Data Structures & Algorithms for Geometry

- Agenda:
  - Data structures for polygons
    - Winged-edge
    - Quad-edge
    - Star-vertex
  - Convex Hulls in 2D
    - Naive
    - Insertion
    - QuickHull

Desirable Mesh Representation Properties

- Low storage space
  - We typically want to acceleration operations on large data sets. If the storage requirement is too high, it can cause various performance problems.

Desirable Mesh Representation Properties

- Low storage space
  - We typically want to acceleration operations on large data sets. If the storage requirement is too high, it can cause various performance problems.
- Simplicity
  - The mesh is the key to many algorithms, if the implementation is too complex, it may hide subtle bugs.

Desirable Mesh Representation Properties

- Low storage space
  - We typically want to acceleration operations on large data sets. If the storage requirement is too high, it can cause various performance problems.
- Simplicity
  - The mesh is the key to many algorithms, if the implementation is too complex, it may hide subtle bugs.
- Fast retrieval of adjacency information
  - Need to know which polygons, vertexes, and edges are connected to each other.

Desirable Mesh Representation Properties

- Ease of manipulation
  - Adding and removing points should not be too expensive.

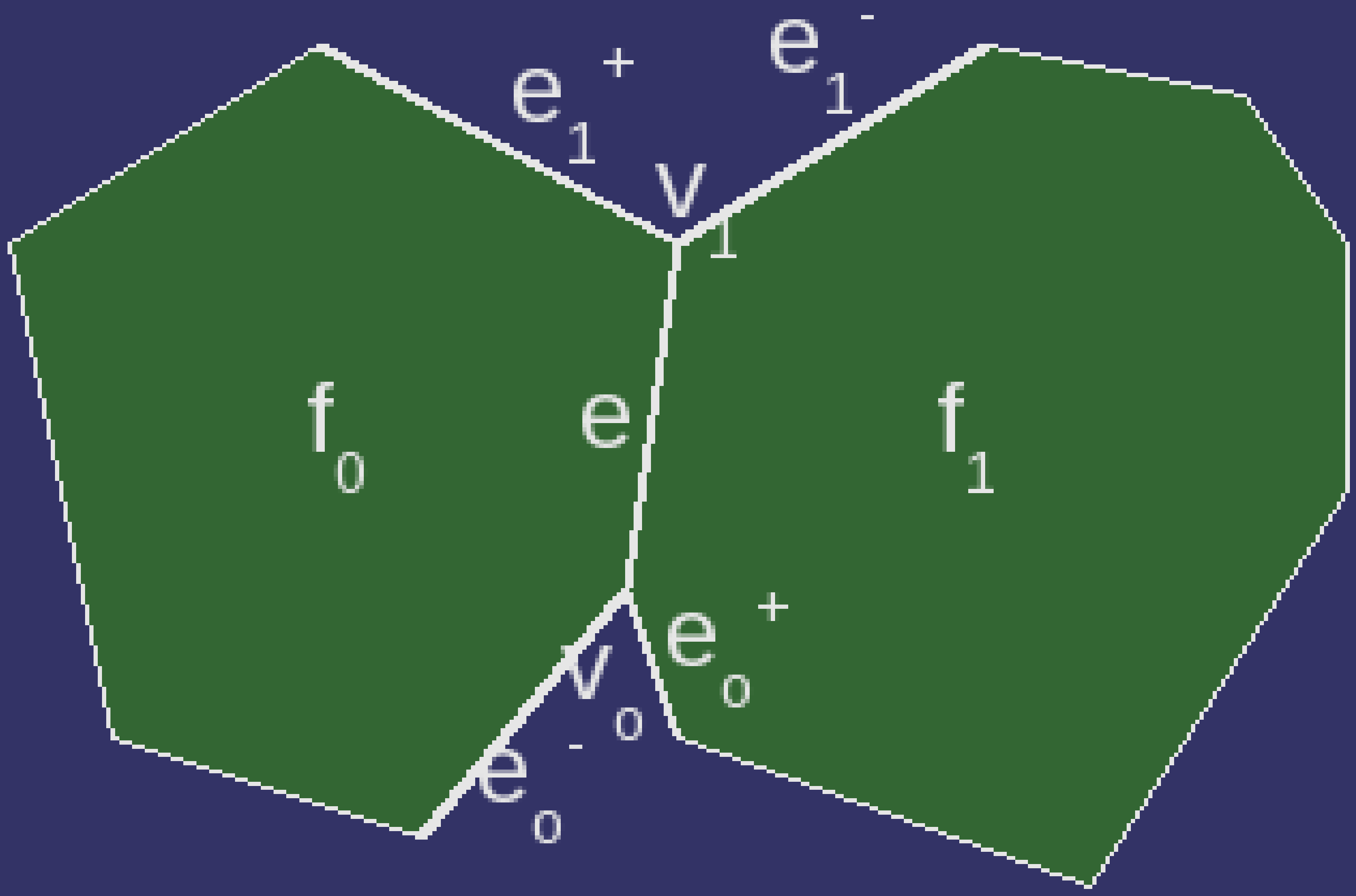
Desirable Mesh Representation Properties

