Geometric Modeling

Riccardo Caprile

April 2023

1 Shape Representation

The shape representation is reached by acquiring real-world objects (discrete sampling , points and meshes) , modeling by hand or procedural modeling (algorithms , grammars).

1.1 Points

Points = unordered set of 3-tuples.

Efficient point processing and modeling requires a spatial partitioning data structure to figure out neighborhoods.

Query : Used to determine whether or not a given point p is inside or outside of the solid bounded by a surface S. Another typical query is the computation of point's distance to a surface.

1.2 Parametric Curves and Surfaces

1.2.1 Parametric Representation

Range of function f : X $\rightarrow$Y , Y , X $\subseteq R^m$, Y $\subseteq R^n$

X is a line from 0 to 1 and t is the parameter varying on this line.

Y is the physical domain on which we describe the line.

Planar curve : m = 1 , n = 2 $\rightarrow$s(t) = x(t),y(t).

Planar curves are curves than lay on a surface , 2D curves.

Space curve : m = 1 , n = 3 $\rightarrow$s(t) = x(t),y(t),z(t))

Surface in 3D : m = 2 , n = 3

s(u,v) = (x(u,v),y(u,v),z(u,v)). As output we get a solution in R^3

1.2.2 Parametric Curves

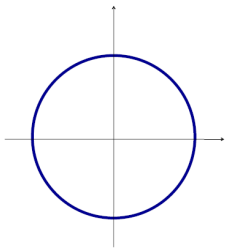
Explicit curve/circle in 2D.

p : R $\rightarrow R^2$

t $\rightarrow$p(t) = (x(t),y(t))

p(t) = r(cos(t),sin(t))

The equation of the circle $x^2 + y^2 = r^2$ is an implicit version.



Parametric Surfaces

Sphere in 3D , s : $R^2 \rightarrow R^3$

s(u,v) = r(cos(u) cos(v), sin(u) cos(v),sin(v))

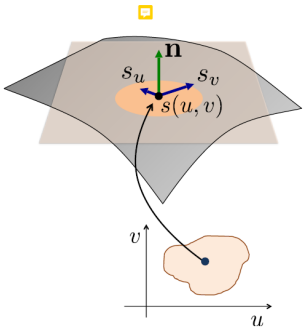
$(u, v) \in [0, 2\pi)x[-\pi/2, \pi/2]$

1.2.3 Tangents and Normal

$$s_u = \frac{\partial s(u,v)}{\partial u}$$

$$s_v = \frac{\partial s(u,v)}{\partial v}$$

$$\mathbf{n} = \frac{s_u \times s_v}{\|s_u \times s_v\|}$$

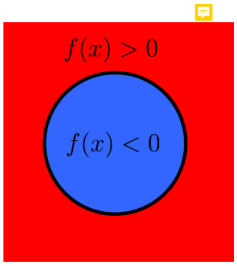


Tangent plane is normal to **n**

Parametric Curves and Surfaces are easy to generate points on the curve/surface.
But it is hard to tell if a point is on the curve/surface

1.3 Implicit Curves and Surfaces

The basic concept of implicit representations for geometric models is to characterize the whole embedding space of an object by classifying each 3D point to lie either inside , outside , or exactly on the surface S that bounds a solid object. An implicit surface does not have any holes as long as the defining function F is continuous.



Kernel of a scalar function f : $R^m \rightarrow$R.

- Curve in 2D : S = $(x \in R^2 | f(x) = 0)$
- surface in 3D : S = $(x \in R^3 | f(x) = 0)$

Space Partitioning :

- Outside : $(x \in R^m | f(x) > 0)$
- Curve/Surface : $(x \in R^m | f(x) > 0)$
- Inside : $(x \in R^m | f(x) > 0)$

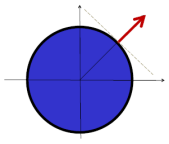
The implicit formula for circle and sphere is : $f(x, y) = x^2 + x^2 - r^2$ and $f(x, y, z) = x^2 + y^2 + z^2 - r^2$

The normal direction to the surface (curve) is given by the gradient of the implicit function.

$$\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)^T$$

- Example

$$f(x, y, z) = x^2 + y^2 + z^2 - r^2$$
$$\nabla f(x, y, z) = (2x, 2y, 2z)^T$$



∂f / ∂x , it represents the partial derivative of the function using x.

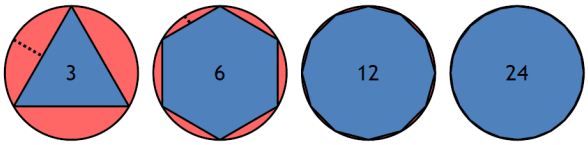
2 Geometric Meshes

2.1 Polygonal Meshes

Polygonal meshes are boundary representations of objects.

Faces can be triangles or quadrilateral or mixed.

Meshes can be used as an approximation of smooth surfaces.



We are trying to approximate the surface of the red circles using , at the beginning , a triangle with three edges. Then we double the number of edges and we can see that the error decreases.

Polygonal meshes are a good representation of approximation , arbitrary topology and piecewise smooth surfaces.

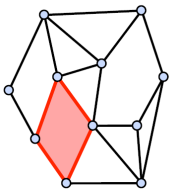
2.1.1 Polygon

Vertices : $v_0, v_1, ..., v_n - 1$

Edges : $\{ (v_0, v_1), ...(V_n - 1, v_0)\}$

All polygons are closed and all vertices are on a plane.

2.1.2 Polygon Mesh



A finite set M of closed , simple polygons Q_i

The intersection of two polygons in M is either empty , or a vertex , or an edge .

M = (V,E,F) , V stands for vertices , E for edges and F for faces.

Every edge belongs to at least one polygon and each Q_i defines a face of the polygonal mesh.

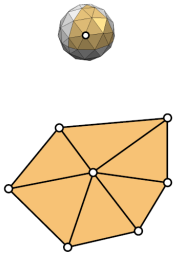
Vertex **degree** = number of incident edges.

Boundary : is the set of all edges that belong to only one polygon. If empty , then the polygonal mesh is said to be closed or watertight.

2.1.3 Triangle Meshes

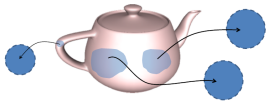
- Connectivity: vertices, edges, triangles
- Geometry: vertex positions

$$V = \{v_1, \ldots, v_n\}$$
$$E = \{e_1, \ldots, e_k\}, \quad e_i \in V \times V$$
$$F = \{f_1, \ldots, f_m\}, \quad f_i \in V \times V \times V$$
$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}, \quad \mathbf{p}_i \in \mathbb{R}^3$$



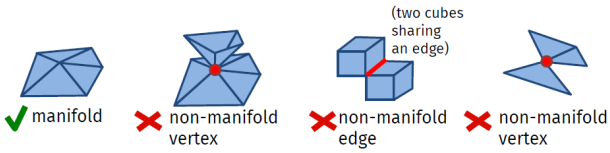
2.1.4 Manifolds

A surface is a closed 2-manifold if it is everywhere locally homeomorphic (deformazione senza strappi,da un punto di vista topologico sono due oggetti uguali) to a disk.



In a manifold mesh , there are at most 2 faces sharing an edge (boundary edges have one incident face , inner edges have two incident faces).

A manifold vertex has 1 connected ring of faces around it, or 1 connected half-ring (boundary)



2.1.5 Polyhedron

If closed and not intersecting , a manifold divides the space into **interior** and **exterior**.

The manifold is the boundary of the object that occupies the interior.

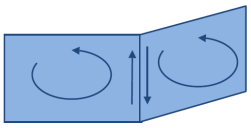
A closed manifold polygonal mesh is called **polyhedron**.

2.1.6 Orientation

Every face of a polygonal mesh is orientable.

Clockwise vs counterclockwise order of face vertices.

Defines sign/direction of the surface normal.



A polygonal mesh is orientable , if the incident faces to every edge can be consistently oriented, if the faces are consistently oriented for every edge , the mesh is oriented.

2.1.7 Global Topology of Meshes

Genus : 1/2 the maximal number of independent closed paths that do not disconnect the graph. Informally , the number of handles.

2.1.8 Euler-Poincarè Formula

Topology : study the properties of the figure , in general of the mathematical objects , that do not change when a deformation without "strappi" , "sovrapposizioni" happen.

Theorem : the sum $\chi(M) = v - e + f$ is constant for a given surface topology , no matter which mesh we choose to cover M

For orientable meshes : $v - e + f = 2(c - g) - b = \chi(M)$

c is the number of connected components , **g** the genus and **b** the number of boundary loops.

2.1.9 Implication for Mesh Storage

Let's count the edges and faces in a closed triangle mesh :

Ratio of edges to faces = $\mathbf{e = 3/2\ f}$, each edge belongs to exactly 2 triangles, each triangle has exactly 3 edges.

Ratio of vertices to faces: f $\sim 2v$ 2v , 2 = v - e + f = v - 3/2f + f , 2 + f/2 = v. The number of triangles is twice the number of vertices

Ratio of edges to vertices : e $\sim$ 3v. The number of edges is three times the number of vertices.

Average degree of a vertex : 6.

2.1.10 Triangulation

Polygonal mesh where every face is a triangle.

It simplifies data structures , rendering , algorithms., Any polygon can be triangulated.

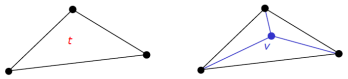
2.2 Editing Operations

- Refinement operators : Producing a triangulation with more vertices/edges/faces
- Simplification operators : Producing a triangulation with less vertices/edges/faces

2.2.1 Refinement Operators

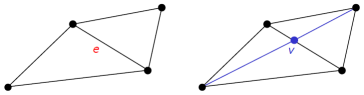
Triangle Split

Insert a new vertex v in a triangle t and split the triangle by connecting v to the vertices of t.



