Convex Hulls in 3D

- Collision detection:

Definitions

- The smallest volume polyhedral that contains a set of points in 3D.
  - Facets
  - Edges
  - Vertices
- Generalizes to higher dimensions.

Size

- Theorem 11.1 + Corollary 11.2:
  - A convex hull of n points in 3D has at most
    - n vertices (of course),
    - 3n-6 edges, and
    - 2n-4 facets.
  - It is "planar".
  - Euler's formula.
Computing a Convex Hull in 3D

- Incremental Algorithm
  - Add points one at a time and maintain a current convex hull.
  - Use DCEL:s to represent the convex hull.
- Start with four (4) input points that form a pyramid.
  - Should these and all other points lie in the same plane, compute a 2D convex hull as in Chapter 1!
  - $O(n \log n)$ time.
- Then
  - add new facets for each point considered that lie outside the current hull.
  - Delete old facets that end up inside.

Adding One Point

Finding a horizon

- The horizon can be found by traversing the current convex hull.
  - Each edge of the horizon bounds two faces of which just one is visible from $p_r$.
- Visible:
  - If a face is visible to $p_r$ they are in conflict.
  - The face has to go when we consider $p_r$. 

Horizon
Deleting/Adding Facets

- Remove visible faces.
  - Easy since, by construction, each facet is oriented counterclockwise when seen from the outside of the convex hull.
- Insert new triangular facets for each edge in the horizon unless the new face would be coplanar with an existing face “behind” the horizon, in which case the two are merged.

How to Find the Visible Facets

- We get an algorithm expected to be faster by using two clever tricks:
  1) Randomization! Permute the input.
     - As we have done before.
     - Lowers the expected time to $O(n \log n)$.
  2) Use a conflict graph to add points and find which faces to remove fast.

Worst-Case Analysis

- Constructing the initial pyramid takes $O(1)$ time.
- Processing each of the remaining n-4 points means:
  - Finding the horizon,
  - removing visible faces, thereby “opening it up”, and
  - adding new triangles to “close” the convex hull again.
- Since each of these steps can be performed in $O(n)$ time, the total time becomes $O(n^2)$.
  - Is this the best we can do…?

Conflict Graphs

- Bipartite.
  - A node per point not yet considered.
  - A node per facet in the current convex hull.
- Arcs between points and facets that are in conflict (“visible”).
- We maintain the conflict graph during the computation.
  - After a point has been considered, we remove its node.
- $P_{conflict}(f)$ is the list of points in conflict with $f$.
- $F_{conflict}(p_i)$ is the list of faces in conflict with $p_i$. 

Points left to insert.
Using the Conflict Graph

- The facets visible to a point are its neighbors in the graph.
- Likewise: The points visible from a facet are its neighbors in the graph.
- The conflict sets of points and facets can be extracted in time linear in their sizes.

Adding a point \( p_t \)

- Using the conflict graph, we get all visible facets and the horizon.
  - \( F_\text{conflict}(p_t) \)
- Remove all visible facets, but keep those next to the horizon in a separate data structure for a while.
  - Update the conflict graph accordingly.
  - For each face \( f \) that is deleted, and for each \( p \) in \( P_\text{conflict}(f) \), delete \( f \) from \( F_\text{conflict}(p) \).
  - If \( F_\text{conflict}(p) \) becomes empty, \( p \) is inside the hull and will never be considered again.
  - Delete the node of \( f \).
- When adding a new point – and new facets – to the convex hull, we add new nodes to the conflict graph as well.
  - The crucial point is how to construct their conflict lists.
- Note: The conflicts of facets not visible from \( p_t \) are not affected.

Constructing conflict lists

Consider adding a new face \( f \).
- \( f \) is coplanar with \( f_1 \) ("behind" the horizon)
  - ConflictList \( f = \) ConflictList \( f_1 \).
  (since a point \( p \) is seen from \( f_1 \) if and only if it is also visible from \( f \))
- \( f \) is not coplanar with \( f_1 \)
  - ConflictList \( f = \) points of ConflictList \( f_1 \) and ConflictList \( f_2 \) visible from \( f \).
  (There could be points visible from \( f_2 \) and \( f \), but not \( f_1 \), and vice versa.)