- Ease of manipulation
  - Adding and removing points should not be too expensive.
- Scalability
  - May want to trade data size for performance per the needs of the application at hand.

Winged-Edge

- The *original* mesh structure to store connectivity information.
- As the name implies, the focus is the edge.
  - Each vertex stores a pointer to one of the edges radiating from it.
  - Each polygon stores a pointer to one of its edges.
  - Each edge has 8 pointers:
    - Pointers to each of its vertexes.
    - Pointers to each of its polygons.
    - Pointers to the 4 connecting edges.

Winged-Edge (cont.)



Quad-Edge

- Slightly more complex, but simplifies many operations.
  - Allows some degenerate (but useful) situations such as both end-points of an edge being the same.
- Each edge is part of 4 circular lists:
  - List of edges for each end point.
  - List of edges for each face.
  - Each edge, therefore, has 4 “next” pointers.

Quad-Edge (cont.)

- Vertex and face structures are minimal.
  - Each vertex stores a pointer to one of the edges radiating from it.
  - Each polygon stores a pointer to one of its edges.

Star-vertex

- Instead of focusing on the edge, this structure focuses on the vertex.
  - Edges and faces aren't explicitly stored *at all*.
- Each vertex stores an array of pointers to its neighbors.
  - The neighbor stores a pointer to the next vertex.
  - It also stores the index in the next vertex's neighbor array that is in the same polygon.

Star-vertex (cont.)

```
struct Neighbor {
    Vertex *v;
    unsigned next;
};

struct Vertex {
    point position;
    unsigned num_neighbors;
    struct Neighbor *neighbors;
};

struct Mesh {
    unsigned num_vertexes;
    struct Vertex *vertexes;
};
```

References

http://graphics.ucmerced.edu/publications/2001_JGI_Kallmann.pdf

http://en.wikipedia.org/wiki/Quad-edge

Break

Convex Hulls in 2D

- What's the obvious, brute force method?

Convex Hulls in 2D

- What's the obvious, brute force method?
  - For each group of 3 non-colinear points:
    - Test each remaining point against the triangle.
    - If the point is inside, mark it as not on the hull.
  - Each point not marked as not-on-the-hull, is on the hull.
- How *slow* is this?

Convex Hulls in 2D

- What's the obvious, brute force method?
  - For each group of 3 non-colinear points:
    - Test each remaining point against the triangle.
    - If the point is inside, mark it as not on the hull.
  - Each point not marked as not-on-the-hull, is on the hull.
- How *slow* is this?
  - $O(n^4)$

