Visibility Culling and Graphical Pipeline Revisited

LESSON 10

Computer Graphics 1

Drawing Wireframe Models

Drawing Wireframe Models

- Apply visibility test to edges
- Discard or draw differently the occluded edges
- Exploit previous algorithms
  - Draw boundary and delete interior
- Better solution
  - Front edges (2 front faces)
  - Back edges (2 back faces)
  - Contour edges (back and front face)

Drawing Wireframe Models

- Back edges
  - Invisible, discard
- Front edges and contour edges
  - Potentially visible
  - Detect and draw only visible parts
- Roberts algorithm
  - Clip potentially visible edges by faces
- Apell algorithm
  - Clip potentially visible edges by contour edges

Visibility Culling

Backface culling, view frustum culling, occlusion culling

Visibility Culling

- Culling – removing triangles from computation
- Visibility culling – culling triangles for the purpose of rendering
- Remove unseen triangles from computation
- Less triangles = faster computation
- Fastest polygon to render is the one that is never sent to renderer

Visibility Culling

- Exact visible set (EVS)
  - All primitives that are partially or fully visible
  - Ideal output of culling
- Potentially visible set (PVS)
  - Primitives that might be visible
- Conservative culling

$$EVS \subseteq PVS$$

- Approximate (aggressive) culling

$$EVS \not\subset PVS$$

Visibility Culling

- Conservative culling
  - Always generate correct images
- Approximate culling
  - Generates incorrect images
  - Minimizing the error
  - Fast computation

Backface Culling

- Every polygon has a front and back face
- Discard backfacing polygons
- Application: closed surfaces
- Determine the angle between viewing direction and polygon normal
- Angle < 90 degrees – discard polygon

