Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

Brian Russin

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

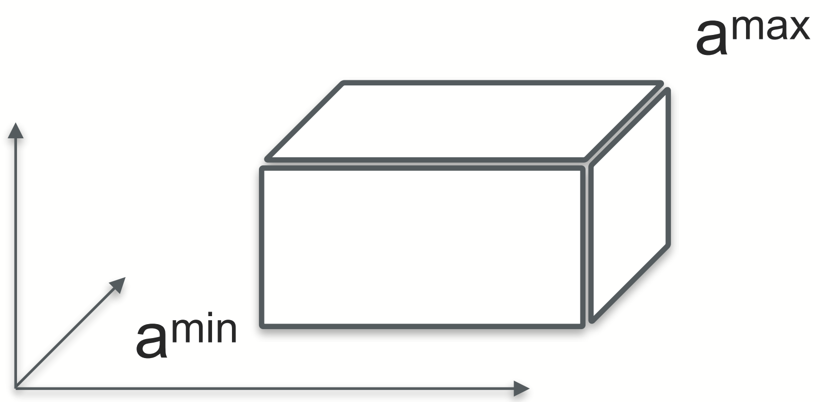
Module 1C
Bounding Volumes

Bounding Volumes

- **Bounding Volumes (BV)** surround complex objects with a simple volume
  - Easier to perform intersection tests
  - Will be important for ray-tracing efficiency, scene graph efficiency improvements, and collision detection methods
- Will discuss 3 different types
  - Sphere
  - Axis Aligned Bounding Box (**AABB**)
    - Aligned to coordinate axes
  - Oriented Bounding Box (**OBB**)
  - Text also discusses k-DOP (Discrete Oriented Polytope)
- Efficiency consideration – BV should tightly bound the objects within
  - Minimizing BV surface area reduces probability an arbitrary ray will intersect
  - Spheres - efficient to create and test intersections against
  - AABB – easy to create and can **more tightly bind** objects that are long and thin
  - OBB – more complex to form

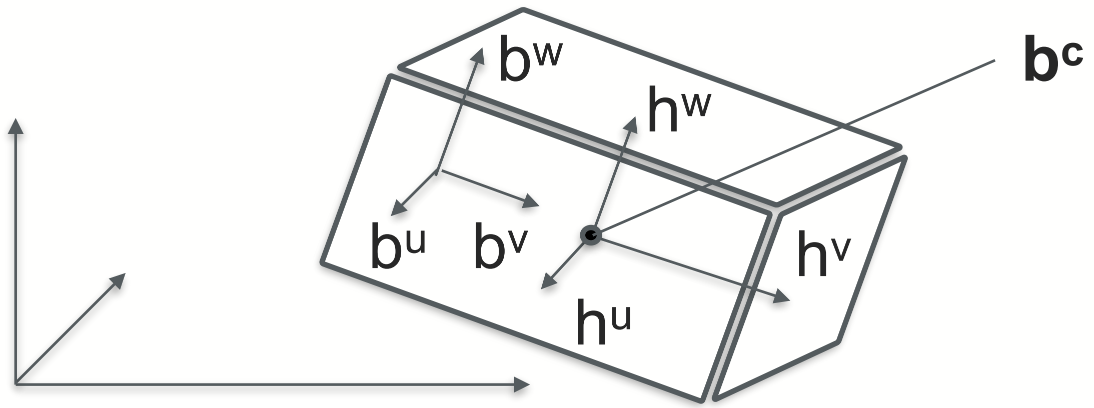
Axis-Aligned Bounding Box

- **Axis Aligned Bounding Box (AABB)**
  - Also called **rectangular box**
  - Faces have normals that coincide with standard coordinate system axes
- Can be described by two extreme points
  - a^{min} and a^{max} $\quad a_i^{min} \le a_i^{max}, \forall i \in \{x, y, z\}$
- Easy to form by finding the extents (min. and max. coordinate component)
  - along each axis from object's vertex list



