CSE 167:
Introduction to Computer Graphics
Lecture #10: View Frustum Culling

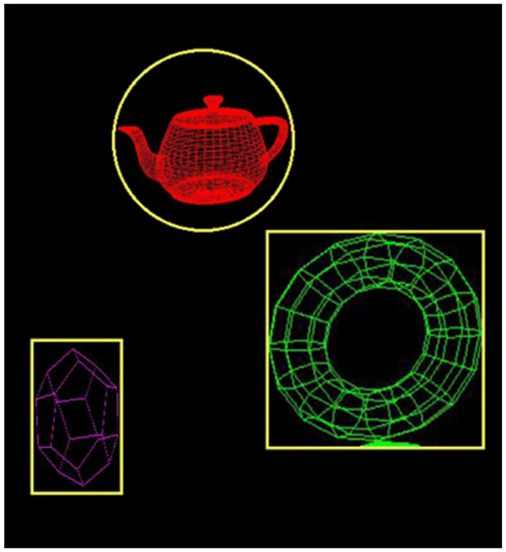
Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2015

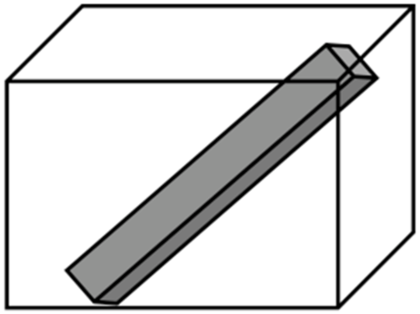
Announcements

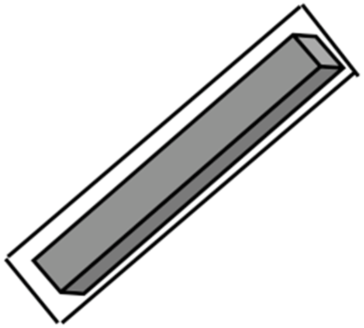
- Project 4 due tomorrow
- Project 5 discussion on Monday
- Midterm:
  - Problem 5 a): no point deduction if R not normalized

Bounding Volumes

- Simple shape that completely encloses an object
- Generally a box or sphere
- We use spheres
  - Easiest to work with
  - But hard to calculate tight fits
- Intersect bounding volume with view frustum instead of each primitive







UCSD

Bounding Box

- How to cull objects consisting of may polygons?
- Cull bounding box
  - Rectangular box, parallel to object space coordinate planes
  - Box is smallest box containing the entire object

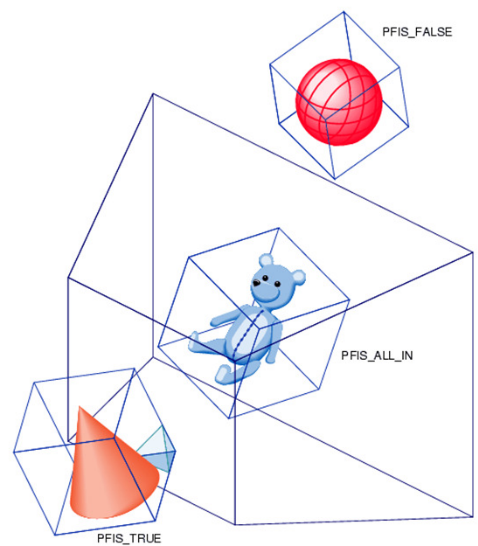
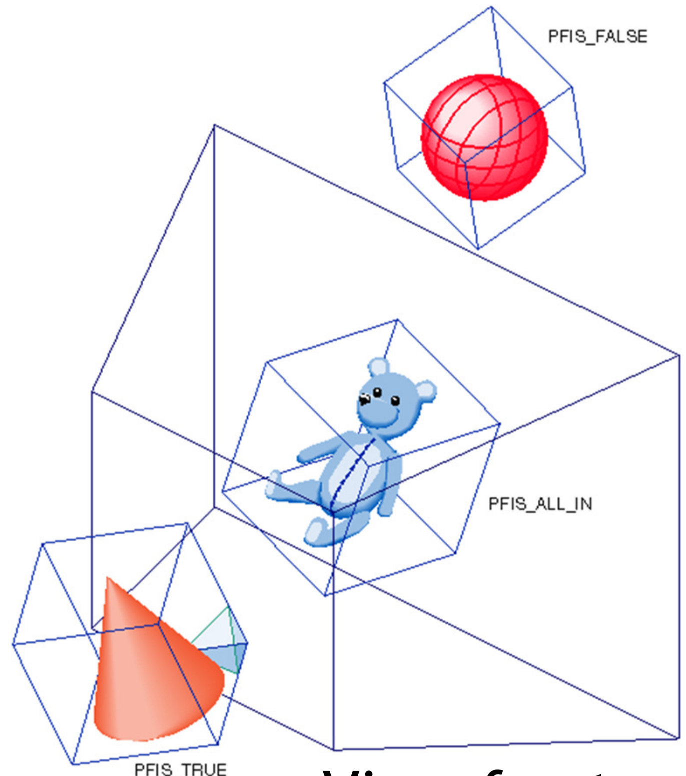


Image: SGI OpenGL Optimizer Programmer's Guide

View Frustum Culling

- Frustum defined by 6 planes
- Each plane divides space into "outside", "inside"
- Check each object against each plane
  - Outside, inside, intersecting
- If "outside" all planes
  - Outside the frustum
- If "inside" all planes
  - Inside the frustum
- Else partly inside and partly out
- Efficiency

