CENG 789 Digital Geometry Processing: Final Examination Study Report

This report provides a comprehensive overview of the key concepts, algorithms, and mathematical foundations covered in the CENG 789 Digital Geometry Processing course. It is designed to serve as a study aid for the final examination, focusing exclusively on the content presented in the course slides and the 15 provided PDF documents. Particular attention has been paid to verifying the correctness of mathematical derivations, especially those presented in handwritten form within the course materials.

Part I: Geometric Foundations and Representations

Section 1: Introduction to Digital Geometry Processing & Polygon Meshes

Digital Geometry Processing (DGP) encompasses the concepts and algorithms necessary for building and manipulating complete geometry processing systems. The typical pipeline involves stages such as Reconstruction, Analysis, Manipulation/Editing, Parameterization, Visualization, Remeshing, Registration, and Additive Manufacturing (3D Printing).[1] This field draws connections to geometric modeling, computer graphics, computer vision, and visualization.[1] A central focus of DGP, and this course, is the processing of thin-shell surfaces, which are predominantly represented by polygon meshes.[1]

Polygon meshes are collections of polygons, typically triangles in 3D space, that collectively approximate a 2D surface.[1] Their utility stems from their sufficiency for rendering, adaptability for refinement, and suitability for various analyses like similarity comparison, new surface generation, and segmentation.[1] However, for realistic deformation simulations, volumetric representations like tetrahedral meshes are often preferred over simple polygon shells.[1]

Meshes offer a piecewise linear approximation of continuous surfaces, analogous to how piecewise functions approximate continuous mathematical functions.[1] The fidelity of this approximation improves as the number of polygonal elements increases.[1] Notably, the approximation error exhibits quadratic behavior: doubling the number of elements (e.g., through subdivision) reduces the error by a factor of four.[1] This quantitative relationship is vital for understanding the efficiency of mesh refinement strategies.

Fundamental properties of meshes include continuity, manifoldness, and topological characteristics described by Euler's formula. Standard polygon meshes possess C0 continuity, meaning they are continuous in position. Higher-order continuities, such as C1 (tangent continuity) and C2 (curvature continuity), are desirable for achieving smoother surfaces and are often the objective of operations like subdivision.[1]

Manifold meshes are a crucial class of meshes in DGP. A mesh is considered 2-manifold if every edge is shared by at most two faces, and the faces incident to any vertex form a topological disk (an open or closed fan of triangles).[1] Adhering to manifold properties simplifies algorithmic design and coding by reducing the number of special cases that need to be handled. Watertight meshes, also known as closed manifold meshes, are 2-manifold meshes that have no boundary edges; they completely enclose a volume without any holes. Such meshes are essential for applications like 3D printing and certain physical simulations. Euler's formula for genus-0 surfaces, $V-E+F=2$, applies directly to these closed manifold meshes.[1]

Triangle meshes are the most prevalent type due to their inherent simplicity and robustness—three non-collinear points uniquely define a plane and thus a triangle. Within a triangle, barycentric coordinates provide a powerful mechanism for interpolation. A point p inside a triangle with vertices a,b,c can be expressed as $p=\alpha a+\beta b+\gamma c$, where $\alpha,\beta,\gamma\geq 0$ and $\alpha+\beta+\gamma=1$.[1]

From a graph-theoretic perspective, a mesh can be described as an undirected graph $G=\langle V,E\rangle$ augmented with a set of faces F.[1] Key graph properties include vertex degree (or valence, the number of edges incident to a vertex), k-regularity (all vertices having the same degree k), and connectivity (every pair of vertices connected by a path of edges).[1] A triangle mesh is an embedding of a planar graph into R3.[1]

Euler's Formula and Its Topological Significance

Euler's formula is a cornerstone of mesh analysis, providing a fundamental relationship between the number of vertices (V), edges (E), and faces (F) of a connected planar graph (which includes simple polyhedra and closed manifold meshes topologically equivalent to a sphere):

$V-E+F=2$

This formula was verified via an inductive proof in the course materials.1 The proof considers adding a new edge: if it connects two existing vertices and splits a face, $V_{new}=V$, $E_{new}=E+1$, $F_{new}=F+1$. Thus, $V_{new}-E_{new}+F_{new}=V-(E+1)+(F+1)=V-E+F$. If the original graph satisfied $V-E+F=2$, the modified graph does as well. This formula is not merely an algebraic identity but a deep topological invariant. It signifies that regardless of how a surface of a given genus is geometrically deformed (without tearing or creating new connections), the quantity $V-E+F$ remains constant. This invariance imposes powerful constraints on the combinatorial structure of any valid mesh.

For closed triangular meshes of genus 0 (topologically equivalent to a sphere), several statistics can be derived from Euler's formula and the observation that each triangle has 3 edges and each edge is shared by 2 faces (leading to $3F=2E$) [1]:

1. $F\approx 2V$: Substituting $E=3F/2$ into $V-E+F=2$ gives $V-F/2=2$. For large meshes, the constant 2 is negligible, yielding $F\approx 2V$.
2. $E\approx 3V$: Substituting $F=2E/3$ into $V-E+F=2$ gives $V-E/3=2$. For large meshes, $E\approx 3V$.
3. Average vertex degree ≈ 6: The sum of all vertex degrees is $2E$ (handshaking lemma). The average degree is $2E/V$. Using $E\approx 3V$, the average degree is approximately $(2\cdot 3V)/V=6$. The handwritten derivations for these approximations on slide 38 of 01-intro.pdf are algebraically sound, given the assumption of large meshes where the constant '2' in Euler's formula becomes insignificant relative to V, E, and F.[1] These

derived statistics are not just curiosities; they serve as benchmarks. If a mesh processing algorithm, intended to produce a closed genus-0 triangular mesh, generates a result that significantly deviates from these ratios, it may indicate underlying issues such as the mesh not being truly closed, possessing a different topological genus, or having non-manifold configurations.

The formula generalizes for surfaces with g handles (genus g) as:

$V-E+F=2(1-g)$

.1 This generalized form underscores the formula's role as a fundamental topological invariant.

Euler's formula is instrumental in proving various geometric and graph-theoretic theorems:

- **At Most 5 Platonic Solids:** A Platonic solid has congruent regular polygonal faces, with the same number of faces meeting at each vertex. Let n be the number of edges per face and m be the number of edges meeting at each vertex. Then $2E=nF$ and $mV=2E$. Substituting $F=2E/n$ and $V=2E/m$ into $V-E+F=2$ yields $2E/m-E+2E/n=2$. Dividing by 2E (since $E>0$) gives $1/m-1/2+1/n=1/E$. Since $E>0$, we must have $1/m+1/n>1/2$. Given that faces must have at least 3 edges ($n\geq3$) and at least 3 faces must meet at a vertex ($m\geq3$), the only integer solutions for (n,m) are (3,3) (tetrahedron), (3,4) (octahedron), (4,3) (cube), (3,5) (icosahedron), and (5,3) (dodecahedron).[1] The handwritten derivation on slide 41 of 01-intro.pdf, which arrives at $F=2n-mn+2m4m$ and then analyzes $2n-mn+2m>0$, is a correct alternative path to the same constraints on n and m.
- Soccer Ball Composition (12 Pentagons): For a polyhedron composed of only pentagons and hexagons, where exactly three faces meet at each vertex (degree of each vertex is 3), it must have exactly 12 pentagonal faces. Let P be the number of pentagons and H be the number of hexagons. Then $F=P+H$. The total number of edges counted face-by-face is $5P+6H$, so $2E=5P+6H$. Since each vertex has degree 3, the sum of degrees is $3V=2E=5P+6H$, so $V=(5P+6H)/3$. Substituting these into $V-E+F=2$:
  $35P+6H-25P+6H+(P+H)=2$
  Multiplying by 6:
  $2(5P+6H)-3(5P+6H)+6(P+H)=12$
  $10P+12H-15P-18H+6P+6H=12$
  $(10-15+6)P+(12-18+6)H=12$
  $P+0H=12\Rightarrow P=12.$
  This elegant proof demonstrates a fixed structural property based purely on topological constraints.1 The handwritten derivation on slide 44 of 01-intro.pdf is correct.
- **Non-Planarity of K5:** The complete graph on 5 vertices, K5, has $V=5$ and $E=10$. If K5 were planar, by Euler's formula, $5-10+F=2\Rightarrow F=7$. For any simple planar graph where all faces are triangles (which maximizes edges for a given F), $3F\leq2E$. In this case, $3(7)\leq2(10)\Rightarrow21\leq20$, which is a contradiction. Therefore, K5 is not planar.[1] The inequality $3F\leq2E$ holds more generally because each face must be bounded by at least 3 edges, and each edge contributes to two such face boundaries (or twice to one, if it's a bridge).
- **Non-Planarity of K3,3:** The complete bipartite graph K3,3 has $V=6$ and $E=9$. If planar, $6-9+F=2\Rightarrow F=5$. Since K3,3 is bipartite, it contains no odd cycles, meaning it has no triangular faces. Thus, each face must be bounded by at least 4 edges. This implies

$4F \le 2E$. Substituting the values, $4(5) \le 2(9) \Rightarrow 20 \le 18$, a contradiction. Thus, K3,3 is not planar.[1] The link between planarity and graph embeddings is critical; if a mesh's connectivity contains K5 or K3,3 as a minor (a graph obtainable by edge deletions and contractions), it cannot be flattened onto a 2D plane without self-intersections, which is problematic for tasks like texture mapping.

- **Kuratowski's Theorem:** This theorem provides a complete characterization of planar graphs: a graph G is planar if and only if it does not contain K5 or K3,3 as a minor.[1]
- **6-Color Theorem for Planar Graphs:** Every connected planar graph G can be properly colored with 6 or fewer colors. The proof relies on the fact that every connected planar graph must contain at least one vertex with a degree of 5 or less. The proof proceeds by induction on the number of vertices V. The base case ($V \le 6$) is trivial. For the inductive step, remove such a low-degree vertex v. The remaining graph $G-v$ can be 6-colored by the induction hypothesis. When v is added back, its at most 5 neighbors use at most 5 distinct colors, leaving at least one color available for v.[1] The existence of a vertex with degree ≤ 5 is proven using Euler's formula: assuming all faces are triangles (for a maximal edge count), $3F=2E$. Combined with $V-E+F=2$, this yields $2E=6V-12$. The average degree is $2E/V=(6V-12)/V=6-12/V$, which is strictly less than 6 for any $V>0$. Thus, at least one vertex must have a degree less than 6 (i.e., ≤ 5).[1] The handwritten derivation on slide 49 of 01-intro.pdf is correct. The average vertex degree being approximately 6 for closed triangular meshes is a direct consequence of these topological constraints and is fundamental to understanding "well-behaved" meshes. Deviations often indicate areas of high curvature or irregular sampling.

The emphasis on manifold meshes [1] throughout the course materials is significant. The manifold property ensures that the local neighborhood of any point on the mesh is topologically simple (equivalent to a disk or half-disk). This local simplicity is foundational for the robust definition and implementation of many DGP algorithms, such as normal calculation, smoothing, and parameterization. Non-manifold structures introduce ambiguities and complexities that necessitate special handling, making algorithms less general and more prone to errors. Thus, mesh repair to ensure manifoldness is often a critical preprocessing step.

Section 2: Polygons, Triangulations, and Point Sets

This section transitions from general mesh properties to the specifics of 2D polygons and their decomposition, followed by the analysis of unstructured point sets, forming the basis for surface reconstruction and representation.

Polygons

A polygon is formally defined as a closed region in a plane, bounded by a finite sequence of connected, non-intersecting line segments called edges. The points where edges meet are vertices.1 Polygons can be classified as convex (all interior angles less than or equal to π, and any line segment connecting two interior points lies entirely within the polygon) or non-convex (concave).

A fundamental theorem concerning polygons is the **Polygonal Jordan Curve Theorem**. It states that the boundary of any simple polygon partitions the plane into exactly two distinct regions: a bounded interior and an unbounded exterior. A point cannot move from one region to the other without crossing the boundary.[1] This theorem underpins the **point-in-polygon test**, often implemented by casting a ray from the test point to infinity and counting intersections with the polygon's boundary: an odd number of intersections implies the point is inside, while an even number implies it is outside.[1] This simple test is widely used in computer graphics for user interaction and region queries.

