Lecture Outline

• Multi-view Stereo part II
  – Slides by G. Vogiatzis and L. Zhang
  – Paper by A. Collet et al. (2015)

• Introduction to Computational Geometry

• Convex Hulls
  – David M. Mount, CMSC 754: Computational Geometry lecture notes, Department of Computer Science, University of Maryland, Spring 2012
  – Slides by:
    • B. Gartner, M. Hoffman and E. Welzl (ETH)
    • M. van Kreveld (Utrecht University)
    • P. Indyk and J.C. Yang (MIT)
Extracting a Surface from Photo-consistency

• Vogiatzis et al. (PAMI 2007)
• Divide the space in voxels
• Compute the photo-consistency of each voxel
  – By robustly combining all pairwise NCC scores
• Problem: \textcolor{red}{find a minimum cost surface that separates interior from exterior of the object}
• Add term that favors large volume, otherwise solution collapses to a point
How to Solve?
Minimum Cut

Can be computed in polynomial time with Ford-Fulkerson (1956) algorithm
Three Equivalent Representations

Binary labelling

Graph partition

Continuous functional

\[ E[S] = \iint_S \rho(x) dS + \iiint_{V(S)} \sigma(x) dV \]
Extracting the Surface

• Marching cubes algorithm can extract isosurfaces
  – Matlab: [tri, pts] = isosurface(V)
  – Where V is a binary volume of 0s and 1s
Results
Results
Volumetric Stereo

- Determine occupancy, "color" of points in $V$
- Slides by L. Zhang
Discrete Formulation: Voxel Coloring

Goal: Assign RGBA values to voxels in $V$ photo-consistent with images

Discretized Scene Volume

Input Images (Calibrated)
Complexity and Computability

Discretized Scene Volume

$N^3$ voxels

$C$ colors

True Scene

Photo-Consistent Scenes

All Scenes ($C^{N^3}$)
Reconstruction from Silhouettes (C=2)

• Approach:
  • *Back-project* each silhouette
  • Intersect back-projected volumes
Volume Intersection

Reconstruction Contains the True Scene

- But is generally not the same
- In the limit (all views) we get \textit{visual hull}
  - Complement of all lines that do not intersect S
Voxel-based Algorithm

Color voxel black if on silhouette in every image

- $O(\ ? )$, for $M$ images, $N^3$ voxels
- Don’t have to search $2^{N^3}$ possible scenes!
Results (Franco and Boyer, PAMI 2009)
Properties of Volume Intersection

Pros

– Easy to implement, fast
– Accelerated via octrees

Cons

– No concavities
– Reconstruction is not photo-consistent
– Requires identification of silhouettes
Space Carving

Space Carving Algorithm

- Initialize to a volume $V$ containing the true scene
- Choose a voxel on the current surface
- Project to visible input images
- Carve if not photo-consistent
- Repeat until convergence
Which Shape do You Get?

The Photo Hull is the UNION of all photo-consistent scenes in V

- It is a photo-consistent scene reconstruction
- Tightest possible bound on the true scene
Results (Kutulakos and Seitz, IJCV 2000)
Free-Viewpoint Video

Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, Steve Sullivan
Microsoft Corporation
SIGGRAPH 2015
Pipeline
Reconstruction

PMVS: system by Furukawa and Ponce, we show last week
PSR: Poisson Surface Reconstruction (last slide last week)
Adaptive Level of Detail
Keyframe-based Mesh Tracking
Synthesized Viewpoints
Synthesized Viewpoints
Computational Geometry
Computational Geometry

• Subfield of the *Design and Analysis of Algorithms*

• Deals with efficient data structures and algorithms for geometric problems

• Only about 30 years old
Surface Reconstruction

- Digitizing 3D objects, such as the Stanford Bunny
Surface Reconstruction

• Step 1: scan the object with a laser scanner to obtain set of points in $\mathbb{R}^3$
Surface Reconstruction

• Step 2: create a triangulation to obtain set of triangles in $R^3$
Surface Reconstruction

• Step 3: process the triangulation to obtain smooth surface in $\mathbb{R}^3$
Good and Bad Triangulations in $\mathbb{R}^2$
Good and Bad Triangulations in $\mathbb{R}^2$
Collision Detection
Collision Detection

• Bounding volume heuristic:
  – Approximate the objects by simple ones that enclose them (bounding volumes)
  – popular bounding volumes: boxes, spheres, ellipsoids, ...
  – if bounding volumes don’t intersect, the objects don’t intersect, either
  – *only* if bounding volumes intersect, apply more expensive intersection test(s)
Boolean Operations

