VGP351 – Week 5

- Agenda:
  - Quiz #2
  - Bounding volumes
    - Axis-aligned bounding boxes
    - Oriented bounding boxes
    - Bounding spheres
  - BV hierarchies

Bounding Volumes

- From Wikipedia:

  "...a bounding volume for a set of objects is a closed volume that completely contains the union of the objects in the set."

- Why is this useful?

Bounding Volumes

- From Wikipedia:

  "...a bounding volume for a set of objects is a closed volume that completely contains the union of the objects in the set."

- Why is this useful?
  - Can represent complex geometry that would be expensive to test with an approximation that is much cheaper to test
  - Visibility, collision detection, etc.

Desirable BV Characteristics

- Inexpensive intersection test
  - BVs are used instead of source geometry to speed up *trivial rejection* (or trivial acceptance) tests

Desirable BV Characteristics

- Inexpensive intersection test
  - BVs are used instead of source geometry to speed up *trivial rejection* (or trivial acceptance) tests
- Tight fitting to source geometry
  - If the BV is a poor fit, tests between BVs may result in false positives or false negatives

Desirable BV Characteristics

- Inexpensive intersection test
  - BVs are used instead of source geometry to speed up *trivial rejection* (or trivial acceptance) tests
- Tight fitting to source geometry
  - If the BV is a poor fit, tests between BVs may result in false positives or false negatives
- Inexpensive to compute
  - If the BV is too expensive to compute, the expense of creating it may cancel the speed-up that it provides

Desirable BV Characteristics

- Easy to transform
  - If the object moves, its BV needs to move. If moving the BV is too expensive, it may cancel out the speed-up.