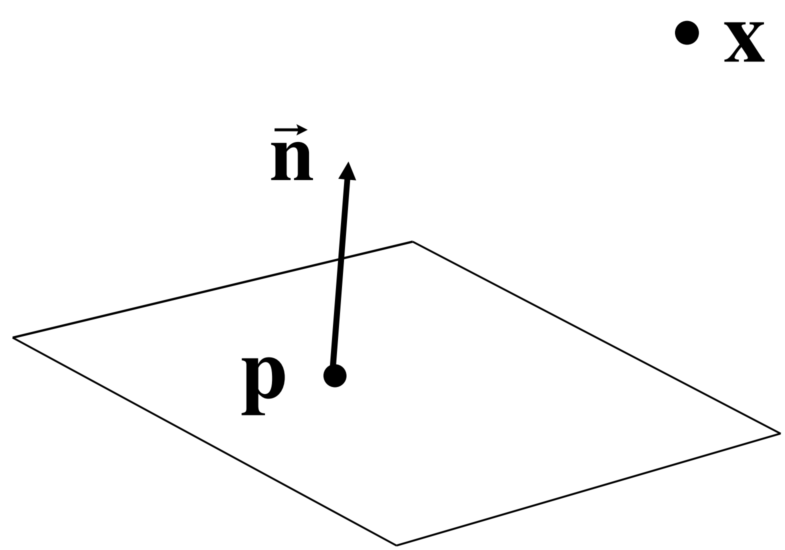


View frustum

UCSD

Distance to Plane

- A plane is described by a point **p** on the plane and a unit normal **n**
- Find the (perpendicular) distance from point **x** to the plane

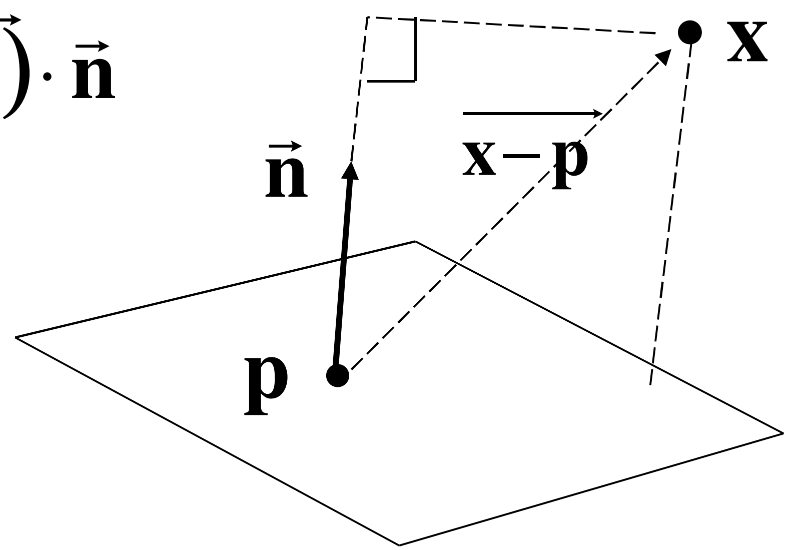


UCSD

Distance to Plane

- The distance is the length of the projection of **x-p** onto **n**

$$dist = \overrightarrow{(\mathbf{x} - \mathbf{p})} \cdot \vec{\mathbf{n}}$$

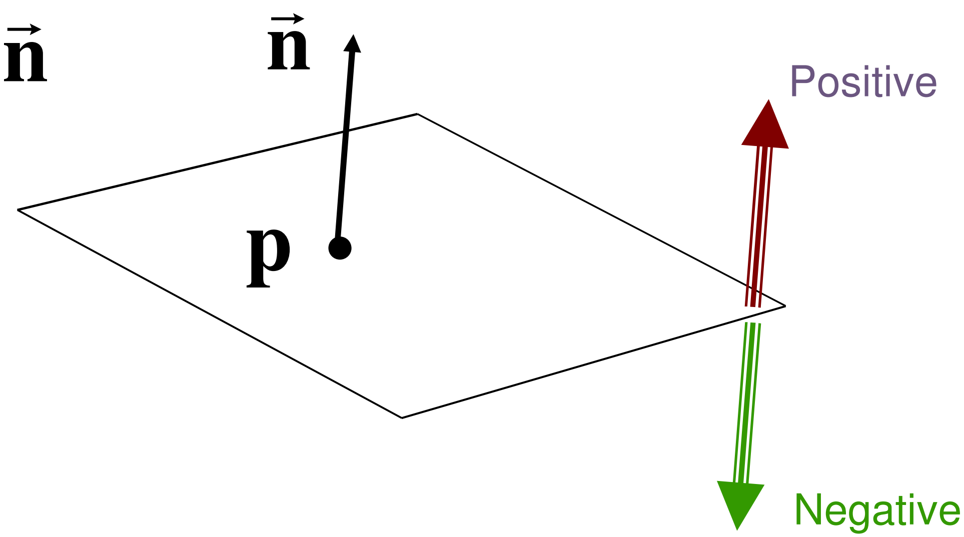


UCSD

Distance to Plane

- The distance has a sign
  - positive on the side of the plane the normal points to
  - negative on the opposite side
  - zero exactly on the plane
- Divides 3D space into two infinite half-spaces

$$dist(\mathbf{x}) = \overrightarrow{(\mathbf{x} - \mathbf{p})} \cdot \vec{\mathbf{n}}$$