• Given two shapes, compute their
Geometric Basics

Mount - Lecture 21
Affine Geometry

• Scalars
• Points
• Free vectors or vectors
  – In contrast to linear algebra where there is no distinction between points and vectors
Affine Geometry

• Can easily derive
  – Vector subtraction
  – Scalar-vector division

• Cannot derive
  – Point-scalar multiplication
  – Point addition

• We can define *affine combination*
  
  \[
  \text{aff}(p_0, p_1; \alpha_0, \alpha_1) = \alpha_0 p_0 + \alpha_1 p_1 = p_0 + \alpha_1 (p_1 - p_0).
  \]

  Why?
Convex Combination

• If $0 \leq a_0, a_1 \leq 1$, then the operation is called convex combination
• Set of all convex combinations traces out line segment

$$\text{aff}(p_0, p_1, p_2; \alpha_0, \alpha_1, \alpha_2) = \alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2 = p_0 + \alpha_1 (p_1 - p_0) + \alpha_2 (p_2 - p_0).$$

• Set of all affine combinations of three non-collinear points generates plane
  – Affine span or affine closure
• Set of all convex combinations of three non-collinear points generates triangle
  – Convex closure
Euclidean Geometry

• Extension of affine geometry that includes inner product
• Maps two real vectors (not points) into real scalar
  – Define length of vector as square root of inner product with itself
  – Normalize vector by dividing with its length
Distance and Angle

- Distance between points: length of the vector between them
- Angle between two non-zero vectors, ranging from 0 to $\pi$

\[
\text{ang}(\vec{u}, \vec{v}) = \cos^{-1} \left( \frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| ||\vec{v}||} \right) = \cos^{-1}(\hat{u} \cdot \hat{v}).
\]
Orientation of Points

• To make discrete decisions, we would like a geometric operation on points that is analogous to the relational operations (<, =, >) with numbers.

• There does not seem to be any natural intrinsic way to compare two points in d-dimensional space,

• but there is a natural relation between ordered (d + 1)-tuples of points in d-space, which extends the notion of binary relations in 1-space:

  ➢ orientation
Orientation of Points

• Given an ordered triple of points \(<p,q,r>\)
• Positive orientation if they define counterclockwise triangle
• Negative orientation if they define clockwise triangle
• Zero orientation if they are collinear
Orientation of Points

- Formally the sign of the determinant of the points in homogeneous coordinates

\[
\text{Orient}(p, q, r) = \det \begin{pmatrix}
1 & px & py \\
1 & qx & qy \\
1 & rx & ry \\
\end{pmatrix}
\]

- 1D case: Orient(p,q)=q-p !!
- Orientation is invariant to translation, rotation and scaling by positive scale
- Sign is reversed by reflection e.g. f(x,y)=(-x,y)
Areas and Angles

• The signed area of triangle is $\frac{1}{2}$ of the determinant
  – Can be extended to any dimension
  – Volume is equal to determinant divided by $d!$

• The sine of the signed angle from vector $p-q$ to $r-q$
  – Can get cosine from inner product

\[
\sin \theta = \frac{\text{Orient}(q, p, r)}{\|p - q\| \cdot \|r - q\|}.
\]
Informal Topology

- Neighborhood
- Open
- Closed
- Unbounded
- Exterior
- Boundary
Convex Hulls

D. Mount - Lecture 3
M. Van Kreveld slides
Polygons

- simple polygon
- polygon with holes
- convex polygon
- non-simple polygon
Convex Hulls

- Simple approximation for set of points
  - Tighter than bounding box, circle or ellipse

**Convexity:** A set $K$ is *convex* if given any points $p, q \in K$, the line segment $pq$ is entirely contained within $K$.

**Convex hull:** The *convex hull* of any set $P$ is the intersection of all convex sets that contains $P$, or more intuitively, the smallest convex set that contains $P$.
  - We will denote it by $\text{conv}(P)$.
Convex Hull Problem

• Given a set of n points P in the plane, output a representation of P’s convex hull.

• The simplest representation is a counterclockwise enumeration of the vertices of the convex hull.
  – Although points of P might lie in the interior of an edge of the boundary of the convex hull, such a point is not considered a vertex.
  – We will assume that the points are in general position, and in particular, no three are collinear. Then, this issue does not arise.

