Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

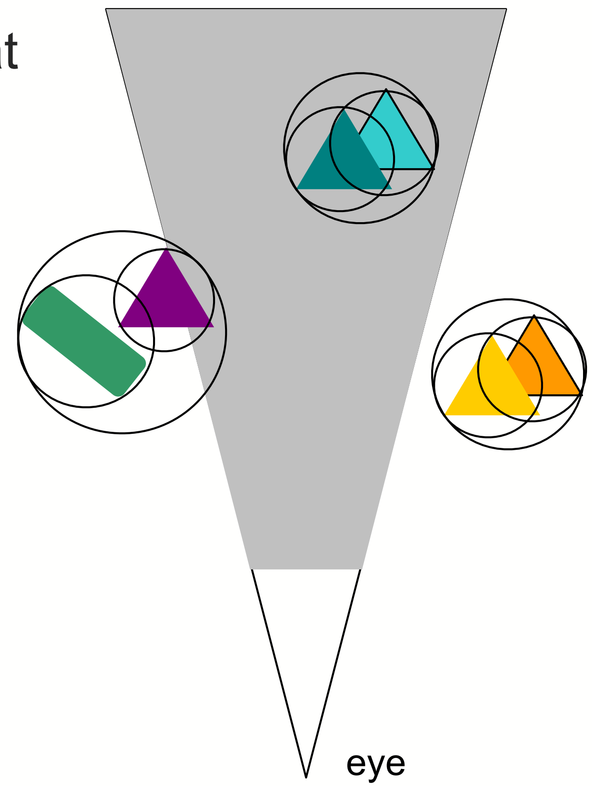
Brian Russin

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Module 5D
View Frustum Culling

Hierarchical View Frustum Culling

- Bound every “natural” group of primitives by a simple volume
  - If a BV is outside the view frustum, then the entire contents of that BV is also outside
    - Do not need to render its contents
    - Do not need to process its subtree
  - If a BV is inside the all its contents must be inside
    - Render the tree
    - No further frustum testing is needed
  - If a BV intersects the frustum
    - Traversal continues – testing children
- View frustum culling occurs in the application
  - Reduces work in geometry and rasterization
  - For large scenes only a fraction of the scene might be visible



View Frustum Intersection

- Complete test is called **exclusion/inclusion/intersection** test
  - Sometimes the test for intersection is costly
    - Classify these cases as “probably inside”
    - Simplified algorithm is called an **exclusion/inclusion test**
      - Err on the side of inclusion
      - Objects that may actually be able to be excluded are included
- General methods described in Haines and Moller
  - Idea is to transform BV/frustum test to point/volume test
    - Moving the BV around the frustum to form a new volume to test against
  - See Figure 22.25 for illustration (16.23 in 3rd Edition)

View Frustum Intersection

- View frustum intersection is essential for rapid scene rendering
  - Scene stored with hierarchy of bounding volumes
  - Need a rapid test for whether view frustum intersects a bounding volume
    - BV totally inside, totally outside, or intersects the frustum
- View frustum is a polyhedron defined by 6 planes
  - Near, far, left, right, bottom, top
  - See Figure 22.24 (16.22 in 3rd Edition)
- Rules
  - If BV is totally outside frustum
    - Any objects and sub-objects do not need to be rendered
  - If BV is totally inside frustum
    - No further BV/frustum intersects need to be performed for any sub-objects
  - If BV intersects frustum
    - Need to recursively test sub-objects BV against the frustum
    - If inside or intersects the frustum, need to render the object

Frustum Plane Extraction

- Store plane equations of the 6 planes of the frustum
  - Oriented so positive half space is outside the frustum
- Geometric solution based on camera definition
  - Near and far plane normals +/- n (view plane normal)
  - Find the 4 corner vertices of the near view plane
    - Find width (W) and height (H)
      - Using near plane distance (N), FOV (y) angle, and aspect ratio
  - Use linear combination of view axes scaled by these distances:
      - topLeft = VRP – Nn – Wu +Hv
  - Find the normals to side and top/bottom planes using cross products
    - Create vectors between vertices on each plane
      - From VRP to 2 corners
    - Orient the cross product so normal faces outwards
    - Normalize the vectors
  - Normal forms A,B,C components of plane equation
  - Solve for D by plugging a vertex on the plane into the plane equation

Alternate Frustum Plane Extraction

- Described in Haines and Moller: section 22.14.1 (16.14.2 in 3rd Edition)
  - Details – look into Gribb and Hartmann method
- Assume view matrix V and projection matrix P
  - Composite matrix is C=PV
    - OpenGL discussion in reference above says to use VP
  - Point s (with s_w=1) is transformed into t as t=Ms
  - t may have t_w != 1 (perspective projection) so divide by tw to obtain point u (u_w=1)
  - For points inside the view frustum -1 <= ui < 1 for i=(x,y,z)
    - Point u is inside the unit (symmetric) cube
  - Equation 22.26 gives derivation of the left plane

$-(m_3, +m_0)\cdot(x,y,z,1)=0$	left
$-(m_3, -m_0)\cdot(x,y,z,1)=0$	right
$-(m_3, +m_1)\cdot(x,y,z,1)=0$	bottom
$-(m_3, -m_1)\cdot(x,y,z,1)=0$	top
$-(m_3, +m_2)\cdot(x,y,z,1)=0$	near
$-(m_3, -m_2)\cdot(x,y,z,1)=0$	far