UCSD

Distance to Plane

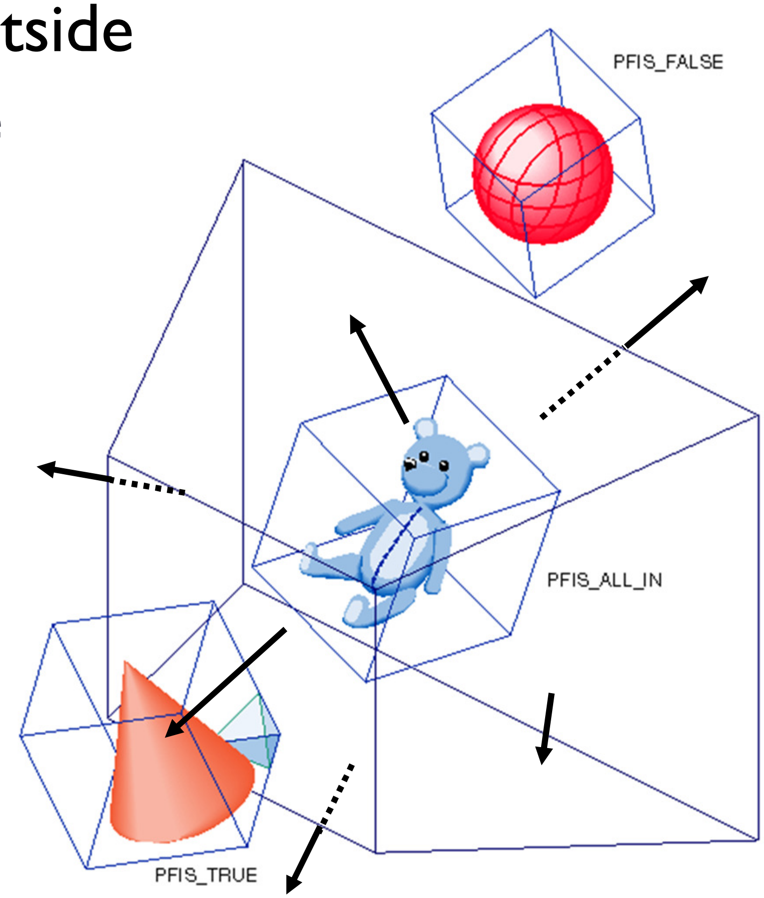
- Simplification

$$dist(\mathbf{x}) = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n}$$
$$= \mathbf{x} \cdot \mathbf{n} - \mathbf{p} \cdot \mathbf{n}$$
$$dist(\mathbf{x}) = \mathbf{x} \cdot \mathbf{n} - d, \quad d = \mathbf{pn}$$

- *d* is independent of **x**
- *d* is distance from the origin to the plane
- We can represent a plane with just *d* and **n**

Frustum With Signed Planes

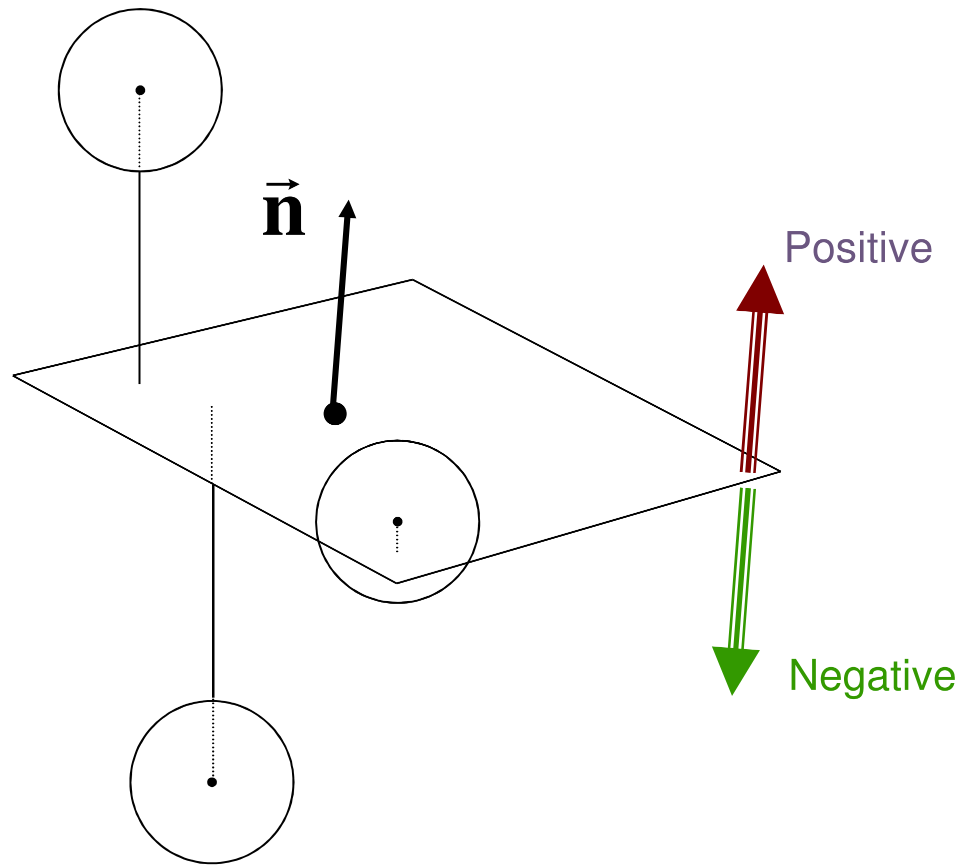
- Normal of each plane points outside
  - "outside" means positive distance
  - "inside" means negative distance



UCSD

Test Sphere and Plane

- For sphere with radius r and origin $\mathbf{x}$, test the distance to the origin, and see if it is beyond the radius
- Three cases:
  - $dist(\mathbf{x})>r$
    - completely above
  - $dist(\mathbf{x})<-r$
    - completely below
  - $-r<dist(\mathbf{x})<r$
    - intersects