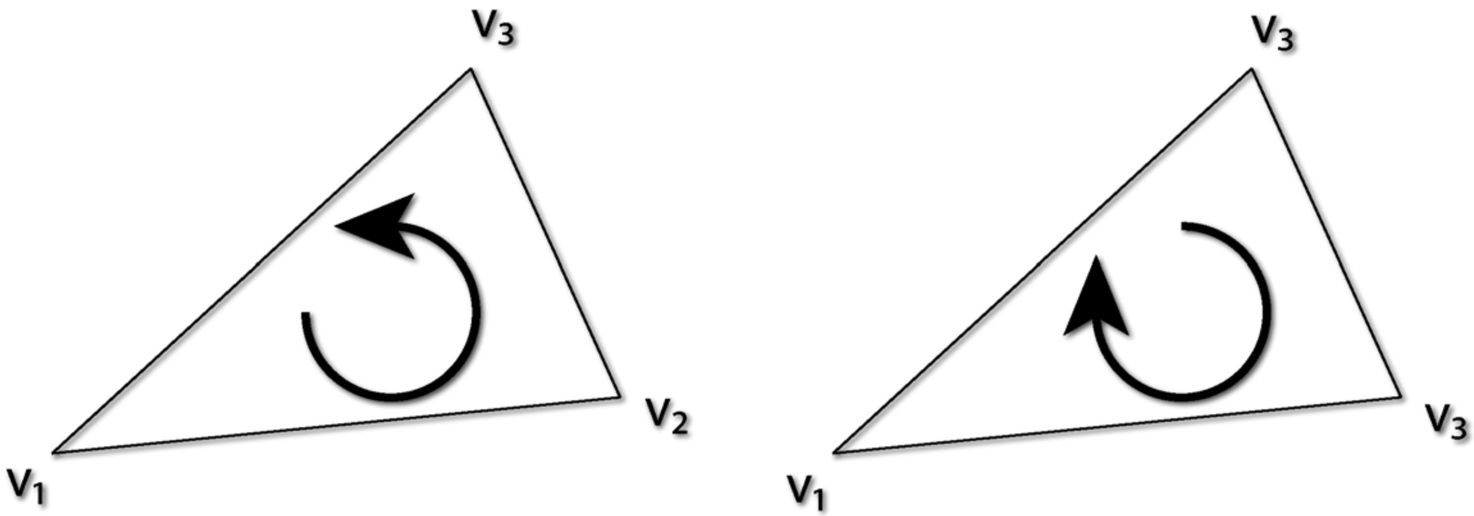
  $$\vec{n} \cdot \vec{v} > 0$$

- Angle > 90 degrees – reserve polygon

  $$\vec{n} \cdot \vec{v} < 0$$

Backface Culling

- Orientation specified by the order of vertices



- Compute normal: $n = (v_2 - v_1) \times (v_3 - v_1)$

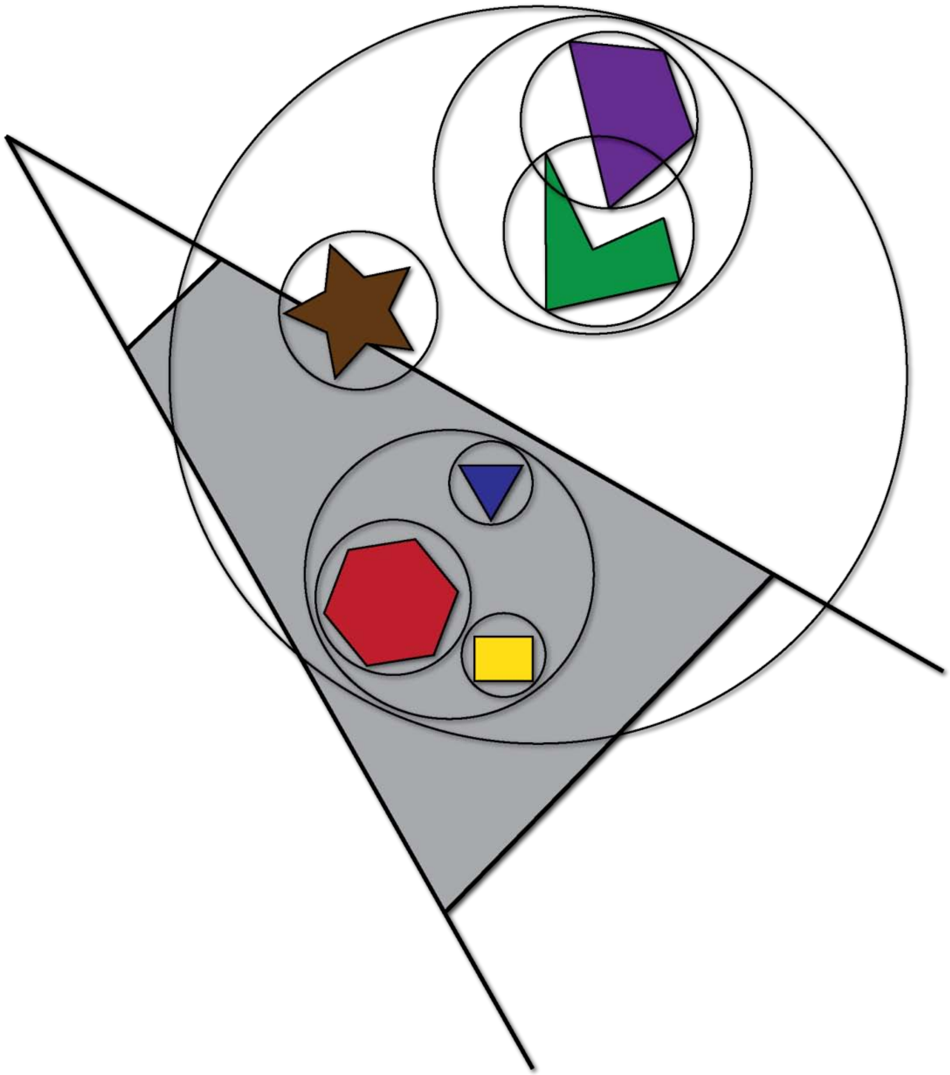
Backface Culling – Conclusion

- Simple algorithm
- Can reduce many polygons
- Suitable for scenes were a lot backfacing polygons appear
  - Very common situation
- Ineffective for terrains or rooms
  - Only few backfacing polygons
- Standard part of graphical APIs (OpenGL, DirectX)
- Need to specify faces which should not be culled

View Frustum Culling

- Draw only objects in view volume
- Clip against cut pyramid
- Clip all objects against clipping edges - O(n)
- Hierarchical culling
  - Hierarchically subdivide space (e. g. Octree, BVH)
  - O (log n)
- Test only bounding volumes
  - Discard if entirely outside view frustum