New Faces lie Next to the Horizon

- Using the conflict graph, we get all visible facets and the horizon.
  - \( F_\text{conflict}(p) \)
- Remove all visible facets, but keep those next to the horizon in a separate data structure for a while.
  - Update the conflict graph accordingly.
  - For each face \( f \) that is deleted, and for each \( p \) in \( P_\text{conflict}(f) \), delete \( f \) from \( F_\text{conflict}(p) \).
  - If \( F_\text{conflict}(p) \) becomes empty, \( p \) is inside the hull and will never be considered again.
  - Delete the node of \( f \).
- When adding a new point – and new facets – to the convex hull, we add new nodes to the conflict graph as well.
  - The crucial point is how to construct their conflict lists.
- Note: The conflicts of facets not visible from \( p_t \) are not affected.
We now want to bound the expected value of \( \deg(\mathcal{C}(p_i)) \). As usual when we analyse a randomized incremental algorithm, we first try to bound the expected number of facets created by \( \mathcal{C}(p_i) \) and their number equals the number of edges incident to \( p_i \). Then the expected degree of a vertex \( p \) is 6. Because \( \deg(p) = 6 \), the expected degree of a vertex \( p \) in \( \mathcal{C}(P_r) \) is 6. Since we treat the vertices in random order, the initial pyramid has at most degree 12 and \( p_i \) is chosen from a subset of the original set of points.

\[
\mathbb{E}[\deg(p_i, \mathcal{C}(P_r))] = 6 + \sum_{j=5}^{n} \mathbb{E}[\deg(p_j, \mathcal{C}(P_r))]
\]

\[
\leq 6 \left( \sum_{j=5}^{n} \mathbb{E}[\deg(p_j, \mathcal{C}(P_r))] \right)
\]

\[
\leq 6 \left( \sum_{j=5}^{n} \frac{3r-6}{r} \right)
\]

\[
= 6 \sum_{j=5}^{n} \frac{3r-6}{r} = 6r - 12 - 12 = 6.
\]
**Total time complexity**

- Initialization takes $O(n \log n)$.
  - If we end up having to compute a 2D convex hull.
- Creating and deleting facets takes $O(n)$ time since this is the expected number of facets created.
- To deleting $p_r$ and facets in $F_{\text{conflict}}(p_r)$ from the conflict graph along with incident arcs also takes $O(n)$ time.
- Our only headache is the time it takes to find new conflicts.

- For each edge $e$ on the horizon we use $O(|P(e)|)$ time, where $P(e) = P_{\text{conflict}}(f_1) \cup P_{\text{conflict}}(f_2)$.
- The total time is $O(\sum_e |P(e)|)$, where $e$ is an edge of a horizon during the computation.

**Lemma 11.6:**
- The expected value of $\sum_e |P(e)|$, where the summation is over all horizon edges that appear at some stage of the algorithm, is $O(n \log n)$.
- (Exactly $96n \log n$.)

**Lemma 11.7:**
- The convex hull of $n$ points in $\mathbb{R}^3$ can be computed in $O(n \log n)$ time.

**Convex Hulls in Dual Space**

- In Chapter 8 we learned about geometric duality.
- The upper (lower) convex hull of a set of points is “essentially” the lower (upper) envelope of a set of lines.

**Why “essentially”?**

- It may seem that convex hulls and intersections of half-planes are dual concepts…
- … and that an algorithm to compute the intersection of half-planes can be given as a convex hull algorithm in the dual plane.
- However, the convex hull of points is always well-defined while the intersection of half-planes could be empty – does not exist.
- Only as long as the intersection exists, the dual does.
Convex Hulls in 3D and Voronoi diagrams/Delaunay Triangulations in 2D

• Ok, so convex hulls dualizes to intersections of half-planes.
• From Chapter 7 we know that cells in Voronoi diagrams are intersections of half-planes.

Conclusion

• The Delaunay triangulation/Voronoi diagram of a set of $n$ points $(x_i, y_i)$ in the plan can be computed by computing the convex hull in 3D of the points $(x_i, y_i, x_i^2 + y_i^2)$.
  − Project the lower hull of the triangulation onto the $xy$-plane.
  − (The dual is the Voronoi diagram.)
• Moreover, the dual of the upper hull projected down onto the $xy$-plane is the furthest-point Voronoi diagram.
  − (Read also the handout you got.)
**Conclusion**

- (Lemma 11.4)
  - Given a set of \( n \) points in 3D, \textsc{ConvexHull} computes a convex hull in \( O(n \log n) \) expected time.

- Comment:
  - There is also a deterministic algorithm by Preparata and Hong that runs in \( O(n \log n) \) time, but that one is more complicated.