Edge Split

Insert a new vertex v on an edge e and split the triangles incident at e connecting v to the vertices opposite to e.



Vertex Split

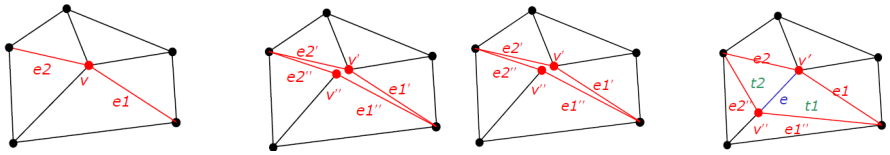
Take a vertex v and two of its incident edges e1 and e2.

Cut along e1 and e2.

Duplicate v,e1 and e2 each in two copies.

Displace either one or both copies of v.

Connect the two copies of v with a new edge e , generating two new faces t1 and t2.



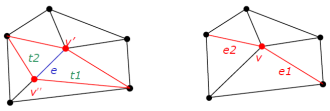
(a) Step 1 (b) Step 2

2.2.2 Simplification Operators

Edge collapse(inverse of vertex split)

Take an edge e and collapse it to a single point

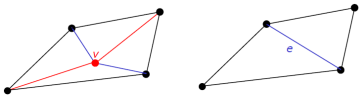
The two faces incident at e also collapse to edges.



Edge merge (inverse of edge split)

Consider an internal vertex v with 4 incident triangle.

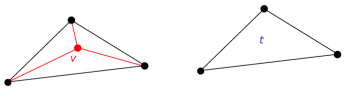
Delete the 4 triangles and create 2 new triangles by a new edge e that is one of the two diagonals of the quadrilateral.



Delete Vertex (inverse of triangle split)

Consider an internal vertex v with 3 incident triangles.

Merge the 3 triangles into a single triangle t.

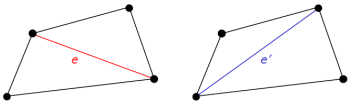


2.2.3 Neutral Editing Operator

Edge Swap

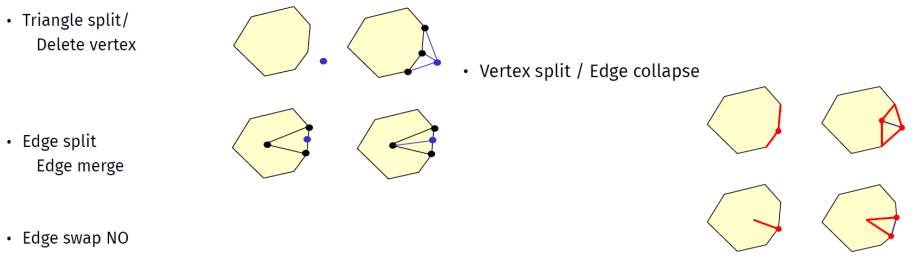
Take and edge e such that its two incident faces form a convex quadrilateral Q.

Replace e with the other diagonal e' of Q



Edge swap cannot be applied on non-convex quadrilaterals.

2.2.4 Boundary Cases



2.3 Data Structures

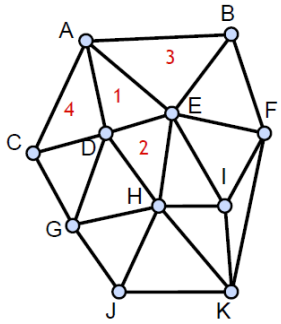
What should be stored?

Geometry : 3D coordinates

Connectivity : Adjacency relationships

Attributes : Normal , color , texture coordinates. Per vertex , face , edge.

What should be supported ?



Rendering (enumeration).

Geometry queries : What are the vertices of face 2? Is vertex A adjacent to vertex H?.

Modifications : Remove /add a vertex or a face etc.

2.3.1 Triangle List

The simplest way to represent a surface mesh consists of storing a set of individual polygonal faces represented by their vertex positions.

4 bytes per coordinate and since due to Euler's formula , the number of faces F is about twice the number of vertices V, this data structure consumes on average 72 bytes

No connectivity information and it is redundant.

Triangles			
0	x0	y0	z0
1	x1	x1	z1
2	x2	y2	z2
3	x3	y3	z3
4	x4	y4	z4
5	x5	y5	z5
6	x6	y6	z6
...	...	...	...

2.3.2 Indexed Face Set

Stores an array of vertices and encodes polygons as a set of indices into this array

Vertices			
v0	x0	y0	z0
v1	x1	x1	z1
v2	x2	y2	z2
v3	x3	y3	z3
v4	x4	y4	z4
v5	x5	y5	z5
v6	x6	y6	z6
...	...	...	...

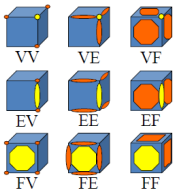
Triangles			
t0	v0	v1	v2
t1	v0	v1	v3
t2	v2	v4	v3
t3	v5	v2	v6
...	...	...	...

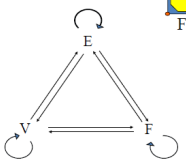
No explicit neighborhood info.

2.3.3 Neighborhood Relations

- All possible neighborhood relationships:
  1. Vertex – Vertex VV
  2. Vertex – Edge VE
  3. Vertex – Face VF
  4. Edge – Vertex EV
  5. Edge – Edge EE
  6. Edge – Face EF
  7. Face – Vertex FV
  8. Face – Edge FE
  9. Face – Face FF

We'd like $O(k)$ time for queries and local updates of these relationships (k = number of found entities)





2.3.4 Half-edge structure

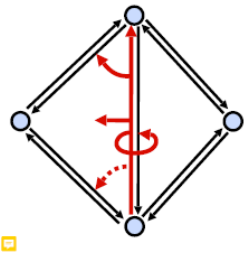
Data structures for general polygon meshes are logically edge-based , since the connectivity primarily relates to the mesh edges.

Half-edge data structures split each edge into two oriented **halfedges**.

Halfedges are oriented consistently in counterclockwise order around each face and along each boundary.

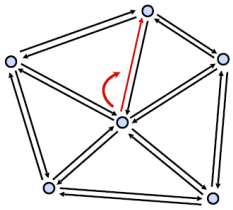
For each halfedge we store a reference to :

- The vertex it points to
- Its adjacent face
- The next halfedge of the face or boundary
- The previous halfedge in the face
- Its opposite halfedge



One ring traversal :

It start at a vertex (in this case the one in the middle). Then we move to the outgoing halfedge , then to the twin halfedge , the next halfedge , the twin etc



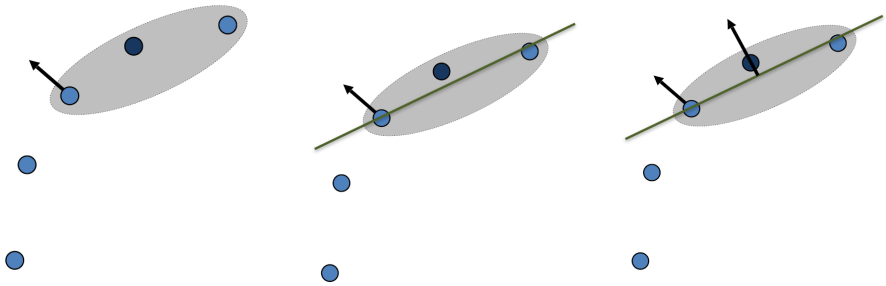
(a) Second step , outgoing halfedge

The problem with this implementation is that it is heavy because requires to store and manage extra pointers.

3 Normal Estimation

Assign a normal vector n at each point cloud x.

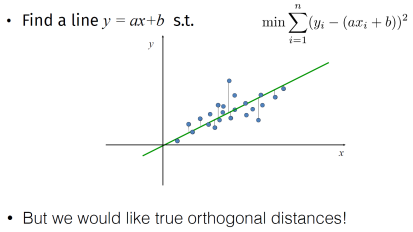
- Estimate the direction by fitting a local plane (in 2D is a line , like in the case below)
- Find consistent global orientation by propagation. We have to check the neighbour and if the dot product between the visited node and the unvisited one is negative , the univisited will be flipped.

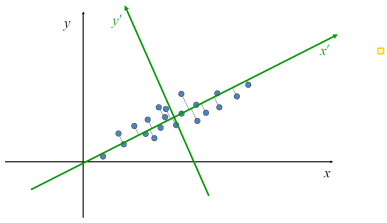


For each point x in the cloud , pick k nearest neighbors or all points in r-ball.

What we need is a plane that minimizes the sum of square distances : $min \sum dist(x_i, \Pi)^2$

And this can be done by using linear least squares.





(a) In this case we have the orthogonal distance between the line and points

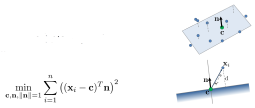
3.1 Principal Component Analysis (PCA)

PCA finds an orthogonal basis that the best represents a given dataset.

PCA finds the best approximating line/plane/orientation in terms of square distance.