View Frustum Culling

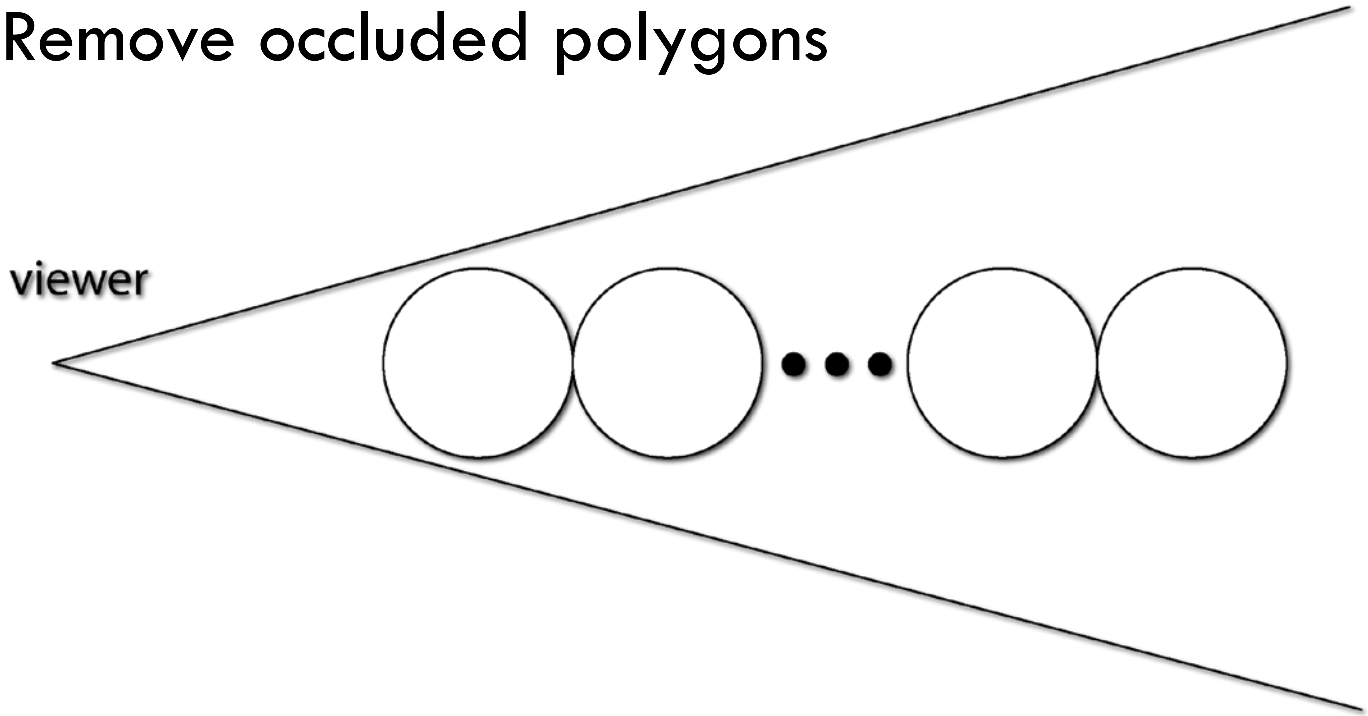


Detail Culling

- Sacrifice quality for speed
- Small detail contribute nothing or very little to the rendered image
- Cull if area of object projection is below a threshold
  - Usually a number of pixels
- Sometimes called screen-size culling
- Usually used by movement of the viewer

Occlusion Culling

- Back-face culling and view-frustum culling can not reduce enough polygons for today games
- Solution: occlusion culling
- Remove occluded polygons



Portal Culling

- Suitable for architectural models
- Walls are often large occluders
- Portal
  - door, window, …
  - Connecting adjacent rooms
- View frustum culling through each portal
- Preprocessing
  - Automated preprocessing Extremely difficult for complex scene
  - Currently done by hand

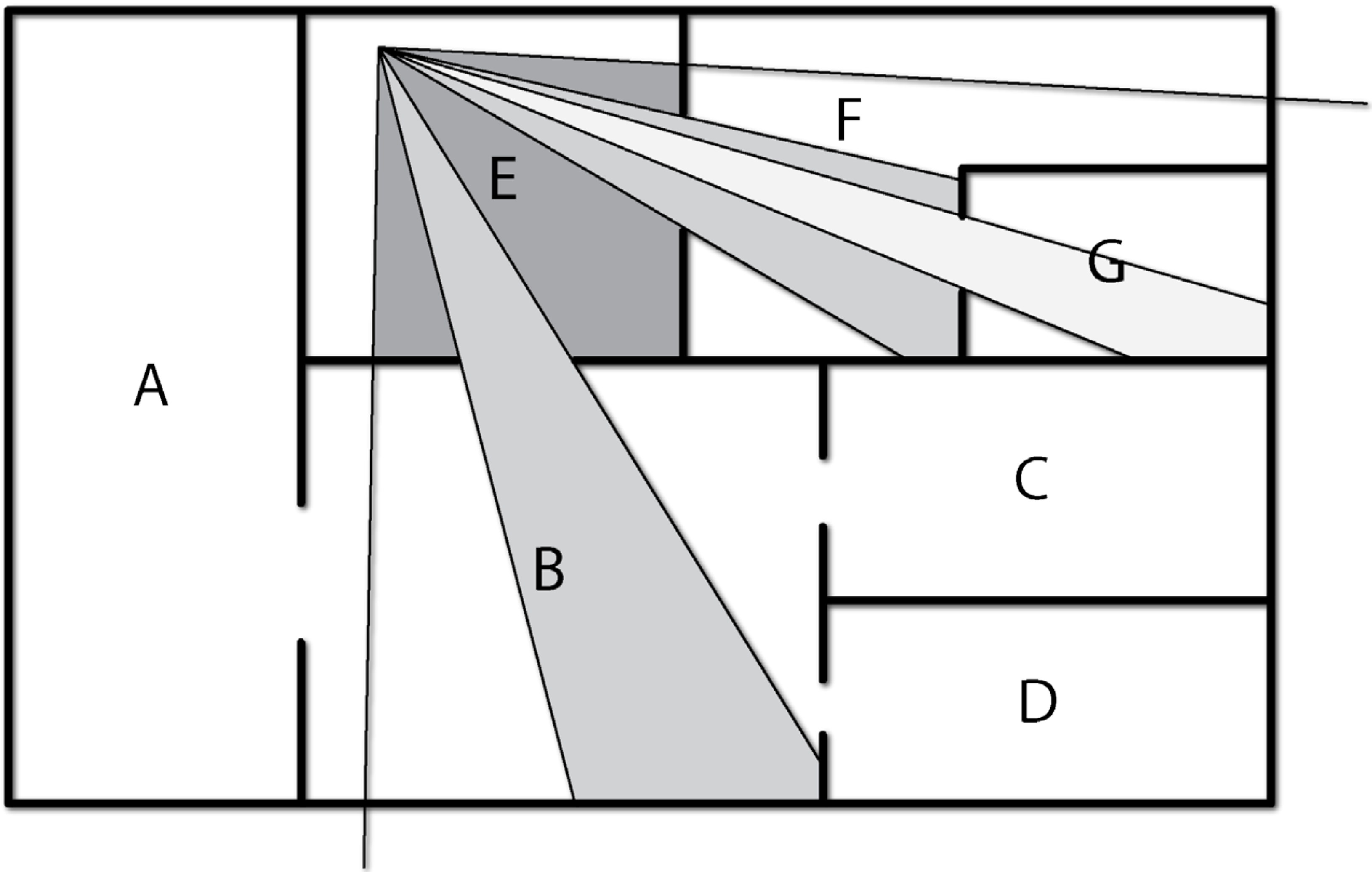
Portal Culling – Algorithm

- 1. locate cell V where the viewer is positioned
- 2. initialize 2D bounding box P to the rectangle of the screen
- 3. render the geometry of the cell V
  - Use view frustum culling
  - Frustum emanates from viewer and goes through P

