on bounding boxes or volumes

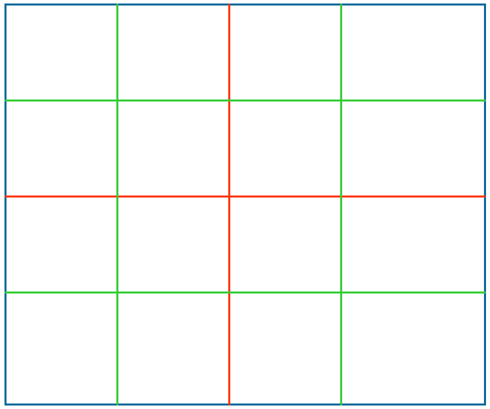
Bounding Boxes and Volumes

- Polygon clipping is overkill if entire polygon outside the window
- Maintain a bounding box
  - Axis-aligned
- Can be a big savings
- Can be easily extended to 3D
  - For volumes in object-space



Hierarchical Spatial Subdivision (2D)

- Quadtree
  - Each node corresponds to a BB
  - It holds the indices of all primitives in that box
  - Divide each box into four equal sized box
    - Four children per node
    - Can be computed from BB of parent
    - BB stored only at root



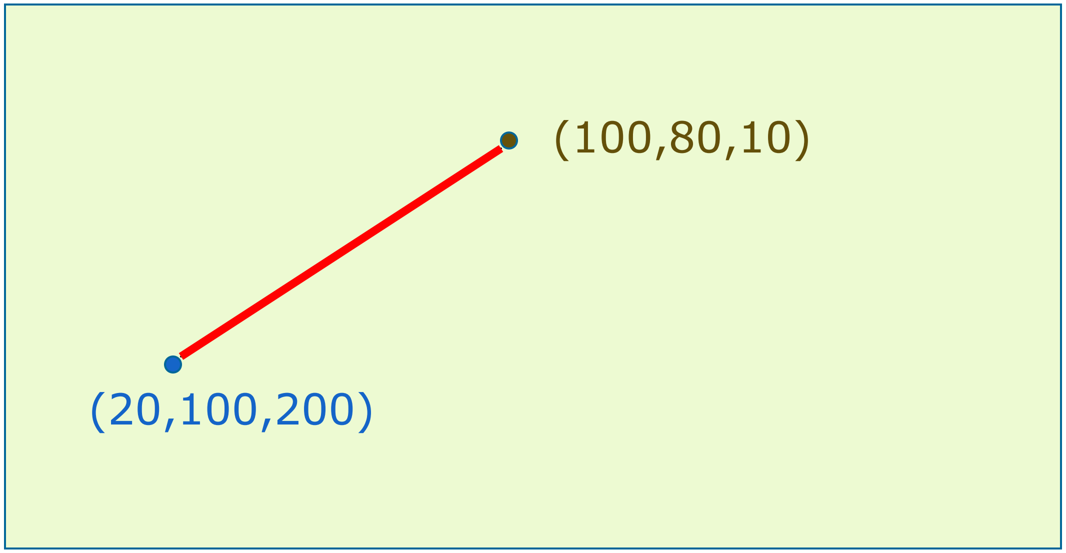
Hierarchical Spatial Subdivision (2D)

- Tree building
- Culling the Model
  - Depth first traversal of nodes
  - If BB inside the view frustum
    - Draw all triangles
  - If BB outside the view frustum
    - Draw nothing
  - If BB intersects the view frustum
    - Go through the children recursively
  - Creates tree cuts

Extending to 3D

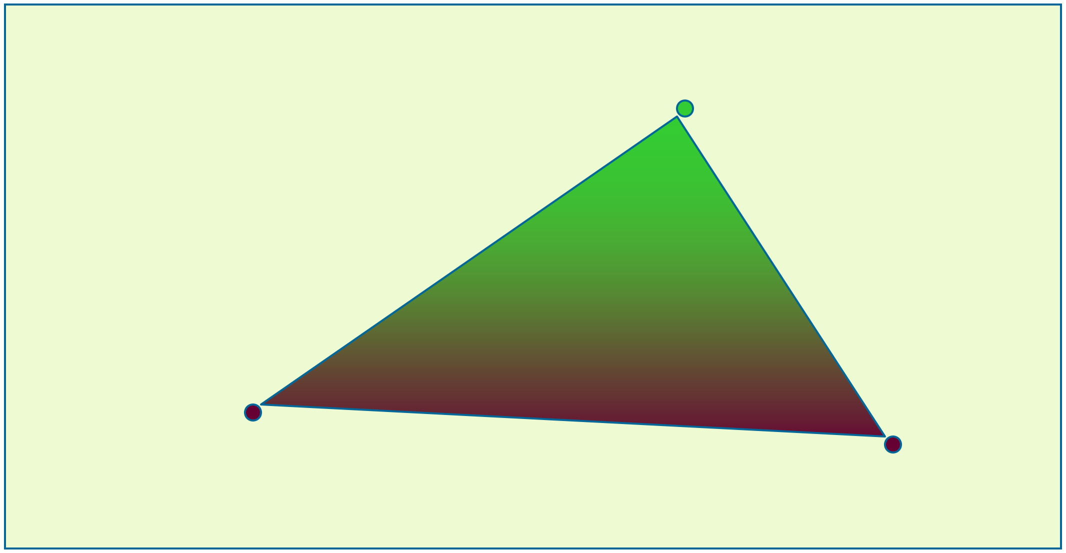
- Cubes instead of boxes
- Octree
  - Eight children
  - Divide in three directions
- Note that may not be optimal
  - Boxes may not be the tightest fit
  - Can have another tree with smaller depth
- Very efficient
  - Since child BB computation is trivial

Scan Conversion



- Which pixels to color?
- What color to put for each pixel?

Scan Conversion



- Which pixels to color?
- What color to put for each pixel?

Which pixels

- Efficient Data Structures
- Integer Operations are preferred
- Hardware adaptability
- Line
  - Bresenham’s
- Polygon
  - Using an edge table and active edge table data structure

How to color them?

- Linear interpolation
- Find the coefficients from the marked pixels
  - Screen space interpolation
- Use these linear coefficients to find a weighted combination of color
- Is screen space interpolation correct?
  - Not really, but we are not sensitive to it

Hidden Surface Removal

- Z buffer (size of the framebuffer)
- Initialize
- Store z when projecting vertices
- During scan conversion
  - Interpolate 1/z
  - If depth is smaller than existing value
    - Set new depth
    - Color pixel

Final Drawing

```
Transform all vertices;
Clear frame buffer;
Clear depth buffer;
for i=1:n triangles
    for all pixels (x_s,y_s) in the triangle
        pixelz = 1/z interpolated from vertex;
        if (pixelz < depthbuffer[x_s][y_s])
            framebuffer[x_s][y_s]= color interpolated
              from vertex attributes;
        endif;
    endfor;
endfor;
```

Back Face Culling

- Do not want to render back facing polygons
- If the normal is pointed towards the viewer
  - $-90 \leq \theta \leq 90$
  - $\cos(\theta) \geq 0$
  - $n.v \geq 0$
- Viewing in -z
  - Culled if normal has negative z