Rendering Software Acceleration

The Need for Acceleration

Constructing detailed models becoming easy

- Re-use models from a “catalog”
- High tesselation for smooth appearance up close

Stereo, anti-aliasing, high resolution all reduce raw performance

Software management required for detailed models and complex environments

Stages to Accelerate

Client-server communications

Per-primitive operations

Per-pixel operations

Acceleration Types

Fast path for machine/API

- Triangle strips/fans, vertex arrays, display lists

Culling Techniques

- View frustum, backface, cell/portal, general occlusion

Replacement Techniques

- Levels of detail, portal textures, general image replacement

Fast Path for Machine/API

Principle: know your architecture and use the methods optimized for speed by the designers whenever possible.

Fast API primitives

Triangle strips/fans

- Reduce triangles from 3 vertices to 1 in limit
- Reduces communications and per-vertex operations (transformation, lighting)

Vertex arrays

- Pack vertex coordinates and properties into arrays
- Avoids communications overhead of individual commands

Display Lists

Cache large sets of commands to be reused

If available, store at server side

- **Saves on communications if it fits in cache**

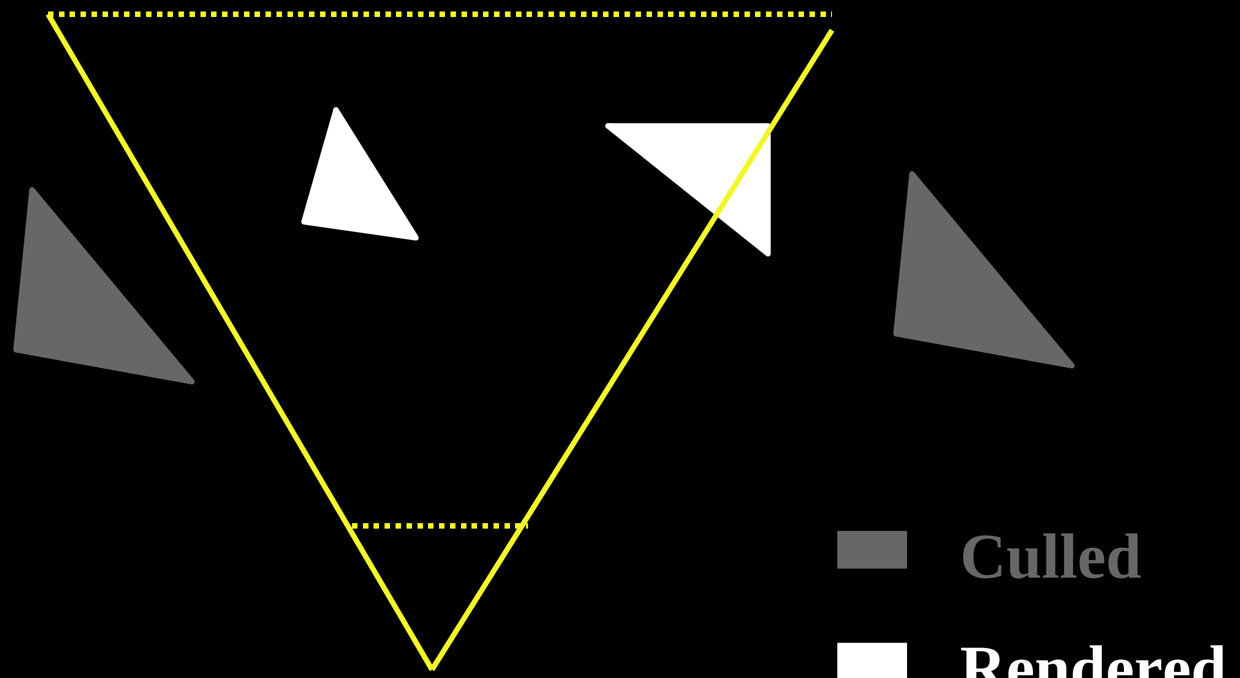
Potential for optimization

- **concatenate matrices**
- **format data for processing**

Culling Techniques

Principle: **Do not render those primitives that will not ultimately be visible to the user.**