Oriented Bounding Box

- **Oriented Bounding Box (OBB)**
  - Faces have normals that are pair wise orthogonal
    - An AABB that is rotated
- Can be described by a center point b^c, three normalized vectors, and three half lengths
  - Unit length vectors b^u, b^v, b^w
    - Describe the normals to the sides of the box
  - Half lengths h^u, h^v, h^w
    - Distance from the center to the respective face



Forming Bounding Spheres

- Several methods – speed vs. quality tradeoff
- Sphere Containing AABB
  - Fast but sometimes gives a poor fit
  - Form an AABB then use the center and diagonal to form sphere
  - Can improve fit by adding a second pass
    - Go through all the vertices once again and find the one furthest away from the new center
      - Use squared distances when comparing to avoid computing a square-root on each vertex!
    - Use that as the new radius
- Sphere Centered at Average of Vertices
  - Slightly better fit than sphere containing AABB
  - Compute average of vertex positions
  - sum x,y,z and divide each by n
  - Find extreme vertex from that center to form the radius
    - Use distance squared metric when comparing

Ritter's Method for Forming Bounding Spheres

- Near-Optimal Bounding Sphere - method by Ritter
  - Find (6) vertices at the minimum and maximum along each axis
    - Vertex with x_{min}, vertex with y_{min}, vertex with z_{min}, vertex with x_{max}, vertex with y_{max}, vertex with z_{max}
  - Find the pair with the largest distance to form a sphere
    - Center at the midpoint between them
    - **diameter** = distance between them
    - This sphere should contain most points
  - Go through all other vertices – check distance d to center
    - If vertex is outside sphere's radius move center and change radius
      - Move center toward vertex by distance (d-r)/2
      - Set radius to (d+r) / 2
      - Use distance squared to do comparison – compute d only if outside
    - Effectively encloses vertex and existing sphere within a new sphere

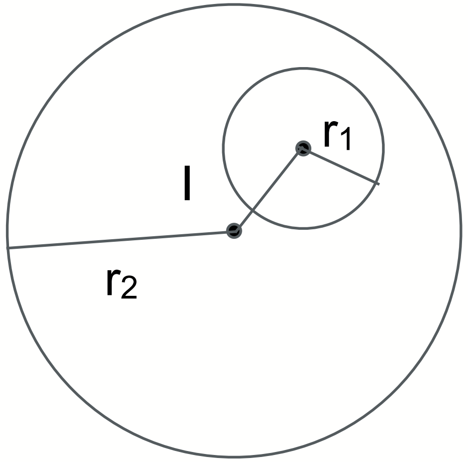
Forming a Bounding Sphere

- Minimum Volume Sphere – method by Welzl
  - More complex but produces an optimal bounding sphere
  - Linear performance for a randomized list of vertices
    - Randomization helps find a good sphere quickly
  - 1) Find a **supporting set** of points defining a sphere
    - Set of 2, 3, or 4 points on its surface
  - 2) Iterate through vertex list: when a vertex is found outside the current sphere
    - Vertex is added to the supporting set
      - Possibly remove old support vertices
    - New sphere computed
  - 3) Repeat step 2) until all vertices are contained within the sphere
- Downloadable code implementation by Bernd Gaertner
  - http://www.inf.ethz.ch/personal/gaertner/miniball.html