Portal Culling – Algorithm

- 4. recurse on portals of the cells neighboring V
  - Project each visible portal of the current cell onto the screen
  - Find 2D axis-aligned BB of the projection
  - Compute intersection of and the BB
- 5. for each intersection
  - Empty intersection – not visible, omit from processing
  - Nonempty intersection – resurse to step 3
    - V – neighboring cell
    - P – intersection BB

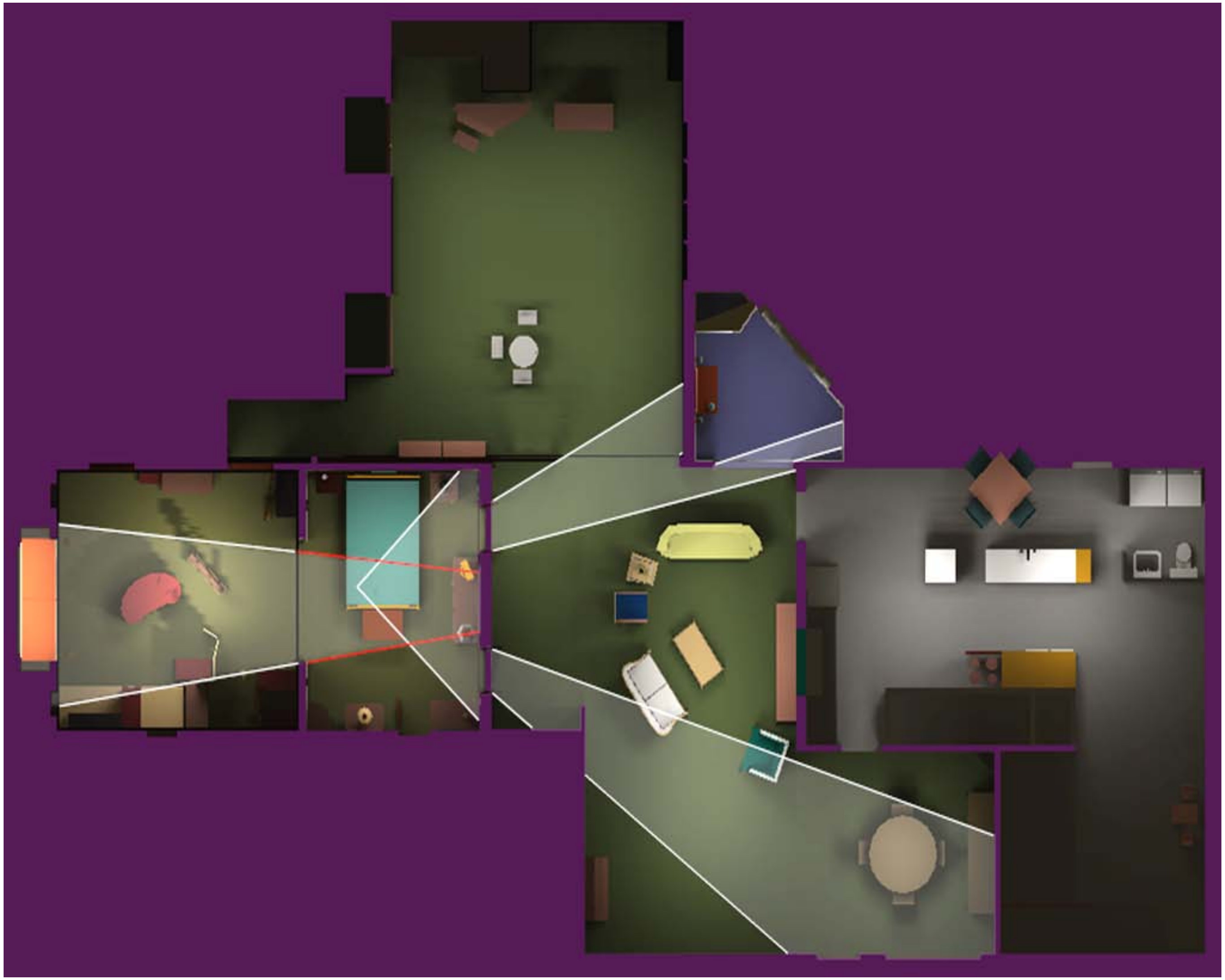
Portal Culling - Example



Portals and Mirrors

David Luebke
Chris Georges

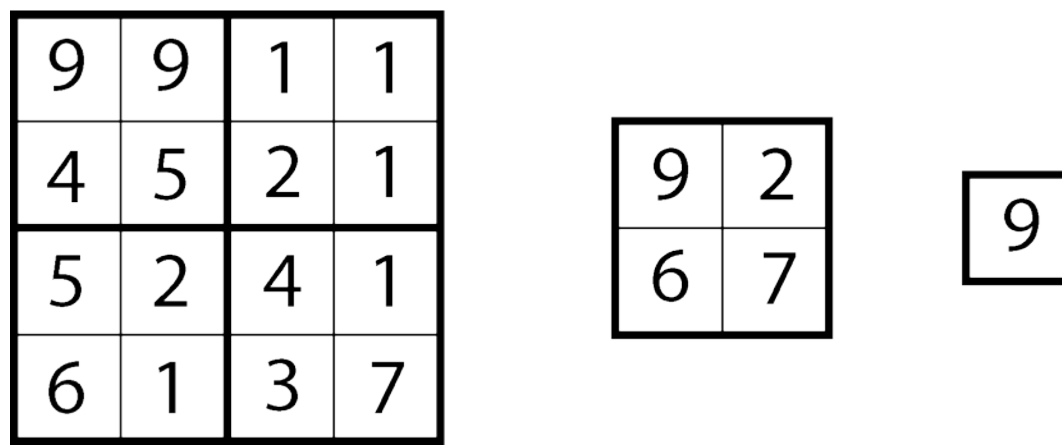
Portals and Mirrors



David Luebke
Chris Georges

Hierarchical Z-Buffering

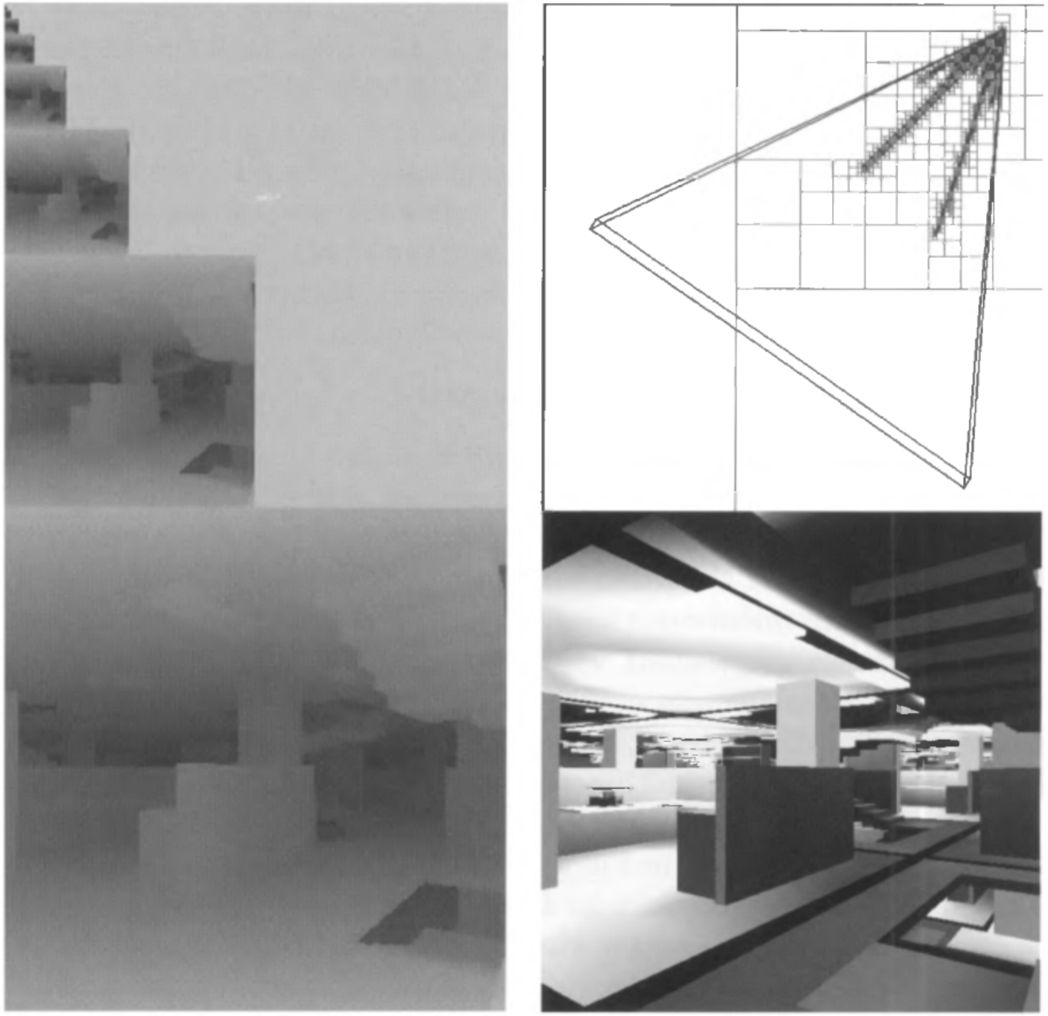
- Scene in octree
- Z-buffer
  - Image pyramid (Z-pyramid)
  - Occlusion representation of the scene
  - Each z-value represents the farthest z-value of the window
  - Overwrite z-value recursively



Hierarchical Z-Buffering

- Hierarchical culling of octree nodes
- Traverse in front-to-back order
- Compare the z-pyramid with the screen projection
  - Z-pyramid cell encloses the octree cell
  - Compare the smallest depth within the cell (z_{near})
  - If z_{near} is larger than the value in z-pyramid the cell is occluded
- Continue recursively down the z-pyramid until
  - Cell is found to be occluded
  - Bottom level of the z-pyramid is reached – cell is visible