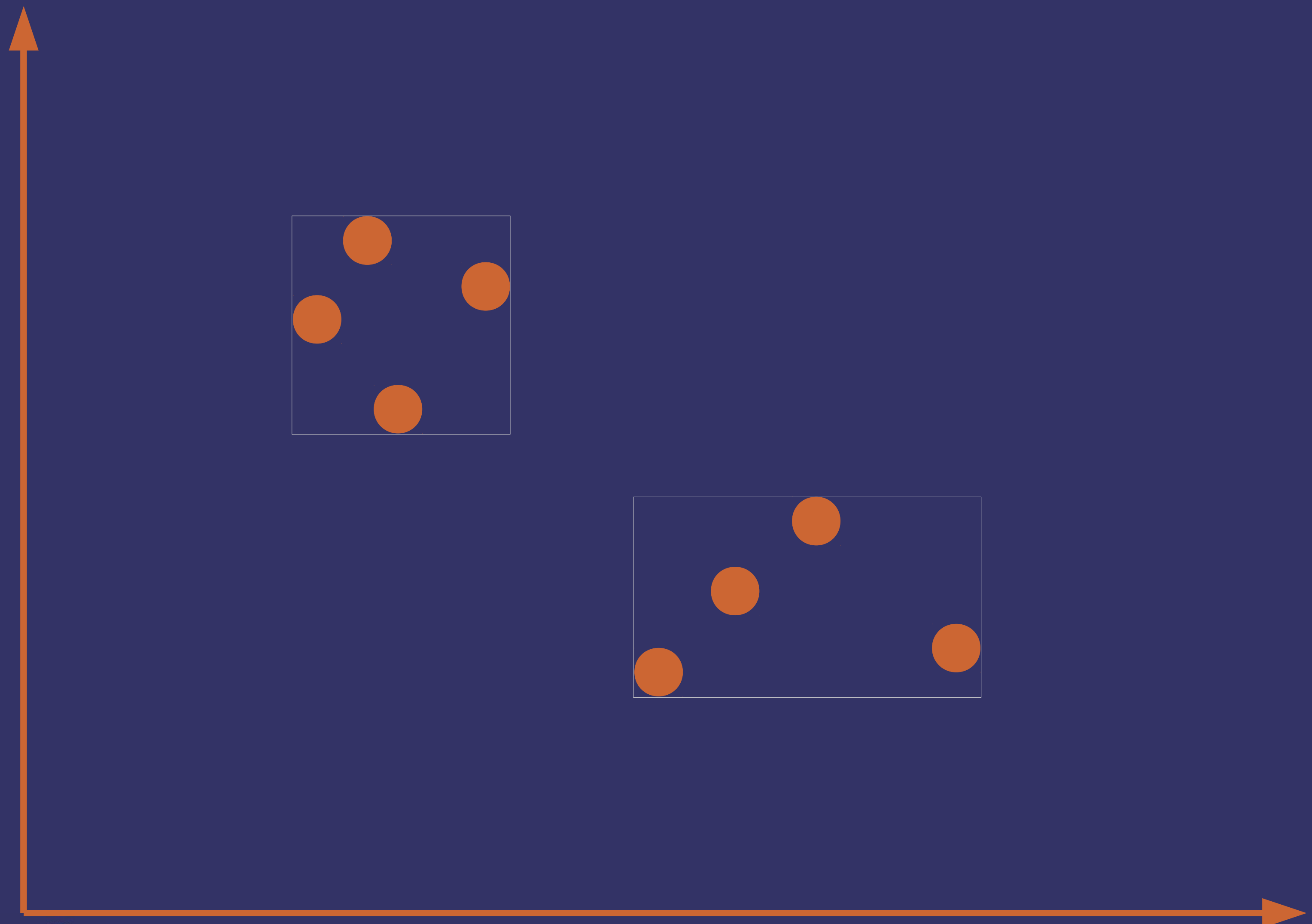
Desirable BV Characteristics

- Easy to transform
  - If the object moves, its BV needs to move. If moving the BV is too expensive, it may cancel out the speed-up.
- Inexpensive to store
  - If the BV requires too much space to store or too much time to access, it can negatively impact performance.

Axis-Aligned Bounding Box

- AABB is probably the most common bounding volume
  - Just an n-dimensional box with sides parallel to the principle axes that encloses all the points

Axis-Aligned Bounding Box



4-May-2010

Axis-Aligned Bounding Box

- Three common representations
  - Easy to translate between them
  - Which is used depends on the source data and the usage of the BV

Axis-Aligned Bounding Box

```
class aabb_min_max {
    // Points such that for every point P in the
    // object:
    //     (min.x <= P.x <= max.x)
    //     && (min.y <= P.y <= max.y)
    //     && (min.z <= P.z <= max.z)
    GLUvec4 min;
    GLUvec4 max;
};
```

Axis-Aligned Bounding Box

```
class aabb_min_diameter {
    // Points such that for every point P in the
    // object:
    //      (min.x <= P.x)
    //      && (min.y <= P.y)
    //      && (min.z <= P.z)
    GLUvec4 min;

    // Dimensions of the box in each direction
    GLUvec4 diameter;
};
```

Axis-Aligned Bounding Box

```
class aabb_center_radius {
    // Center of the bounding box
    GLUvec4 center;

    // Radius of the box in each direction
    GLUvec4 radius;
};
```

AABB Creation

- Trivial $O(n)$ problem:
  - Scan all points tracking minimum and maximum value in each dimension

AABB Update

- Translation is trivial
  - Rotation is problematic
- Three common techniques:
  - Recalcuation
  - AABB of an AABB
  - Hill climbing

AABB Update

- Recalculation:
  - Transform source data, calculate new AABB
- Advantages / disadvantages?

AABB Update

- Recalculation:
  - Transform source data, calculate new AABB
- Advantages / disadvantages?
  - Creates a tight-fitting AABB
  - O(n) per transformation is probably much too slow
    - Can speed up by using only points on the convex hull

AABB Update

- Hill climbing:
  - Track the extreme points of the object
  - To update, check neighboring points for new extrema
- Advantages / disadvantages?

AABB Update

- Hill climbing:
  - Track the extreme points of the object
  - To update, check neighboring points for new extrema
- Advantages / disadvantages?
  - Creates a tight-fitting AABB
  - Average case performance is good
    - Requires precalculation of convex hull
    - Requires data structure to store connectivity amoung points on hull

AABB Update

- AABB of AABB:
  - Calculate AABB of base orientation of object
  - Apply transformations to object and AABB
  - Calculate AABB of transformed AABB
- Advantages / disadvantages?

AABB Update

- AABB of AABB:
  - Calculate AABB of base orientation of object
  - Apply transformations to object and AABB
  - Calculate AABB of transformed AABB
- Advantages / disadvantages?
  - Creates a loose-fitting AABB
  - Very fast!
- This is probably the most commonly used technique

Oriented Bounding Boxes

- Arbitrarily oriented box that encloses the object
  - Can lead to much tighter bounding volume
- How would you represent an OBB?