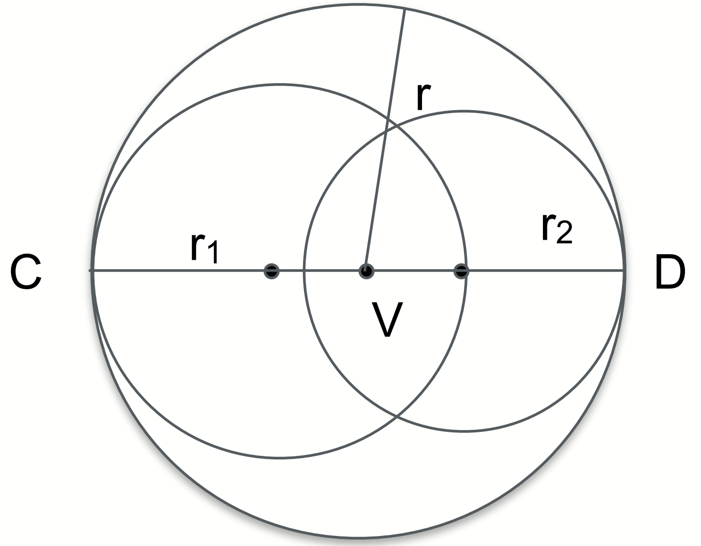
Merging Two Bounding Spheres

```
BoundingSphere Merge(BoundingSphere& s2){
      // Form a vector between the 2 centers, find its squared length
      Vector3 v = s2.center - center;
      float vsqr  = v.NormSquared();

      // Compare that length against the difference in the 2 sphere radii
      float rDiff = s2.radius - radius;
      if ((rDiff*rDiff) >= vsqr)
         return (rDiff >= 0.0f) ? BoundingSphere(s2) : BoundingSphere(*this);
      else {
         // Partial overlap or spheres are disjoint
         // Find diameter and radius of new bounding sphere
         float normv = sqrt(vsqr);
         float r = radius + s2.radius + normv;   // r1 + r2 + norm(v)

         // New center is bisector of the diameter line from C to D
         v *= (1.0f / normv);                    // Normalize v
         return BoundingSphere(center – (radius * v) + (r * v)),  r);
      }
   }
```

Can be useful with dynamic objects or forming hierarchy of BVs.



Spheres merge into S2



Overlap: create new sphere

Forming an OBB

- Optimal approaches are not feasible, good approximations are best
- Can be found by first computing the convex hull
  - Text describes method
  - finding the convex hull is O (n log n) where n is number of primitives
- Main concern is finding the axes of the OBB
  - Trivial computation after that
- Eberly presents a method using a minimization technique without needing a convex hull
  - An iterative solution to solve a minimization problem
    - Initial guess determines how fast the solution converges
  - Uses Powell's direction set method
  - http://www.geometrictools.com
- Ericson presents a reasonable PCA (principle component analysis) method to determine 3 good candidate axes
  - PCA is similar to statistical regression analysis
  - Select a primary axis, project primitives onto a plane and compute 2D OBB for the remaining axes
- Moller, Haines, Hoffman presents an iterative approach using an initial segment guess based on a k-DOP
  - Projects primitives onto the initial axis (from the line segment) and forms a triangle
  - Chosen axes are among those aligned with the triangle
- OOB constructed from the axes
  - Find min. and max. (denote as k) along each axis
  - Center of this box is: $a^c = \frac{k^u_{\text{min}} + k^u_{\text{max}}}{2} a^u + \frac{k^v_{\text{min}} + k^v_{\text{max}}}{2} a^v + \frac{k^w_{\text{min}} + k^w_{\text{max}}}{2} a^w$
  - Half lengths are: $h_l = \frac{k^l_{\text{min}} + k^l_{\text{max}}}{2}$ for each axes u,v,w

Intersection Calculations: Efficiency Considerations

- Perform computations and comparisons that lead to **trivial rejection** or **trivial acceptance**
  - Provides an early exit – avoids further, unnecessary calculations
- Postpone expensive computations until they are needed
  - Trigonometric methods, sqrt, division
- Sometimes can reduce the dimension of the problem
  - Turn a 3D intersection into a 2D intersection
    - An example is the ray-polygon intersection test
- Compute terms that are constant over a scene in advance
  - e.g., projection of polygons onto planes
- Organize bounding volumes in nested hierarchies

Hierarchy of Bounding Volumes

- Often can group objects together and form BV for entire group
  - e.g., car consists of body, 4 wheels
- Tree structure
  - Figure 19.2 (14.2, 3rd Edition)
  - BV a bounds volumes b (an entire object), c1,c2, c3, c4 (bounding volumes for more complex objects c1,c2, c3, c4)
- If ray misses a BV, do not need to intersect object(s) within
  - Including those in sub tree