3.1.1 Notations

Input points : $x_1, x_2, ..., x_n \in R^d$

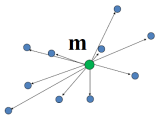


Looking for a plane passing through c with normal n
Centroid: $m = 1/n \sum_{i=1}^{n} x_i$
Vectors from the centroid : $y_i = x_i - m$, a point which is in average position between all the points considered

- It can be shown that:

$$\mathbf{m} = \underset{\mathbf{c}}{\operatorname{argmin}} \sum_{i=1}^{n} \left((\mathbf{x}_i - \mathbf{c})^T \mathbf{n} \right)^2$$

$$\mathbf{m} = \underset{\mathbf{c}}{\operatorname{argmin}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{c}\|^2$$



$$\mathbf{m} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

- **m** minimizes SSD
- **m** will be the origin of the (hyper)-plane
- Our problem becomes: $\min_{\|\mathbf{n}\|=1} \sum_{i=1}^{n} \left(\mathbf{y}_i^T \mathbf{n} \right)^2$

To find the best fitting plane we have to :

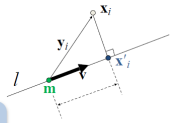
- Input points
- Compute the centroid , which is the plane origin
- Compute the scatter matrix S = YY^T $\rightarrow$y = $(y_1 y_2 ... y_n)$, $y_i = x_i - m$

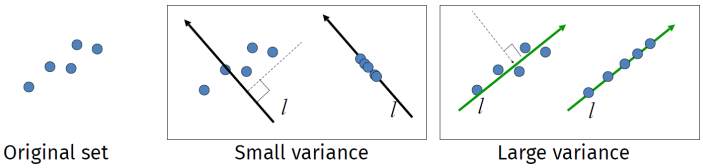
The plane normal n is the eigenvector of S with the smallest eigenvalue.
Si dice che uno scalare λ_0 é un autovalore**(eigenvalue)** della matrice quadrata A se esiste un vettore colonna non nullo v **(eigen vector)** tale che $Av = \lambda_0 v$

The Scatter matrix measures the variance of our data point along the direction v.

$$\text{var}(\mathbf{x}_1, \dots \mathbf{x}_n; \mathbf{v}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i' - \mathbf{m}\|^2 =$$

$$= \frac{1}{n}\sum_{i=1}^{n} \|(\mathbf{m} + \mathbf{v}^T\mathbf{y}_i) - \mathbf{m}\|^2 = \frac{1}{n}\sum_{i=1}^{n} (\mathbf{y}_i^T\mathbf{v})^2 = \frac{1}{n}\mathbf{v}^T\mathbf{S}\mathbf{v}$$

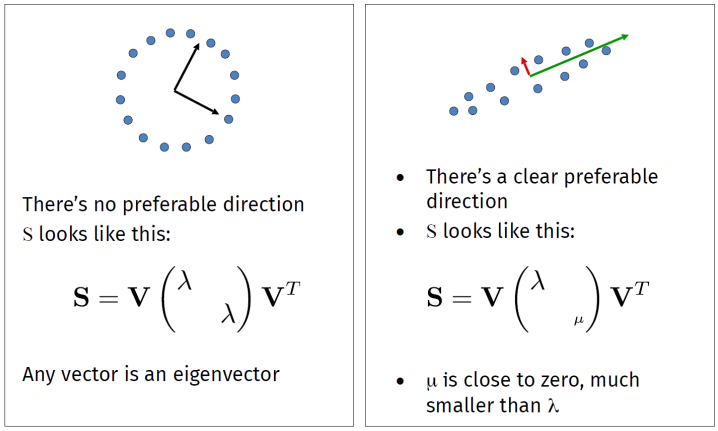




Original set | Small variance | Large variance

Eigenvectors of S that correspond to big eigenvalues are the directions in which the data has strong components (= large variance)

If the eigenvalues are more or less the same , there is no preferable direction.



4 Surface Reconstruction

How to get a mesh out of a cloud of points.

The first step is the Scanning , which results in range images.

Then Registration that bring all range images to one coordinate system

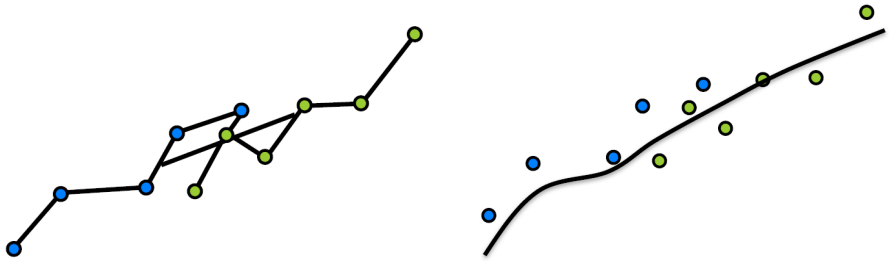
In the end , Reconstruction that is the integration of scans into a single mesh

Explicit Reconstruction

1. Each range scan is meshed independently
2. Stitch the range scans together

Connect sample points by triangles.

Bad for noisy or misaligned data and can lead to holes or non manifold situations.



(a) Explicit Reconstruction (b) Implicit Reconstruction

Implicit Reconstruction

1. Estimate a signed distance function (SDF)
2. Extract 0-level set mesh

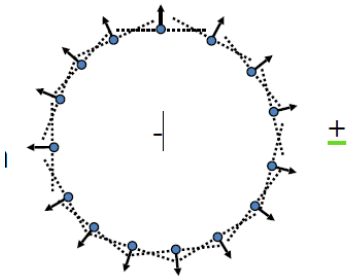
Approximation of input points. Watertight manifold by construction.

Implicit function approach, with a value > 0 outside the shape and < 0 inside. The zero set is the boundary where f(x) = 0

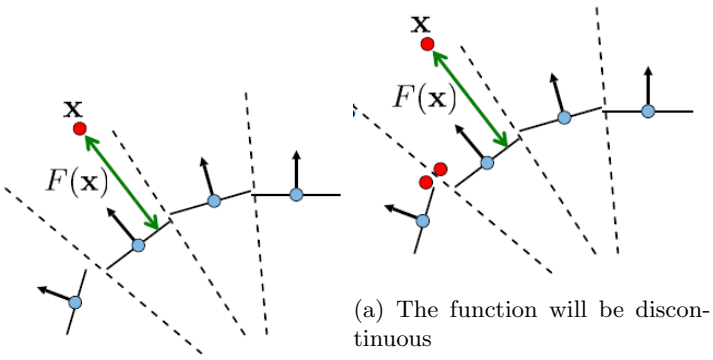
4.1 SDF from Points and Normals

Compute signed distance to the tangent plane of the closest point.

Normals help to distinguish between inside and outside.



Compute signed distance to the tangent plane of the closest point.

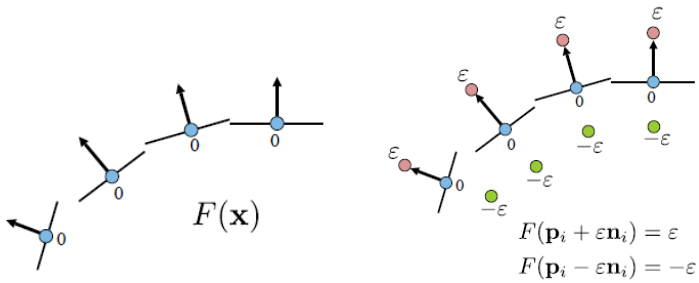


(a) The function will be discontinuous

4.1.1 Smooth SDF

Instead find a smooth formulation for F.

Scattered data interpolation : F(p_i) = 0 , F is smooth.



(a) (b) Add off-surface constraints

4.1.2 Radial Basis Function Interpolation

RBF : Weighted sum of shifted , smooth kernels

$F(x) = \sum_{i=0}^{N-1} \omega_i \phi(||x - c_i||)$

Where ϕ is the basis function corresponding to the center c_i

To find and RBF function that interpolates the displacement constraints , we use as many RBF kernels as we have constraints and place them on the constraints.

$\{c_{3i}, c_{3i+1}, c_{3i+2}\} = \{p_i, p_i + \epsilon n_i, p_i - \epsilon n_i\}$

The weights w_i are then found as the solution of the symmetric linear system.

$$\begin{pmatrix} \varphi(\|\mathbf{c}_0 - \mathbf{c}_0\|) & \dots & \varphi(\|\mathbf{c}_0 - \mathbf{c}_{N-1}\| \\ \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_0\|) & \dots & \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_{N-1}\|) \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_{N-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ \vdots \\ d_{N-1} \end{pmatrix}$$

Once the weight has been computed , the RBF function d has been fit to the constraints.

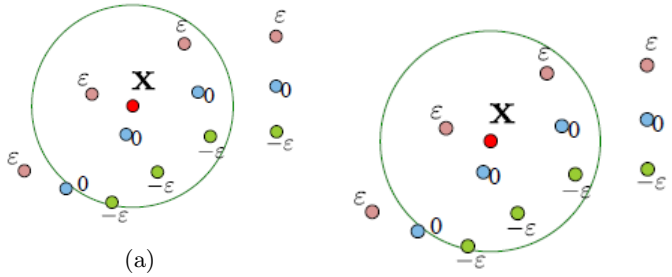
RBF is a global definition. A global optimization of the weights , even if the basis function are local.

4.2 Moving Least Squares (MLS)

Do purely **local** approximation of the SDF.

Weights change depending on where we are evaluating.

The beauty: the stitching of all local approximations, seen as one function F(x) , is smooth everywhere! We get a globally smooth function but only do local computation.



(a)

4.2.1 MOVING Least-Squares Approximation

Polynomial least-squares approximation

- Choose degree, k

$$f \in \Pi_k^3 : f(x,y,z) = a_0 + a_1 x + a_2 y + a_3 z + a_4 x^2 + a_5 xy + \ldots + a_* z^k$$

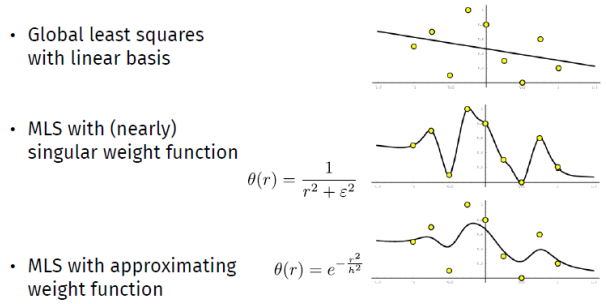
$$f(\mathbf{x}) = \mathbf{b}(\mathbf{x})^T \mathbf{a}$$

$$\mathbf{a} = (a_1, a_2, \ldots, a_*)^T,\ \mathbf{b}(\mathbf{x})^T = (1, x, y, z, x^2, xy, \ldots, z^k)$$

- Find $\mathbf{a}_\mathbf{x}$ that minimizes WEIGHTED sum of squared differences
- The value of the SDF is the obtained approximation evaluated at $\mathbf{x}$:

$$F(\mathbf{x}) = f_\mathbf{x}(\mathbf{x}) = \mathbf{b}(\mathbf{x})^T \mathbf{a}_\mathbf{x}$$

4.2.2 Dependence on Weight Function

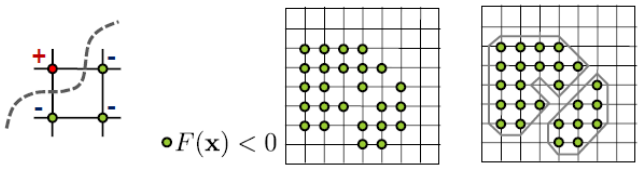


4.3 Tessellation

We want to approximate an implicit surface with a mesh.