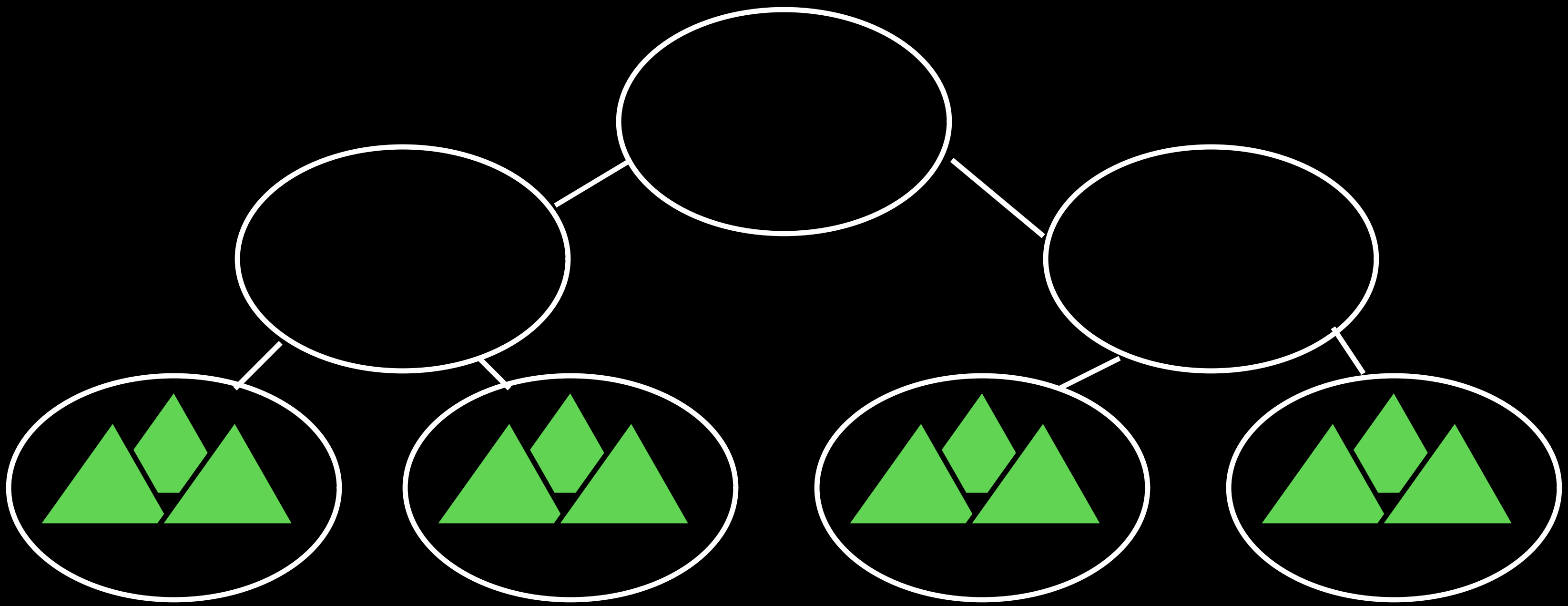
View Frustum Culling Diagram



View Frustum Culling

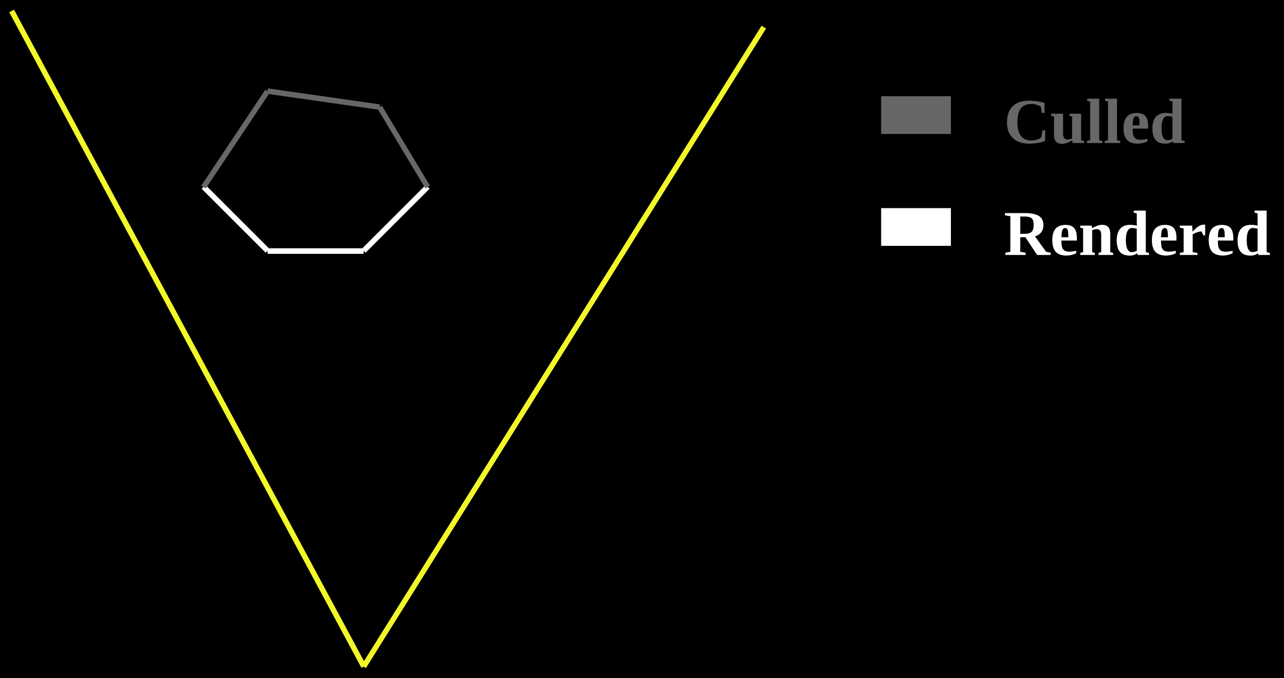
- Do not render primitives that lie outside viewing frustum
- May be determined exactly or conservatively
- May or may not consider near and far planes
- May be performed hierarchically
  - Cluster primitives
  - Cull bounding volumes of entire clusters

Database Hierarchy



Leaf nodes may have one or more primitives