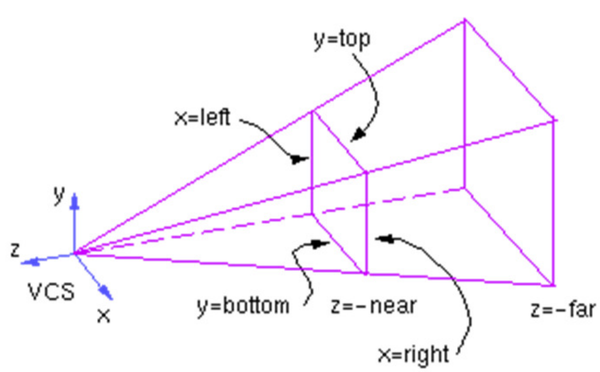


Culling Summary

- Pre-compute the normal $\mathbf{n}$ and value d for each of the six planes.
- Given a sphere with center $\mathbf{x}$ and radius r
- For each plane:
  - if $dist(\mathbf{x}) > r$: sphere is outside! (no need to continue loop)
  - add 1 to count if $dist(\mathbf{x}) < -r$
- If we made it through the loop, check the count:
  - if the count is 6, the sphere is completely inside
  - otherwise the sphere intersects the frustum
  - *(can use a flag instead of a count)*

Culling Groups of Objects

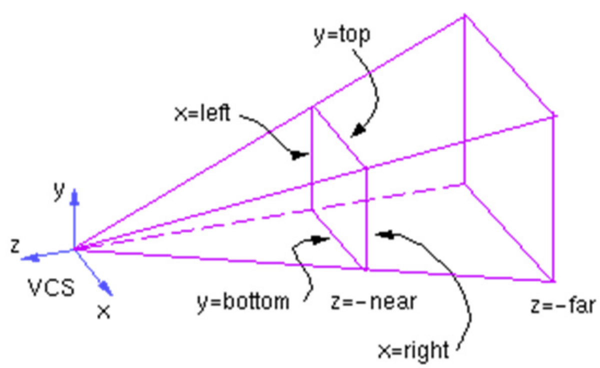
- Want to be able to cull the whole group quickly
- But if the group is partly in and partly out, want to be able to cull individual objects

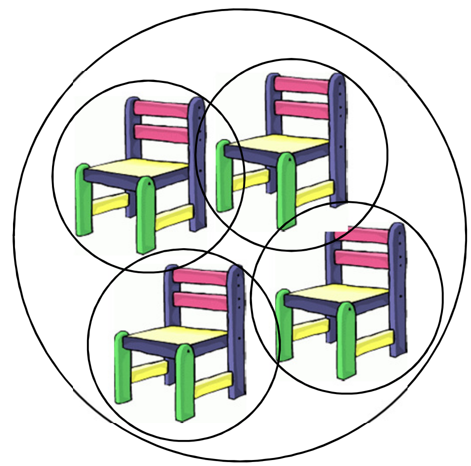


UCSD

Hierarchical Bounding Volumes

- Given hierarchy of objects
- Bounding volume of each node encloses the bounding volumes of all its children
- Start by testing the outermost bounding volume
  - If it is entirely outside, don't draw the group at all
  - If it is entirely inside, draw the whole group

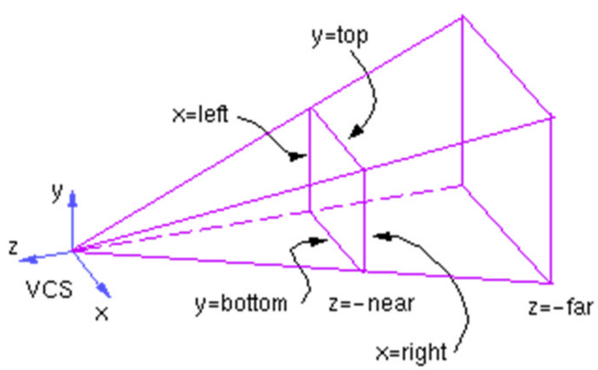


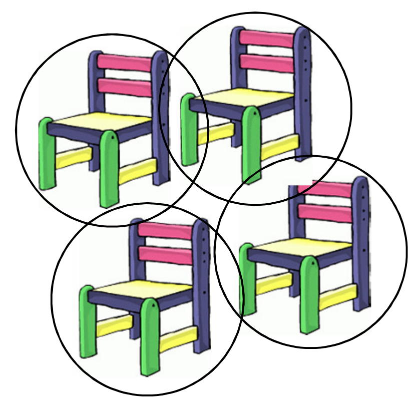


UCSD

Hierarchical Culling

- If the bounding volume is partly inside and partly outside
  - Test each child's bounding volume individually
  - If the child is in, draw it; if it's out cull it; if it's partly in and partly out, recurse.
  - If recursion reaches a leaf node, draw it normally

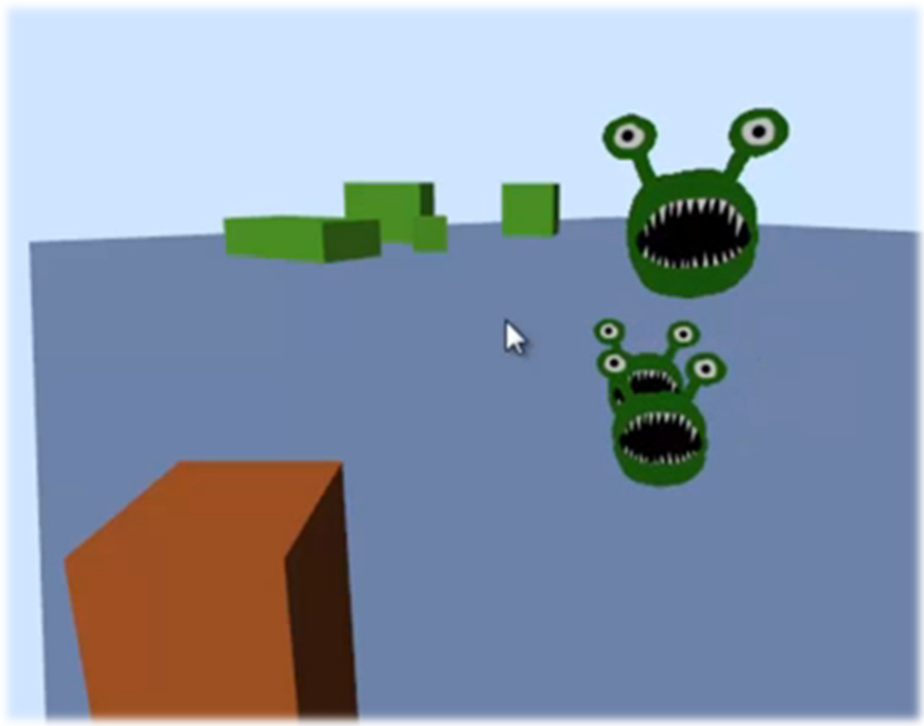




UCSD

Video

- Math for Game Developers - Frustum Culling
  - http://www.youtube.com/watch?v=4p-E_31XOPM