Oriented Bounding Boxes

```
class obb_base_vectors {
    // Base point of box
    GLUvec4 base;

    // X, Y, and Z axes
    GLUvec4 axis[3];
};
```

Oriented Bounding Boxes

```
class obb_basis_radius {
    // Radius in each direction
    GLUvec4 radius;

    // Transformation to the OBB's coordinate
    // system
    GLUmat4 basis;
};
```

4-May-2010

OBB Creation

- One common method:
  - Calculate 3D convex hull
    - One of the OBB faces must be coplanar with a face of the convex hull
  - For each face of the 3D convex hull:
    - Project points onto its plane
    - Calculate 2D convex hull
    - Use "rotating calipers" to find minimal bounding rectangle
      - This defines one face of the OBB
    - Calculate distance of farthest point from the convex hull face
  - Use the OBB with the smallest resulting volume

OBB Creation

- References:

http://cbloomrants.blogspot.com/2009/04/04-24-09-convex-hulls-and-obb.html

OBB Update

- Trivial!
  - Apply transformation to the OBB's basis matrix

Bounding Spheres

- Sphere surrounding the object
  - Ideally it's the *minimal* sphere
  - Representation is trivial
  - Update is trivial

Bounding Sphere Creation

- Generating a good sphere is a non-trivial exercise
  - Brute-force is $O(n^5)$
  - Statistical methods can produce a good approximation in $O(n)$
  - A recursive method can produce minimum sphere in $O(n)$, but a robust implementation is complex.
  - An iterative approach can get within 5% of minimum in $O(n)$, but has a higher constant factor.

Bounding Sphere Creation

- Generating a good sphere is a non-trivial exercise
  - Brute-force is $O(n^5)$
  - Statistical methods can produce a good approximation in $O(n)$
  - A recursive method can produce minimum sphere in $O(n)$, but a robust implementation is complex.
  - An iterative approach can get within 5% of minimum in $O(n)$, but has a higher constant factor.

We won't talk about these methods today

Bounding Sphere Creation

- Brute-force:
  - A plane is defined by 3 non-colinear points
  - A sphere is defined by 3 points on a plane and one point not on the plane
    - i.e., a tetrahedron
  - Consider the sphere defined by all combinations of 4 non-coplanar points, keep the smallest that contains all the points.

Bounding Sphere Creation

- Ritter's algorithm:
  - Given an initial guess that is too small, can find bounding sphere within 10% of minimum
  - Easy to understand and easy to implement
    - I did a version in 68000 assembly language many years ago

Bounding Sphere Creation

```
void bounding_sphere(Sphere &sphere, GLUvec4 *p, unsigned num)
{
    float r_squared = sphere.radius * sphere.radius;

    for (unsigned i = 0; i < num; i++) {
        const GLUvec4 d = p[i] – sphere.center;
        const float dist_squared = gluDot3(d, d);

        if (dist_squared > r_squared) {
            const float dist = sqrt(dist_squared);
            const float r = (sphere.radius + dist) / 2.0f;
            const float k = (r – sphere.radius) / dist;

            sphere.radius = r;
            sphere.center += d * k;
            r_squared = r * r;
        }
    }
}
```

Bounding Sphere Creation

- What's the big assumption?

Bounding Sphere Creation

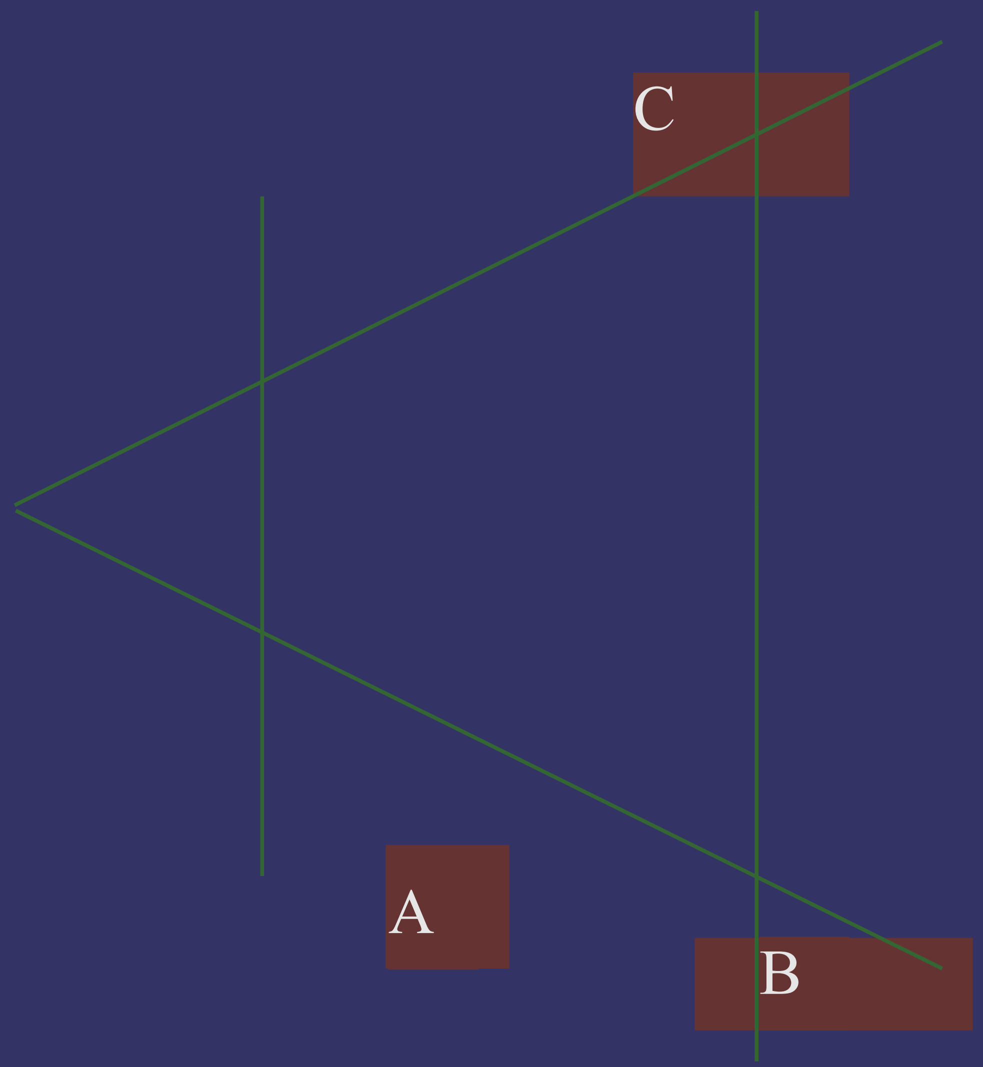
- What's the big assumption?
  - That we have a good way to come up with an initial sphere
    - The initial sphere must be a little bit too small
    - The better the initial sphere, the better the final sphere