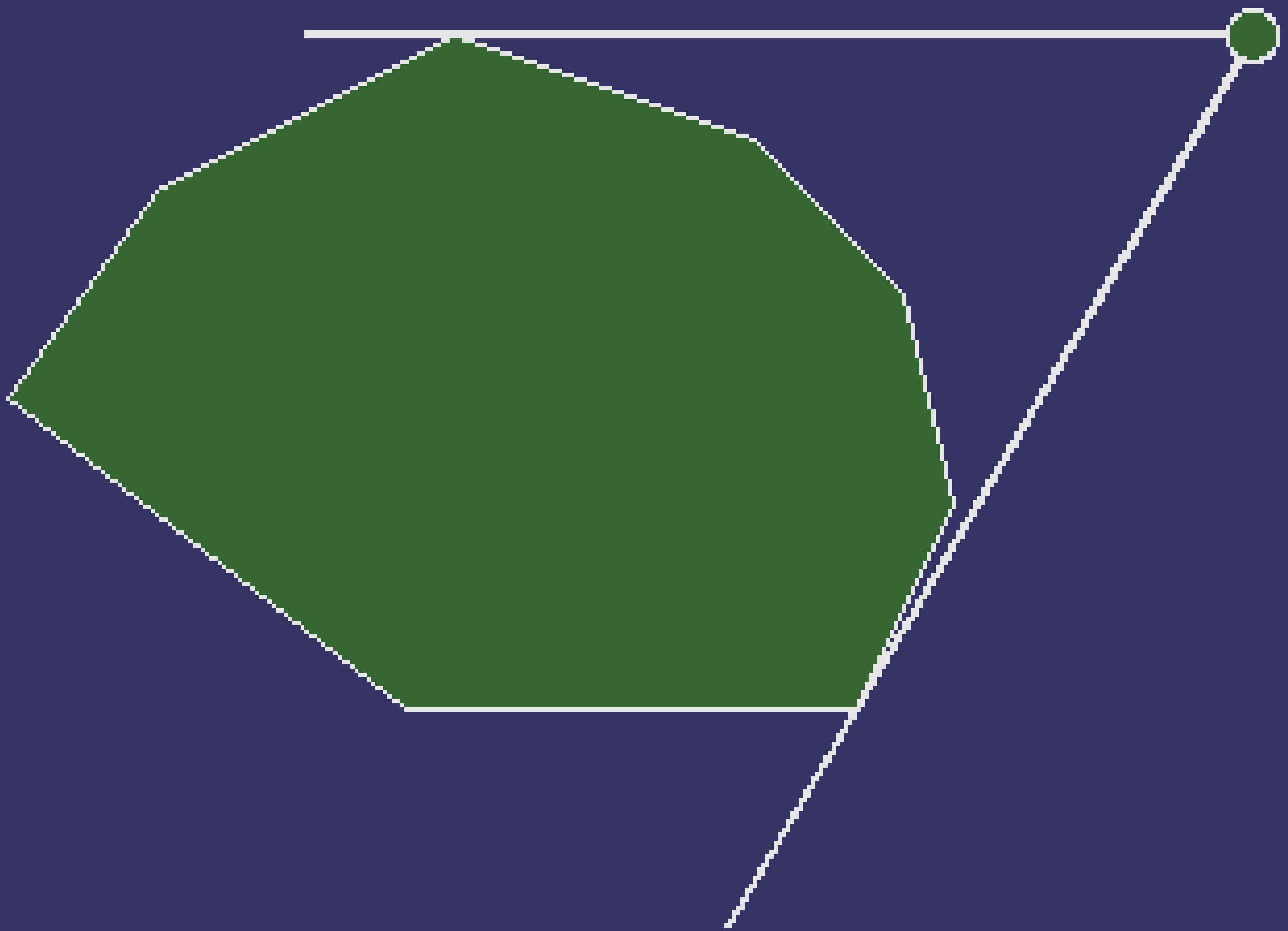
Incremental Hull in 2D

- Assume we already have a partial hull. Can we incrementally add points?

Incremental Hull in 2D

- Assume we already have a partial hull. Can we incrementally add points?
- Determine which pair of points on the hull for a tangent line with the new point.

Tangent Lines



Incremental Hull in 2D

- Assume we already have a partial hull. Can we incrementally add points?
- Determine which pair of points on the hull for a tangent line with the new point.
  - If p_{new} is to not on the same side of (p_{i-1}, p_i) and (p_i, p_{i+1}), then p_i is a tangent point.
  - If there are no tangent points, then p_{new} is inside the existing hull.
  - If we know p_i and p_j are tangent points, we know where add p_{new} *and* which points to remove.

Incremental Hull in 2D

- As-is, this algorithm in $O(n^2)$.
  - How can we make it O(n log n)?

Incremental Hull in 2D

- As-is, this algorithm in $O(n^2)$.
  - How can we make it O(n log n)?
- If we sort the points on the hull by their X coordinate...
  - Start the search for tangent points with the point with the nearest X coordinate.
  - This reduces the search for tangent points from O(n) to O(log n).
  - Total run-time is dominated by the sort step. Sorting is O(n log n).