• Although the output consists only of the boundary of the hull, the convex hull includes both the boundary and interior of the polygon
Illustration
Developing an Algorithm

- Property: The vertices of the convex hull are always points from the input
- Consequently, the edges of the convex hull connect two points of the input
- Property: The supporting line of any convex hull edge has all input points to one side
Slow Algorithm

**Algorithm** \texttt{SLOWCONVEXHULL}(P)

*Input.* A set \( P \) of points in the plane.

*Output.* A list \( \mathcal{L} \) containing the vertices of \( CH(P) \) in clockwise order.

1. \( E \leftarrow \emptyset \).
2. \textbf{for} all ordered pairs \((p, q) \in P \times P\) with \( p \) not equal to \( q \)
3. \hspace{1em} \textbf{do} valid \leftarrow \text{true}
4. \hspace{1em} \textbf{for} all points \( r \in P \) not equal to \( p \) or \( q \)
5. \hspace{2em} \textbf{do} if \( r \) lies left of the directed line from \( p \) to \( q \)
6. \hspace{3em} \textbf{then} valid \leftarrow \text{false}
7. \hspace{1em} \textbf{if} valid \textbf{then} Add the directed edge \( \vec{pq} \) to \( E \)
8. \hspace{1em} From the set \( E \) of edges construct a list \( L \) of vertices of \( CH(P) \), sorted in clockwise order.

**Complexity ?**
Incremental Algorithm - Graham’s Scan

• Incremental, from left to right
• First compute the upper boundary of the convex hull
  – property: on the upper hull, points appear in x-order
• Main idea: Sort the points from left to right
• Insert the points in this order, and maintain the upper hull
• Then complete the lower hull
Graham’s Scan

Observation: from left to right, there are only right turns on the upper hull
Graham’s Scan

Initialize by inserting the leftmost two points
Graham’s Scan

If we add the third point there will be a right turn at the previous point, so we add it.
Graham’s Scan

If we add the fourth point we get a left turn at the third point
Graham’s Scan

... so we remove the third point from the upper hull when we add the fourth
Graham’s Scan

If we add the fifth point we get a left turn at the fourth point
Graham’s Scan

... so we remove the fourth point when we add the fifth
Graham’s Scan

If we add the sixth point we get a right turn at the fifth point, so we just add it
Graham’s Scan

We also just add the seventh point
Graham’s Scan

When adding the eight point
... we must remove the seventh point
Graham’s Scan

... we must remove the seventh point
Graham’s Scan

... and also the sixth point
Graham’s Scan

... and also the fifth point
Graham’s Scan

After two more steps we get:
The Algorithm

**Algorithm** $\text{ConvexHull}(P)$

*Input.* A set $P$ of points in the plane.

*Output.* A list containing the vertices of $CH(P)$ in clockwise order.

1. Sort the points by $x$-coordinate, resulting in a sequence $p_1, \ldots, p_n$.
2. Put the points $p_1$ and $p_2$ in a list $L_{\text{upper}}$, with $p_1$ as the first point.
3. **for** $i \leftarrow 3$ **to** $n$
   4. **do** Append $p_i$ to $L_{\text{upper}}$.
5. **while** $L_{\text{upper}}$ contains more than two points and the last three points in $L_{\text{upper}}$ do not make a right turn
   6. **do** Delete the middle of the last three points from $L_{\text{upper}}$. 
Lower Convex Hull

Then we do the same for the lower convex hull, from right to left.
We remove the first and last points of the lower convex hull.
... and concatenate the two lists into one.
Correctness

• Does the sorted order matter if two or more points have the same x-coordinate?
• What happens if there are three or more collinear points, in particular on the convex hull?
Efficiency

• The sorting step takes $O(n \log n)$ time
• Adding a point takes $O(1)$ time for the adding-part. Removing points takes constant time for each removed point. If due to an addition, $k$ points are removed, the step takes $O(1+k)$ time
• Total time:

$$O(n \log n) + \sum_{i=3}^{n} O(1 + k_i)$$

• if $k_i$ points are removed when adding $p_i$
• Since $k_i = O(n)$, we get

$$O(n \log n) + \sum_{i=3}^{n} O(n) = O(n^2)$$
Efficiency

- Sometimes there are global arguments why an algorithm is more efficient than it seems, at first
- Global argument: each point can be removed only once from the upper hull
- This gives us the fact:

$$
\sum_{i=3}^{n} k_i \leq n
$$

- Hence:

$$
O(n \log n) + \sum_{i=3}^{n} O(1 + k_i) = O(n \log n) + O(n) = O(n \log n)
$$
Jarvis’s March Algorithm