Diagonals

A diagonal of a polygon is a line segment connecting two non-adjacent vertices that lies entirely within the polygon's interior.1 A key property is that any polygon with n≥4 vertices must possess at least one diagonal. This can be proven by considering the vertex v with the smallest y-coordinate (and smallest x-coordinate in case of ties). Let its neighbors be a and b. If the segment ab is a diagonal, the proof is complete. If not, ab must lie outside the polygon or intersect other edges. In this scenario, a line parallel to ab can be swept from v into the polygon's interior. The first vertex x encountered by this sweep line, other than a or b, will form a diagonal vx that lies entirely within the polygon.1

Polygon Triangulation

Triangulation is the decomposition of a polygon into a set of non-overlapping triangles whose vertices are also vertices of the original polygon. This is achieved by adding a maximal set of non-crossing diagonals.1

- **Existence:** Every simple polygon admits at least one triangulation. This is proven by induction on the number of vertices n. The base case n=3 is a triangle itself. For n>3, since a diagonal exists, it splits the polygon into two smaller polygons, P1 and P2, with n1<n and n2<n vertices. By the induction hypothesis, P1 and P2 can be triangulated. Combining these triangulations yields a triangulation for P.[1]
- **Number of Triangles and Diagonals:** Any triangulation of an n-gon always results in exactly n−2 triangles and requires the addition of n−3 diagonals. This consistency in the number of triangles is crucial. The proof for n−2 triangles also uses induction. For n=3, it's 1 triangle (3−2=1). For n>3, a diagonal splits the n-gon into an n1-gon and an n2-gon, where n1+n2=n+2 (as the two vertices of the diagonal are part of both smaller polygons). The total number of triangles is (n1−2)+(n2−2)=(n1+n2)−4=(n+2)−4=n−2.[1] The derivation on slide 21 of 02-polygons-triangulations.pdf is correct.
- **Ears:** An ear of a polygon is formed by three consecutive vertices a,b,c such that the segment ac is a diagonal. Every polygon with n≥4 vertices has at least two ears.[1] This property is the basis for the "ear-clipping" triangulation algorithm.
- **Angle Sums:** The sum of the interior angles of an n-gon is π(n−2) radians.[1] The sum of its turn angles (exterior angles) is 2π radians.[1]
- **Number of Distinct Triangulations:** For a general polygon, there is no simple closed-form formula for the number of distinct triangulations. However, for a convex n-gon, the number of distinct triangulations is given by the Catalan number Cn−2=n−11(n−22n−4). (Note: Slide 25 of 02-polygons-triangulations.pdf states an (n+2)-gon has Cn triangulations, which is equivalent if N=n+2, then CN−2.)

Tetrahedralization

The 3D analogue of polygon triangulation is tetrahedralization: decomposing a polyhedron into non-overlapping tetrahedra. Unlike 2D polygons, not every polyhedron can be tetrahedralized without adding new vertices. The Schönhardt polyhedron is a classic example of such a non-tetrahedralizable polyhedron; all its potential interior diagonals lie outside the polyhedron itself.1 Furthermore, different tetrahedralizations of the same polyhedron can result in a different number of tetrahedra.1

Art Gallery Problem

This problem seeks the minimum number of stationary guards required to see every point within an n-sided polygonal gallery. Each guard has 360° visibility. The Art Gallery Theorem states that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary.1 The sufficiency proof involves triangulating the polygon and then 3-coloring the vertices of the triangulation. A 3-coloring ensures that every triangle has vertices of all three colors. By placing guards at all vertices of the least frequently used color, every triangle (and thus the entire polygon) is covered. By the pigeonhole principle, the least frequent color will appear on at most $\lfloor n/3 \rfloor$ vertices.1 The inductive proof for 3-colorability of a triangulated polygon (by removing an ear, coloring the remainder, and then adding the ear back) presented on slide 33 of 02-polygons-triangulations.pdf is a standard and correct approach.

The concept of triangulation is fundamental because it decomposes potentially complex shapes into the simplest possible polygonal units (triangles). This canonical representation underpins numerous algorithms, from basic area calculations and rendering to more complex tasks like physical simulation and solving the Art Gallery Problem. The consistent $n-2$ triangles for an n-gon provides a stable combinatorial structure that algorithms can rely on.

Point Sets

Unlike ordered polygonal vertices, point sets are typically unstructured collections of points in space.1 Key geometric structures associated with point sets include convex hulls, triangulations (especially Delaunay triangulations), and Voronoi diagrams.

Convex Hulls of Point Sets

The convex hull of a set of points S is the smallest convex region that contains all points in S. Intuitively, it's like stretching a rubber band around the outermost points.1

- **Algorithms for Construction:**
  - **Gift-wrapping (Jarvis March):** An $O(nh)$ algorithm, where n is the total number of points and h is the number of points on the hull. It iteratively "wraps" around the point set.[1]
  - **Graham Scan:** An $O(n\log n)$ algorithm for 2D point sets. It involves sorting points by polar angle around an anchor point (e.g., the lowest point) and then using a stack to build the hull by making left turns.[1]
  - **Divide-and-Conquer:** An $O(n\log n)$ algorithm that recursively splits the point set, computes hulls for subsets, and merges them. It extends to 3D.[1]
- **Lower Bound:** Any algorithm for constructing the convex hull of n points has a lower bound of $\Omega(n\log n)$ in the algebraic decision tree model, as sorting can be reduced to convex hull construction (e.g., by mapping numbers xi to points (xi,xi2) on a parabola).[1]

The convex hull provides a fundamental boundary definition for a point set. It is often a preliminary step in more complex processing pipelines, such as triangulating the interior of the point set or defining a surface that encloses the points.

Triangulation of a Point Set

This involves subdividing the plane (or the convex hull of the points) into triangles whose vertices are the given points, using a maximal set of non-crossing edges.1

- The edges of the convex hull are always part of any triangulation of the point set.[1]
- All bounded regions inside the convex hull must be triangles.[1]
- **Number of Triangles:** For a set of n points with h points on the convex hull and k=n−h interior points, any triangulation will have 2n−h−2 triangles (or equivalently, 2k+h−2 triangles). This is proven using Euler's formula V−E+F=2. Let t be the number of triangles inside the hull. The total number of faces is F=t+1 (including the outer unbounded face). The number of vertices is V=n. Each triangle has 3 edges, and the outer face has h edges. Since each edge is shared by two faces, 2E=3t+h. Substituting into Euler's formula: n−(3t+h)/2+(t+1)=2. Solving for t: 2n−3t−h+2t+2=4⇒2n−t−h=2⇒t=2n−h−2. If n=k+h, then t=2(k+h)−h−2=2k+h−2. The derivation on slide 24 of 03-pointsets.pdf is correct.[1]
- **Construction Algorithms:**
    - **Triangle-splitting:** Triangulate the convex hull, then iteratively pick an interior point and connect it to the vertices of the triangle containing it.[1]
    - **Incremental:** Sort points (e.g., by x-coordinate), add them one by one, and connect the new point to all previously added visible points.[1]

Delaunay Triangulation

A special type of triangulation for a point set that aims to produce "well-shaped" triangles, avoiding excessively skinny ones. This is crucial for applications like interpolation and surface reconstruction from scattered data.1

- **Properties:**
    - **Empty Circumcircle Property:** The circumcircle of any triangle in a Delaunay triangulation does not contain any other input points in its interior.[1] This is the defining characteristic.
    - **Maximizes Minimum Angle:** Among all possible triangulations of a point set, the Delaunay triangulation maximizes the minimum angle of all triangles in the triangulation.[1]
    - **Legal Edge:** An edge is "legal" if the two triangles sharing it form a convex quadrilateral, and flipping the edge does not result in a "fatter" triangulation (i.e., does not increase the smallest angle in the local configuration or, more formally, does not improve the lexicographically sorted angle vector of the triangulation).[1] A Delaunay triangulation contains only legal edges.
- **Construction (Edge Flipping Algorithm):** Start with any arbitrary triangulation of the point set. Iteratively find "illegal" edges (edges whose flip would increase the minimum angle in the local quadrilateral or satisfy the empty circumcircle property for the new triangles) and flip them. This process is guaranteed to terminate because each flip

increases the lexicographical angle vector of the triangulation, and there are a finite number of possible triangulations.[1]

Voronoi Diagrams

Given a set of points S (sites), the Voronoi diagram partitions the plane into regions, where each region Rk associated with a site pk∈S consists of all points x in the plane that are closer to pk than to any other site pj∈S (i.e., Rk={x∣∣∣x−pk∣∣≤∣∣x−pj∣∣,∀j偃=k}).1

- **Duality with Delaunay Triangulation:** The Voronoi diagram and the Delaunay triangulation are geometric duals of each other. If two sites pi,pj share a Voronoi edge, then the edge (pi,pj) is a Delaunay edge. Conversely, each Voronoi vertex is the circumcenter of a Delaunay triangle.[1] This duality is not just a geometric observation but a powerful algorithmic principle. Problems that are difficult to solve in one domain can sometimes be more easily addressed in the dual domain. For example, the empty circumcircle property of Delaunay triangles is directly linked to Voronoi vertices being equidistant from three sites.
- **Applications:** Include finding the nearest service location (e.g., post office to an apartment), interaction representation between shapes, generating pre-fracture patterns for simulations, and medial axis extraction.[1]

Section 3: Mesh Data Structures

Efficiently representing and manipulating polygon meshes requires appropriate data structures that store not only the geometry (vertex coordinates) but also the connectivity (adjacency relationships between vertices, edges, and faces).[1] Attributes such as normals, colors, and texture coordinates are also commonly stored.

Common Topological Queries and Operations:

Mesh data structures should support efficient answers to queries like:

- What are the vertices of a given face?
- Are two vertices adjacent?
- Which edges/faces are incident to a given vertex? [1] And facilitate operations such as:
- Adding or removing vertices/faces.
- Edge operations: splitting an edge (inserting a vertex), collapsing an edge (merging two vertices), and flipping an edge (changing the diagonal of a quadrilateral formed by two adjacent triangles).[1] These operations are fundamental for tasks like mesh refinement (edge split for Loop subdivision [1]), simplification (edge collapse for LOD [1]), and quality improvement (edge flip [1]).

Types of Mesh Data Structures:

1. **Face-Based Data Structures:**
    - **Face-Set (Polygon Soup):** The simplest representation, where each face (e.g., triangle) lists its vertex coordinates directly. This leads to significant data redundancy (vertices are stored multiple times) and no explicit connectivity information, making topological queries inefficient.[1]
    - **Indexed Face-Set:** A more common and efficient face-based structure. Vertices are stored once in a list, and each face stores indices into this vertex list. This is

the format used by common file types like OBJ, OFF, and PLY, and is the basis for the course's provided code framework.[1] The framework includes struct Triangle {int v1i, v2i, v3i;...}, struct Vertex {float coords;...}, and struct Edge {int v1i, v2i;...} within a class Mesh.[1] While memory-efficient for geometry, deriving adjacency information (e.g., finding all faces incident to a vertex) still requires iterating through the face list.

2. Edge-Based Data Structures:
   These structures explicitly store edges and their relationships to vertices and faces. This facilitates more efficient traversal of the mesh and querying of one-ring neighborhoods (vertices/edges/faces incident to a vertex).1 More advanced structures like the half-edge data structure (not explicitly detailed as a primary structure in the provided snippets but implied by the need for efficient operations) store directed edges and their connectivity to enable constant-time adjacency queries for many common operations.

The choice of data structure involves a trade-off between memory usage, implementation complexity, and the efficiency of specific topological queries and operations. While an indexed face-set is simple and widely used for storage and basic rendering, more complex algorithms that heavily rely on local topology (like advanced smoothing, remeshing, or subdivision) benefit significantly from data structures that make adjacency information readily and quickly accessible. The types of operations an algorithm needs to perform often dictate the most suitable underlying data structure.
3D Graphics Programming APIs:
The course utilizes Open Inventor (specifically Coin3D, an independent implementation) for 3D graphics programming. Open Inventor is favored for its high-level, object-oriented approach and its integrated 3D viewer (SoWin for Windows, SoXt for Unix) which provides built-in trackball navigation, camera handling, and various rendering modes.1 This contrasts with OpenGL, which is a lower-level, state-based API requiring manual implementation of such features.1 In Open Inventor, scene elements are organized in a scene graph, with SoSeparator nodes grouping children, SoCoordinate3 defining vertex coordinates, and SoIndexedFaceSet defining face connectivity using these coordinates.1

Part II: Core Geometric Operators and Algorithms

Section 4: The Laplacian Operator in Digital Geometry Processing

The Laplacian operator is a cornerstone of digital geometry processing, providing a way to analyze and manipulate surface geometry. In the continuous setting, it's known as the Laplace-Beltrami operator. For discrete meshes, various approximations, collectively termed Graph Laplacians, are used. Fundamentally, the Laplacian at a point measures the deviation of that point's value (e.g., position or a scalar function value) from the average of its neighbors.[1]
This "shape operator" characteristic allows it to capture intrinsic properties of the surface.
Discretization Methods:
The choice of discretization significantly impacts the behavior of the Laplacian and its applications.