Can't explicitly compute all the roots.

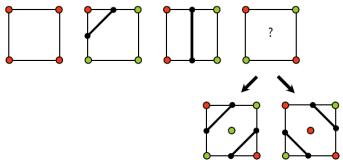
Solution: find an approximate roots by trapping the implicit surface in a grid.



4.3.1 Marching Squares

The marching squares algorithm aims at drawing lines between interpolated values along the edges of a square.

In order to display figure we want , we can consider individually each square of the grid using 16 different configuration which allows the representation of all kinds of lines in 2D space.



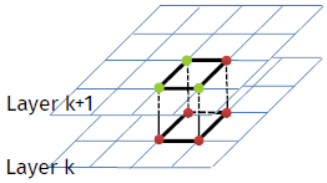
(a) 4 equivalence classes

Case 4 is ambiguous , always pick consistently to avoid problems with the resulting mesh.

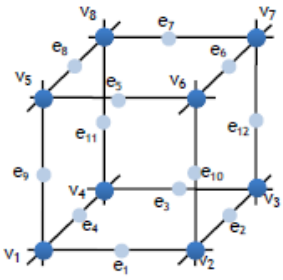
4.4 Marching Cubes

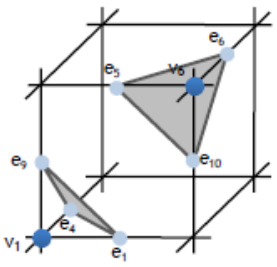
Following the marching squares algorithm we can adapt our approach to the 3D case. In a 3D space we enumerate 256 different situations for the marching cubes representation

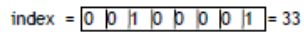
1. Load 4 layers of the grid into memory
2. Create a cube whose vertices lie on the two middle layers
3. Classify the vertices of the cube according to the implicit function (outside,inside or on the surface)



4. Compute case index. We have $2^8 = 256$ cases (0/1 for each of the eight vertices). If the vertices is inside the bit relative to the vertex is set to 1 , 0 otherwise. When all the vertices are checked we got the configuration wanted expressed as a 8 bit number







5. Using the case index , retrieve the connectivity in the look up table. In this case cut edges $e_1, e_4, e_5, e_6, e_9 and e_1 0$
6. Then we have to compute the position of the cut vertices by linear interpolation
7. Move to the next cube

As before we have to make consistent choices for neighboring cubes , otherwise we get holes.

We have problems with short triangle edges.

Triangles with short edges waste resources but do not contribute to the surface mesh representation.

4.4.1 Grid Snapping

Solution : threshold the distances between the created vertices and the cube corners.

When the distance is smaller than d_snap we snap the vertex to the cube corner.

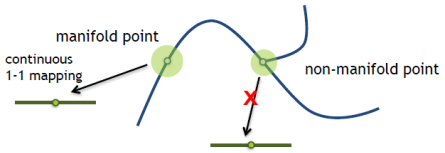
If more than one vertex of a triangle is snapped to the same point , we discard that triangle altogether.

5 Curves

5.1 Differential Geometry Basics

Geometry of manifolds.

Things can be discovered by local observations : point + neighborhood.



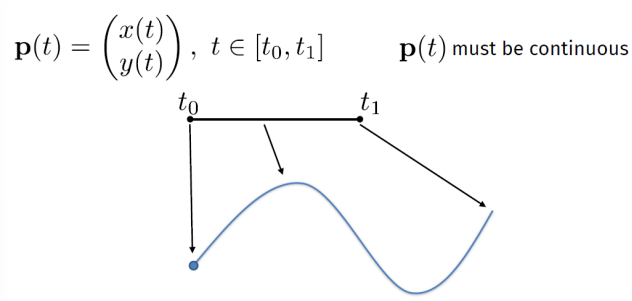
If it is a sufficiently smooth mapping can be constructed , we can look at its first and second derivatives to find : tangent , normals , curvatures etc.

5.2 Curves

We need mathematical concepts to characterize the desired curve properties.

Notions from curve geometry help with designing user interfaces for curve creation and editing.

5.2.1 2D Parametric Curve



(a) A curve is defined as **the image function of a function x**

The coordinates x and y are assumed to be differentiable functions (continuous, gradient exists) of t.

The same curve can be obtained with a different parametrization. For example $P_1(t) = (u, u)^T and P_2 = (u^2, u^2)^T$ describe the same curve for $t \in [0, 1]$, but their parametrization is different. This illustrates that we can change the parametrization without changing the shape of the curve.

The tangent vector to the curve is at point p(t) is defined as the first derivative of the coordinate function.

The **tangent vector** corresponds to the velocity vector a time t, norm of the derivatives.

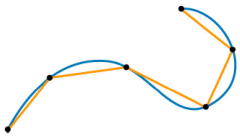
5.2.2 Arc Length

How long is the curve between t_0 and t?

This can be computed as the integral of the tangent vector.

- Speed is $\|\mathbf{p}'(t)\|$, so:

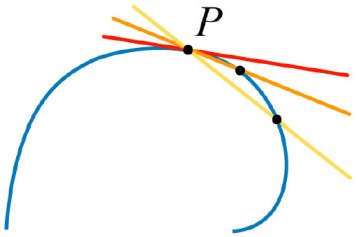
$$s(t) = \int_{t_0}^{t} \|\mathbf{p}'(t)\| dt$$



5.3 Tangent : Geometric Construction

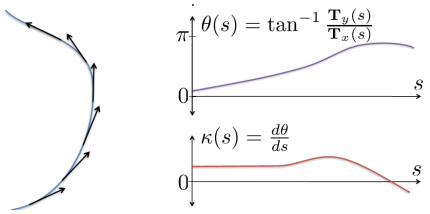
Secant : a line through two points on the curve

Tangent: the limiting secant as two points come together



5.4 Curvature

How much does the curve turn per unit S?

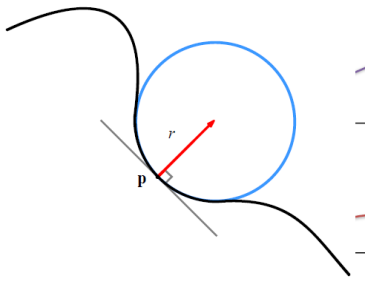


k(s) , in the book , is $||x''(s)||$, this formula define the curvature at a point x(s).

Curvature measures how strongly a curve deviates from a straight line.

5.4.1 Curvature: Geometric Construction

We need to find the "best fitting circle".



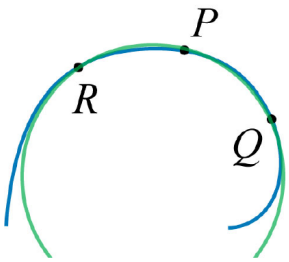


Figure 48: Consider the circle passing through three points on the curve

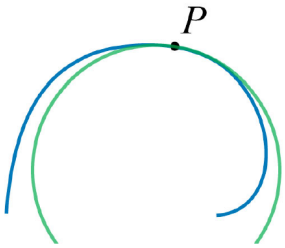


Figure 49: The limiting circles as Q and R move to P : **Osculating Circle**

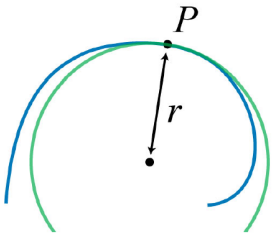


Figure 50: Take the radius of the osculating circle

5.4.2 Curvature of a circle

- Curvature of a circle:

$$\mathbf{p}(s) = \begin{pmatrix} r\cos(s/r) \\ r\sin(s/r) \end{pmatrix}$$

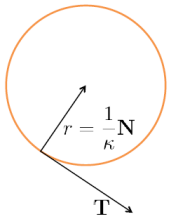
$$\mathbf{p}'(s) = \begin{pmatrix} -\sin(s/r) \\ \cos(s/r) \end{pmatrix}$$

$$\theta(s) = \tan^{-1}\frac{\cos(s/r)}{-\sin(s/r)}$$

$$= s/r - \pi/2$$

$$\kappa(s) = 1/r$$

$$\mathbf{N}(t) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\mathbf{T}(t) \text{ = Unit Normal}$$



5.4.3 Curvature Normal

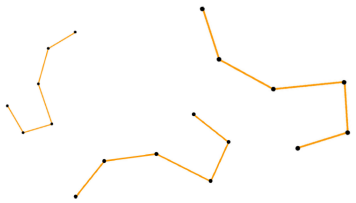
The curve normal vector can be defined using the relation x" = k(s)n(s).

6 Discrete Differential Geometry of Curves

6.1 Discrete Planar Curves

Piecewise linear curves , not smooth at vertices and can't take derivatives.

Generalize notions from the smooth world for the discrete case.



6.1.1 Tangents , Normals

For any point on the edge , the tangent is simply the unit vector along the edge and the normal is the perpendicular vector.