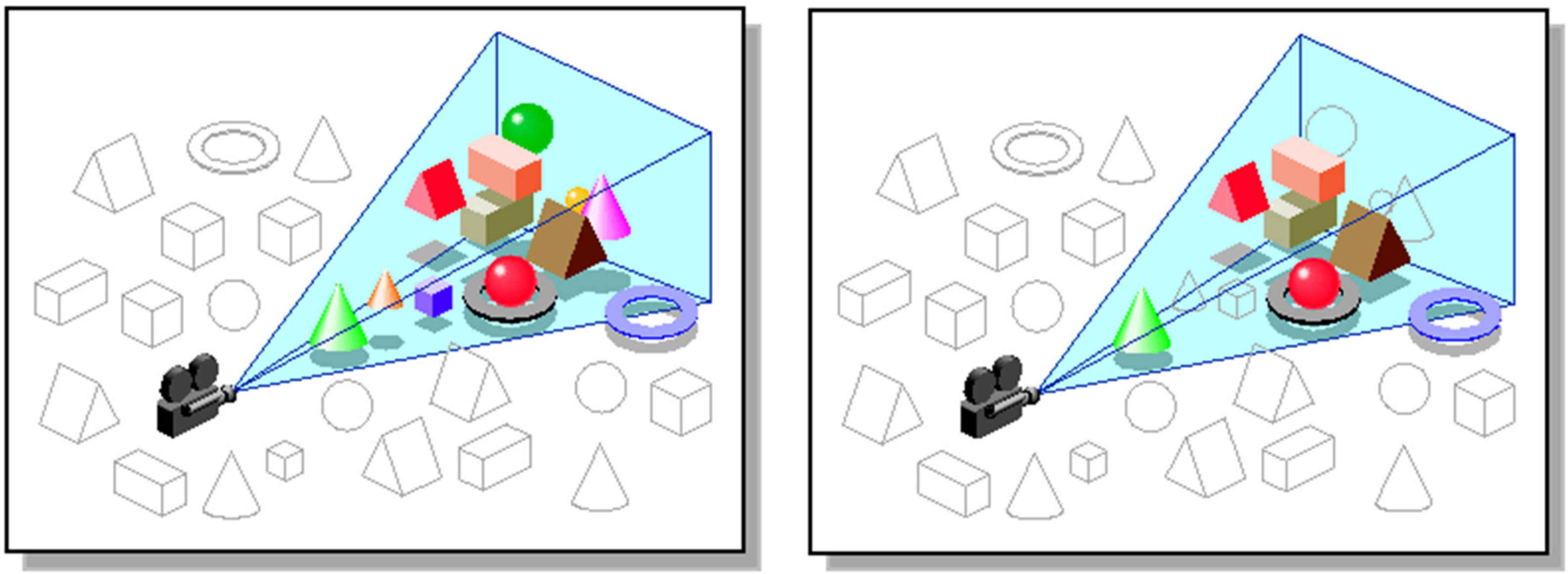
UCSD

Culling

- Goal:
  Discard geometry that does not need to be drawn to speed up rendering
- Types of culling:
  - View frustum culling
  - Occlusion culling
  - Small object culling
  - Backface culling
  - Degenerate culling

Occlusion Culling

- Geometry hidden behind occluder cannot be seen
  - Many complex algorithms exist to identify occluded geometry



Images: SGI OpenGL Optimizer Programmer's Guide

Video

- Umbra 3 Occlusion Culling explained
  - http://www.youtube.com/watch?v=5h4QgDBwQhc

Small Object Culling

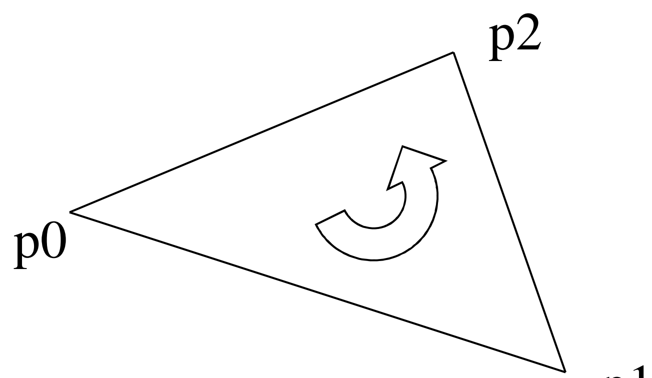
- Object projects to less than a specified size
  - Cull objects whose screen-space bounding box is less than a threshold number of pixels

Backface Culling

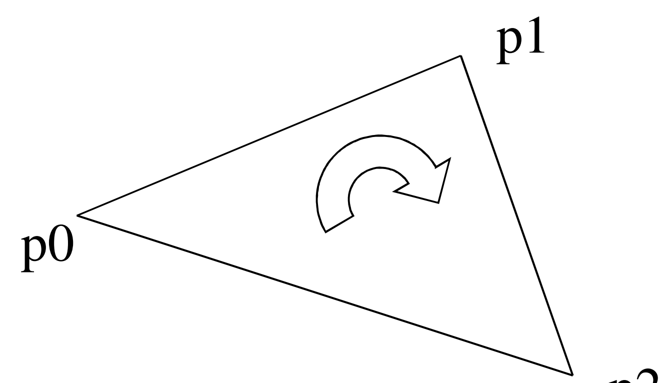
- Consider triangles as "one-sided", i.e., only visible from the "front"
- Closed objects
  - If the "back" of the triangle is facing the camera, it is not visible
  - Gain efficiency by not drawing it (culling)
  - Roughly 50% of triangles in a scene are back facing