Bounding Sphere Creation

- What's the big assumption?
  - That we have a good way to come up with an initial sphere
    - The initial sphere must be a little bit too small
    - The better the initial sphere, the better the final sphere
- Apply the algorithm iteratively
  - Generate a sphere from an AABB
  - Apply Ritter's algorithm
  - Shrink the output sphere
  - Apply again adding the points in random order

AABB / Frustum Intersection

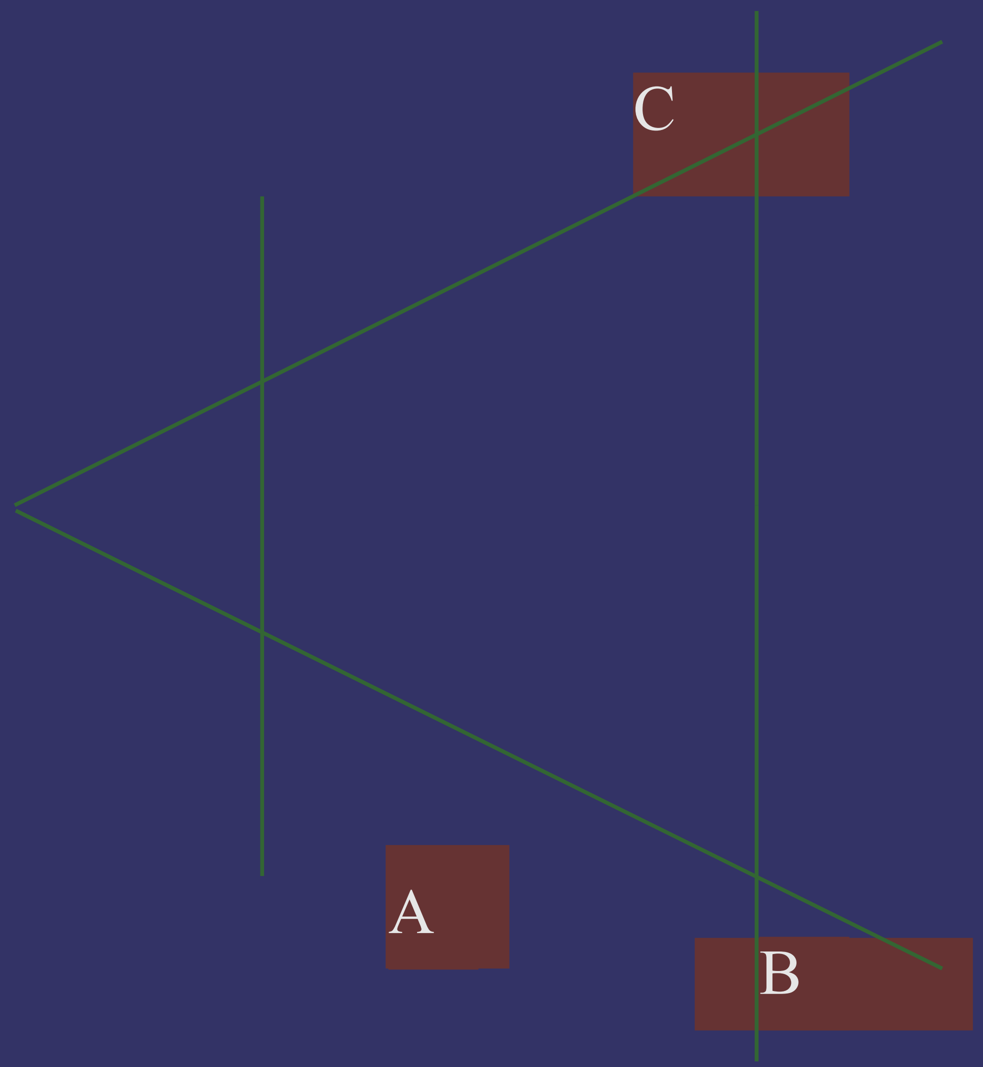
- Test each corner of the box. If all corners are outside the frustum, then box is outside.