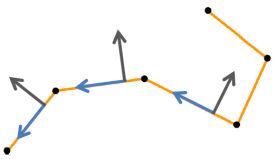


Figure 53: In gray Normal , in blue the Tangent

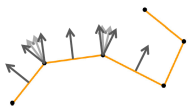


Figure 54: For vertices , we have many options

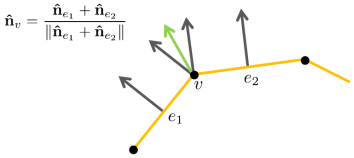
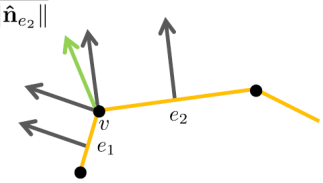


Figure 55: Can choose to average the adjacent edge normals for obtaining the normal of the vertex v

$$\hat{\mathbf{n}}_v = \frac{|e_1|\hat{\mathbf{n}}_{e_1} + |e_2|\hat{\mathbf{n}}_{e_2}}{\||e_1|\hat{\mathbf{n}}_{e_1} + |e_2|\hat{\mathbf{n}}_{e_2}\|}$$

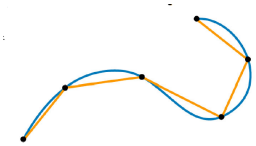


(a) Or we can weight by edge lengths

6.1.2 Inscribed Polygon , p

Connection between discrete and smooth.

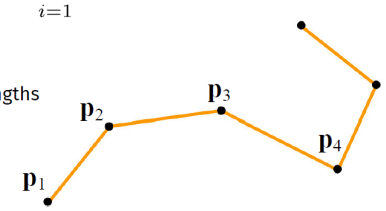
Finite number of vertices each lying on the curve, connected by straight edges.



6.1.3 The length of a Discrete Curve

$$\text{len}(p) = \sum_{i=1}^{n-1} \|\mathbf{p}_{i+1} - \mathbf{p}_i\|$$

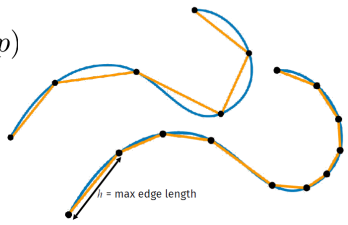
- Sum of edge lengths



6.1.4 The Length of a Continuous Curve

- Take limit over a refinement sequence

$$\lim_{h \to 0} \text{len}(p)$$



6.1.5 Curvature of a Discrete Curve

Curvature is the change in normal direction as we travel along the curve.

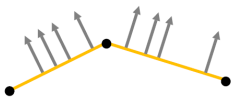


Figure 60: No change along each edge, curvature is zero along edges

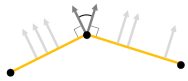


Figure 61: Normal changes at vertices , record the turning angle!

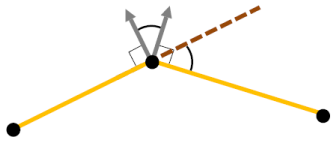
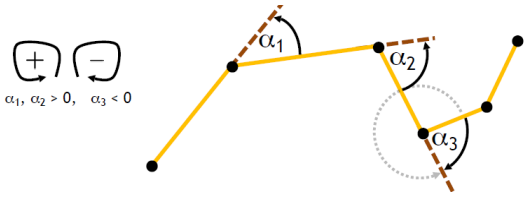


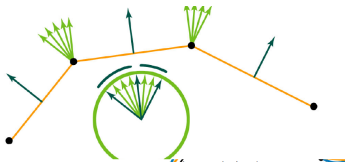
Figure 62: Same as the turning angle between the edges

Zero along the edges.
Turning angle at the vertices = the change in normal direction



$tsc(p) = \sum_{i=1}^{n} a_i$, sum of turning angles

6.1.6 Discrete Gauss Map



(a) Edges map to points , vertices map to arcs

(b) Turning number well-defined for discrete curves

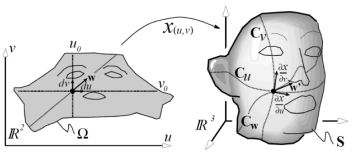
7 Surfaces

7.1 Parametric Representation of surfaces

- Surface : S $\subset R^3$
- Parameter Domain : $\Omega \subset R^2$
- Mapping: $f : \Omega \rightarrow$S

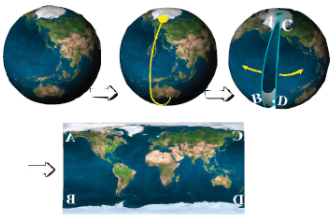
f is a bi-variate vector function mapping points of the 2D plane to points in 3D space

We can define an injective parametrization iff surface S is homeomorphic to a (punctured) disk.



For a surface S that is no homeomorphic to a punctured disk , parametrization needs to cut S at the image of borders of Ω

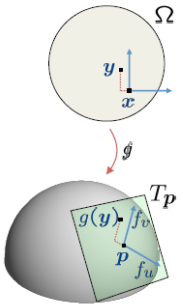
We need to find a way to unfold the surface of the world, in order to obtain a flat 2D surface.



7.2 Tangent Plane

f(u,v) $= (x(u,v), y(u,v), z(u,v))^T$

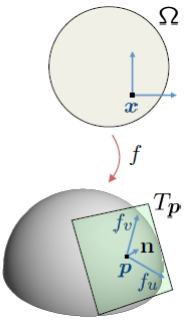
- Jacobian of f : $J_f = [f_u . f_v]$, derivate parziali di u e v
- Tangent plane at p : $T_p = \{p + af_u + bf_v : a, b \in R\}$
- Taylor expansion of f : $f(y) = f(x) + J_f(y - x) + ...$



7.3 Surface Normal

Surface normal at a point p is the unit-length normal vector n of tangent plane T_p pointing outwards from S.

$$\mathbf{n} = \frac{f_u \times f_v}{||f_u \times f_v||}$$

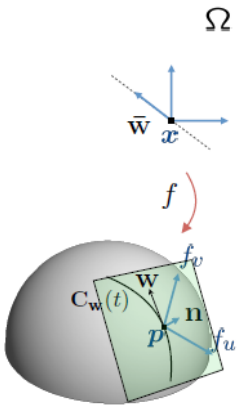


7.4 Directional derivatives

Given a direction vector $\bar{w} = (u_w, v_w)^T$

Straight line in parameter space (u,v) $= (u_x, v_x) + t\bar{w}$

Corresponds to a curve on S : $C_w(t) = f(u_x + tu_w, v_x + tv_w)$



Directional derivative w of f at (u_x, v_x) wrt $\bar{w}$ is the tangent to $C_w(t)$ computed at t=0

By the chian rule w $= J_f \bar{w}$

7.5 First Fundamental Form

Basically the Jacobian matrix correspond to the linear map that transforms a vector $\bar{w}$ in parameter space into a tangent vector w on the surface

$$\mathbf{I} = J_f^T J_f = \begin{bmatrix} E & F \\ F & G \end{bmatrix} = \begin{bmatrix} f_u^T f_u & f_u^T f_v \\ f_u^T f_v & f_v^T f_v \end{bmatrix}$$

The first fundamental form I defines an inner product on the tangent space of S.

Beside measuring angles , we can use this inner product to determine the squared length of a tangent vector w as $||w||^2 = \bar{w}^T I \bar{w}$

7.6 Measuring Areas

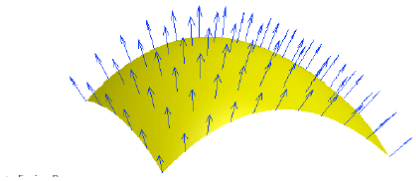
The first fundamental form can be used to measure areas on the surface : Area of a region f(U) where U is a region in Ω:

$$A = \iint_U \sqrt{\det(\mathbf{I})} du dv = \iint_U \sqrt{EG - F^2} du dv$$

7.7 Gauss Map

The Gauss map sends a point on the surface to the outward pointing unit normal vector :

$$N(p) = N(f(x)) = \frac{f_u(x) \times f_v(x)}{||fu(x) \times fv(x)||} = \mathbf{n}$$

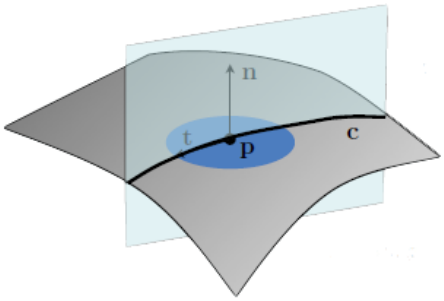


7.8 Normal Curvature

To extend the notion of curvature from curves to surfaces, we look at the curvature of curves embedded in the surface.

Let $t = u_t x_u + v_t x_v$ be a tangent vector at a surface point p represented as $\bar{t} = (u_t, v_t)$ in parameter space.

The normal curvature $k_n(\bar{t})$ at p is the curvature of the planar curve created by intersecting the surface at p with the plane spanned by t and the surface normal n



(a) t is the tangent vector

We can choose a specific curve c that has the normal aligned to n.
The curvature of c curve at p is the normal curvature wrt t.
We can express normal curvature in direction $\bar{t}$ as :

$$\kappa_n(\bar{\mathbf{t}}) = \frac{\bar{\mathbf{t}}^T \mathbf{II} \bar{\mathbf{t}}}{\bar{\mathbf{t}}^T \mathbf{I} \bar{\mathbf{t}}} = \frac{L u_t^2 + 2M u_t v_t + N v_t^2}{E u_t^2 + 2F u_t v_t + G v_t^2}$$

(a) t is the tangent vector

Where II denotes the **second fundamental form** defined as :

$$\mathbf{II} = \begin{bmatrix} L & M \\ M & N \end{bmatrix} = \begin{bmatrix} f_{uu}^T \mathbf{n} & f_{uv}^T \mathbf{n} \\ f_{uv}^T \mathbf{n} & f_{vv}^T \mathbf{n} \end{bmatrix}$$

7.9 Principal Curvatures

Consider all possible directions t on the tangent plane at p.