Culling traversal may go down to leaves or terminate early
Bounding volumes used to cull-test an entire node

Backface Culling Diagram



Backface Culling Basics

Don't render primitives facing away from the viewer

- For solid objects, front occludes back

Polygon is backfacing iff ray from polygon to eye is more than 90° from polygon normal

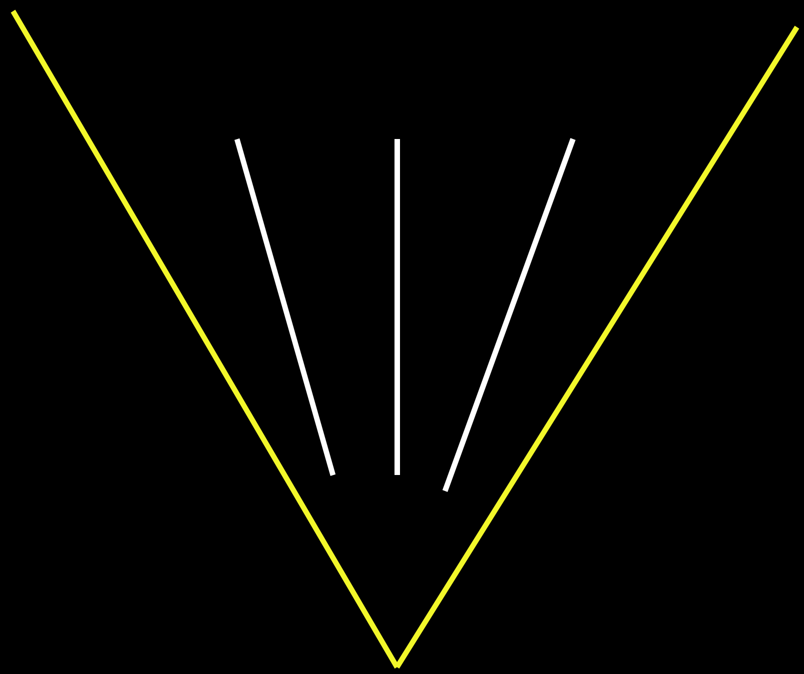
Orthogonal Projection

- Test sign of (-viewdir . normal)