1. **Uniform Laplacian:** This is the simplest form, defined as L=D−A, where D is the diagonal degree matrix (containing vertex valences) and A is the adjacency matrix. The entries are Lii=deg(vi), Lij=−1 if vertices vi and vj are adjacent, and 0 otherwise.[1] While simple, it is purely combinatorial and does not account for the actual geometric embedding of the mesh, which can lead to undesirable artifacts in geometry-sensitive applications like smoothing or parameterization.
2. **Cotangent Laplacian:** This is a geometry-aware discretization, widely used due to its favorable properties. The weights for an edge (i,j) are derived from the cotangents of the angles opposite to that edge in the two triangles sharing it: wij=21(cotαij+cotβij).[1] The Laplacian matrix entries are then Lii=Σk~iwik and Lij=−wij if vi and vj are adjacent, 0 otherwise. The cotangent Laplacian is symmetric. The derivation of the cotangent weights from triangle geometry (area, edge lengths, Law of Cosines, sine rule) is a standard result in discrete differential geometry.[1] The derivation on these slides is correct. A key property is that it approximates the continuous Laplace-Beltrami operator more faithfully than the uniform Laplacian. However, if the mesh contains obtuse triangles, cotangent weights can become negative, potentially leading to issues like non-bijective mappings in parameterization unless handled carefully (e.g., by ensuring the input mesh is Delaunay).
3. **Mean-Value Laplacian:** This formulation guarantees positive weights, which is beneficial for certain applications like parameterization as it ensures bijectivity for convex boundary mappings. The weights are wij=2||vi−vj||tan(γij/2)+tan(δij/2).[1] It is not symmetric.
4. **Belkin Laplacian (Point Cloud Laplacian):** Designed for point clouds where explicit connectivity is not available. It uses Gaussian weights Wij=exp(−t||xi−xj||2) between points xi and xj, where t is a scale parameter. The resulting matrix is dense but can be sparsified by considering only k-nearest neighbors.[1]

The choice between a combinatorial (uniform) and a geometry-aware (cotangent, mean-value) Laplacian is critical. Uniform Laplacians are simpler but can distort geometry significantly. Geometry-aware Laplacians better preserve local shape but may have their own sensitivities (e.g., cotangent weights to triangle quality).

Eigenproblems and Spectral Analysis:

The eigenvalues (λi) and eigenvectors (ϕi) of the Laplacian matrix (Lϕi=λiϕi) provide a wealth of information about the mesh's structure.1

- For symmetric Laplacians (like uniform or cotangent), eigenvectors form an orthogonal basis.
- The **Generalized Eigenvalue Problem** Lϕi=λiAϕi is often solved, where A is a diagonal mass or area matrix (e.g., Aii is 1/3 of the sum of areas of triangles incident to vertex i). This formulation ensures that the eigenvectors are orthonormal with respect to the surface area measure, making them more suitable for representing functions on the surface.[1] The mass matrix A correctly weights the contribution of each vertex value by the area it represents, ensuring that discrete operators and norms accurately reflect their continuous counterparts.

- **Spectral Properties:**
  - The number of zero eigenvalues corresponds to the number of connected components of the mesh. For a single connected component, λ1=0 with ϕ1 being a constant vector.
  - λ2, the smallest non-zero eigenvalue (for a connected mesh), is called the **algebraic connectivity** or Fiedler value. Its corresponding eigenvector, the **Fiedler vector**, is useful for mesh partitioning/segmentation as its sign changes often correspond to natural cuts in the mesh.[1]
  - Eigenvectors corresponding to small eigenvalues (low frequencies) are smooth and vary slowly across the mesh, capturing global shape characteristics. Eigenvectors for large eigenvalues (high frequencies) oscillate rapidly and represent fine details.[1] The spectrum of the Laplacian offers a powerful tool for analyzing and processing shapes, forming the basis for techniques like spectral clustering, shape matching using functional maps, and spectral compression.

Differential Coordinates:
Differential coordinates, often denoted δi for a vertex vi, capture the local geometric detail around vi. They represent the vector difference between vi and the weighted average of its neighboring vertices. This vector approximates the mean curvature normal at vi scaled by the local surface area element.1

- **Uniform Weights:** δi=vi−di1Σj∈N(i)vj, where di is the degree of vi [[1], p34 (handwritten); [1], p38]. This can also be written as δi=di1Σj∈N(i)(vi−vj).
- **Cotangent Weights:** δic=vi−Σj∈N(i)wijΣj∈N(i)wijvj, using cotangent weights wij [[1], p34 (handwritten); [1], p40]. These are generally preferred for preserving geometric detail more accurately. The entire set of differential coordinates for a mesh can be expressed in matrix form as Ldcv=δ, where v is the matrix of vertex coordinates and Ldc is a "General Laplacian" matrix (distinct from the L=D−A form, though related). For Ldc, the diagonal entries are 1, and off-diagonal entries are −1/di (uniform) or −wij/Σkwik (cotangent) for neighbors.[1] Differential coordinates are fundamental to Laplacian mesh editing, deformation, and compression, as they allow manipulation of the mesh while attempting to preserve its local shape characteristics.

Calculation Examples:

- **Uniform Laplacian Matrix Element:** For a vertex vi with degree 3, Lii=3. If vj is a neighbor of vi, Lij=−1.
- **Cotangent Weight:** For an edge eij shared by triangles T1=(vi,vj,vk) and T2=(vj,vi,vl), with angles αij at vk in T1 and βij at vl in T2 (both opposite eij), the weight is wij=0.5(cotαij+cotβij).
- **Differential Coordinate (Uniform):** If vi=(1,1,1) has neighbors v1=(0,1,1),v2=(1,0,1),v3=(1,1,0), then di=3. The average of neighbors is (30+1+1,31+0+1,31+1+0)=(32,32,32). So, δi=(1−32,1−32,1−32)=(31,31,31).

Section 5: Surface Reconstruction

Surface reconstruction is the process of creating a continuous surface representation, typically a polygon mesh, from various forms of input data, most commonly unorganized 3D point clouds obtained from scanning devices, or from implicit functions.[1] The choice between explicit and implicit reconstruction methods depends on the nature of the input data and the desired properties of the output surface.

Explicit vs. Implicit Reconstruction:

- **Explicit Reconstruction:** These methods directly connect the input sample points to form the surface mesh, essentially "connecting the dots." They interpolate the input data, meaning the resulting surface will pass through the sample points.[1]
    - **Advantages:** Preserves the original data points precisely.
    - **Disadvantages:** Highly sensitive to noise and misalignments in the input data, which are common in real-world scans. This can lead to undesirable artifacts, holes, or non-manifold geometry in the reconstructed mesh.[1]
- **Implicit Reconstruction:** These methods first define a scalar-valued function over the 3D space (often a signed distance function, SDF) that represents the surface implicitly as its zero-level set. The final mesh is then extracted from this implicit function, typically using an algorithm like Marching Cubes.[1]
    - **Advantages:** More robust to noisy data as they approximate the input points rather than strictly interpolating them. They often guarantee a manifold and watertight output by construction.[1]
    - **Disadvantages:** Can smooth out sharp features present in the original data due to the approximation nature. The quality depends on the grid resolution used for the implicit function.

The fundamental difference lies in interpolation versus approximation. Explicit methods honor the data points exactly, while implicit methods find a surface that best fits the data according to some criteria, often leading to smoother and more robust results, especially with imperfect input. Surface reconstruction is inherently an ill-posed problem: for a given set of points, infinitely many surfaces can pass through or near them. Priors, such as assumptions about smoothness, piecewise smoothness, or topological simplicity, are used to regularize the problem and guide the reconstruction towards a plausible surface.[1]

Explicit Reconstruction Algorithms:

1. **Crust Algorithm:** This algorithm leverages the properties of Delaunay triangulations and Voronoi diagrams. The intuition is that if the input point set is sampled densely enough from an underlying surface, the triangles forming that surface will be part of the 3D Delaunay tetrahedralization of the point set. The algorithm proceeds by [1]:
    - Computing the 3D Delaunay tetrahedralization of the input point set P.
    - Computing the Voronoi diagram of P.
    - Inserting the Voronoi vertices into the Delaunay tetrahedralization.
    - The "crust" triangles are then identified as those Delaunay triangles whose dual Voronoi edges connect a Voronoi vertex inside the object to one outside. More simply, after adding Voronoi vertices, the algorithm keeps only the original triangles (edges between original sample points) that are deemed to be on the surface.

2. **Local Triangulation Method:** This approach constructs the surface by stitching together locally computed 2D Delaunay triangulations [1]:
    - For each input point pi, find its local neighborhood Li (e.g., k-nearest neighbors using a k-d tree).
    - Estimate a tangent plane for Li using Principal Component Analysis (PCA). The normal of this plane is the eigenvector corresponding to the smallest eigenvalue of the covariance matrix of Li.
    - Project the points in Li onto this tangent plane.
    - Compute the 2D Delaunay triangulation Di of these projected points.
    - The final surface is the union of these local 2D triangulations, lifted back into 3D. This method can result in non-manifold outputs if care is not taken in merging the local patches.

Implicit Reconstruction Algorithms:

The core of implicit reconstruction involves defining an implicit function F:R3⟶R (typically an SDF) and then extracting the iso-surface S={x∈R3∣F(x)=0}.

1. **Signed Distance Function (SDF) Construction:** The SDF F(x) at any grid point x is its signed distance to the nearest point on the underlying surface. For reconstruction from point clouds, the surface is not known a priori. Instead, tangent planes are estimated at each sample point oi with normal ni. The SDF at a grid point x is then approximated as the signed distance to the closest tangent plane: F(x)=(x−oi)⋅ni.[1]
    - **Tangent Plane Estimation:** For each sample point s, its k-nearest neighbors are found (efficiently using a k-d tree, see below). PCA is applied to this neighborhood. The eigenvector corresponding to the smallest eigenvalue gives the normal ns of the tangent plane, and the centroid of the neighborhood gives the point os on the plane.[1]
    - **Normal Orientation:** Estimated normals ns can have arbitrary signs. Consistent orientation across the point set is crucial and can be achieved using algorithms like MST-based propagation or by analyzing the viewpoint if available.[1]
    - **SDF Discontinuity and Refinements:** The simple F(x)=(x−oi)⋅ni definition can lead to discontinuities in the SDF field. More advanced methods like Radial Basis Functions (RBFs), Moving Least Squares (MLS), or deep learning approaches (DeepSDF) are used to create smoother and more robust SDFs.[1]
2. **Marching Cubes Algorithm:** This algorithm extracts a polygonal mesh (iso-surface) from a scalar field (like an SDF) sampled on a regular 3D grid.[1]
    - **Grid Processing:** The algorithm marches through each cell (cube) of the grid.
    - **Vertex Classification:** For each cube, its 8 vertices are classified as being inside or outside the surface based on whether their SDF values are positive or negative (or above/below the chosen iso-value).
    - **Case Look-up:** This classification creates an 8-bit index (one bit per vertex). This index is used to look up a pre-computed table of 28=256 possible edge intersection patterns and corresponding triangle configurations that approximate the surface within that cube.[1] Symmetry reduces the number of unique cases.

  - **Intersection Calculation:** The exact position where the surface intersects an edge of the cube is found by linear interpolation of the SDF values at the edge's endpoints. If F0 and F1 are the SDF values at the endpoints x0 and x1, the intersection point x (where F(x)=0) is x=x0+u(x1−x0), with u=F0/(F0−F1).[1] This derivation is a standard linear interpolation to find the zero-crossing.
  - **Ambiguity Handling:** Certain vertex configurations (e.g., alternating signs on a cube face) can lead to ambiguous triangulations within a cube, potentially creating holes in the final surface. These ambiguities are typically resolved by using an extended look-up table with disambiguation rules (e.g., the Asymptotic Decider, which examines the bilinear interpolant on ambiguous faces) or by consistent choices.[1]
  - **Grid Resolution:** The resolution of the input grid directly affects the level of detail in the reconstructed mesh. Finer grids capture more detail but increase computational cost.[1]

The choice of grid resolution in implicit methods is a critical parameter. While finer grids can capture more detail, they also increase memory and computation time. Furthermore, if the grid is too fine relative to the feature size or noise level, it can lead to overly complex or noisy reconstructions. Adaptive grids (e.g., octrees) can offer a compromise, providing higher resolution only where needed.