There are exactly two directions :

- t_1 such that $k_1 = k(t_1)$ is maximum
- t_1 such that $k_2 = k(t_2)$ is maximum

k1 and k2 are called the principal curvatures and t1,t2 are called the principal directions of curvature.

7.10 Euler Theorem

7.11 Intrinsic Geometry

In differential geometry , properties that only depend on the first fundamental form are called intrinsic.

7.12 Darboux Frame

Principal directions have arbitrary orientations.

Orientations can be selected such that (t_1, t_2, n) form a right-handed 3D frame with origin at p and n points outward.

The surface in a sufficiently small neighborhood of p can be expressed in explicit form wrt to the Darboux frame.

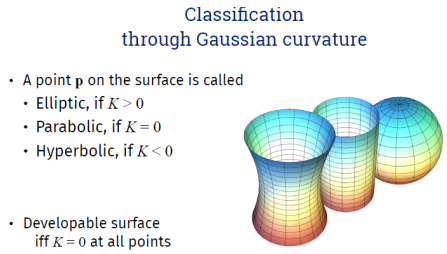
7.13 Shape Operator

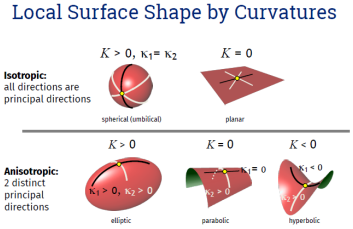
It is a linear operator that , when applied to a tangent vector t tells how the surface normal varies when moving along t on S.

In local coordinates given by frame (f_u, f_v) it is represented as a 2x2 matrix defined at each point p through the coefficients of the first and second fundamental form.

The principal directions of curvature are the eigenvector of Sh , the principal curvatures are the eigenvalues of Sh.

The shape operator is described by a diagonal matrix of principal curvatures in the frame (t_1, t_2)





8 Discrete Differential Geometry of Surfaces Part 1

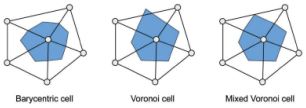
8.1 Differential Geometry on Meshes

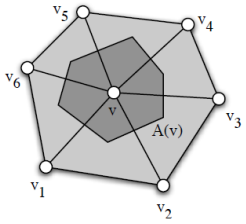
Assumption : Meshes are piecewise linear approximations of smooth surfaces.
Meshes have straight line edges and flat faces.
Differential properties of the approximated surface are estimated on the mesh

8.2 Discrete Integration

Vertex neighborhood that partition the mesh into disjoint regions are often preferred :

- Barycentric Cells , connects the triangle barycenters with the edge midpoints
- Voronoi Cells, replace the triangle barycenters with triangle circumcenter (il centro di una circonferenza circoscritta ad un triangolo)
- Mixed Voronoi Cells





8.3 Surface Normal via Discrete Integration

Many operations require normal vectors.

Normal vectors for individual triangles t $= (v_i, v_j, v_j)$ is the normal of the plane containing t :

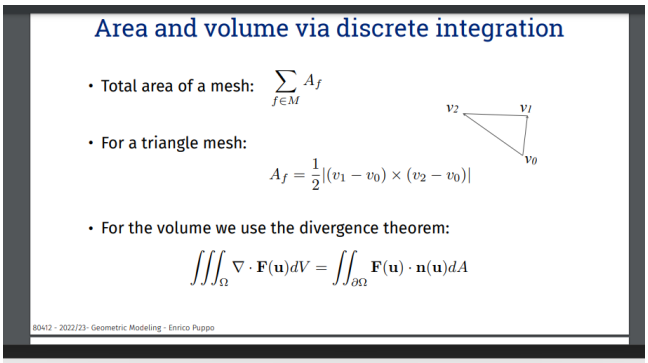
$$\mathbf{n}(t) = \frac{(v_j - v_i) \times (v_k - v_i)}{||(v_j - v_i) \times (v_k - v_i)||}$$

- Surface normal at a vertex v is computed as a weighted average of normals of its incident faces:

$$\mathbf{n}(v) = \frac{\sum_{t \in N_1(v)} \alpha_t \mathbf{n}(t)}{||\sum_{t \in N_1(v)} \alpha_t \mathbf{n}(t)||}$$

where weight α_t is the area of the portion of averaging region related to t

8.4 Area and Volume via Discrete Integration



Area and volume via discrete integration

- Set $\mathbf{F}(\mathbf{x}) = \frac{1}{3}\mathbf{x}$ then we have $\mathrm{div}\mathbf{F}(\mathbf{x}) = 1$ everywhere
- Therefore the volume inside a mesh M is

$$V_M = \iiint_{\Omega_M} 1 dV = \iiint_{\Omega_M} \nabla \cdot (\frac{1}{3}\mathbf{u}) dV = \iint_M \frac{1}{3}\mathbf{u} \cdot \mathbf{n}(\mathbf{u}) dA$$

- For a polygonal mesh, the last integral is computed face-by-face:

$$V_M = \frac{1}{3} \sum_{f \in M} \mathbf{x}_f \cdot \mathbf{n}_f A_f$$

where $\mathbf{x}_f$ is any point inside face f (all points have the same projection on $\mathbf{n}_f$)

80412 - 2022/23- Geometric Modeling - Enrico Puppo

8.5 Discrete Curvature

8.5.1 Shape Operator via Surface Fitting

The radius r of the neighborhood of each point p is used as a scale parameter.

1. Gather all vertices in a local neighborhood of radius r
2. set a local frame at p: (u, v, n_p) where n_p is the surface normal at p and u,v are any two orthogonal unit vectors spanning the tangent plane.
3. Discard all vertices v_i such that $n_i \cdot w < 0$

4. Express all vertices of the neighborhood the local frame

5. Fit to these points a polynomial of degree two through p, $f(u,v) = au^2 + bv^2 + cuv + du + ev$

6. Shape operator at p is computed analytically via first and second fundamental forms of f at the origin.

Since f is a polynomial , coefficients E,F,G,L,M,N of the fundamental forms can be computed easily in closed form.

SVD decomposition of the shape operator provides principal curvatures directions.

8.5.2 Shape Operator via normal Curvature

1. Project all edges (v_i, v_j) incident at v_i on the tangent plane to obtain tangent directions w_{ij}.

2. For each direction w_{ij} evaluate normal curvature k_{ij} by local curve fitting

3. Average results from normal curvatures to compute the shape operator

- Step 2: normal curvature at v_i along edge (v_i, v_j) can be estimated by fitting an osculating circle through v_i and v_j tangent to the plane defined by normal $\mathbf{n}_i$ and vector $\mathbf{w}_{ij}$:

$$\kappa_{ij} = 2\frac{(v_i - v_j)\cdot \mathbf{n}_i}{||v_i - v_j||^2}$$

- Step 3: once κ_{ij} has been computed for all v_j's neighbors of v_i, the second fundamental form can be estimated by least squares fitting of equations

$$\kappa_{ij} = \mathbf{t}_{ij}^T \begin{bmatrix} L & M \\ M & N \end{bmatrix} \mathbf{t}_{ij}$$

where $\mathbf{t}_{ij}$ denotes the normalized $\mathbf{w}_{ij}$ (unit-length projection to the tangent plane at v_i of vector v_j-v_i) expressed in a local coordinate frame $(\mathbf{u},\mathbf{v},\mathbf{n}_i)$ with origin at v_i

- Note: the first fundamental form w.r.t. $(\mathbf{u},\mathbf{v},\mathbf{n}_i)$ is the identity

9 Discrete Differential Geometry of Surfaces - Functions on Surfaces

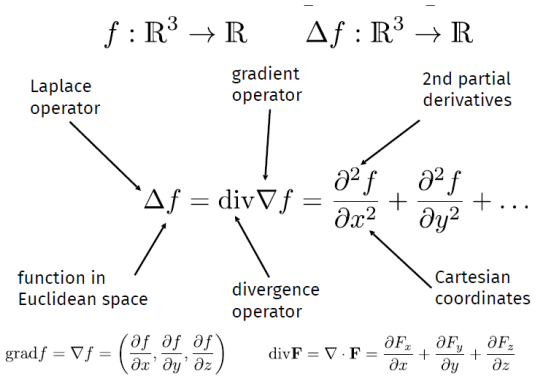
9.1 Gradient

The gradient of f is defined as: $\bigtriangledown \mathrm{f} = (f_u, f_v)^T$

The gradient $\bigtriangledown$f(p) of a function at a point p is a vector on the plane pointing to the direction of maximal ascent of f and having a size proportional to the rate of ascent.

f: M $\rightarrow$R defined on a manifold , the gradient $\bigtriangledown f_M$ at a point p is a vector in the tangent plane at p pointing to the direction of maximal ascent of f and having a size proportional to the rate of ascent.

9.2 Laplace Operator



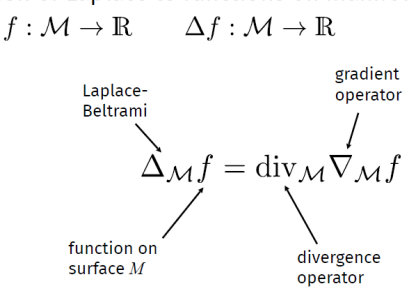
Intuitive Explanation

The Laplacian $\Delta f(p)$ of a function f at a point p is the rate at which the average value of f over spheres centered at p deviates from $f(p)$ as the radius of the sphere grows, up to a constant depending on the dimension.