4-May-2010

AABB / Frustum Intersection

- ~~Test each corner of the box. If all corners are outside the frustum, then box is outside.~~ Wrong!!!
- If all corners are on positive side of any one plane, then the box is outside.



AABB / Frustum Intersection

- Can we do better than testing all 8 corners?

AABB / Frustum Intersection

- Can we do better than testing all 8 corners?
  - Pick the “most positive” point and “most negative” point relative to each plane
    - Call these the *p-vertex* and the *n-vertex*
  - Test just those points
    - If both are on the same side of the plane, then **all** of the points must be on that same side

AABB / Frustum Intersection

- Finding p-vertex and n-vertex:
  - Look at the signs of the components of the plane's normal
  - The signs determine which corner the normal points towards
    - Example: If the normal signs are { +, +, - }, then the p-vertex is { box.radius.x, box.radius.y, -box.radius.z }
    - The n-vertex is always the opposite corner

AABB / Frustum Intersection

```
int frustum_aabb(Plane *planes, Aabb &aabb)
{
    bool intersect = false;
    for (unsigned i = 0; i < 6; i++) {
        const GLUvec4 vn =
          get_negative_far_point(planes[i], aabb);
        if (gluDot3(vn, planes[i].n) + planes[i].d > 0)
            return OUTSIDE;

        const GLUvec4 vp =
          get_positive_far_point(planes[i], aabb);
        if (gluDot3(vp, planes[i].n) + planes[i].d > 0)
            intersect = true;
    }

    return (intersect) ? INTERSECTING : INSIDE;
}
```

AABB / Frustum Intersection

- References:
  - http://www.ce.chalmers.se/~uffe/vfc_bbox.pdf
  - http://www.ce.chalmers.se/~uffe/vfc.pdf

OBB / Frustum Intersection

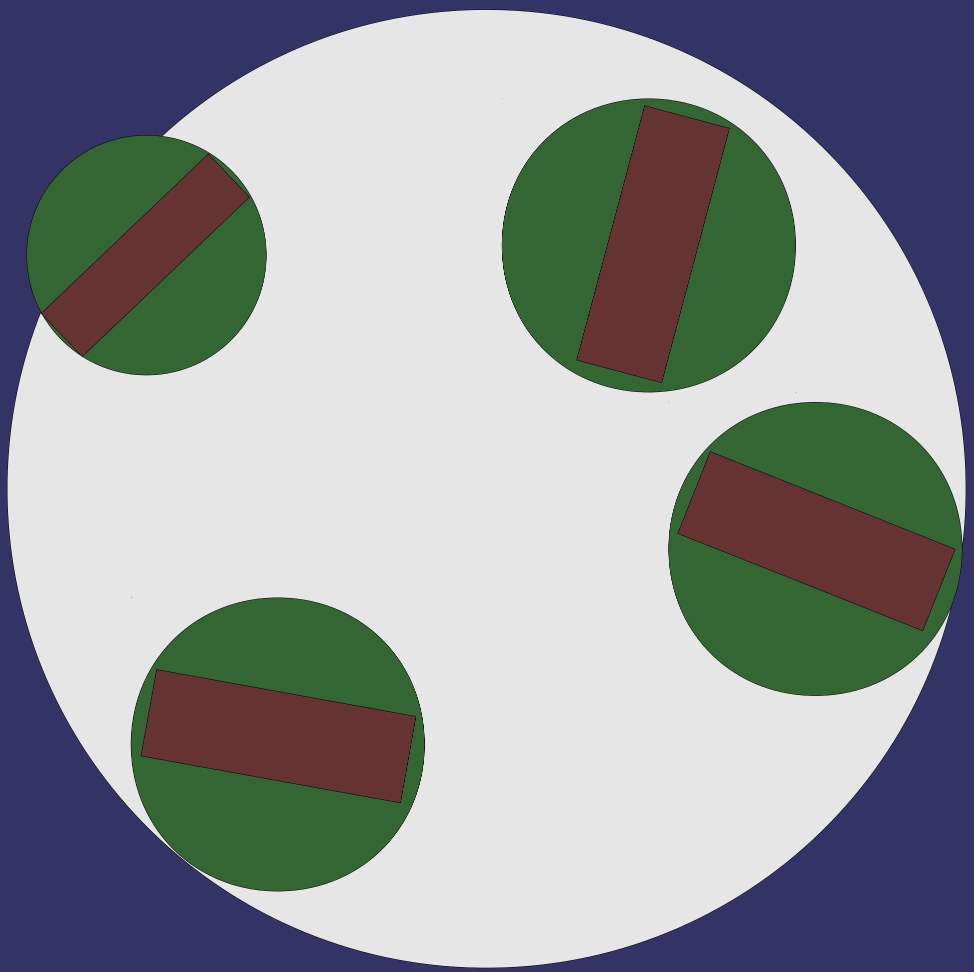
- Same!
  - Transform the frustum to the coordinate space of the OBB
  - This effectively transforms the OBB to an AABB