Backface Culling

- Convention:
  Triangle is front facing if vertices are ordered counterclockwise



Front-facing



Back-facing

- OpenGL allows one- or two-sided triangles
  - One-sided triangles:
    glEnable(GL_CULL_FACE); glCullFace(GL_BACK)
  - Two-sided triangles (no backface culling):
    glDisable(GL_CULL_FACE)

Backface Culling

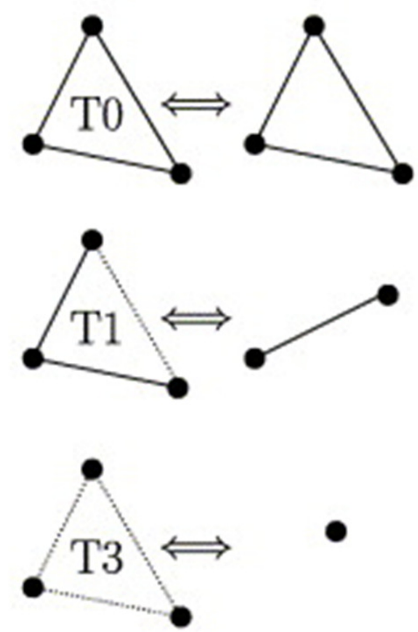
- Compute triangle normal after projection (homogeneous division)

$$\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$$

- Third component of $\mathbf{n}$ negative: front-facing, otherwise back-facing
  - Remember: projection matrix is such that homogeneous division flips sign of third component

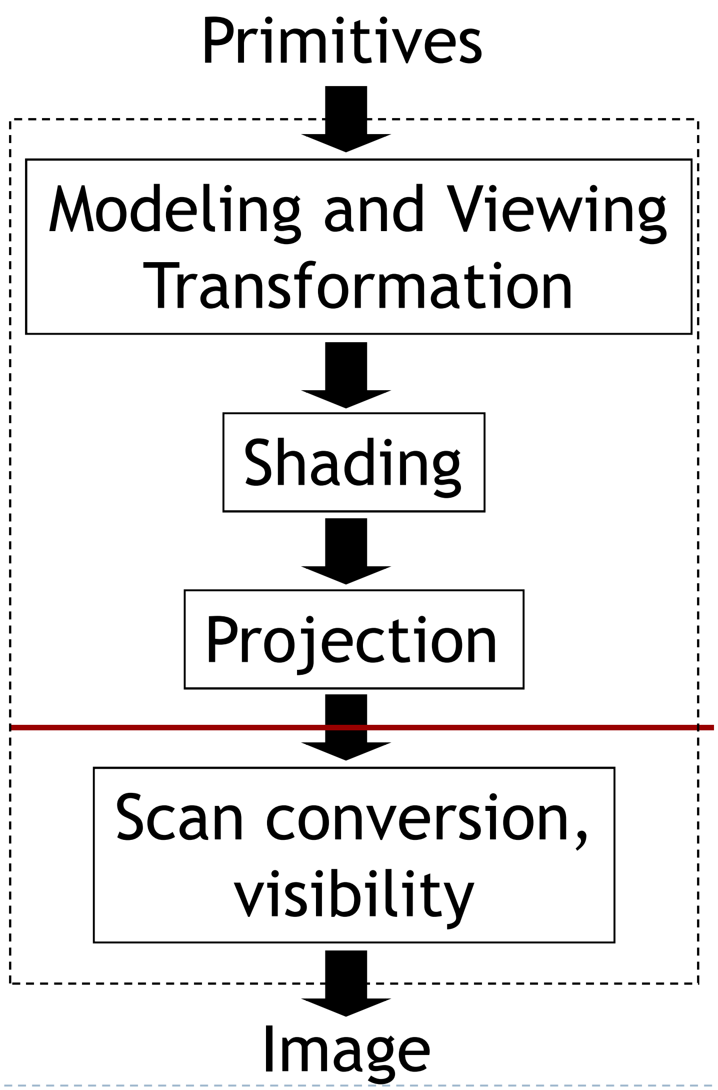
Degenerate Culling

- Degenerate triangle has no area
  - Vertices lie in a straight line
  - Vertices at the exact same place
  - Normal $\mathbf{n}$=0



Source: Computer Methods in Applied Mechanics and Engineering, Volume 194, Issues 48–49

Rendering Pipeline



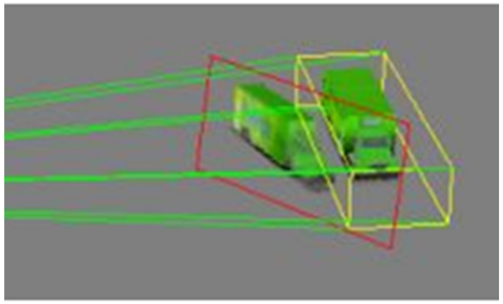
Culling, Clipping

- Discard geometry that will not be visible

UCSD

Level-of-Detail Techniques

- Don't draw objects smaller than a threshold
  - Small feature culling
  - Popping artifacts
- Replace 3D objects by 2D impostors
  - Textured planes representing the objects