9.3 Laplace-Beltrami Operator

Extends the concept explained before to functions defined on surface

- Extension of Laplace to functions on manifolds



9.3.1 Harmonic Functions and Surfaces

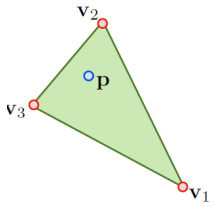
A function is said to be **harmonic** if its Laplacian is null.

A **minimal surface** has null mean curvature.

Harmonic functions and minimal surfaces are especially smooth.

9.4 Discrete Setting

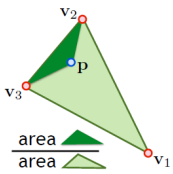
9.4.1 Barycentric Coordinates



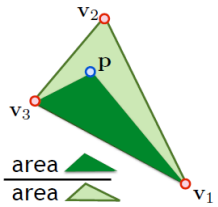
$$\mathbf{p} = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + w_3\mathbf{v}_3$$

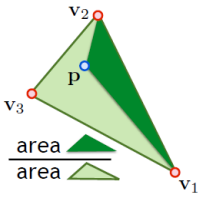
Partition of unity: $w_1 + w_2 + w_3 = 1$

$$\mathbf{p} = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + (1 - w_1 - w_2)\mathbf{v}_3$$



$$\mathbf{p} = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + (1 - w_1 - w_2)\mathbf{v}_3 \qquad \mathbf{p} = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + (1 - w_1 - w_2)\mathbf{v}_3$$



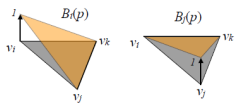


9.4.2 Piecewise Linear Functions on Meshes

We assume a piecewise linear function f that is given at each mesh vertex and interpolated linearly within each triangle.

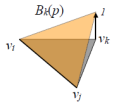
Hat functions and PL interpolation

$$f(\mathbf{p}) = B_i(\mathbf{p})f_i + B_j(\mathbf{p})f_j + B_k(\mathbf{p})f_k$$
$$B_i(\mathbf{p}) + B_j(\mathbf{p}) + B_k(\mathbf{p}) = 1$$



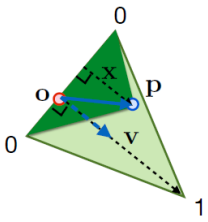
Gradients

$$\nabla f(\mathbf{p}) = \nabla B_i(\mathbf{p})f_i + \nabla B_j(\mathbf{p})f_j + \nabla B_k(\mathbf{p})f_k$$
$$\nabla B_i(\mathbf{p}) + \nabla B_j(\mathbf{p}) + \nabla B_k(\mathbf{p}) = 0$$



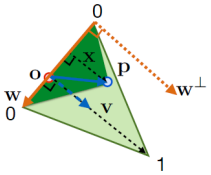
$$\nabla f(\mathbf{p}) = (f_j - f_i)\nabla B_j(\mathbf{p}) + (f_k - f_i)\nabla B_k(\mathbf{p})$$

9.4.3 The Hat Function



$$B(\mathbf{p}) = \frac{\text{area}}{\text{area}} = \frac{\|\mathbf{x}\|}{\|\mathbf{v}\|} = \frac{(\mathbf{p}-\mathbf{o})\cdot\frac{\mathbf{v}}{\|\mathbf{v}\|}}{\|\mathbf{v}\|} = \frac{(\mathbf{p}-\mathbf{o})\cdot\mathbf{v}}{\|\mathbf{v}\|\|\mathbf{v}\|}$$

9.4.4 Gradient of the Hat Function



$$B(\mathbf{p}) = \frac{(\mathbf{p}-\mathbf{o})\cdot\mathbf{v}}{\|\mathbf{v}\|\|\mathbf{v}\|}$$

$$\nabla B(\mathbf{p}) = \frac{\mathbf{v}}{\|\mathbf{v}\|\|\mathbf{v}\|}$$

$$\nabla B(\mathbf{p}) = \frac{\mathbf{w}^{\perp}}{\|\mathbf{w}^{\perp}\|\|\mathbf{v}\|}$$

$$\nabla B(\mathbf{p}) = \frac{\mathbf{w}^{\perp}}{\|\mathbf{w}\|\|\mathbf{v}\|}$$

$$\nabla B(\mathbf{p}) = \frac{\mathbf{w}^{\perp}}{2A}$$

$$\frac{\mathbf{v}}{\|\mathbf{v}\|} = \frac{\mathbf{w}^{\perp}}{\|\mathbf{w}^{\perp}\|}$$

$$\|\mathbf{w}^{\perp}\| = \|\mathbf{w}\|$$

$$A = \frac{\|\mathbf{v}\|\|\mathbf{w}\|}{2}$$

$\perp$ denotes a counterclockwise rotation by 90 degrees in the triangle plane and A is the Area of the triangle.

9.5 Discrete Laplace-Beltrami uniform

Directly measures the difference between function at a vertex wrt the average of its neighbors.

$$\Delta f(v_i) = \frac{1}{\mathcal{N}_1(v_i)} \sum_{v_j \in \mathcal{N}_1(v_i)} (f_j - f_i) = \left(\frac{1}{\mathcal{N}_1(v_i)} \sum_{v_j \in \mathcal{N}_1(v_i)} f_j \right) - f_i$$

Simple and efficient.

One assumption is that all triangles are equilateral.

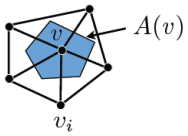
The resulting vector can be a non zero even for a planar configuration of vertices. However, in such a setting we would expect a zero Laplacian since the mean curvature over the entire mesh is zero.

This indicates that the uniform Laplacian is not an appropriate discretization for non-uniform meshes.

9.6 Discrete Laplace-Beltrami cotangent

We integrate the value of the Laplacian over the averaging region A(v) about a vertex v.

$$\Delta f(v) = \frac{1}{A(v)} \iint_{A(v)} \Delta f(\mathbf{u}) dA$$



To simplify the integrate we make use of the divergence theorem for a vector-valued function F

$$\iint_A \nabla \cdot \mathbf{F}(\mathbf{u}) dA = \oint_{\partial A} \mathbf{F}(\mathbf{u}) \cdot \mathbf{n}(\mathbf{u}) ds$$

where n(u) is the outward pointing unit normal at each point u of the boundary of region A

9.7 Laplacian Matrix

w_i : vertex weight

w_{ij} : edge weights

Mass matrix: M = diag$(w_1, ..., w_n)$ **diagonal matrix of vertex weights**

Stifness matrix : L_w defined as

$$l_{i,j} = \begin{cases} -\sum_{v_j \in N_1(v_i)} w_{ij} & \text{if } i = j \\ w_{ij} & \text{if } v_j \in N_1(v_i) \\ 0 & \text{otherwise} \end{cases}$$

is the symmetric matrix of edge weights.

$\mathbf{L} = M^{-1} L_w$

Then we have :

both L and L_w are sparse and M is diagonal

$$\begin{pmatrix} \Delta f(v_1) \\ \cdot \\ \cdot \\ \Delta f(v_n) \end{pmatrix} = \mathbf{L} \begin{pmatrix} f(v_1) \\ \cdot \\ \cdot \\ f(v_n) \end{pmatrix} = \mathbf{M}^{-1}\mathbf{L}_w \begin{pmatrix} f(v_1) \\ \cdot \\ \cdot \\ f(v_n) \end{pmatrix}$$

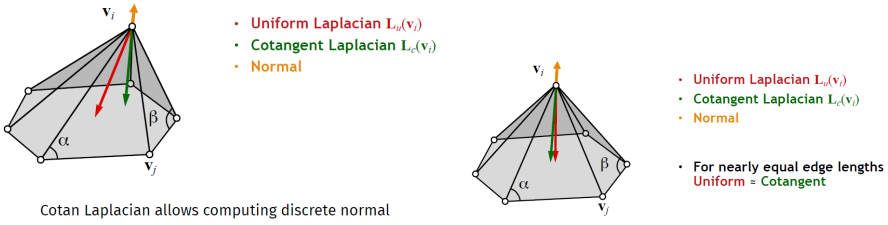
9.7.1 Linear system involving the Laplacian

Suppose we have a linear system : Lx = b using the asymmetric cotangent Laplacian

The right hand side vector b represents a scalar field , storing the desired Laplacian value at each vertex.

Solving symmetric sparse linear systems matrices is much more efficient so we prefer to solve : $L_w x = Mb$

9.8 Discrete Laplace-Beltrami Geometric meaning



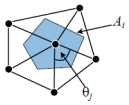
Recap: discrete curvatures

- Mean curvature (sign defined according to normal)

$$|H(\mathbf{v}_i)| = \|L_c(\mathbf{v}_i)\|/2$$

- Gaussian curvature

$$K(\mathbf{v}_i) = \frac{1}{A_i}(2\pi - \sum_j \theta_j)$$



- Principal curvatures

$$\kappa_1 = H - \sqrt{H^2 - K} \qquad \kappa_2 = H + \sqrt{H^2 - K}$$

10 Smoothing and Fairing

10.1 Surface Smoothing - Motivation

Scanned surfaces can be noisy.

Noise produces artefacts at high frequency.
Removing noise makes the surface smoother.
Marching Cubes meshes can be ugly.

