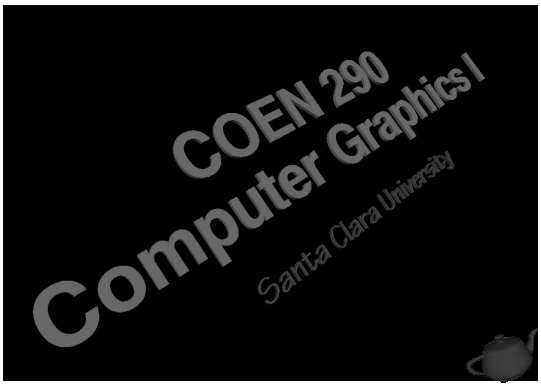


Evening's Goals

- Discuss application bottleneck determination
- Discuss various optimizations for making programs execute faster
  - 2D
  - 3D

2D Applications

- "Scroller" games
- Photo / Image viewing software
- *Photoshop*-type applications

How We've Rendered So Far …

❶ Clear the window
❷ Redraw all contents using new positions
❸ Swap Buffers
❹ Go to ❶

- Forced to redraw every pixel in the viewport
  - Potentially very wasteful for certain types of applications

"Dirty Rectangles"

- For 2D applications, clearing the entire window may be wasteful
- Only update pixels that were "damaged"
  - Refill where players were with background image
  - Use *Painter's Algorithm* to determine drawing order

"Dirty Rectangles"

What Makes Applications Slow

- Poor data structure use in application
- Too much geometry per frame
  - Vertex transformations
  - Lighting computations
  - Texture coordinate generation
- Too many pixels to fill
  - Depth testing
  - Texture mapping

Keeping things "Real Time"

- What's your hardware capable of doing?
  - Polygons / second
  - Pixels / second
- Determining what's realistic
  - Computing the *polygon budget*
    - Controlling model complexity
  - Estimating how many pixels need to be filled per frame
    - Need to consider rejected pixels from depth testing

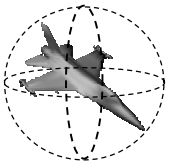
Polygon Budget

- Need to consider
  - How many frames you want a second (Hertz)
  - How many geometric primitives do you need to render per frame

$$\frac{\text{polygons}}{\text{frame}} = \frac{polygons/\sec}{frames/\sec}$$

Computing Bounding Volumes

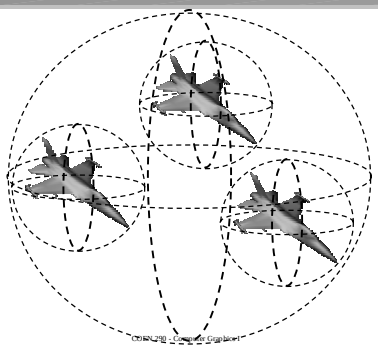
- Bounding sphere
  - Easy to compute
  - Easy to work with
  - Not the most accurate representation



$$\vec{c} = \tfrac{1}{n}\sum_{i=1}^{n} \vec{p}_i$$

$$r = \sqrt{(x_{\max} - x_{\min})^2 + (y_{\max} - y_{\min})^2 + (z_{\max} - z_{\min})^2}$$

Hierarchical Bounding Volumes

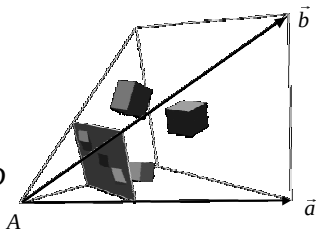


View Frustum Culling

$$\hat{n} = \frac{\vec{a} \times \vec{b}}{\left\| \vec{a} \times \vec{b} \right\|}$$

$$D = -A \bullet \hat{n}$$

$$Plane_i = \hat{n} \bullet \vec{p} + D$$



Needs to be computed in either eye space or world coordinates

Portal Culling

- Determine what geometry's visible from a certain vantage point
  - Useful for interior scenes: rooms, hallways, etc.
- For each room
  - For each wall (and optionally floor and ceiling)
    - A list of what's visible when looking at each wall
    - Include what can be viewed through each *portal*
      - May be possible to depth sort these primitives so that they don't need to be depth buffered

Binary Space Partition Trees

- Store primitives in a binary tree based on their location to their parent
- Partition space based on a primitive's line or plane equation
- Use faced-ness of primitive to determine front and back sides