Impostor generation

Original vs. impostor

- Adapt triangle count to projected size

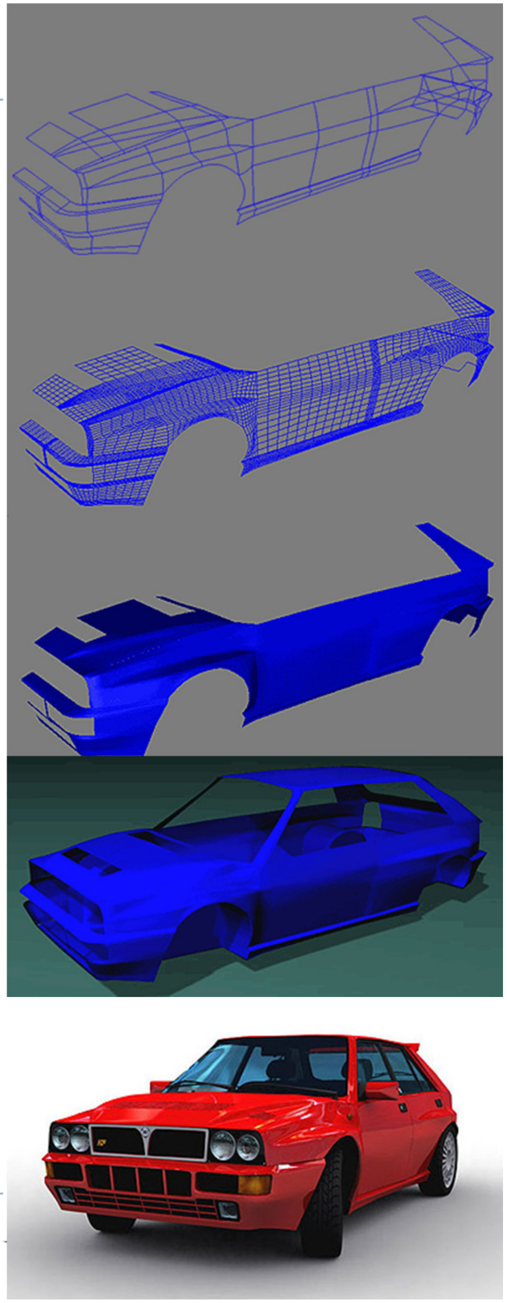
Size dependent mesh reduction
(Data: Stanford Armadillo)

Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves

Modeling

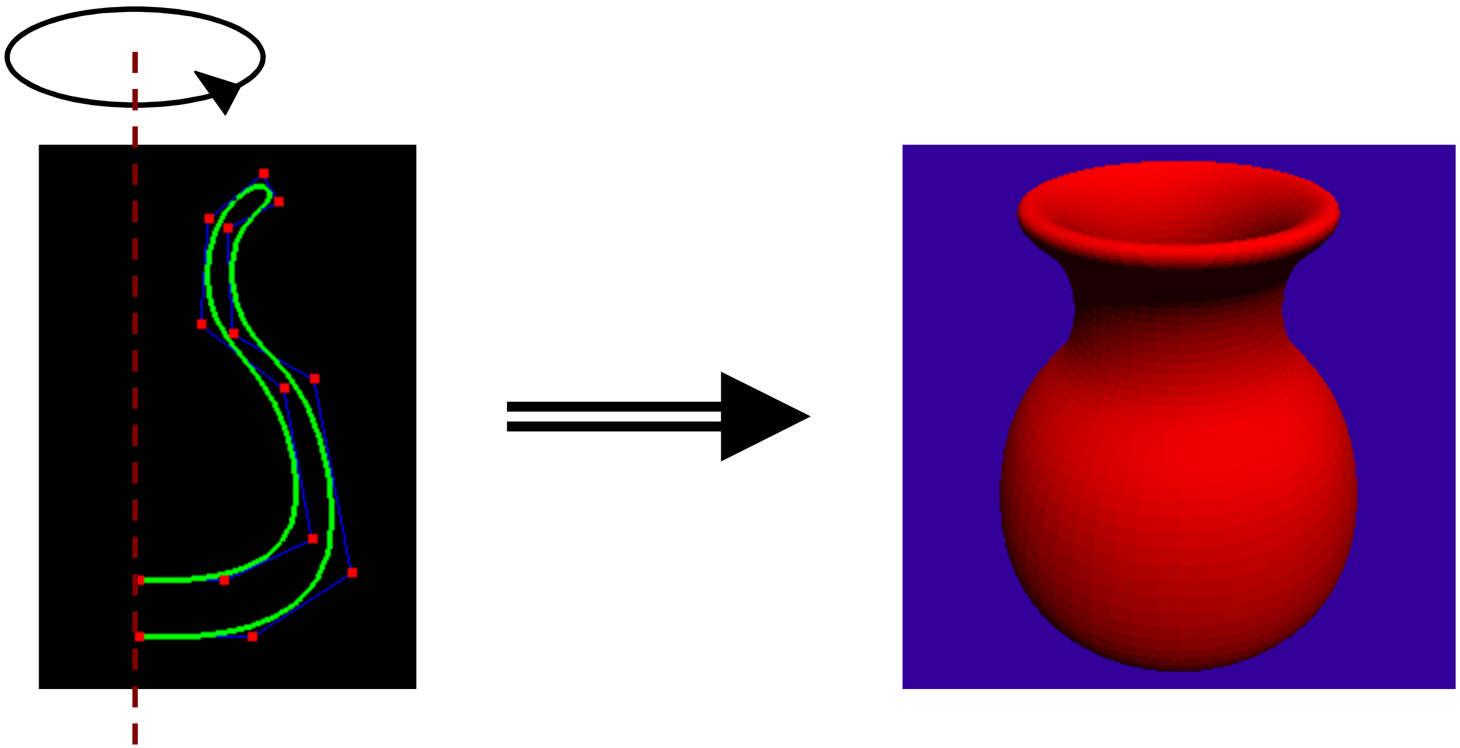
- Creating 3D objects
- How to construct complex surfaces?
- Goal
  - Specify objects with control points
  - Objects should be visually pleasing (smooth)
- Start with curves, then generalize to surfaces



- Next: What can curves be used for?