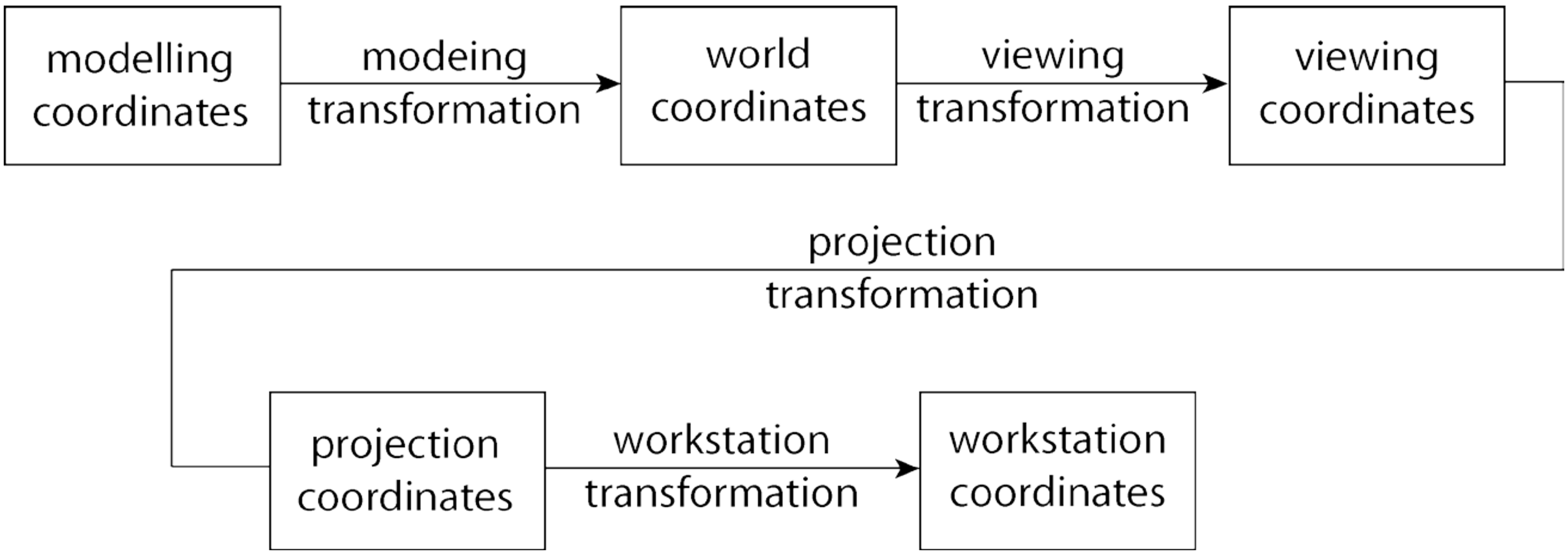
Hierarchical Z-Buffering



25

Graphical Pipeline (Revisited)

Graphical Pipeline



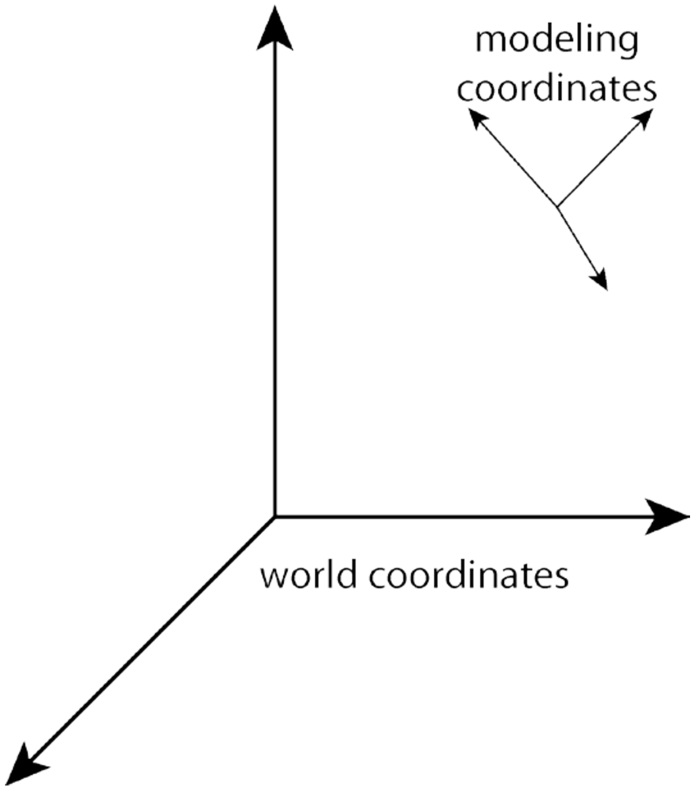
Modeling Coordinates

- Local coordinates
- Specific for every object
- Simplify modeling of object
  - Make the representation easier

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \qquad \frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} = 1$$

World Coordinates

- Specify position of object
- User defines object with respect to this coordinates