BV Hierarchies

- Bounding volume containing bounding volumes containing bounding volumes, etc.
  - Arrange the BVs in a tree-like structure
  - Sibling BVs may occupy overlapping space

BV Hierarchies

- Parent-child property:
  - Each parent BV contains its child BVs
  - Not required, but makes somethings easier
    - Parent BV need only contain objects in child BVs
    - Top level circle (right) contains all boxes but not all sub-circles.



Desirable BVH Characteristics

- Nodes within a subtree should be “near” each other
  - Farther down the tree, the nodes should be closer

Desirable BVH Characteristics

- Nodes within a subtree should be “near” each other
  - Farther down the tree, the nodes should be closer
- Each node should be tight-fitting
  - Just like non-hierarchical bounding volumes

Desirable BVH Characteristics

- Nodes within a subtree should be “near” each other
  - Farther down the tree, the nodes should be closer
- Each node should be tight-fitting
  - Just like non-hierarchical bounding volumes
- Nodes near the root are more important than nodes near the leaves
  - Trivial reject (or trivial accept) as many objects as possible as with as little work as possible

Desirable BVH Characteristics

- Minimal overlap of sibling nodes
  - Overlap can force traversal of multiple subtrees

Desirable BVH Characteristics

- Minimal overlap of sibling nodes
  - Overlap can force traversal of multiple subtrees
- Hierarchy should be balance w.r.t. node structure and content
  - Balanced structure just like regular search trees
  - Balanced content (i.e., number of objects in nodes) allows earlier trivial rejection

Desirable BVH Characteristics

- Minimal overlap of sibling nodes
  - Overlap can force traversal of multiple subtrees
- Hierarchy should be balance w.r.t. node structure and content
  - Balanced structure just like regular search trees
  - Balanced content (i.e., number of objects in nodes) allows earlier trivial rejection
- Worst-case performance should not be much worse than average-case performance
  - Avoid stuttering framerates

Desirable BVH Characteristics

- Generate *without* human intervention
  - Automatically generate without artist or programmer guiding the process

Desirable BVH Characteristics

- Generate *without* human intervention
  - Automatically generate without artist or programmer guiding the process
- Memory overhead should be low
  - Just like non-hierarchical bounding volumes

BVH Creation

- Three common strategies:
  - Insertion
  - Top-down
  - Bottom-up

BVH Creation – Top-Down

- Start with single BV and recursively subdivide
  - Easy to implement
  - Doesn't result in optimal BVH

BVH Creation – Top-Down

```
BVHNode *build_BVH(Entity *e, int num_e)
{
    BoundingVolume *bv = new BoundingVolume(e, num_e);
    BVHNode *node = new BVHNode(bv);

    if (num_entity < threshold) {
        node->is_leaf = true;
    } else {
        int first_half_count = divide_entities(e, num_e);
        node->child[0] = build_BVH(& e[0],
            first_half_count);
        node->child[1] = build_BVH(& e[first_half_count],
            num_e - first_half_count);
    }

    return node;
}
```