• Builds the convex hull in $O(nh)$ time
  – $h$ is the number of vertices in the output
• “Gift-wrapping” process
• Analogous to selection sort
Jarvis’s March Algorithm

• Start with any one point on the convex hull, e.g. the lowest point
• Then find the “next” edge on the hull in counterclockwise order
  – Assuming that \( p_k \) and \( p_{k-1} \) were the last two points added to the hull, compute the point \( q \) that maximizes the angle \( p_{k-1}p_kq \)
  – \( q \) can be found in \( O(n) \) time
Jarvis’s March Algorithm

• If \( h \) is \( o(\log n) \), Jarvis’s march is better than Graham’s algorithm
• Why?
Divide and Conquer

- Analogous to merge sort
- Sort points by x coordinate, then

1. If \(|P| \leq 3\), then compute the convex hull by brute force in \(O(1)\) time and return.

2. Otherwise, partition the point set \(P\) into two sets \(A\) and \(B\), where \(A\) consists of half the points with the lowest x-coordinates and \(B\) consists of half of the points with the highest x-coordinates.

3. Recursively compute \(H_A = \text{conv}(A)\) and \(H_B = \text{conv}(B)\).

4. Merge the two hulls into a common convex hull, \(H\), by computing the upper and lower tangents for \(H_A\) and \(H_B\) and discarding all the points lying between these two tangents.
Divide and Conquer

- LowerTangent(H_A, H_B):
  (1) Let a be the rightmost point of H_A.
  (2) Let b be the leftmost point of H_B.
  (3) While (ab is not a lower tangent for H_A and H_B) do
      (a) While (ab is not a lower tangent to H_A) do a ← a.pred (move a clockwise).
      (b) While (ab is not a lower tangent to H_B) do b ← b.succ (move b counterclockwise).
  (4) Return ab.
Divide and Conquer

• “ab is not a lower tangent to $H_A$” is equivalent to $\text{Orient}(b,a,a.\text{pred}) \geq 0$
  – Vertical gap between two partial convex hulls needed for this test to hold
  – Each vertex is visited at most once
Complexity

- Partitioning points is $O(n)$ since they are sorted according to $x$
- Returning final results is $O(n)$
- Tangent computation is $O(n)$
- Running time is:

\[
T(n) = \begin{cases} 
1 & \text{if } n \leq 3 \\
1/n + 2T(n/2) & \text{otherwise.}
\end{cases}
\]

- Therefore, $T(n) = O(n \log n)$
Quick Hull

1. Find the points with minimum and maximum x coordinates, those are bound to be part of the convex hull.
2. Use the line formed by the two points to divide the set in two subsets of points, which will be processed recursively.
3. Add this line in both directions to the convex hull.
4. For each line, find the point with the maximum positive distance from the line. Form a triangle with this point and the initial endpoints which replaces the line in the convex hull.
5. The points lying inside of the triangle cannot be part of the convex hull and can therefore be ignored in the next steps.
6. Repeat the previous two steps on the two lines formed by the triangle (not the initial line).
7. Repeat until no more points are left. The recursion has come to an end and the points selected constitute the convex hull.
Convex Hulls in 3D

Slides by P. Indyk and J.C. Yang
Problem Statement

• Given P, a set of n vertices in 3D

• Return convex hull of P: CH(P), i.e. the smallest polyhedron such that all elements of P are on or in the interior of CH(P)
Complexity (Spatial)

• Complexity of CH for $n$ points in 3D is $O(n)$
• ..because the number of edges of a convex polytope with $n$ vertices is at most $3n-6$ and the number of facets is at most $2n-4$
• ..because the graph defined by vertices and edges of a convex polytope is planar
• Euler’s formula: $n - n_e + n_f = 2$
Randomized Incremental Algorithm

• Initialize the algorithm

• Loop over remaining points
  – Add $p_r$ to the convex hull of $P_{r-1}$ to transform $CH(P_{r-1})$ to $CH(P_r)$
  – [for integer $r \geq 1$, let $P_r := \{p_1, \ldots, p_r\}$ ]
Initialization

• Need a CH to start with
• Build a tetrahedron using 4 points in P
  – Start with two distinct points in P, say, $p_1$ and $p_2$
  – Walk through P to find $p_3$ that does not lie on the line through $p_1$ and $p_2$
  – Find $p_4$ that does not lie on the plane through $p_1$, $p_2$, $p_3$
  – Special case: No such points exist? Planar case!
• Compute random permutation $p_5, \ldots, p_n$ of the remaining points
Inserting Points into CH