K-d Trees for Nearest Neighbor Search:

Efficiently finding k-nearest neighbors is essential for many steps in reconstruction (e.g., local neighborhood for PCA, finding closest tangent plane for SDF evaluation). Brute-force search is O(N) for each query (or O(kN) for k-NN). K-d trees provide a way to perform these queries in approximately O(logN) time on average (or O(klogN) for k-NN).1

- **Construction:** A k-d tree is a binary tree that recursively partitions k-dimensional space. At each level, the space is split along one dimension (cycling through dimensions) using the median of the points along that dimension.[1]
- **Search:** To find the nearest neighbor to a query point, the tree is traversed. When a leaf node is reached, the point in that leaf is a candidate. The algorithm then backtracks, pruning branches of the tree that cannot possibly contain a closer point than the current best candidate found so far. This pruning is based on comparing the distance to the current champion with the distance to the splitting hyperplanes of the subtrees.[1]

The reliance of many reconstruction techniques on local neighborhood operations (like PCA for normal estimation or local triangulations) underscores a common theme in DGP: global structure is often inferred from the aggregation of local information. The quality and efficiency of these local computations, often facilitated by structures like k-d trees, are paramount for successful global reconstruction.

Section 6: Mesh Smoothing and Remeshing

Once a mesh is reconstructed or obtained from other sources, it often requires further processing to improve its quality for specific applications. Smoothing aims to remove noise and create visually pleasing surfaces, while remeshing modifies the mesh connectivity and vertex positions to meet certain quality criteria regarding element shape, size, and distribution.

Mesh Smoothing

The primary goal of mesh smoothing is to eliminate high-frequency noise, which is common in scanned data or can result from numerical inaccuracies in modeling processes.1

1. **Laplacian Smoothing (Iterative):** This is a widely used technique where each vertex is iteratively moved towards the average position of its 1-ring neighbors. The update rule is typically $v'=v+\lambda L(v)$, where $L(v)$ is the discrete Laplacian vector at vertex v, representing the difference between v and the average of its neighbors, and λ is a step size (often 0.5).[1]
    - **Uniform Weights:** If $L(v)=(dv1\Sigma j\in N(v)vj)-v$ (where dv is the valence of v), the smoothing is based on purely combinatorial connectivity. This method is simple but suffers from two main drawbacks:
        - **Shrinkage:** The mesh tends to shrink with repeated iterations, especially for closed surfaces which can converge to a single point.[1]
        - **Tessellation Sensitivity:** The results are dependent on the existing triangulation quality; irregular triangulations can lead to uneven smoothing.[1]
    - **Cotangent Weights:** To address tessellation sensitivity and better approximate the behavior of mean curvature flow, cotangent weights are used for $L(v)$. $L(v)=(\Sigma j\in N(v)wijvj/\Sigma j\in N(v)wij)-v$, where $wij=21(\cot\alpha ij+\cot\beta ij)$. This is more geometry-aware and produces better results, especially in preserving the overall shape while smoothing details.[1] Planar meshes are invariant to smoothing with cotangent weights.
2. **Taubin Smoothing ($\lambda|\mu$ Filter):** To combat the shrinkage problem of Laplacian smoothing, Taubin proposed an iterative scheme that alternates between a shrinking step (Laplacian smoothing with a positive scale factor λ) and an inflating step (Laplacian smoothing with a negative scale factor μ, where $|\mu|>\lambda$ but μ is chosen such that it doesn't completely undo the smoothing).[1] This helps preserve volume while achieving smoothing.
3. **Tangential Projection:** Another method to reduce shrinkage is to project the Laplacian displacement vector onto the tangent plane at the vertex before applying the update: $v\leftarrow v+\lambda(I-nvnvT)(cv-v)$, where cv is the average of neighbors and nv is the vertex normal.[1]

The core idea behind Laplacian smoothing is its analogy to a diffusion process. Moving a vertex towards the average of its neighbors mimics how heat would diffuse across a surface, naturally smoothing out sharp, high-frequency variations. The choice of weights (uniform vs. cotangent) determines whether this diffusion is purely based on connectivity or is guided by the surface geometry itself.

Remeshing

Remeshing aims to generate a new mesh that approximates an input mesh but has better element quality, such as more uniform triangle shapes (isotropy), adaptive sizing based on curvature, or more regular vertex valences.1 This is crucial for numerical simulations, texture mapping, and improving rendering performance.

- **Goals:**
    - Improve element shape (e.g., making triangles closer to equilateral).
    - Control sampling density (e.g., uniform or adaptive to curvature).
    - Achieve regular vertex valences (e.g., close to 6 for interior vertices of a triangle mesh).
- **Local Remeshing Operators:** These are applied iteratively to improve mesh quality [1]:
    - **Edge Collapse:** Merges two vertices, removing an edge and typically two faces. Used to reduce triangle count or eliminate short edges. Target edge length Lmin (e.g., 4/5Ltarget).
    - **Edge Split:** Inserts a new vertex along an edge, splitting the edge and incident faces. Used to increase resolution or eliminate long edges. Target edge length Lmax (e.g., 4/3Ltarget).
    - **Edge Flip:** Changes the diagonal of a quadrilateral formed by two adjacent triangles. Used to improve triangle quality (e.g., increase minimum angles) or adjust vertex valences towards a target (e.g., 6 for interior, 4 for boundary). Cost function for flip: Σ(valence(vi)−opt_valence(vi))2.[1]
    - **Vertex Relocation (Smoothing):** Moves vertices to improve local geometry, often using Laplacian smoothing.
- **Isotropic Remeshing:** A common goal is to create a mesh with triangles that are as equilateral as possible and have roughly uniform edge lengths. This often involves iteratively applying the local operators mentioned above until desired criteria are met.[1]
- **Mesh Decimation (Simplification):** A specific form of remeshing focused on reducing the number of triangles while preserving the overall shape and important features.[1]
    - **Generic Algorithm:** Iteratively collapse edges based on a cost metric, typically managed with a priority queue.[1]
    - **Cost Metrics for Edge Collapse:**
        - **Edge Length:** Simple but not robust; can destroy important features.[1]
        - **Curvature-based:** Penalizes collapses that would flatten high-curvature areas.[1] The cost function cost(u,v)=∣∣u−v∣∣×maxf∈Tuming∈Tuv{(1−nf⋅ng)/2} incorporates this.
        - **Quadric Error Metrics (Garland & Heckbert '97):** This is a very effective and widely used metric. For each vertex v, a 4×4 symmetric matrix Qv (the error quadric) is stored. Qv represents the sum of squared distances from v to the planes of its incident triangles. Specifically, for a plane p=[a b c d]T (where ax+by+cz+d=0), the squared distance from v=[x y z 1]T is (pTv)2. Qv=Σp∈planes(v)ppT. The error at v is vTQvv. When collapsing an edge (v1,v2) to a new vertex v′, the new quadric is Q′=Q1+Q2. The optimal position for v′ is the one that minimizes v′TQ′v′. This position can be found by solving a small linear system derived from setting the gradient ∇(v′TQ′v′)=0. If the system is singular (e.g., in flat regions), v′ can be chosen as v1,v2, or their midpoint.[1] The derivation for Q′=Q1+Q2 correctly

    reflects that the new vertex v′ should ideally satisfy the plane equations of both original vertices. The minimization of v′TQ′v′ to find the optimal v′ is a standard quadratic minimization, leading to a linear system for v′. The initial error vTQvv is 0 for all original vertices as they lie on their incident planes.
- **Progressive Meshes (Hoppe '96):** A representation that stores a base mesh and a sequence of vertex split operations (inverse of edge collapses) to progressively add detail. This allows for efficient level-of-detail (LOD) rendering.[1]

Remeshing strategies often balance improving element quality with preserving the original geometry. Aggressive remeshing for ideal element shapes might smooth out or distort important features if not guided by appropriate error metrics or feature preservation schemes (e.g., not flipping feature edges, collapsing along features).[1] The iterative application of local operators is a common paradigm, where the cumulative effect of these simple changes leads to a global improvement in mesh quality or a desired level of simplification.

Section 7: Subdivision Surfaces

Subdivision surfaces provide a powerful method for generating smooth surfaces from coarse polygonal control meshes (base meshes). The process is iterative: in each step, the mesh is refined by adding new vertices and faces (topological refinement), and then the positions of both new and existing vertices are updated according to specific geometric rules (geometry update).[1] This iterative process converges to a smooth limit surface. Subdivision is an alternative to NURBS patches, particularly advantageous for modeling complex topologies as it avoids the difficulties of stitching multiple NURBS patches with high continuity.[1]

Key Subdivision Schemes:

1. **Loop Subdivision (for Triangle Meshes):**
    - **Topological Refinement:** Each triangle is split into four smaller triangles by inserting a new vertex at the midpoint of each edge and connecting these midpoints.[1]
    - **Geometric Update Rules:** [1]
        - **Edge Vertex (New Vertex):** For an edge (v0,v2) shared by triangles (v0,v2,v1) and (v0,v2,v4), the new vertex u on this edge is positioned at: u=83(v0+v2)+81(v1+v4)
        - **Vertex Vertex (Updated Old Vertex):** An existing vertex vi with valence d and neighbors vneighborj is updated to vi′: vi′=(1−dβ)vi+βΣj=1dvneighborj The coefficient β depends on the valence d: If d=3, β=3/16. If d>3, β=d1(85−(83+41cosd2π)2). A common simplification (Warren's rule) is β=d(d+2)3 for d>3, though the original Loop formula is more precise for achieving C2 continuity at regular valence 6 vertices and C1 elsewhere. The formulas on slide 8 of 04-structures-programming.pdf use α instead of β, where α=β.
    - Loop subdivision is an approximating scheme (the limit surface does not generally interpolate the original control points except at extraordinary vertices under certain conditions). It produces C2 continuous surfaces at regular vertices

(valence 6) and C1 continuous surfaces at extraordinary vertices.

2. **Catmull-Clark Subdivision (for Quadrilateral Meshes, also handles triangles):**
   - **Topological Refinement:**
     - A **face point** is created for each face (average of its vertices).
     - An **edge point** is created for each edge (average of its endpoints and the two adjacent new face points).
     - Each original vertex is moved to a **vertex point**.
     - New faces are formed by connecting face points to edge points, and edge points to vertex points. This typically turns an original quadrilateral face into four new quadrilaterals. Triangles are handled by first converting them to quads or using special rules.
   - **Geometric Update Rules:** [1]
     - **Face Point fp:** Average of all original vertices of the face.
     - **Edge Point ep:** For an edge with endpoints v1,v2 and adjacent face points fp1,fp2: ep=4v1+v2+fp1+fp2.
     - **Vertex Point vp′ for original vertex vp with valence d:** vp′=dQ+d2R+dS(d−3) where Q is the average of the new face points of faces incident to vp, R is the average of the midpoints of original edges incident to vp, and S is vp. (The slide uses slightly different notation/weights: Face point weights 1/4 for each vertex. Edge point weights 3/8 for endpoints, 1/16 for "wing" vertices. Vertex point weights depend on valence, e.g., for valence 4: 9/16 for original vertex, 3/32 for each direct edge neighbor, 1/64 for each diagonal neighbor in the 1-ring quad patch.)
   - Catmull-Clark is also an approximating scheme, producing C2 surfaces at regular vertices (valence 4 for quads) and C1 at extraordinary vertices. It is widely used in animation (e.g., Pixar's Geri's Game).
3. **Other Subdivision Schemes:**
   - **3-Subdivision (Triangles):** Involves adding new vertices at face midpoints, connecting them, and flipping original edges. Vertex positions are also updated.[1]
   - **Doo-Sabin Subdivision (General Polygons, typically Quads):** A dual scheme where new vertices are created within each original face, effectively "shrinking" original faces and creating new faces at original vertices and edges.[1]
   - **Butterfly Subdivision (Triangles):** An interpolating scheme, meaning the original vertices remain on the limit surface.[1]
   - **Phong Tessellation (Triangles):** An interpolating scheme that uses tangent planes at mesh vertices to create quadratic patches, aiming for smooth surfaces without the shrinkage typical of approximating schemes.[1]

Subdivision schemes are powerful because they allow the creation of complex, smooth surfaces from simple, coarse control meshes. The specific rules of each scheme determine the properties of the limit surface, such as its order of continuity (C1,C2, etc.) and whether it interpolates or approximates the initial control points. The local nature of the rules means that changes to one control point have a localized effect on the limit surface, making them intuitive for interactive modeling.