BVH Creation – Top-Down

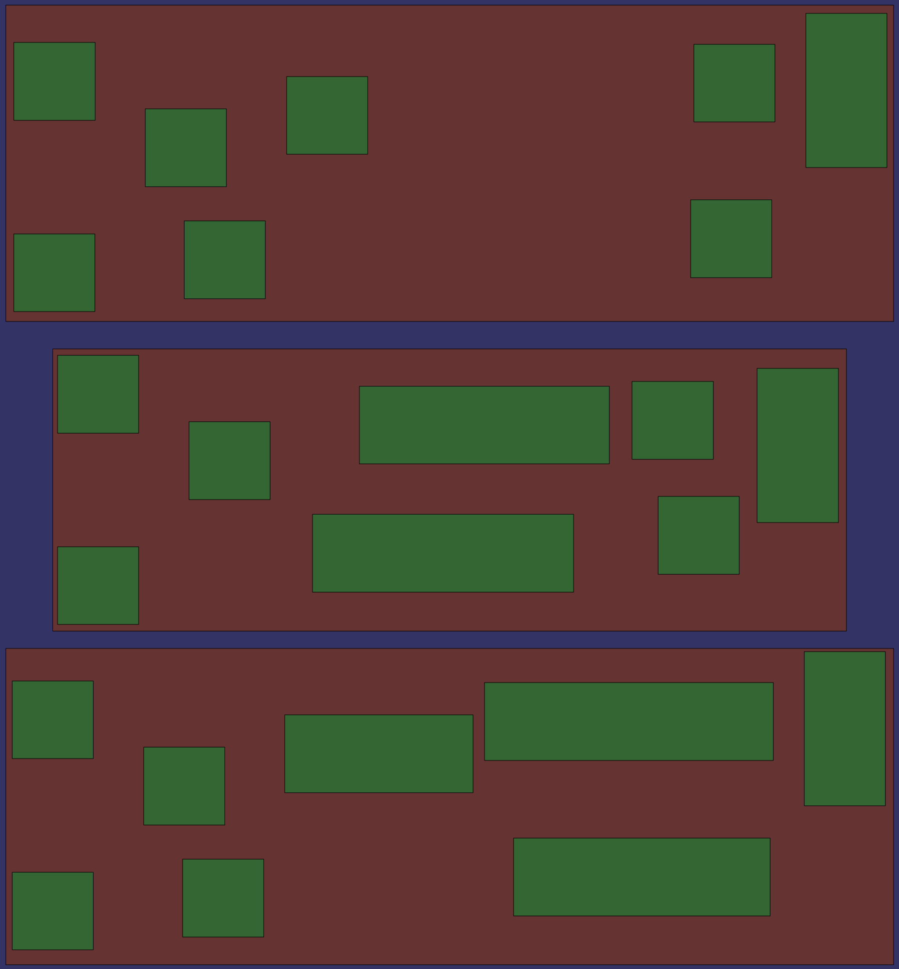
- The key element is `divide_entities`
  - As coded, assumes each entity is in exactly one set
  - *Not* the only strategy
- How do we decide where to divide the set?

BVH Creation – Top-Down

- Median-cut is a common strategy
  - Select an axis
    - Longest axis of the BV being partitioned is a common choice
  - Project all entities onto this axis
  - Sort projected entities by position
  - First half goes in the first node, second half goes in the second node