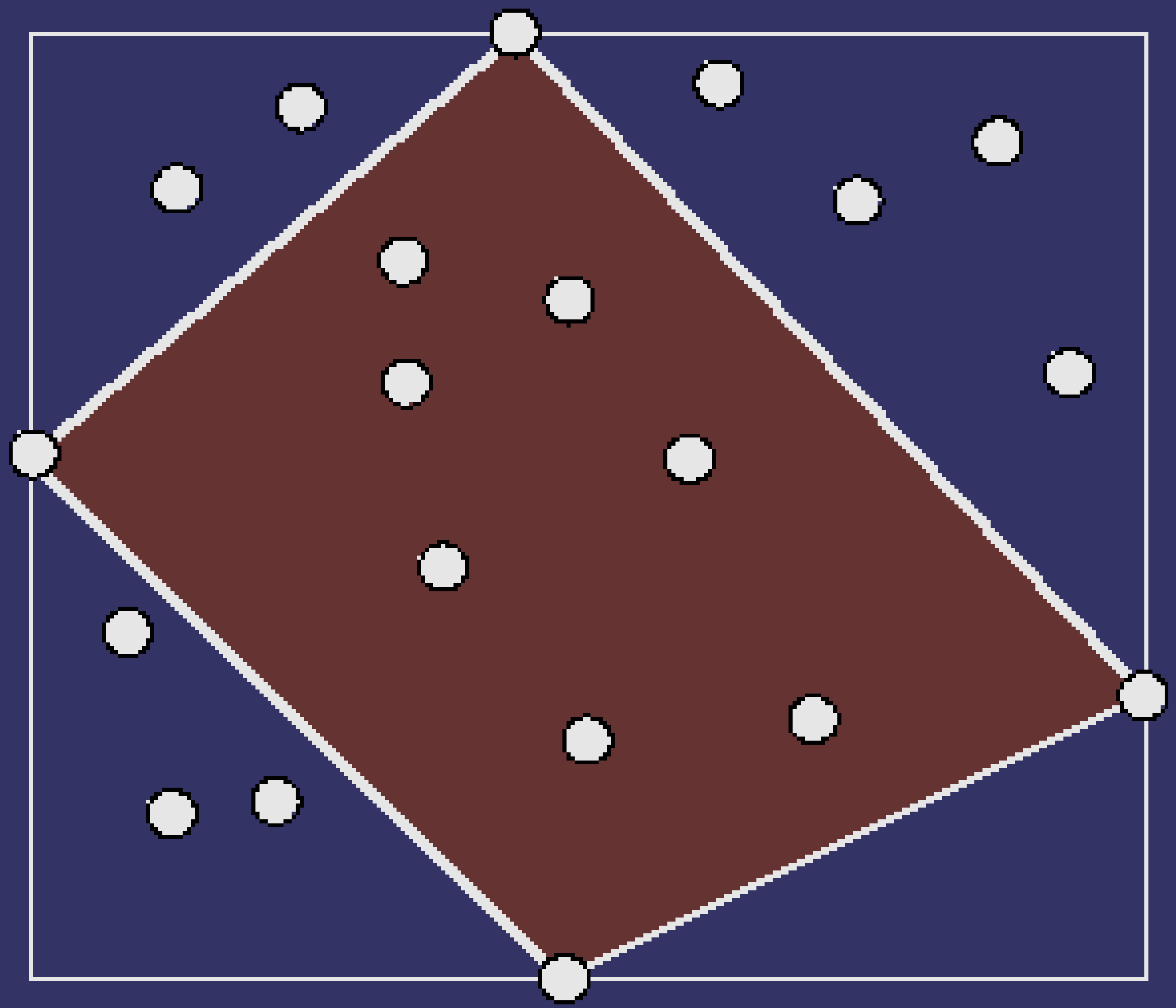
QuickHull in 2D

- QuickHull is named because of similarities to the QuickSort algorithm.
  - Like qsort, it is $O(n \log n)$ in the *average* case, and $O(n^2)$ in the worst case.
  - Like qsort, its worst case is a seemingly trivial case.
- Algorithm has two distinct phases.
  - First phase prepares the data for the second phase.
  - Second phase is recursive.

QuickHull: phase 1

- Calculate the extreme quadrilateral of the points
  - Calculate the AABB.
  - The points on the AABB define the extreme quad.
    - If a point is at the corner of the AABB, it may be an extreme triangle.
- Divide the points into 5 groups:
  - Points outside each of the 4 edges of the extreme quad.
  - Points inside the extreme quad.