Perspective Projection

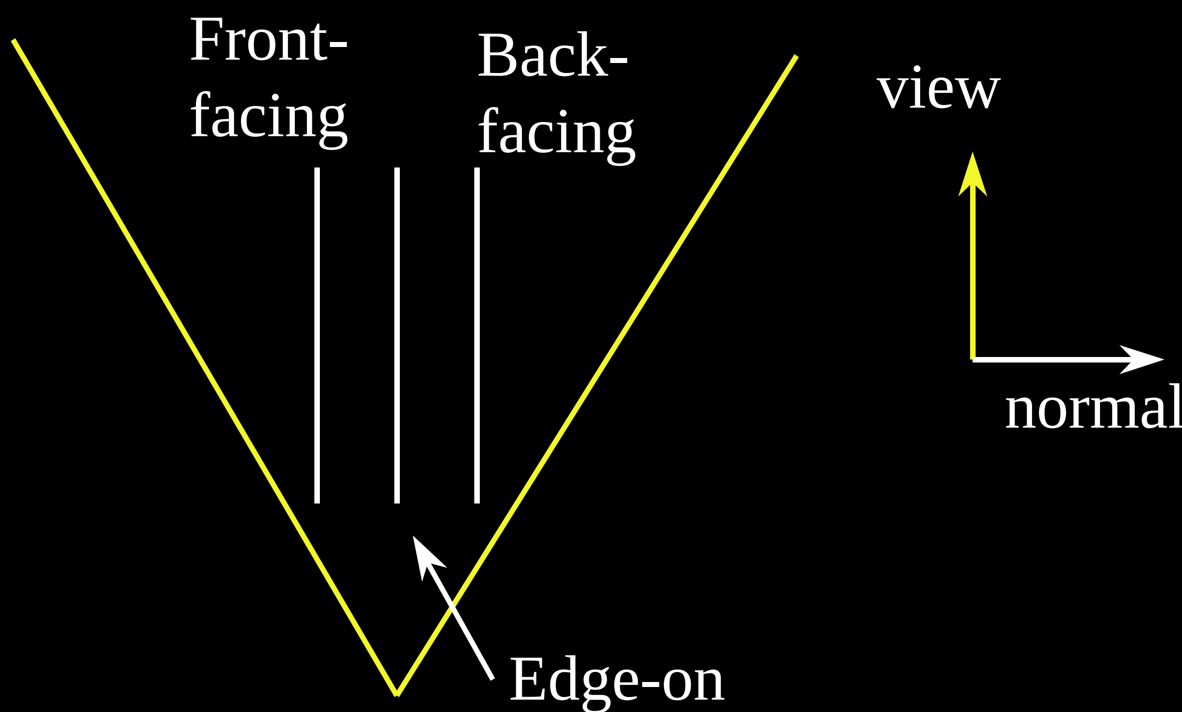
- Test sign of (eye-polygon . normal)

Edge-on Polygons in Perspective



Critical angle between front- and back-facing

90 Degrees from View Direction



Options

- In graphics engine
  - After client-server communications
  - Often after transformation - eye or screen space
- On client
  - In object space
  - Avoids communicatiosn and transformation
- May be performed hierarchically
  - Restricts cluster organization of hierarchy
- May be accelerated with normal masks
  - Quantize normals, classify quantizations
  - Conservative backface culling on client

Backface Culling Video

"Fast Backface Culling using Normal Masks"