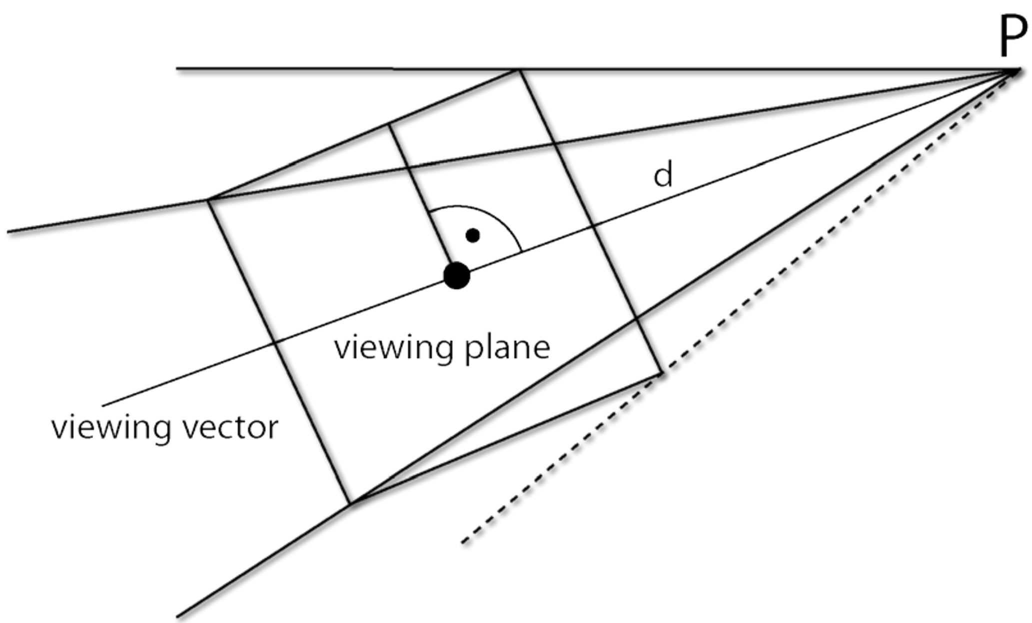


Viewing Coordinates

- Camera coordinates
- Analogy to pinhole camera
- Specified by:
  - Camera position (vector or view at point)
  - Viewing direction
  - View plane (distance from the camera)
  - Upward vector

Viewing Coordinates – View Plane

- Viewing plane
  - Perpendicular to the viewing vector
  - Specified by the distance from the camera
  - In front of the camera



Viewing Coordinates - Computation

- v – viewing direction
- w – upward vector
- It is difficult to define upward vector parallel to the viewing plane
- Solution
  - Project arbitrary vector onto the viewing plane

$$\vec{w}_u = \vec{w} + c\vec{n}$$

$$0 = \vec{w}_u \cdot \vec{n} = \vec{w} \cdot \vec{n} + c\vec{n} \cdot \vec{n} \qquad \vec{n} \cdot \vec{n} = \|\vec{n}\| = 1$$

$$c = -\vec{w} \cdot \vec{n}$$

$$\vec{w}_u = \vec{w} - (\vec{w} \cdot \vec{n})\vec{n}$$

Viewing Coordinates - Computation

- Origin p = camera position
- Upward vector
  - User specified
  - Projection of a vector from the base
- Coordinate system (u_1, u_2 , u_3)

$$u_3 = \frac{v}{\|v\|}$$

$$u_2 = \frac{w_u}{\|w_u\|}, \qquad w_u = w - (w \cdot u_3)u_3$$

$$u_1 = u_3 \times u_2$$

Viewing Transformation

- World coordinates to viewing coordinates

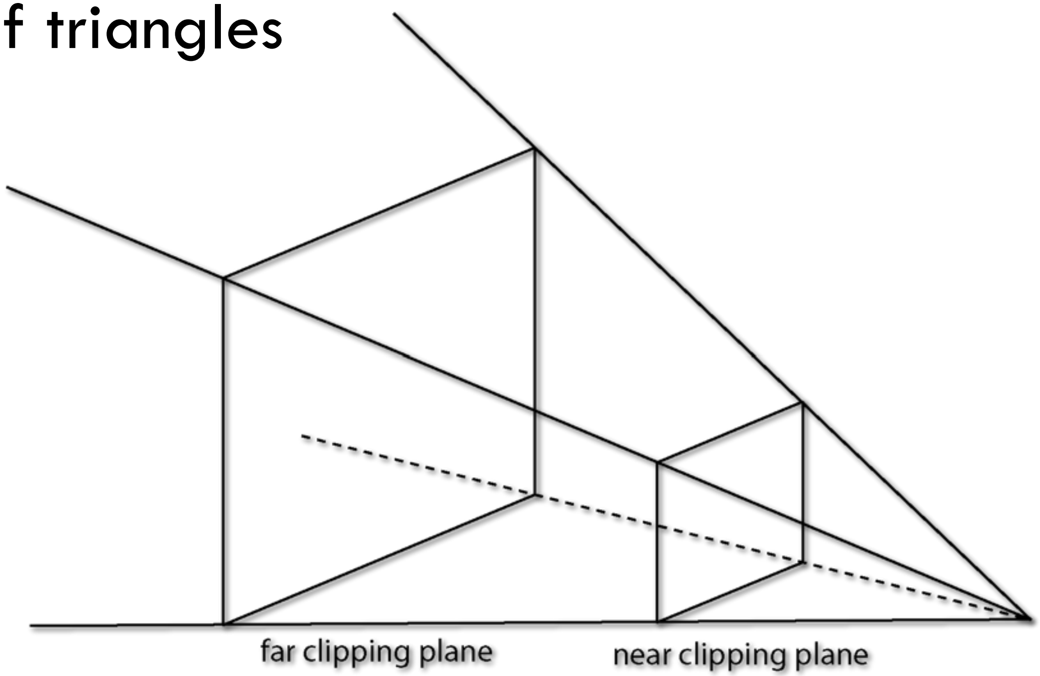
$$q \rightarrow (q - p)\left(u_1^T u_2^T u_3^T\right) = (q - p)M$$

$$u_1 = (u_{11}, u_{12}, u_{13})$$
$$u_2 = (u_{21}, u_{22}, u_{23})$$
$$u_3 = (u_{31}, u_{32}, u_{33})$$

$$M = \begin{pmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{pmatrix}$$

Clipping

- View frustum clipping
- Far and near clipping planes
  - Limiting visibility
  - Limiting number of triangles