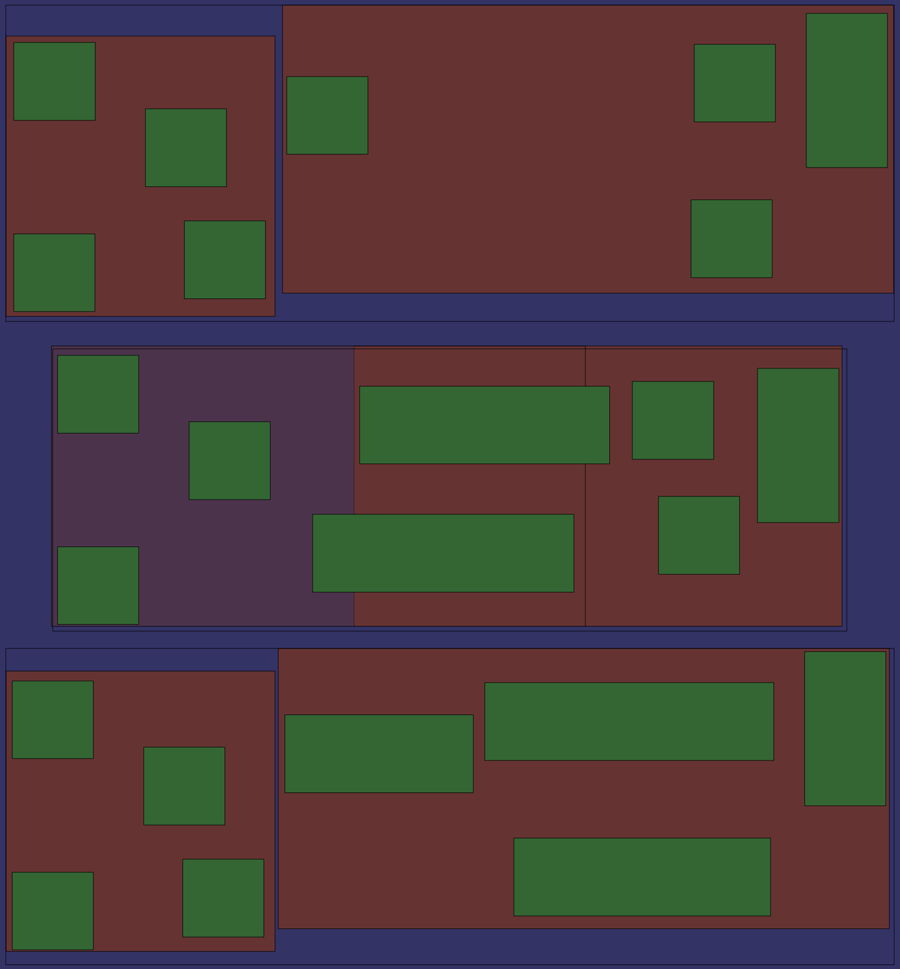
BVH Creation – Top-Down

- Median-cut is easy to implement, but it poorly partitions some sets:



BVH Creation – Top-Down

- Median-cut is easy to implement, but it poorly partitions some sets:



Too much empty space

Too much overlap

Unbalanced node sizes

BVH Creation – Top-Down

- Other heuristics:
  - Minimize sum of volumes
  - Minimize largest volume
  - Minimize overlap volume
  - Maximize child node separation
- No single heuristic is perfect
  - Implement a primary heuristic and adjust choice if secondary heuristic scores very poorly
  - Repeat for all heuristics or until a heuristic passes without adjustment

BVH Creation – Top-Down

- Infinite number of possible partition axes
  - Similar to the problem of selecting the basis of OBB
- Common choices:
  - Aligned axes of BV
  - Axes of parent BV
  - Axis through most distant points
  - Axis of greatest variance

BVH Creation – Top-Down

- Once an axis is selected, a split-point must also be selected
  - Median of projected object centroids
  - Mean of projected object centroids
  - Median of projected BV extents
  - Pick best of n evenly spaced points along axis

BVH Creation – Bottom-Up

- Repeatedly merge individual BVs:
  - Create a BV for each object
    - Store in an "active" BV list
  - Select 2 or more BVs to merge
    - Remove old BVs from active list
    - Add new, merged BV to active list
  - Lather, rinse, repeat until only one BV remains
- Tradeoffs:
  - Often much, much slower
  - More complex implement
  - Usually results in *much* better hierarchies

BVH Creation – Bottom-Up

- The key element is the algorithm for node selection

BVH Creation – Bottom-Up

- The key element is the algorithm for node selection
- Obvious, brute-force approach: search active list for pair of nodes that form least-volume BV
  - $O(n^2)$ for the search which is repeated $(n\text{-}1)$ times... $O(n^3)$ for the lose. :(

BVH Creation – Bottom-Up

- The key element is the algorithm for node selection
- Obvious, brute-force approach: search active list for pair of nodes that form least-volume BV
  - $O(n^2)$ for the search which is repeated $(n-1)$ times... $O(n^3)$ for the lose. :(

Other heuristics can also be used

BVH Creation – Bottom-Up

- Use the brute-force method as basis for an improved method:
  - For each node, determine the best node for it to pair with
    - Store both nodes with heuristic score in a priority queue
  - Loop, removing the head from the queue:
    - Validate stored size
      - May have changed if either node was already removed
    - If size is still smallest, calculate pairing for new node and add to queue
    - Otherwise, re-insert the original node in the queue

BVH Creation – Insertion

- Find location to insert node with least cost
  - Heuristic is usually along the lines of volume added to BV and all parent BVs
  - Large objects will be added near the root, small objects will be added near the leaves
  - Far away (isolated) objects will be added near the root

BVH Creation – Insertion

- Common insertion strategies:
  - Depth first:
    - At each step, pick the child with the least cost.
    - Recur on its children
    - Search cost is $O(\ln n)$ with n searches → $O(n \ln n)$
  - Guided breadth first:
    - Keep track of cost at each visited depth, recur on branch with current best cost
    - Worst-case search cost is $O(n)$ → $O(n^2)$
      - Average case is still $O(n \ln n)$
    - Results in much better tree
      - Uses global information instead of just local information

Next week...

- Texture mapping, part 1

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.