Hansong Zhang and Kenneth E. Hoff III

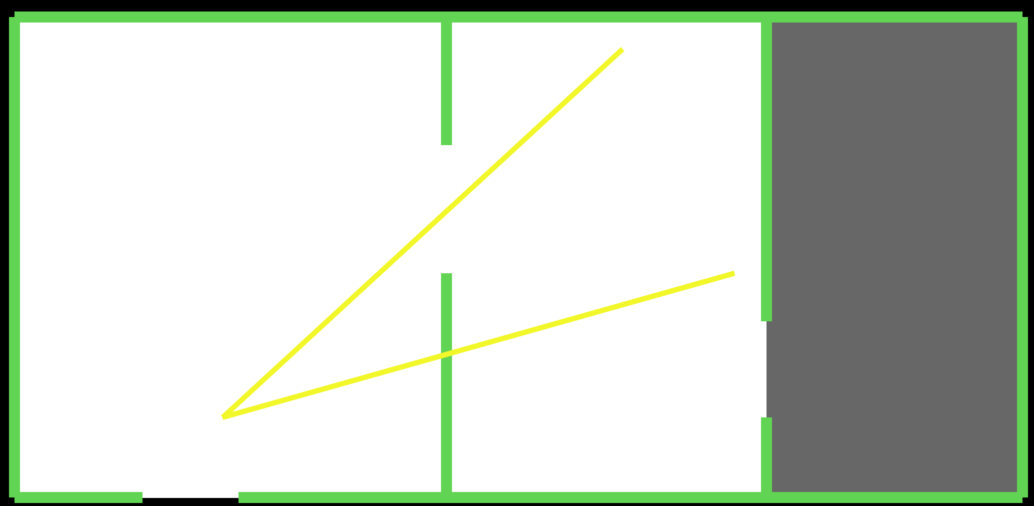
1997 Symposium on Interactive 3D Graphics

Tradeoffs

Culling at client end

- Reduces bandwidth requirements
- Adds conditional to tight rendering loop
- May be difficult with triangle strips