Section 8: Mesh Parameterization

Mesh parameterization is the process of finding a bijective (one-to-one and onto) mapping between a 3D surface mesh and a simpler 2D domain, typically a planar region like a disk or square, or sometimes a sphere for closed genus-0 meshes.[1] This mapping essentially "unwraps" or "flattens" the 3D surface onto the 2D domain.
Motivation and Applications:
The primary motivation is to transfer information or operations between the 2D domain and the 3D surface. Key applications include:

- **Texture Mapping:** Applying a 2D image (texture) to the 3D surface. The (u,v) coordinates from the parameterization are used to look up colors in the texture image.[1]
- **Remeshing:** A high-quality 2D triangulation (e.g., Delaunay) or grid pattern can be generated on the 2D parameter domain and then mapped back to the 3D surface to create a well-structured 3D mesh.[1]
- **Shape Correspondence and Morphing:** Parameterizing two shapes to a common domain (e.g., a sphere) can help establish correspondences between them, which can then be used for morphing.[1]

Types of Parameterization and Distortion:
An ideal parameterization would preserve geometric properties like angles, areas, and lengths.

- **Conformal (Angle-Preserving):** Angles between intersecting curves on the 3D surface are preserved in their 2D mapping.
- **Equiareal (Area-Preserving):** The area of any region on the 3D surface is proportional to the area of its 2D mapping.
- **Isometric (Length-Preserving):** The length of any curve on the 3D surface is the same as the length of its 2D mapping. An isometric mapping is both conformal and equiareal. .[1] Achieving a perfect isometric mapping is generally impossible unless the surface is developable (can be flattened without stretching or tearing, like a cylinder or cone). Therefore, practical parameterization methods aim to minimize some form of **distortion**. The deformation gradient Ji (a 2×2 matrix for each 2D triangle, derived from the mapping of its 3D counterpart) is used to quantify distortion. Its singular values σ1,σ2 are key:
- Conformal Distortion: diconf=21(σ2σ1+σ1σ2). Perfect if σ1=σ2.
- Areal Distortion: diarea=21(det(Ji)+(det(Ji))−1). Perfect if det(Ji)=1.
- Isometric Distortion can be a weighted sum: diiso=αdiconf+(1−α)diarea.[1] The deformation gradient F (or Ji) for a triangle maps its 3D edge vectors (in a local frame) to its 2D edge vectors. If the 3D triangle has vertices v1,v2,v3 and their 2D images are u1,u2,u3, then F transforms edges like (v2−v1) and (v3−v1) to (u2−u1) and (u3−u1). It can be computed as F=DsDm−1, where Ds contains the 2D edge vectors and Dm contains the 3D edge vectors (possibly in a local tangent frame).[1] The derivation on slide 76 is standard for finding the affine map between two triangles.

Linear Parameterization Methods (Disk Parameterization):
These methods typically involve two stages: fixing the 2D positions of the boundary vertices of the 3D mesh, and then solving a linear system $Wx=b$ for the positions of the interior vertices.1

1. **Boundary Mapping:** The boundary loop of the 3D mesh (assuming it's disk-like) is mapped to a convex polygon in 2D, often a circle. A common technique is to distribute the boundary vertices around the circle proportionally to their original 3D edge lengths: the angle ψi subtended by edge ei is $\psi i=\Sigma j||ej||2\pi||ei||$. The 2D position (ui,vi) is then $r(\cos(\Psi i),\sin(\Psi i))$ where $\Psi i=\Sigma j=1i\psi j$.[1]
2. **Interior Vertex Placement (Solving Wx=b):**
   - **Uniform Weights (Tutte Embedding):** $wij=1$ for connected vertices (i,j). Each interior vertex is placed at the average of its neighbors. This guarantees a bijective mapping if the boundary is mapped to a convex polygon, but can result in significant angular and areal distortion.[1] The system matrix W has $Wii=di$ (degree of vi) and $Wij=-1$ if vi,vj are connected (for interior rows), or $Wii=1,Wij=0$ for boundary rows. The vector b contains zeros for interior rows and fixed boundary coordinates for boundary rows.
   - **Harmonic (Cotangent) Weights:** $wij=21(\cot\alpha ij+\cot\beta ij)$. This aims for a conformal (angle-preserving) map. It can produce non-bijective results (flipped triangles) if the 3D mesh has obtuse triangles, unless the input mesh is Delaunay or the weights are modified.[1]
   - **Mean-Value Weights:** $wij=2||vi-vj||\tan(\gamma ij/2)+\tan(\delta ij/2)$. These weights are guaranteed to be positive for non-degenerate polygons, ensuring a bijective mapping for convex boundaries. They also aim for conformal mapping but can sometimes introduce more angular distortion than cotangent weights.[1] The choice of weights in $Wx=b$ significantly influences the distortion characteristics of the resulting parameterization. Uniform weights are simple but often produce poor quality. Cotangent weights are theoretically linked to conformality but have issues with obtuse triangles. Mean-value weights offer a robust alternative for bijectivity.

Free-Boundary Parameterization:
Fixed-boundary methods can perform poorly if the 3D mesh boundary is complex or very different from the target 2D boundary. Free-boundary methods compute the 2D boundary shape as part of the optimization, often leading to lower distortion.1 Multidimensional Scaling (MDS) based methods, which aim to preserve geodesic distances from 3D in the 2D Euclidean distances, are inherently free-boundary.1
Parameterization of Closed Meshes:
Closed meshes (like spheres or tori) must be cut to create one or more boundaries before they can be mapped to a 2D plane. These cuts, or seams, can introduce artifacts in applications like texture mapping.1

- **Cutting:** Algorithms often find cuts along paths of high curvature or use spanning trees to create a disk-topology surface.[1]
- **Charts:** Alternatively, the surface can be segmented into multiple patches (charts),

each parameterized independently. This is common for complex surfaces to minimize distortion within each chart.[1]

- **Spherical Parameterization:** For genus-0 closed meshes, mapping directly to a sphere avoids cuts and provides a seamless parameterization. This is useful for inter-surface mapping and morphing.[1] Methods include:
  1. Cutting one triangle, mapping to a planar triangle, then inverse stereographic projection to sphere.[1]
  2. Cutting into two disks, mapping each to a hemisphere.[1]
  3. Direct projection from an interior point (kernel point) for star-shaped objects.[1]
  4. Iterative smoothing and projection for general genus-0 meshes.[1]

The parameterization of complex topologies or geometries often necessitates a "cut-and-flatten" approach, where the surface is decomposed into simpler patches. While this reduces distortion within individual patches, managing continuity and minimizing artifacts across the seams (cuts) becomes a significant challenge.

Section 9: Shape Registration (Rigid Alignment)

Shape registration, or alignment, is the process of transforming different sets of data (e.g., point clouds, meshes) into a common coordinate system. This section focuses on **rigid alignment**, where shapes differ only by a rigid transformation (rotation R and translation t).[1] This is easier than non-rigid alignment due to fewer degrees of freedom and is a core task in applications like merging multiple 3D scans.

PCA-based Alignment (for complete, overlapping shapes):

Principal Component Analysis (PCA) can be used for a global alignment if the two shapes are largely complete and have significant overlap.1

1. **Translation Alignment:** The centroids (mean of vertex positions) of the two shapes are computed and aligned, typically by translating both shapes so their centroids are at the origin.[1]
2. **Rotation Alignment:** PCA is performed on the vertex coordinates of each (centered) shape. The eigenvectors of the covariance matrix of the point coordinates define the principal axes of inertia (directions of largest variance) for the shape.[1] The shapes are then rotated so that their corresponding principal axes align with the global Cartesian axes, or with each other. The rotation matrix R can be formed by the eigenvectors.
   - The covariance matrix C for a set of centered points pi=(xi,yi,zi) has entries like C11=Σxi2, C12=Σxiyi, etc..[1] The eigenvectors of C are the principal directions.
   - To align a vector (e.g., a principal axis) with a Cartesian axis (e.g., y-axis), a sequence of rotations around other Cartesian axes can be computed using trigonometry (e.g., zAngle = acos(y/xyLength), xAngle = acos(xyLength/vecLength)).[1]

- **Limitations:** PCA-based alignment is a global method. If shapes have only partial overlap or significantly different extents, their principal axes may not correspond meaningfully, and the alignment will likely fail.[1] However, it can still provide a reasonable initial guess for iterative methods.

Alignment with Known Correspondences:

If a set of n corresponding point pairs {(pi,qi)} between the source shape P and target shape Q is known, the optimal rigid transformation (R,t) that minimizes the sum of squared distances E(R,t)=Σi=1nwi||(Rpi+t)−qi||2 (where wi are weights) can be found.1

1. **Optimal Translation t:** Given R, t is found by setting ∂t∂E=0. This yields t=q¯−Rp¯, where p¯ and q¯ are the weighted centroids of the point sets P and Q respectively.[1] The handwritten derivation on slide 34 of 10-shapereg.pdf correctly derives this result.
2. **Optimal Rotation R:** After translating both point sets so their centroids are at the origin (i.e., xi=pi−p¯, yi=qi−q¯), the problem reduces to finding R that minimizes Σwi||Rxi−yi||2. This is equivalent to maximizing trace(RS), where S=XWYT is the weighted cross-covariance matrix (X,Y are matrices of centered points). If S=UΣVT is the Singular Value Decomposition (SVD) of S, then the optimal rotation is R=VUT (assuming det(VUT)=1; if not, a correction is needed for reflection cases, typically by flipping the sign of the column of V corresponding to the smallest singular value if det(VUT)=−1).[1] The handwritten derivation on slide 35 of 10-shapereg.pdf correctly outlines this SVD-based solution to the orthogonal Procrustes problem.

Iterative Closest Point (ICP) Algorithm:

ICP is used when correspondences are unknown, especially for aligning partially overlapping scans.1 It's an iterative algorithm:

1. **Initialization:** Start with an initial guess for the transformation (R,t).
2. **Find Correspondences:** For each point in the (transformed) source shape, find its closest point in the target shape. K-d trees are often used for efficient closest point queries.[1]
3. **Filter Correspondences (Optional):** Reject pairs that are too far apart, or whose normals are too different, or that are on boundaries.[1]
4. **Compute Transformation:** Given the current set of correspondences, compute the optimal (R,t) using the method for known correspondences (SVD-based).
5. **Apply Transformation:** Apply the computed (R,t) to the source shape.
6. **Iterate:** Repeat steps 2-5 until convergence (e.g., change in error is small, or max iterations reached).[1]

- **Error Metrics:**
  - **Point-to-Point:** Minimize Σ||Rpi+t−qi||2.[1]
  - **Point-to-Plane:** Minimize Σ((Rpi+t−qi)⋅ni)2, where ni is the normal at qi. This often converges faster and more accurately, especially for planar regions, as it uses surface normal information.[1] The derivation often involves linearizing the rotation for small angles.
- **Improvements:** Using feature points (detected using local shape descriptors like Spin Images, HKS, SDF) to establish more reliable initial correspondences can significantly improve ICP's robustness and speed.[1] Sampling strategies that ensure a good distribution of normals can prevent bias towards large flat areas.[1]

ICP's effectiveness hinges on the "chicken-and-egg" problem: good transformations require

good correspondences, and good correspondences require the shapes to be nearly aligned. A good initial alignment (e.g., from manual placement, PCA, or feature matching) is crucial for ICP to converge to the correct global minimum rather than a local one. The choice of error metric also plays a significant role, with point-to-plane often preferred for its use of surface information, leading to better guidance during the alignment process.
RANSAC (Random Sample Consensus) for Alignment:
RANSAC is a robust method for estimating transformation parameters in the presence of a large number of outliers (incorrect correspondences).1

1. Randomly select a minimal set of point pairs required to define the transformation (e.g., 3 non-collinear pairs for 3D rigid transformation).
2. Compute the transformation (R,t) from this minimal set.
3. Count the number of "inliers": other point pairs from the dataset that agree with this transformation within a certain tolerance.
4. Repeat steps 1-3 many times.
5. The transformation that has the largest number of inliers is chosen as the best estimate. This transformation can then be refined using all inliers with a standard least-squares fit (like ICP on the inlier set).[1] RANSAC is powerful because it focuses on finding a consensus among a subset of data, making it less susceptible to being skewed by outliers compared to methods like standard least-squares that consider all points.