Projection Coordinates

- Visible space = unit cube
- Simple to clip against a unit cube
  - Simple equations of clipping planes
- Clipping algorithm is independent of boundary dimensions
- Clipping in homogenous coordinates

Projective Transformation

- See lesson 4
- Transform clipped frustum to cube
  - Scale, translate, T_{persp}

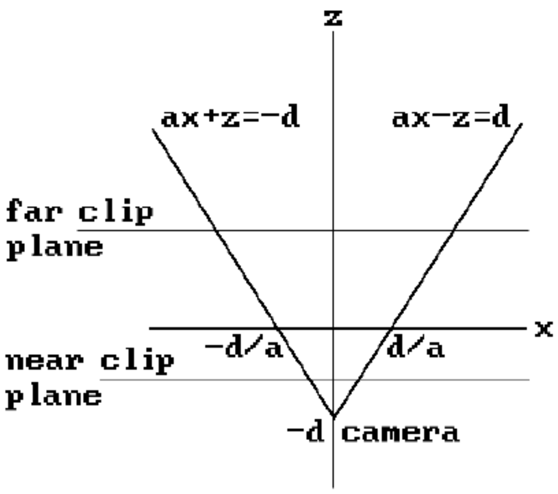
$$T_{persp} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

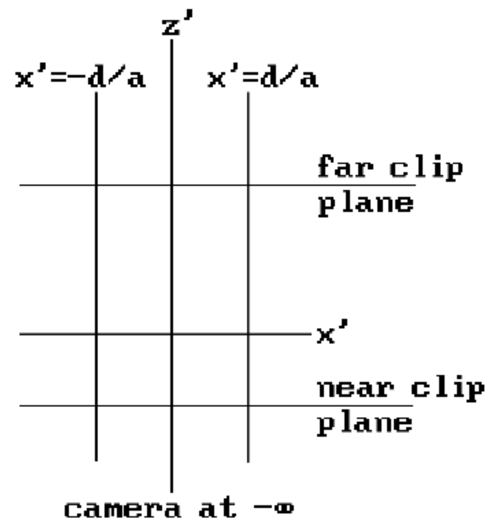
Projective Transformation

$$(0,0,-d,1)T_{persp} = (0,0,-d,0)$$

$$(x,0,-d-ax,1)T_{persp} = \left(x,0,-d-ax,-\frac{ax}{d}\right) = -\frac{d}{ax}\left(-\frac{d}{a},0,d+\frac{d^2}{ax},1\right)$$

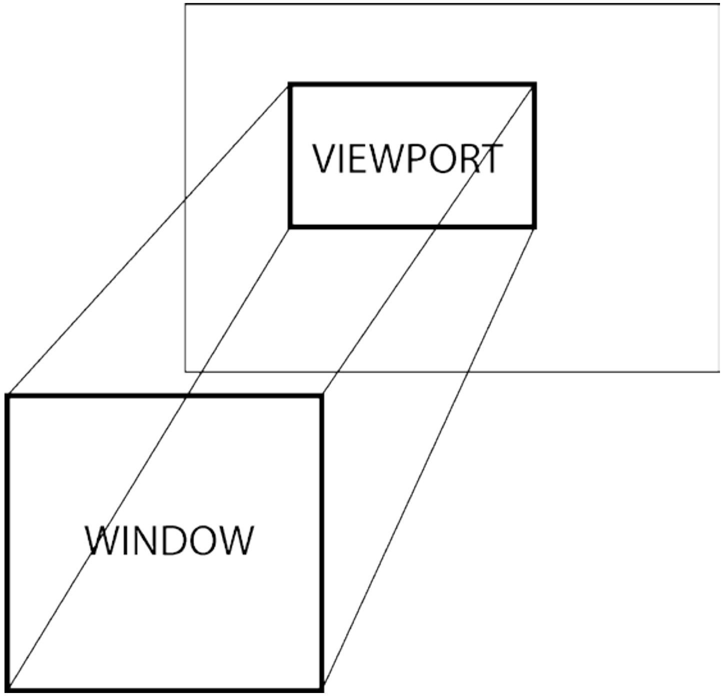
$$(x,0,-d+ax,1)T_{persp} = \left(x,0,-d+ax,\frac{ax}{d}\right) = \frac{d}{ax}\left(\frac{d}{a},0,d-\frac{d^2}{ax},1\right)$$





Workstation Transformation

- From homogenous to Euclidian
- Parallel projection along z axis
- Scale and transform in order to map to viewport



Graphical Pipeline - Conclusion

- Lighting and shadows
  - Global coordinates
- Clipping
  - Projection coordinates
- Visibility
  - Depends on algorithm
  - Image space – workstation coordinates or projection coordinates
  - Object space – viewing coordinates, world coordinates

2D Graphical Pipeline

- Similar to 3D pipeline
- Viewing transformation
  - Make window axis aligned
- Projection coordinates (normalized coordinates)
  - Window is square with side of length = 1
  - Separate modeling from displaying
- Clipping
  - Viewing coordinates
  - World coordinates – join viewing and projection (normalization) coordinates

41

Questions ???