Implementation Details

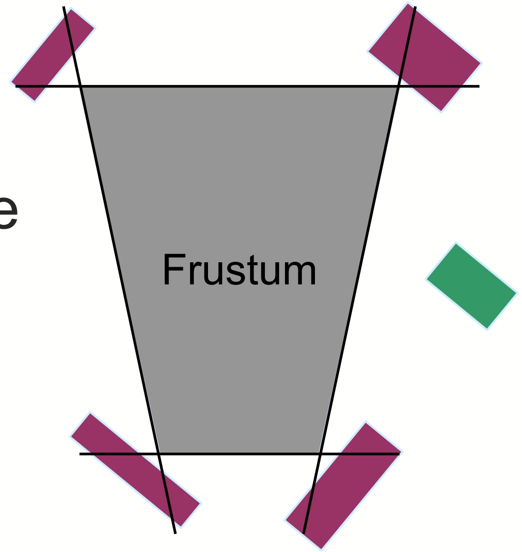
- Use C = PV as the combo matrix
- Multiply each plane equation component by -1 to get outward facing normals
  - Reference code assumes normals point to the inside half space
- Normalize the plane equations
  - Use Normalize method of the Plane class

Frustum/Sphere Intersection

- Loop over the 6 planes of the frustum
  - Find the signed distance d of the sphere center to the frustum plane
    - Insert sphere center into plane equation
  - If d>r the sphere is outside with respect to the frustum plane
    - Sphere is outside frustum (exclusion)
    - Early exit case if any one plane found where sphere is outside
  - If d<-r for **all** 6 planes then sphere is inside frustum (inclusion)
  - Otherwise the sphere intersects the frustum
- If frustum is symmetric about the view direction can simplify
  - Divide frustum into 8 octants
  - Find the octant that the sphere center is in
    - Test against the 3 outer planes of the octant
  - Does not improve performance much since algorithm is already fast

Frustum / Box Intersection

- Check the bounding box against the 6 frustum planes
  - If **all** corner points of AABB or OBB are outside with respect to **any** 1 plane the BB is outside the frustum
  - If box is inside all 6 planes then the box is inside the frustum
  - Otherwise it intersects the frustum
- Can classify boxes as intersecting that are actually fully outside
  - A more complex test can be derived using the separating axis theorem
- Algorithm in Haines for AABB
  - Tests box against the 6 planes
  - Using AABB/plane intersect
  - Early return if outside any 1 plane

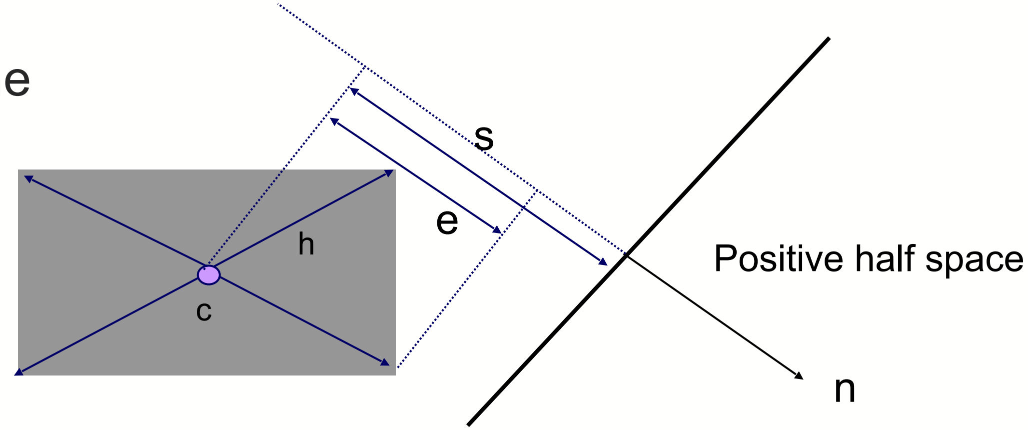


Plane/Box Intersection

- Can determine if an intersect occurs by inserting the 8 vertices of the box into the plane equation
  - If at least one positive solution and one negative solution, then vertices are located on both sides of the plane
    - An intersect occurs
- Faster methods for AABB and OBB
  - See Section 22.10 (16.10 in 3rd Edition)
  - Idea is that only two points need to be inserted into plane equation
    - The 2 points form the longest diagonal of the box when measured along the plane's normal
    - Found by taking the dot product of each diagonal's direction with the plane's normal
      - Largest value identifies the diagonal with the furthest points
  - Text includes improved and simplified pseudo code (compared to Edition 2)

Plane/AABB Intersection

- Store the box center c and half diagonal vector h
  - $c=(b^{max}+b^{min}) / 2$
    - Midpoint between two extreme points (affine combination)
  - $h=(b^{max}-b^{min}) / 2$
- Find the box's extent when projected onto the n (plane normal)
  - Could project each of the 8 half diagonals
  - However can express as: $e = h_x\left|n_x\right| + h_y\left|n_y\right| + h_z\left|n_z\right|$
    - Since half diagonals are combinations of $+/-h_x$, $+/-h_y$, $+/-h_z$
    - Dot product reaches maximum when all are +
- Compute signed distance s of center point to the plane
  - If $s-e > 0$ then AABB is outside
  - If $s+e < 0$ then AABB is inside
  - Else it intersects



Plane / OBB Intersection

- Use same test as plane/AABB intersect
- "Extent" of box is changed

$$e = h_u^B \left| n \cdot b^u \right| + h_v^B \left| n \cdot b^v \right| + h_w^B \left| n \cdot b^w \right|$$

  - b are the coordinate system axes for the OBB
  - h are the lengths along these axes

View Frustum Culling – Frame Coherency

- Frame to frame coherency may be utilized
  - If a BV is outside a specific plane it is likely to be outside that plane in the next frame
    - Assuming view updates are small
  - Can store an index to the plane along with the BV
  - Test against this plane first
    - Slight performance improvement