Section 10: Mesh Deformation

Mesh deformation involves modifying the shape of a mesh by changing its vertex positions while maintaining a plausible appearance and, ideally, preserving local details.[1]
Free-Form Deformation (FFD):
FFD deforms an object by deforming the ambient space in which it is embedded. A lattice or cage is defined around or within the object. The user manipulates the control points of this lattice/cage, and the embedded object deforms accordingly.1

- **Mechanism:** Each vertex of the object is expressed in terms of local coordinates within its containing cell of the lattice/cage (e.g., using trilinear interpolation for a cubic lattice, barycentric coordinates for tetrahedral or triangular cells, or more general coordinates like Wachspress or Mean Value Coordinates for arbitrary polygonal/polyhedral cells).[2] When the lattice/cage deforms, these local coordinates are used to recompute the vertex's global position within its (now deformed) cell.
  - **Barycentric Coordinates:** For a triangle with vertices A,B,C, any point P=αA+βB+γC where α,β,γ≥0 and α+β+γ=1. α=Area(PBC)/Area(ABC), etc..[1]
  - **Mean Value Coordinates:** For an arbitrary polygon with vertices vj and point v inside, v=Σλjvj, where λj=wj/Σwk and wj=||vj−v||tan(αj/2)+tan(αj−1/2) (αj are angles at v formed by edges (v,vj) and (v,vj+1)).[1]

Skinning (Linear Blend Skinning - LBS):
A common technique in character animation where an underlying "skeleton" (hierarchy of bones/joints) controls the deformation of the surface "skin" mesh.1

- **Mechanism:** Each skin vertex pi is assigned weights wij that determine its influence by each bone j. When a bone j undergoes a transformation T_j (relative to its initial "bind pose" transformation B_j), the new position of vertex p_i is a weighted sum of its transformed positions according to each bone: $p_i' = \Sigma_j w_{ij}(T_j B_j^{-1} p_i^{bind})$.[1]

Laplacian-based Deformation:

These methods aim to preserve the local shape details of the mesh, which are captured by differential coordinates δ_i. Recall $\delta_i = v_i - \text{avg_neighbors}(v_i)$ using uniform or cotangent weights.1 The goal is to find new vertex positions v' such that their differential coordinates Lv' are close to the original differential coordinates $\delta_0 = Lv_0$, while satisfying user-imposed positional constraints $v_k' = c_k$ for handle vertices.

- **Energy Minimization:** The problem is typically formulated as minimizing an energy function: $E(v') = \Sigma_{\text{handles } k} ||v_k' - c_k||^2 + \alpha ||Lv' - \delta_0||^2$ The first term is the **constraint/match term** (pulling handles to target positions c_k). The second is the **regularization term** (preserving shape). α balances these terms.
- **Solving:** This quadratic energy function leads to a sparse linear system of equations when its gradient is set to zero. $(A^T A + \alpha L^T L)v' = A^T c + \alpha L^T \delta_0$ (general form, where A selects handle vertices and c contains their target positions). The slides show specific forms: $(I + \alpha L^T L)v' = c + \alpha L^T \delta_0$ if all vertices are soft-constrained towards c.[1] $(W + \alpha L^T L)v' = Wc + \alpha L^T \delta_0$ if W is a diagonal matrix selecting hard or weighted soft constraints.[1] The handwritten derivations for these linear systems by taking the gradient of the energy function and setting it to zero are standard least-squares procedures and are correctly outlined.
- **Limitations:** Standard Laplacian deformation is not rotation-invariant. Large rotations of handles can cause unnatural shearing or volume loss because the differential coordinates δ_0 are defined in the global frame of the rest pose.[1]

Physically-based Deformation:

Simulates deformation based on physical laws, typically Newton's $F=ma$. Vertex positions and velocities are updated over time using numerical integration (e.g., Euler integration).1

$v_{new} = v_{old} + \Delta t \cdot a_{old}$

$p_{new} = p_{old} + \Delta t \cdot v_{new}$

Forces can include springs between vertices, gravity, user interaction, etc.

As-Rigid-As-Possible (ARAP) Deformation:

A powerful technique that aims to make each local region (e.g., triangle or its 1-ring neighborhood) deform as rigidly as possible. It allows for large, intuitive deformations while preserving details well.1

- **Energy:** $E_{ARAP}(v') = \Sigma_i \Sigma_{j \in N(i)} w_{ij} ||(v_i' - v_j') - R_i(v_i - v_j)||^2$. This energy penalizes deviations from local rigidity. For each vertex i (or cell i), an optimal rotation R_i is found that best aligns its deformed 1-ring edges $(v_i' - v_j')$ with its rest-pose edges $(v_i - v_j)$.
- **Alternating Optimization:** [1]
    1. **Local Step (Rotation Fitting):** For fixed vertex positions v', find the optimal rotation R_i for each local region i. This is typically done by computing a local covariance matrix for the edges and finding its closest rotation matrix using SVD (similar to rigid alignment rotation finding).

2. **Global Step (Vertex Update):** For fixed rotations Ri, find new vertex positions v′ that minimize the ARAP energy. This step involves solving a sparse linear system, similar in structure to Laplacian deformation systems. These two steps are iterated until convergence (usually 5-6 iterations suffice).

- **Deformation Gradient Fj:** For an element j (e.g., triangle), Fj is the 2×2 (or 3×3 for tets) matrix that transforms its rest-pose edge vectors to its deformed edge vectors. Fj=Ds,jDm,j−1.[1] The closest rotation Rj to Fj is found via SVD of Fj.

The balance between local detail preservation and global user control is a central challenge in mesh deformation. Laplacian methods achieve this by preserving differential coordinates. ARAP takes this further by aiming for local rigidity, which is more robust for large rotations. FFD and Skinning offer intuitive control by manipulating a simpler structure (cage/skeleton) but might not always offer the finest control over surface details without very dense control structures.

Part III: Advanced Topics, Applications, and Exam Specifics

Section 11: Geometric Fitting and Generation

This section covers algorithms for fitting geometric primitives to data and generating new shapes, often based on statistical models learned from examples.
Least-Squares Fitting:
A fundamental technique for finding the model parameters that best fit a set of observed data by minimizing the sum of the squares of the residuals (differences between observed and predicted values).

1. **Line Fitting (y=ax+b, Vertical Distance):** The error function is E(a,b)=Σ(yi−(axi+b))2. Setting partial derivatives ∂a∂E=0 and ∂b∂E=0 yields a 2×2 system of linear equations (normal equations) for a and b: [Σxi2ΣxiΣxin][ab]=[ΣxiyiΣyi] .[1] The handwritten derivation on slide 3 of 13-leastsquares-ransac-hough.pdf is correct.
2. **Polynomial Fitting (y=Σakxk):** A generalization where the error E(a0,...,am)=Σ(yi−Σakxik)2 is minimized. This also leads to a system of linear normal equations for coefficients ak.[1]
3. **Plane Fitting (n·x+d=0, Normal Distance):** The error is E(n,d)=Σ(nTpi+d)2 subject to ∣∣n∣∣=1. Minimizing with respect to d gives d=−nTp¯ (where p¯ is the centroid of points pi). Substituting d, the error becomes E(n)=Σ(nT(pi−p¯))2=nTCn, where C=Σ(pi−p¯)(pi−p¯)T is the covariance matrix of the centered points. Minimizing nTCn subject to nTn=1 is an eigenvalue problem: Cn=λn. The normal n that minimizes the error is the eigenvector corresponding to the smallest eigenvalue of C.[1] The handwritten derivation on slides 6-7 of 13-leastsquares-ransac-hough.pdf correctly uses Lagrange multipliers to arrive at this eigenvalue problem.

RANSAC (Random Sample Consensus):
An iterative method robust to outliers when fitting models.1

1. Randomly select a minimal subset of data points required to fit the model.
2. Fit the model to this subset.
3. Count the number of "inliers" – other data points that fit this model within a tolerance.
4. Repeat for many iterations. The model with the most inliers is chosen. RANSAC is effective when the percentage of inliers is reasonably high.

Hough Transform:
A feature extraction technique that uses a voting procedure in a parameter space to identify instances of specific shapes (e.g., lines, circles, planes).1

1. Each data point "votes" for all possible model parameters that could generate it.
2. Votes are accumulated in a discretized parameter space (accumulator array).
3. Peaks in the accumulator correspond to the parameters of the most likely shapes. It is robust to noise and occlusion but can be computationally intensive for models with many parameters.

Least-squares is efficient but highly sensitive to outliers. RANSAC and Hough Transform offer robustness to outliers but at a higher computational cost. The Hough Transform's power lies in its ability to convert a detection problem in data space into a peak-finding problem in a more structured parameter space.
Shape Generation (PCA-based):
PCA can be used not only for alignment but also to learn a statistical "shape space" from a collection of example shapes, enabling the generation of new, plausible instances.1

1. **Learning the Shape Space:** Given N example shapes (aligned and in correspondence, each represented as a long vector of vertex coordinates), compute the mean shape m and the principal components (eigenvectors p_k of the covariance matrix of the shape vectors). Any shape can then be approximated as $x \approx m + \sum_{k=1}^{K} b_k p_k = m + Pb$, where b_k are PCA coefficients and $K \ll$ dimensionality of x.[1]
2. **Karhunen-Loeve Transform (KLT):** For high-dimensional shape vectors (e.g., 3V for V vertices), the covariance matrix YY^T is huge ($3V \times 3V$). If the number of shapes $N \ll 3V$, KLT provides an efficient way to find the $N-1$ non-zero eigenvalues and corresponding eigenvectors. It solves the eigenproblem for the smaller $N \times N$ matrix Y^TY. If $Y^TYq_i = \lambda_i q_i$, then the eigenvectors p_i of YY^T are $p_i = Yq_i$.[1] The derivation on slide 47 of 07-shapegen.pdf is correct.
3. **Correlating Semantics with Shape:** If semantic attributes s (e.g., height, weight) are available for the example shapes, a mapping C can be learned such that $b = Cs$. For new semantic attributes s_{new}, new PCA coefficients $b_{new} = Cs_{new}$ can be computed, and a new shape $x_{new} = m + Pb_{new}$ generated.[1]

PCA provides a powerful method for dimensionality reduction and for building generative models of shape, capturing the main modes of variation within a dataset.

Section 12: Mesh Comparison, Analysis, and 3D Printing

Mesh Comparison and Analysis:
Comparing and analyzing shapes often requires robust distance measures and descriptive features.

- **Distance Measures:**
  - **Euclidean Distance:** Straight-line distance, sensitive to pose and deformation.
  - **Geodesic Distance:** Shortest path distance along the surface of the mesh. Computed using graph algorithms like Dijkstra's or A* on the mesh edges, or more accurately with methods like Fast Marching that consider paths through faces.[1]
  - **Diffusion Distance:** Based on heat flow simulation. $k_t(x,y)=\sum e^{-\lambda_i t}\phi_i(x)\phi_i(y)$ is the amount of heat transferred from x to y in time t (where λ_i,ϕ_i are Laplacian eigenpairs). Diffusion distance $d_D(x,y)^2=\sum e^{-2t\lambda_k}(\phi_k(x)-\phi_k(y))^2$ is more robust to topological noise than geodesic distance.[1]
- **Shape Descriptors:** Compact representations of shape features.
  - **Global Descriptors:** Characterize the entire shape (e.g., Shells histograms, Shape Distributions, LightField descriptors).[1]
  - **Local Descriptors:** Characterize the geometry around a point (e.g., Shape Contexts, Spin Images, curvature, Average Geodesic Distance (AGD), Shape Diameter Function (SDF), Intrinsic Girth Function (IGF), Heat Kernel Signature (HKS) $k_t(x,x)$).[1] These are crucial for tasks like feature matching in registration or correspondence.

Intrinsic measures like geodesic and diffusion distances, along with intrinsic descriptors, are generally preferred for shape comparison when invariance to pose or non-rigid deformation is desired, as they capture the inherent structure of the surface.
3D Printing (Additive Manufacturing):
The process of creating physical 3D objects from digital models, typically by adding material layer by layer.1

- **Technologies:**
  - **FDM (Fused Deposition Modeling):** Extrudes molten filament.[1]
  - **SLA (Stereolithography):** Cures liquid resin with a UV laser.[1]
  - **SLS (Selective Laser Sintering):** Fuses powder with a laser.[1]
  - **SLA/DLP (Digital Light Processing):** Cures resin using a projector.[1]
- **Processing Pipeline:** Design → Tessellation (mesh creation) → Orientation (optimizing build direction) → Support Structure Generation → Slicing → Toolpath Generation → Printing.[1]
- **Input Requirements:** Meshes must typically be **watertight** (closed, 2-manifold, no holes) to define a clear interior/exterior. Thin sheets must be thickened.[1]
  - **Hole Filling:** Often necessary. Algorithms like the one by Liepa (2003) use dynamic programming for coarse triangulation, followed by refinement and smoothing. The DP recurrence is $w[a,c]=\min_b(w[a,b]+w(\Delta abc)+w[b,c]')$, where $w(\Delta abc)$ is the cost of triangle (a,b,c) (e.g., area + max dihedral angle penalty).[1]
- **Printing Fidelity and Issues:**
  - **Staircase Effect:** Visible layers due to the discrete slicing process, especially on shallow slopes.[1]
  - **Cusp-Height Error:** A measure of the staircase effect, $h\approx l|\cos\theta|$ where l is layer

height and θ is the angle between surface normal and build direction.[1]

- **Support Structures:** Needed for overhangs (parts not supported by lower layers) to prevent collapse during printing. Minimizing supports is an optimization problem.[1]

The requirements of 3D printing (e.g., watertightness, manufacturability of overhangs) drive significant research in DGP, including algorithms for mesh repair, geometric optimization for printability, and support structure generation.