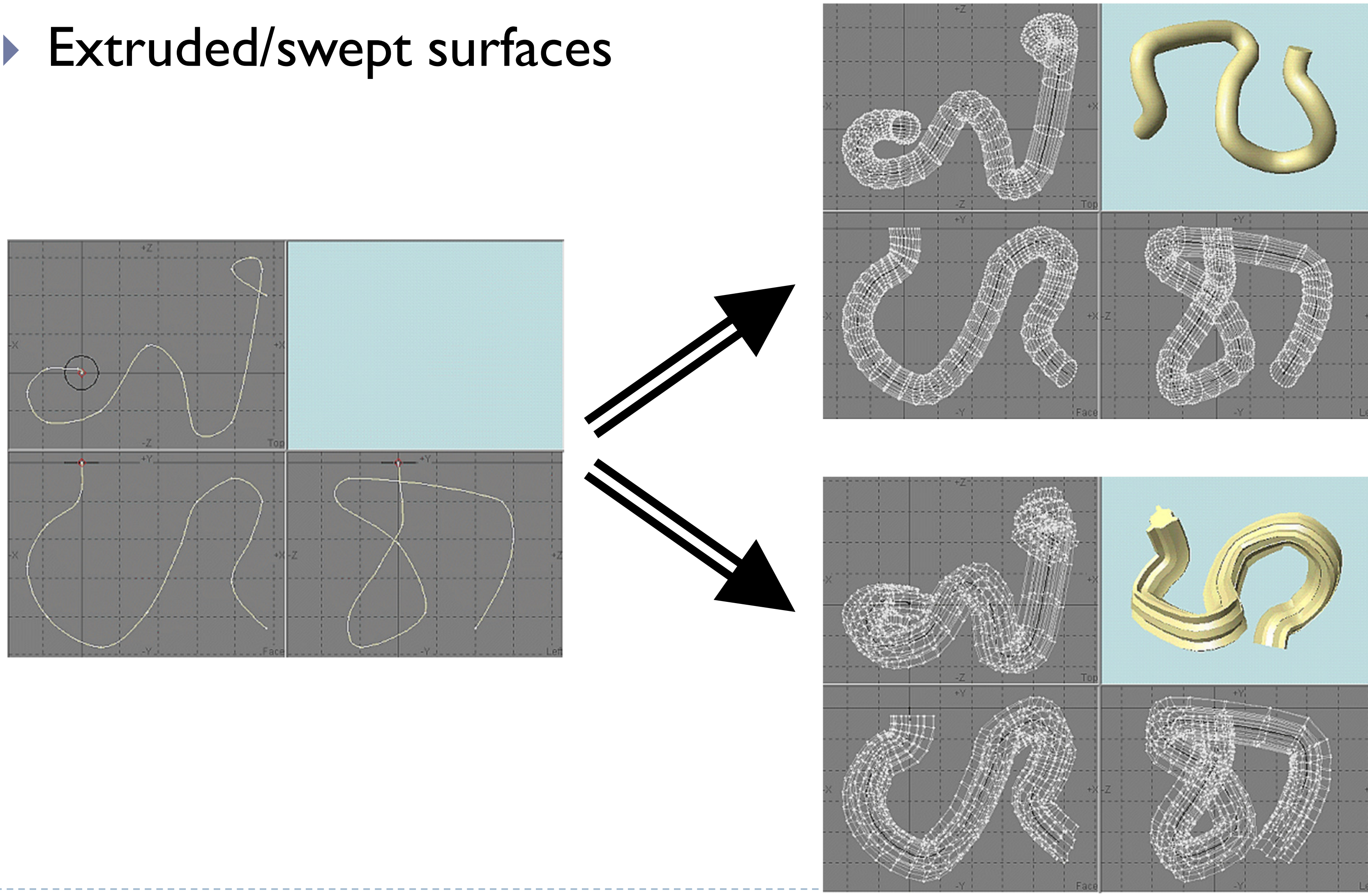
Curves

- Surface of revolution



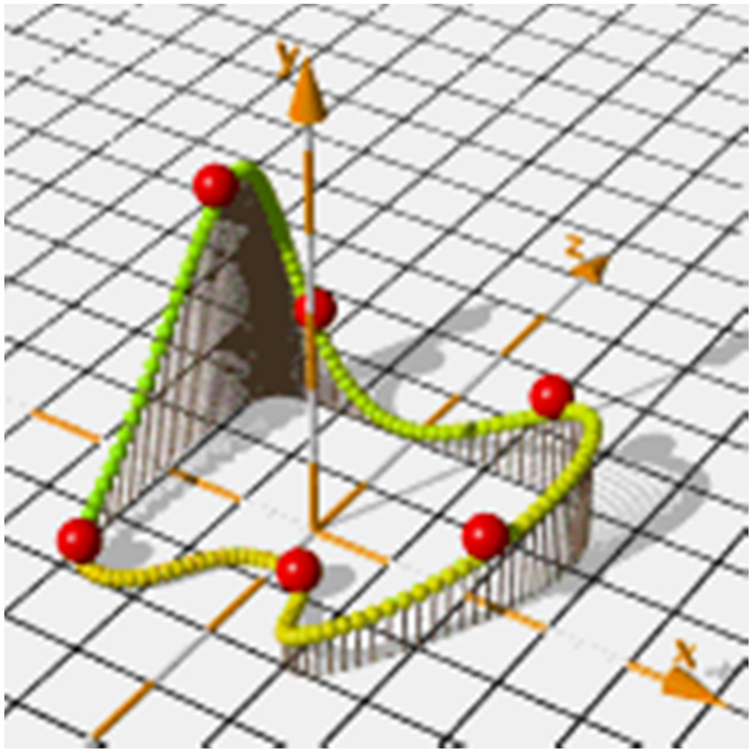
Curves

- Extruded/swept surfaces



Curves

- Animation
  - Provide a "track" for objects
  - Use as camera path



UCSD