Cell and Portal Culling



Cells associated with rooms
Portals associated with windows and doors

Cell and Portal Basics

Partition model into cells with portals between them

Reduce model to *potentially visible set*

- Primitives in current cell and cells visible through portals

Hierarchy of viewing frusta

- Each portal narrows the viewing frustum into adjacent cells

Useful for models with wall-like occlusions

- Difficult to determine structure automatically

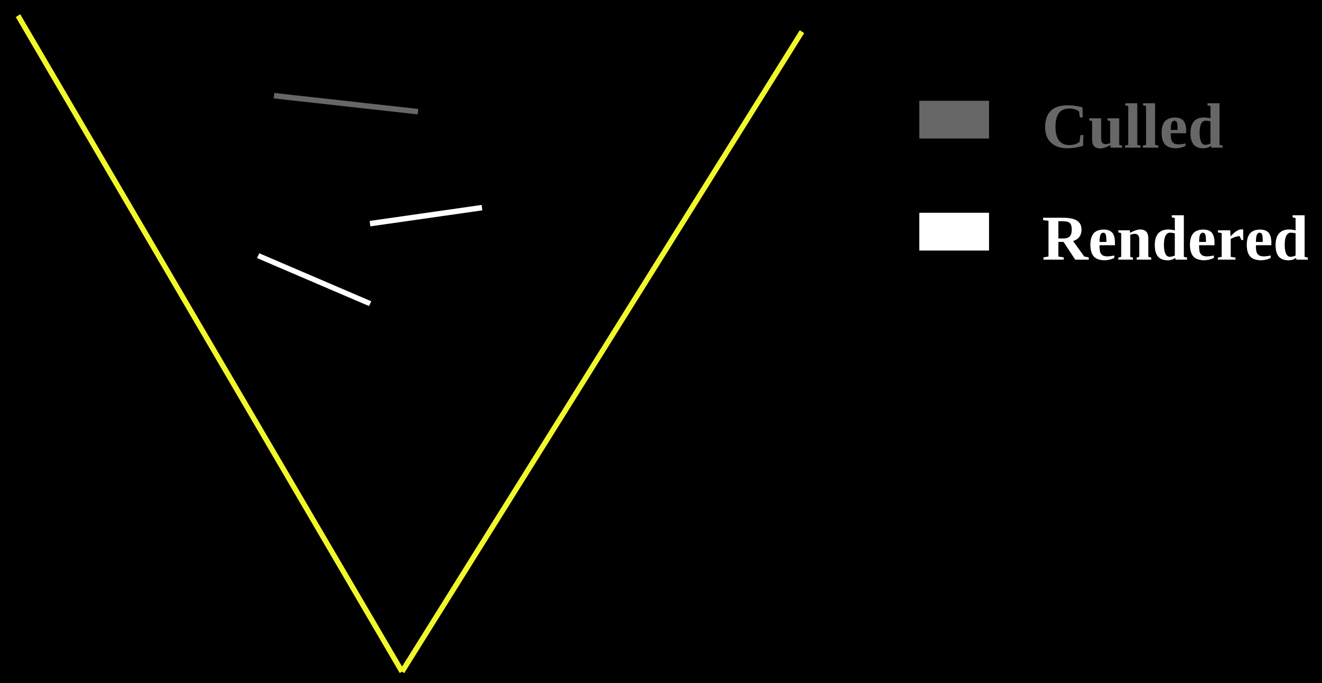
Cell and Portal Video

“Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets”

David Luebke and Chris Georges

1995 Symposium on Interactive 3D Graphics

Occlusion Culling Diagram



Occlusion Culling Basics

Don't render occluded primitives

- Expensive to compute exactly

Algorithm components

- Occluder selection
- Occlusion test

Object-space approach

- Shadow frusta

Image-space approach

- Hierarchical occlusion maps

Occluder Selection

Prune database to potentially good occluders

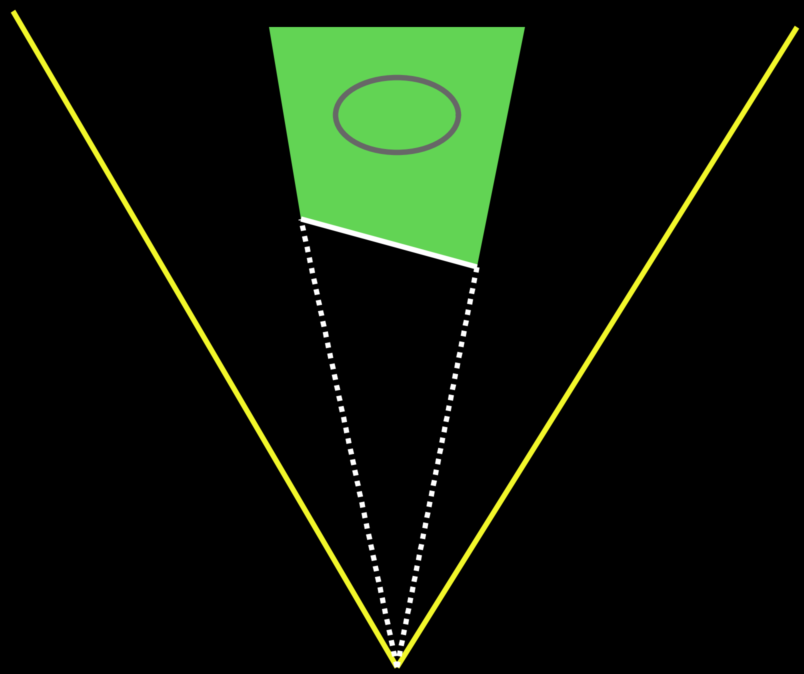
Preprocessing: create occluder database

- Retain large, simple objects
- Discard small objects
- Discard or replace complex objects

Run-time processing: view-dependent criteria

- Distance from viewpoint
- Size
- Temporal coherence

Shadow Frustum Diagram



Shadow Frustum Approach

Construct frustum for current occluders

Perform hierarchical frustum cull for each

- Database nodes fully contained in a frustum are occluded

Easily implemented using same code as view frustum culling

Not so good for *occluder fusion*

Occlusion Maps

Render occluders into highest resolution map

- Covered pixels get opacity of 1

Filter maps by averaging

- For higher resolutions, accelerate with texture map hardware

Culling with Occlusion Maps

Culling requires depth + map coverage

Depth

- only cull primitives farther than farthest occluder

Coverage

- Compute bounding screen-space rectangle for database node
- Cull all map pixels in rectangle (at some resolution) are above opacity threshold

Occlusion Culling Video

"Visibility Culling using Hierarchical Occlusion Maps"

Zhang, Manocha, Hudson, and Hoff

Proceedings of SIGGRAPH 97