Section 13: Key Algorithms and Concepts Summary

The following table summarizes some of the most critical algorithms and concepts discussed, highlighting their purpose and core ideas.

Concept/Algorithm	**Purpose**	**Core Idea / Key Formula/Steps**	**Typical Complexity / Notes**	**Relevant Slides (Primary)**
Euler's Formula	Relate V, E, F for planar graphs/polyhedra	V−E+F=2−2g	Fundamental topological invariant.	[1], p36-p50
Polygon Triangulation	Decompose polygon into triangles	Add n−3 non-crossing diagonals to get n−2 triangles.	Ear clipping is one method.	[1], p13-p27
Delaunay Triangulation	"Good" triangulation of point sets	Empty circumcircle property; maximizes minimum angle. Edge flips to achieve.	O(nlogn) for construction.	[1], p31-p38
Voronoi Diagram	Partition space based on closest sites	Each region $R_k = \{x \mid \$	\	x-p_k\
Indexed Face-Set	Common mesh data structure	List of unique vertices; faces store indices to vertices.	Memory efficient for geometry; topology queries can be slow.	[1], p14-p15
Laplacian (Cotangent)	Discrete Laplace-Beltrami operator	Lii=Σkwik, Lij=−wij, wij=21(cotαij+cotβij)	Symmetric, geometry-aware.	[1], p17, p20-21
Laplacian Eigenanalysis	Analyze mesh properties via spectrum	Solve Lϕ=λϕ or Lϕ=λAϕ.	Eigenvalues reveal connectivity, Fiedler vector for	[1], p23-p33

			partitioning.	
Marching Cubes	Extract iso-surface from scalar field	Process grid cells, use look-up table based on 8 vertex signs to generate triangles.	$O(\text{num_cells})$. Ambiguity handling is important.	[1], p32-p37, p62-p69
Laplacian Smoothing	Remove mesh noise	v′=v+λL(v). Cotangent weights preferred. Taubin smoothing for anti-shrink.	Iterative. Uniform weights cause shrinkage.	[1], p4-p20
Error Quadric Simpl.	Mesh decimation preserving features	Collapse edges minimizing v′T(Q1+Q2)v′. Qi from sum of squared plane distances.	Greedy, O((E+V)logE) with heap.	[1], p61-p70
Loop Subdivision	Smooth triangle meshes	Split edges, update old vertex vi′=(1−dβ)vi+βΣvneigh, new edge vertex u=83(va+vb)+81(vc+vd).	Approximating, C2 at regular (valence 6) vertices.	[1], p7-p8; [1], p103-p105, p116-117
Disk Parameterization	Map 3D mesh to 2D disk	Fix boundary vertices in 2D, solve Wx=b for interior (e.g., using cotangent weights).	Quality depends on weights and boundary map.	[1], p24-p43
ICP (Iterative Closest Pt)	Rigidly align point clouds	Iterate: 1. Find closest points. 2. Compute R, t. 3. Transform.	Converges locally. Point-to-plane often better.	[1], p36-p53
Laplacian Deformation	Deform mesh preserving local detail	Minimize $E(v') = \$	\	Lv' - L v_0\
ARAP Deformation	As-Rigid-As-Possible deformation	Minimize $\sum \$		
\	(v'_i - v'_j) - R_i (v_i - v_j) \	\	^2. Alternatingly solve for Ri (SVD) and v′ (linear	Rotation-invariant, good for large deformations.

			system).	
Least-Squares Fitting	Fit model to data by min sum of squared errors	Line: y=ax+b, solve normal equations. Plane: n·x+d=0, n is eigenvector of smallest eigenvalue of covariance matrix.	Efficient, but sensitive to outliers.	[1], p3-p7
RANSAC	Robust model fitting with outliers	Iteratively sample minimal set, fit model, count inliers.	Probabilistic, handles high outlier rates.	[1], p8-p11
Hough Transform	Detect shapes by voting in parameter space	Points vote for parameters of shapes they could belong to. Peaks in accumulator = shapes.	Robust to noise/occlusion, but parameter space can be large.	[1], p12-p15

Section 14: Tackling Calculation-Based Questions

The final examination may include questions requiring direct calculations. Based on the course material, these are likely to involve small, illustrative examples rather than extensive numerical computation.

- **Laplacian Computations:**
  - **Uniform Laplacian Matrix:** Given a small graph (e.g., 3-4 vertices, 3-4 edges), construct its adjacency matrix A and degree matrix D. Then calculate L=D−A. For example, for a triangle graph with vertices 1,2,3 and edges (1,2), (2,3), (3,1): D=200020002, A=011101110, L=2−1−1−12−1−1−12.
  - **Cotangent Weights:** Given two triangles sharing an edge (vi,vj), with specified angles αij and βij opposite this edge, calculate wij=0.5(cotαij+cotβij). For instance, if αij=60◦ and βij=45◦, then wij=0.5(cot60◦+cot45◦)=0.5(1/3+1).
  - **Differential Coordinates:** For a vertex vk with neighbors v1,...,vd:
    - Uniform: δk=vk−d1Σi=1dvi.
    - Cotangent: δk=vk−(Σi=1dwkivi)/(Σi=1dwki).
- **Euler's Formula (V−E+F=2−2g):**
  - Given V,E,g, calculate F. E.g., for a closed genus-0 triangular mesh with V=10, E≈3V=30. Then 10−30+F=2⇒F=22. (Check: F≈2V=20; 3F=66≈2E=60. The approximations hold well).
  - Verify plausibility: If given V=8,E=12,F=6 for a claimed single-component closed mesh, this matches a cube (genus 0), as 8−12+6=2.
- Least-Squares Line Fitting (y=ax+b):

  Given 3-4 data points (xi,yi), set up the normal equations:
  (Σxi2)a+(Σxi)b=Σxiyi
  (Σxi)a+nb=Σyi
  Solve this 2×2 system for a,b. For example, points (0,0), (1,2), (2,1):
  n=3. Σxi=0+1+2=3. Σyi=0+2+1=3.
  Σxi2=02+12+22=5. Σxiyi=0∗0+1∗2+2∗1=4.
  5a+3b=4
  3a+3b=3⇒a+b=1⇒b=1−a.
  5a+3(1−a)=4⇒5a+3−3a=4⇒2a=1⇒a=0.5.
  b=1−0.5=0.5. So, y=0.5x+0.5.

- **SVD for Rotation (R=VUT):** Full SVD calculation by hand is unlikely. Questions would be conceptual, e.g., "Given S=UΣVT, what is the optimal rotation R?" or "What property of S is used to find R?"
- Deformation Gradient (F=DsDm−1):
  Given a 2D triangle with rest vertices v~1,v~2,v~3 and deformed vertices v1,v2,v3.
  Form edge vectors: e~1=v~2−v~1,e~2=v~3−v~1. Form Dm=[e~1∣e~2] (a 2×2 matrix).
  Similarly, e1=v2−v1,e2=v3−v1. Form Ds=[e1∣e2].
  Calculate Dm−1 (for a 2×2 matrix (acbd), inverse is ad−bc1(d−c−ba)).
  Then F=DsDm−1.
- **Barycentric Coordinates:** Given triangle vertices A,B,C and a point P. Calculate areas of △PBC,△PCA,△PAB,△ABC. Then α=Area(PBC)/Area(ABC), β=Area(PCA)/Area(ABC), γ=Area(PAB)/Area(ABC) (or γ=1−α−β). Area of a 2D triangle (x1,y1),(x2,y2),(x3,y3) is 0.5∣x1(y2−y3)+x2(y3−y1)+x3(y1−y2)∣.

For more complex numerical methods like SVD or solving large linear systems, the exam will likely focus on understanding the setup of the problem (e.g., forming the correct matrix L or covariance matrix C) and interpreting the meaning of the solution, rather than demanding full manual computation. Small-scale examples will serve to test the application of core formulas and algorithmic steps.

Section 15: Insights for Coding Questions

Coding questions on paper will likely test understanding of algorithms at a conceptual level, possibly requiring pseudocode or outlining the main steps of an implementation.

- **Data Structures:** A core requirement is often an indexed face-set (separate lists for vertex coordinates and faces defined by vertex indices).[1] For more advanced operations, pre-calculating or dynamically querying adjacency information (e.g., vertex-to-face lists, 1-ring neighbors) is common if a full half-edge structure is not assumed.
- **Common Loops:**
  - Iterating over all vertices for v_i in mesh.vertices:...
  - Iterating over all faces for f_j in mesh.faces:...
  - For a vertex v_i, iterating over its 1-ring neighbors (requires adjacency info).
  - For a face f_j, iterating over its vertices or edges.
- **Geometric Calculations:**

    - Vector operations: addition, subtraction, dot product, cross product, normalization, distance calculation.
    - Triangle properties: area, normal vector.
    - Angles: between vectors (using dot product and acos), or interior angles of polygons.
- **Algorithm Patterns:**
    - **Iterative Refinement:** Many algorithms (ICP, ARAP, iterative smoothing) follow a pattern of: initialize -> loop { compute something -> update state } until convergence.
    - **Solving Linear Systems:** For least-squares problems (Laplacian deformation, parameterization, fitting), the core is setting up a matrix A and vector b, then solving Ax=b. On paper, one might describe how to populate A and b.
    - **Priority Queues:** Used in algorithms like Dijkstra's shortest path or greedy mesh decimation (for edge collapses based on cost).
- **Example Pseudocode Snippet (Laplacian Smoothing - Uniform):**

  Code snippet
  ```
  function smooth_mesh(mesh, iterations, lambda):
    for iter from 1 to iterations:
      new_positions = array of size num_vertices
      for each vertex v_i in mesh:
        sum_neighbors = vector(0,0,0)
        degree = 0
        for each neighbor v_j of v_i:
          sum_neighbors = sum_neighbors + v_j.position
          degree = degree + 1
        if degree > 0:
          laplacian_vector = (sum_neighbors / degree) - v_i.position
          new_positions[i] = v_i.position + lambda * laplacian_vector
        else:
          new_positions[i] = v_i.position // Isolated vertex
      // Update all vertex positions simultaneously
      for each vertex v_i in mesh:
        v_i.position = new_positions[i]
    return mesh
  ```

Many DGP algorithms directly translate mathematical formulas into code. A solid grasp of the underlying mathematics (e.g., vector calculus for gradients, linear algebra for matrix operations) is crucial for correct implementation. Efficiently querying topological relationships is also key; if not provided by a rich data structure, helper functions to compute adjacency on-the-fly from an indexed face-set might be necessary.

Section 16: Consolidated Verification of Handwritten Derivations

This section consolidates the verification status of handwritten derivations found in the provided course materials. All referenced derivations were checked against standard texts, mathematical principles, or re-derived.

1. **Euler's Formula Inductive Proof [1]: Correct.** Standard inductive step for planar graphs.
2. **Mesh Statistics from Euler's Formula [1]: Correct.** Algebraic derivation for $F \approx 2V, E \approx 3V$, avg. degree ≈ 6 for closed triangular genus-0 meshes is sound, assuming large meshes.
3. **Platonic Solids Proof [1]: Correct.** The derivation using F=2n−mn+2m4m and analyzing $2n-mn+2m>0$ correctly constrains (n,m).
4. **Soccer Ball (12 Pentagons) Proof [1]: Correct.** Algebraic manipulation of Euler's formula with constraints from face types and vertex degrees is sound.
5. **K5 Non-Planarity Inequality [1]: Correct.** The use of $3F \leq 2E$ is appropriate for testing planarity limits.
6. **Proof of Vertex Degree ≤5 in Planar Graphs [1]: Correct.** Derivation from Euler's formula and $3F=2E$ (for triangulated version) correctly shows average degree <6.
7. **Polygon Triangulation (n−2 triangles) Inductive Proof [1]: Correct.** Standard inductive proof using diagonal splitting.
8. **Point Set Triangulation (Number of Triangles, 2n−h−2) Euler Proof [1]: Correct.** Correct application of Euler's formula with appropriate counting of V, E, F for a point set triangulation.
9. **Cotangent Laplacian Formula Derivation Elements [1]: Correct.** Derivation of cotαij from triangle area, law of cosines, and sine rule is standard.
10. **Differential Coordinates (Uniform & Cotangent) and General Laplacian L [1]: Correct.** Formulas represent standard definitions. Matrix L correctly captures the linear combination for δi.
11. **Least-Squares for Deformation/Compression (I+αLTL)v=c+αLTLv0 [1]: Correct.** Standard result from setting the gradient of the quadratic energy function to zero.
12. **Least-Squares for Deformation with W matrix (W+αLTL)v=Wc+αLTLv0 [1]: Correct.** Standard result for weighted least-squares with constraints.
13. **Least-Squares Line Fitting (Vertical Distance) Normal Equations [1]: Correct.** Standard derivation by minimizing sum of squared vertical errors.
14. **Least-Squares Plane Fitting (Normal Distance) Eigenvalue Problem [1]: Correct.** Standard derivation using Lagrange multipliers, leading to PCA-like solution.
15. **KLT Derivation for PCA [1]: Correct.** Algebraic manipulation showing pi=Yqi.
16. **Loop Subdivision Formulas [1]: Correct.** Matches standard Loop subdivision rules for edge and vertex point updates.
17. **Error Quadric Optimal Contraction Point [1]: Correct as a representation.** The formula shown is the solution to the linear system derived from setting the gradient of v′TQ′v′ to zero. The matrix shown is the coefficient matrix for the x,y,z coordinates, augmented for the homogeneous coordinate w=1.
18. **Deformation Gradient F=DsDm−1 [1]: Correct.** Standard method to find the affine transformation between two sets of corresponding vectors (edges).
19. **Optimal Translation $t=\bar{q}-R\bar{p}$ for Rigid Alignment [1]: Correct.** Standard result from

minimizing squared distances w.r.t. translation.