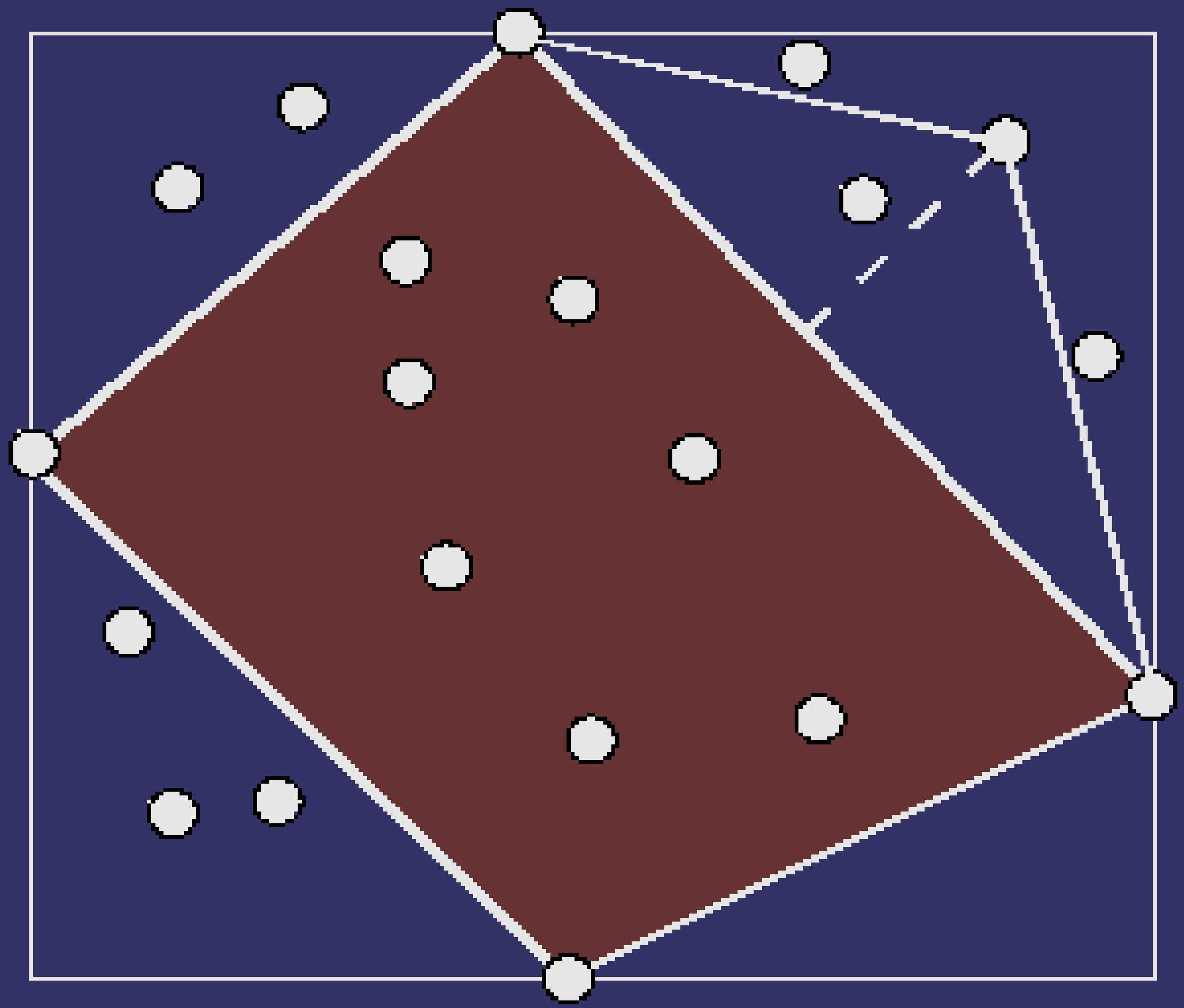
QuickHull: phase 1



QuickHull: phase 2

- For each partitioning line segment
  - Find the point that is the farthest outside the line segment. This point forms a triangle with the existing segment (2 points)
  - Divide the group of points outside the segment into 3 groups:
    - The points outside each edge of the triangle.
    - The points inside the triangle.
  - Repeat phase 2 on each group of points outside the triangle.

QuickHull: phase 2



QuickHull Performance

- What makes it fast?

QuickHull Performance

- What makes it fast?
  - Being able to cull many points at each step.

QuickHull Performance

- What makes it fast?
  - Being able to cull many points at each step.
- What makes it slow? Or...what is the worst case?

QuickHull Performance

- What makes it fast?
  - Being able to cull many points at each step.
- What makes it slow? Or...what is the worst case?
  - *Not* being able to cull many points at each step.
  - We can't cull any points at any step if the original point set defines a convex hull.
    - Just like qsort! The worst case there is trying to sort a sorted list.

Break

Next week...

- Space partitioning
  - Uniform grids
  - Octrees (one of my favs)
  - k-d trees
- Quiz #2

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM 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.