- Add $p_r$ to the convex hull of $P_{r-1}$ to transform $\text{CH}(P_{r-1})$ to $\text{CH}(P_r)$
- Two Cases:
  1. $P_r$ is inside or on the boundary of $\text{CH}(P_{r-1})$
     - Simple: $\text{CH}(P_r) = \text{CH}(P_{r-1})$
  2. $P_r$ is outside of $\text{CH}(P_{r-1})$ - the hard case
Case 2: $P_r$ outside $CH(P_{r-1})$

- Determine **horizon** of $p_r$ on $CH(P_{r-1})$
  - Closed curve of edges enclosing the visible region of $p_r$ on $CH(P_{r-1})$
Visibility

- Consider a plane $h_f$ containing a facet $f$ of $\text{CH}(P_{r-1})$
- $f$ is visible from a point $p$ if that point lies in the open half-space on the other side of $h_f$
Rethinking the Horizon

• The boundary of polygon obtained from projecting $\text{CH}(P_{r-1})$ onto a plane with $p_r$ as the center of projection
CH(Pr⁻¹) to CH(Pr)

- Remove visible facets from CH(Pr⁻¹)
- Found horizon: Closed curve of edges of CH(Pr⁻¹)
- Form CH(Pr) by connecting each horizon edge to pr to create a new triangular facet
Algorithm So Far

• Initialization
  – Form tetrahedron CH(P₄) from 4 points in P
  – Compute random permutation of remaining pts

• For each remaining point in P
  – pr is point to be inserted
  – If pr is outside CH(P_{r-1}) then
    • Determine visible region
    • Find horizon and remove visible facets
    • Add new facets by connecting each horizon edge to p_r
    • How do we determine the visible region?
How to Find the Visible Region

• Naïve approach:
  – Test every facet with respect to $p_r$
  – $O(n^2)$ work

• Trick is to work ahead:
  – Maintain information to aid in determining visible facets.
Conflict Lists

• For each facet $f$ maintain
  – $P_{\text{conflict}}(f)$ subset of $\{p_{r+1}, \ldots, p_n\}$
    containing points to be inserted that can see $f$

• For each $p_t$, where $t > r$, maintain
  – $F_{\text{conflict}}(p_t)$ containing facets of $CH(P_r)$ visible
    from $p_t$

• $p$ and $f$ are in conflict because they cannot
  coexist on the same convex hull
Conflict Graph G

• Bipartite graph
  – points not yet inserted
  – facets on CH(P_r)
• Arc for every point-facet conflict
• Conflict sets for a point or facet can be returned in linear time
• At any step of our algorithm, we know all conflicts between the remaining points and facets on the current CH
Initializing G

- Initialize G with CH(P₄) in linear time
- Walk through P₅₋ₙ to determine which facet each point can see
Updating G

- Discard visible facets from $p_r$ by removing neighbors of $p_r$ in G
- Remove $p_r$ from G
- Determine new conflicts
Determining New Conflicts

- If $p_t$ can see new $f$, it can see edge $e$ of $f$
- $e$ on horizon of $p_r$, so $e$ was already in and visible from $p_t$ in $CH(P_{r-1})$
- If $p_t$ sees $e$, it saw either $f_1$ or $f_2$ in $CH(P_{r-1})$
- $p_t$ was in $P_{conflict}(f_1)$ or $P_{conflict}(f_2)$ in $CH(P_{r-1})$
Determining New Conflicts

- Conflict list of $f$ can be found by testing the points in the conflict lists of $f_1$ and $f_2$ incident to the horizon edge $e$ in $\text{CH}(P_{r-1})$
What About the Other Facets?

- $P_{\text{conflict}}(f)$ remains unchanged for any $f$ unaffected by $p_r$
Final Algorithm

• Initialize CH(P₄) and G
• For each remaining point
  – Determine visible facets for pᵣ by checking G
  – Remove F_{conflict}(pᵣ) from CH
  – Find horizon and add new facets to CH and G
  – Update G for new facets by testing the points in existing conflict lists for facets in CH(P_{r-1}) incident to e on the new facets
  – Delete pᵣ and F_{conflict}(pᵣ) from G
Fine Point

• Coplanar facets
  – $p_r$ lies in the plane of a face of $\text{CH}(P_{r-1})$

• $f$ is not visible from $p_r$ so we merge created triangles coplanar to $f$
• New facet has same conflict list as existing facet