20. **Optimal Rotation R=VUT for Rigid Alignment [1]: Correct.** Standard SVD-based solution to the orthogonal Procrustes problem.
21. **Physically-based Deformation Energy Terms [1]: Correct.** Standard quadratic energy formulations and their gradient derivations for least-squares minimization.
22. **Cusp-Height Error Formula [1]: Correct.** Standard geometric formula for cusp height in layered manufacturing.
23. **Hole Filling DP Recurrence [1]: Conceptually Correct.** The dynamic programming structure w[a,c]=min(w[a,b]+w(Δabc)+w[b,c]′) is standard for optimal polygon triangulation problems. The specific cost w(Δabc) can vary.

All significant handwritten derivations presented in the course materials have been reviewed and found to be consistent with established mathematical principles and derivations in the field of digital geometry processing.

Conclusions

This report has systematically reviewed the core topics of the CENG 789 Digital Geometry Processing course, focusing on the material pertinent to the final examination. Key areas including geometric foundations, mesh data structures, fundamental operators like the Laplacian, surface reconstruction, smoothing, remeshing, subdivision, parameterization, registration, deformation, and geometric fitting have been detailed.

A critical aspect of this study has been the verification of mathematical derivations presented in the course materials, particularly those in handwritten form. All such derivations were found to be sound and consistent with established principles in the field. This provides a strong foundation for understanding the theoretical underpinnings of the algorithms discussed.

The study of various algorithms, from basic polygon triangulation to advanced deformation techniques like ARAP, highlights recurring themes:

1. **The Importance of Discretization:** Continuous geometric concepts are adapted to discrete mesh representations, with the choice of discretization (e.g., for the Laplacian) having significant impact.
2. **Trade-offs:** Many algorithms involve trade-offs, such as between computational efficiency and accuracy/robustness (e.g., least-squares vs. RANSAC), or between geometric fidelity and element quality (e.g., in remeshing).
3. **Local Operations for Global Results:** Complex global shape changes or analyses often arise from the iterative application of simple local rules or operators.
4. **Linear Algebra as a Core Tool:** Many problems are formulated as solving systems of linear equations or eigenvalue problems.

For the final examination, a strong grasp of the definitions, the purpose and high-level steps of key algorithms, and the ability to perform calculations on small examples (especially related to the Laplacian, Euler's formula, and least-squares fitting) will be essential. Understanding the principles behind coding these algorithms, particularly data structure access and fundamental geometric computations, will also be beneficial. The insights provided throughout this report aim to connect disparate topics and provide a deeper understanding of why certain methods are used and what their implications are.

Works cited

1. 13-leastsquares-ransac-hough.pdf
2. en.wikipedia.org, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Barycentric_coordinate_system#:~:text=The%20barycentric%20coordinates%20of%20a,Barycentric%20coordinates
3. Barycentric Coordinates - Ray-Tracing: Rendering a Triangle, accessed on June 5, 2025, https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates.html
4. Mesh Basics: Definitions, Topology & Data Structures, accessed on June 5, 2025, https://www.cs.ubc.ca/~sheffa/dgp/ppts/date_structures.pdf
5. Planar graph - Wikipedia, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Planar_graph
6. Euler's Formula & Platonic Solids, accessed on June 5, 2025, https://sites.math.washington.edu/~julia/teaching/445_Spring2013/Euler_Presentation.pdf
7. What is the Euler's Formula for Polyhedrons? - BYJU'S, accessed on June 5, 2025, https://byjus.com/maths/eulers-formula-for-polyhedra/
8. The soccer ball is made of 12 black pentagonal panels and 20 white hexagon panels. There are 90 edges where the panels joined together. How many edges join two white panels? - Quora, accessed on June 5, 2025, https://www.quora.com/The-soccer-ball-is-made-of-12-black-pentagonal-panels-and-20-white-hexagon-panels-There-are-90-edges-where-the-panels-joined-together-How-many-edges-join-two-white-panels
9. Counting the Number of Faces on A Soccer Ball - Mathematics Stack Exchange, accessed on June 5, 2025, https://math.stackexchange.com/questions/1620497/counting-the-number-of-faces-on-a-soccer-ball
10. Remarks on Kuratowski's Planarity Theorem, accessed on June 5, 2025, https://math.mit.edu/~djk/18.310/Lecture-Notes/Remarks-Kuratowski.pdf
11. Math 228: Kuratowski's Theorem, accessed on June 5, 2025, https://www.math.cmu.edu/~mradclif/teaching/228F16/Kuratowski.pdf
12. Euler concepts. Lecture 6. Planar graphs. Euler's Formula. - EECS 70, accessed on June 5, 2025, https://www.fa21.eecs70.org/assets/lec/lec-6-6up.pdf
13. 5.5 Map Colorings - Penn Math, accessed on June 5, 2025, https://www2.math.upenn.edu/~mlazar/math170/notes05-5.pdf
14. Polygon Triangulation, accessed on June 5, 2025, https://www.cs.jhu.edu/~misha/Spring16/02.pdf
15. Computational Geometry Triangulations and Guarding Art Galleries, accessed on June 5, 2025, https://ics.uci.edu/~goodrich/teach/geom/notes/Triangulation-slides.pdf
16. en.wikipedia.org, accessed on June 5, 2025,

https://en.wikipedia.org/wiki/Sch%C3%B6nhardt_polyhedron#:~:text=In%20geometry%2C%20a%20Sch%C3%B6nhardt%20polyhedron,tetrahedra%20without%20adding%20new%20vertices.

17. Schönhardt polyhedron - Wikipedia, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Sch%C3%B6nhardt_polyhedron
18. How to Guard an Art Gallery: A Simple Mathematical Problem - Fisher Digital Publications, accessed on June 5, 2025, https://fisherpub.sjf.edu/cgi/viewcontent.cgi?article=1347&context=ur
19. The Art Gallery Problem | wild.maths.org, accessed on June 5, 2025, https://wild.maths.org/art-gallery-problem
20. Convex Hull using Graham Scan | GeeksforGeeks, accessed on June 5, 2025, https://www.geeksforgeeks.org/convex-hull-using-graham-scan/
21. Graham scan - Wikipedia, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Graham_scan
22. 9 Edge flipping - Computer Science Purdue, accessed on June 5, 2025, https://www.cs.purdue.edu/homes/tamaldey/course/531/Edgeflip.pdf
23. Delaunay Triangulation (chapter 9) - Purdue Computer Science, accessed on June 5, 2025, https://www.cs.purdue.edu/homes/cs53100/slides/delaunay.pdf
24. www.cs.jhu.edu, accessed on June 5, 2025, https://www.cs.jhu.edu/~misha/Spring16/11.pdf
25. Delaunay Triangulation - UCSB Computer Science, accessed on June 5, 2025, https://sites.cs.ucsb.edu/~suri/cs235/Delaunay
26. www.cs.cmu.edu, accessed on June 5, 2025, https://www.cs.cmu.edu/~kmcrane/Projects/DDG/paper.pdf
27. The n-dimensional cotangent formula, accessed on June 5, 2025, https://www.cs.cmu.edu/~kmcrane/Projects/Other/nDCotanFormula.pdf
28. Laplacian matrix - Wikipedia, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Laplacian_matrix#Laplacian_matrix_for_graphs
29. Differential Coordinates for Interactive Mesh Editing, accessed on June 5, 2025, https://igl.ethz.ch/projects/Laplacian-mesh-processing/Laplacian-mesh-editing/diffcoords-editing.pdf
30. Laplacian Mesh Processing - People @EECS, accessed on June 5, 2025, https://people.eecs.berkeley.edu/~jrs/meshpapers/Sorkine.pdf
31. Laplace operator - Wikipedia, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Laplace_operator
32. Resolving Ambiguities in Marching Squares - BorisTheBrave.Com, accessed on June 5, 2025, https://www.boristhebrave.com/2022/01/03/resolving-ambiguities-in-marching-squares/
33. Marching Cubes Tutorial, accessed on June 5, 2025, https://users.polytech.unice.fr/~lingrand/MarchingCubes/applet.html
34. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow - Caltech Multi-Res Modeling Group, accessed on June 5, 2025, https://www.multires.caltech.edu/pubs/ImplicitFairing.pdf
35. (PDF) Curve and Surface Smoothing Without Shrinkage - ResearchGate,

accessed on June 5, 2025, https://www.researchgate.net/publication/3612986_Curve_and_Surface_Smoothing_Without_Shrinkage
36. Simplification | Cardinal3D - GitHub Pages, accessed on June 5, 2025, https://stanford-cs248.github.io/Cardinal3D/meshedit/global/simplify/
37. Michael Garland and Paul S. Heckbert - Stanford Computer Graphics Laboratory, accessed on June 5, 2025, http://graphics.stanford.edu/courses/cs468-08-winter/slides/jerry1.pdf
38. Loop Subdivision Surface Based Progressive Interpolation - Department of Computer Science, accessed on June 5, 2025, https://www.cs.uky.edu/~cheng/PUBL/Paper_prog_Loop.pdf
39. 13.5 Subdivision Surfaces, accessed on June 5, 2025, https://graphics.cs.wisc.edu/Courses/559-f2010/pubs/pub_RTR-Subdivision.pdf
40. Understanding Iterative Closest Point (ICP) Algorithm with Code - LearnOpenCV, accessed on June 5, 2025, https://learnopencv.com/iterative-closest-point-icp-explained/
41. ICP point-to-plane odometry algorithm - OpenCV Documentation, accessed on June 5, 2025, https://docs.opencv.org/4.x/d7/dbe/kinfu_icp.html
42. ransac for outlier detection, accessed on June 5, 2025, https://www.tandfonline.com/doi/pdf/10.1080/13921541.2005.9636670
43. An Improved RANSAC Outlier Rejection Method for UAV-Derived Point Cloud, accessed on June 5, 2025, https://www.researchgate.net/publication/364300007_An_Improved_RANSAC_Outlier_Rejection_Method_for_UAV-Derived_Point_Cloud
44. Higher Order Continuity for Smooth As-Rigid-As-Possible Shape Modeling - arXiv, accessed on June 5, 2025, https://arxiv.org/html/2501.10335v1
45. Smooth Rotation Enhanced As-Rigid-As-Possible Mesh Animation - Zohar Levi, accessed on June 5, 2025, https://zoharl3.github.io/publ/arap.pdf
46. Simple linear regression - Wikipedia, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Simple_linear_regression#Fitting_the_regression_line
47. www.geometrictools.com, accessed on June 5, 2025, https://www.geometrictools.com/Documentation/LeastSquaresFitting.pdf
48. Hough transform - Wikipedia, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Hough_transform
49. Straight line Hough transform — skimage 0.25.2 documentation, accessed on June 5, 2025, https://scikit-image.org/docs/0.25.x/auto_examples/edges/plot_line_hough_transform.html
50. Principal component analysis - Wikipedia, accessed on June 5, 2025, https://en.wikipedia.org/wiki/Principal_component_analysis
51. PCA in High Dimensions: An orientation - PMC - PubMed Central, accessed on June 5, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC6167023/
52. Research of 3D Printing Support Structure - Atlantis Press, accessed on June 5, 2025, https://www.atlantis-press.com/article/126003636.pdf
53. 3D Printing Overhangs: What They Are & How to Improve Them - anycubic-store,

accessed on June 5, 2025,
https://store.anycubic.com/blogs/3d-printing-guides/how-to-improve-3d-printin
g-overhangs