10.2 How to measure smoothness?

C

10.2.1 Which Curvature

- Prinicipal curvature : Nonlinear and discountinuous operator in the definition
- Gauss curvature K : Intrinsic only , insensitive to embedding in 3D space
- Mean curvature H : Relatively simple to extract on meshes via Laplace-Beltrami

10.3 Smoothing by Diffusion Flow

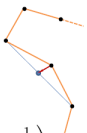
10.3.1 Example of Smoothing Curves

- Laplace in 1D = second derivative:

$$L(\mathbf{p}_i) = \frac{1}{2}(\mathbf{p}_{i-1} - \mathbf{p}_i) + \frac{1}{2}(\mathbf{p}_{i+1} - \mathbf{p}_i)$$

- In matrix-vector form for the whole curve

$$L\mathbf{p} \qquad \mathbf{p} = [\mathbf{x}\ \mathbf{y}] \in \mathbb{R}^{n\times 2} \qquad L = \frac{1}{2}\begin{pmatrix} -2 & 1 & & & 1 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 1 & & & 1 & -2 \end{pmatrix}$$



- Flow to reduce curvature:

$$\tilde{\mathbf{p}}_i = \mathbf{p}_i + \lambda \frac{d^2}{ds^2}(\mathbf{p}_i)$$

- Scale factor $0 < \lambda < 1$
- Matrix-vector form:

$$\tilde{\mathbf{p}} = \mathbf{p} + \lambda L\mathbf{p}, \quad \mathbf{p} \in \mathbb{R}^{n\times 2}$$

- May shrink the shape; can be slow

Going on with more iterations we are going to smooth the mesh

10.4 Diffusion Flow - General Scheme

Model for the time-dependent process of smoothing a given signal.

Diffusion flow is modeled by the **diffusion equation** , describing a signal over time :

$$\frac{\partial f(\mathbf{x},t)}{\partial t} = \lambda \Delta f(\mathbf{x},t)$$

A function f obeying to to the above equation becomes smoother and smoother for increasing values of t.

f(x,0) is the function at its initial state.

Parameter λ sets the speed at which the function is smoothed.

This technique is used to blur images and smooth terrain surfaces

The diffusion equation is a Partial Differential Equation.

We discretize it both in space and in time.

For the spatial discretization we replace the function f by its sample values at the mesh vertices f(t) $= f(v_1,t),...,f(v_n,t))^T$

For the temporal discretization we divide the time axis into regular intervals of size h, yielding time steps $\{t, t+h, t+2h..\}$, approximating the time derivative by finite differences :

$$\frac{\partial f(v_i,t)}{\partial t} \approx \frac{f(v_i,t+h) - f(v_i,t)}{h}$$

10.4.1 On meshes : smoothing as mean curvature flow

- Model smoothing as a diffusion process

$$\boxed{\frac{\partial \mathbf{p}}{\partial t} = \lambda \Delta \mathbf{p}} = -2\lambda H \mathbf{n}$$

- Discretize in time, forward differences:

$$\frac{\mathbf{p}^{(n+1)} - \mathbf{p}^{(n)}}{dt} = \lambda L \mathbf{p}^{(n)}$$

$$\mathbf{p}^{(n+1)} - \mathbf{p}^{(n)} = dt\, \lambda L \mathbf{p}^{(n)}$$

$$\boxed{\mathbf{p}^{(n+1)} = (I + dt\, \lambda L)\mathbf{p}^{(n)}}$$

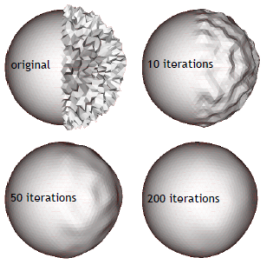
Explicit integration! Unstable unless time step *dt* is small

10.4.2 Taubin Smoothing : Explicit Steps

- Per-vertex iterations

$$\tilde{\mathbf{p}}_i = \mathbf{p}_i + \lambda L(\mathbf{p}_i)$$
$$\tilde{\mathbf{p}}_i = \mathbf{p}_i + \mu L(\mathbf{p}_i)$$

- Simple to implement
- Requires many iterations
- Need to tweak λ, u



$\lambda > 0$ to smooth and $\mu < 0$ to inflate

10.5 Smoothing as Optimization

- Let's go for $H = 0$ goal: $\Delta_{\mathcal{M}}\mathbf{p} = -2H\mathbf{n}$ $H = 0$ or $H = \text{const}$
- only trivial solution, no connection to initial surface $\mathbf{p}$
- Let's regularize!

$$\Delta_{\mathcal{M}}\tilde{\mathbf{p}} = 0$$ ← LAPLACE EQUATION

$$\min_{\tilde{\mathbf{p}}} \int_{\mathcal{M}} \underbrace{\|\Delta_{\mathcal{M}}\tilde{\mathbf{p}}\|^2}_{\text{small } H} + \underbrace{w}_{\text{weighting factor (like } 1/\lambda)} \underbrace{\|\tilde{\mathbf{p}} - \mathbf{p}\|^2}_{\text{stay close to original surface}}$$

10.6 Smoothing as filtering

10.6.1 Fourier Analysis

Represent a function as weighted sum of sines and cosines

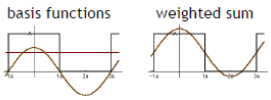


Figure 102: f(x) = $a_0 + a_1 cos(x)$

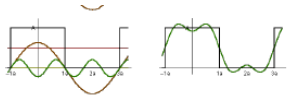


Figure 103: f(x) = $a_0 + a_1 cos(x) + a_2 cos(3x)$

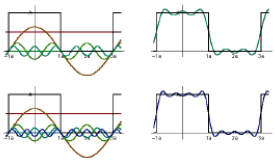


Figure 104: f(x) = $a_0 + a_1 cos(x) + a_2 cos(3x) + a_3 cos(5x) + a_4 cos(7x) + ...$

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \omega x} \, dx$$

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{2\pi i \omega x} \, d\omega$$

The Fourier transform maps a function from its representation f(x) in the spatial domain to its representation $F(\omega)$ in the frequency domain.

10.7 Extend Fourier to meshes?

Fourier basis functions are eigenfunctions of the Laplace operator.

On meshes : take the eigenvectors of the Laplace-Beltrami matrix

Take you L-B matrix L , compute the eigenvectors with k smallest eigenvalues, and reconstruct the mesh geometry from these eigenvectors.

10.8 Fairing

Fairing has the purpose to compute surfaces that are as smooth as possible.

Principle of simplest shape : the surface should be free of any unnecessary details or oscillations.

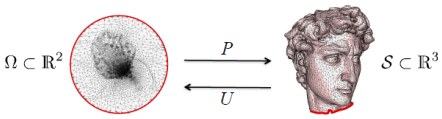
11 Mesh Parametrization

11.1 Parametrization - Definition

The notion of parametrization attaches a geometric coordinate system to the object

Mapping P between a 2D domain Ω and the mesh S embedded in 3D.

Each mesh vertex has a corresponding 2D position : $U(v_i) = (u_i, v_i)$

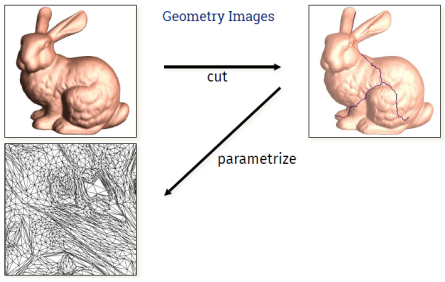


Parametrization allows us to do many things in 2D and then map those actions onto the 3D surface.

One goal of parametrization is Texture Mapping : put the sruface into a one-to-one correspondence with an image.

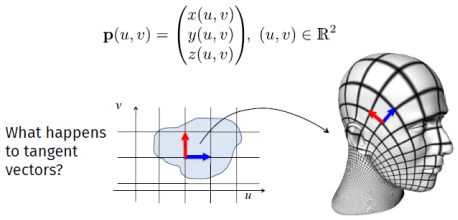
Another goal is Remeshing : the coordinate system defined by the parametrization facilitates converting from a mesh representation into an alternative one.

To summarize , a parametrization of a 3D surface is a function putting this surface in one-to-one correspondence with a 2D domain.



11.2 How to Measure Distortion?

11.2.1 Measures of Local Distortio



$p_u = \partial p(u,v)/\partial u$, $p_v = \partial p(u,v)/\partial v$

- How do lengths and angles of tangents change?
  - First fundamental form!

$$\mathbf{I} = \begin{pmatrix} E & F \\ F & G \end{pmatrix} = \begin{pmatrix} \mathbf{p}_u^T \mathbf{p}_u & \mathbf{p}_u^T \mathbf{p}_v \\ \mathbf{p}_u^T \mathbf{p}_v & \mathbf{p}_v^T \mathbf{p}_v \end{pmatrix}$$

- The eigenvalues of $\mathbf{I}$ tell us the maximal/minimal stretching of a tangent vector

$$\lambda_{1,2} = \tfrac{1}{2}\left((E+G) \pm \sqrt{4F^2 + (E-G)^2}\right)$$

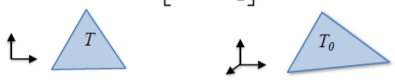
11.2.2 Distortion on Triangle Meshes?

Triangle in 3D is mapped to triangle in 2D.

Unique affine mapping.

SVD of the Jacobian reveals directions of extreme stretching

$$A = U \begin{bmatrix} \sigma_1 & \\ & \sigma_2 \end{bmatrix} V^T$$



$$P : \mathbb{R}^2 \to \mathbb{R}^3, \quad P(\mathbf{u}) = A\mathbf{u} + \mathbf{c}$$

$$[P_u \; P_v] = A \Longrightarrow \mathbf{I} = A^T A$$

The possible distortion measures are : $E(T) = \sqrt{\sigma_1^2 + \sigma_2^2}$ or $E(T) = max\{\sigma_1, 1/\sigma_2\}$

11.3 How to Compute (Good) Parametrizations? and Quickly?

11.3.1 Harmonic-Mapping - Idea

Want to flatten the mesh - no curvature → Laplace operator gives zero

Because the Laplacian measures the regularity of a function. For instance , for a linear function the Laplacian is equal to zero. Minimizing the Laplacian of u and v results in smooth parametric coordinates, in other words , this also minimizes the distortion of the parametrization.

11.3.2 Tutte's barycentric mapping Theorem

Given a triangulated surface homeomorphic to a disk :

1. If the (u,v) coordinates at the boundary vertices lie on a convex polygon

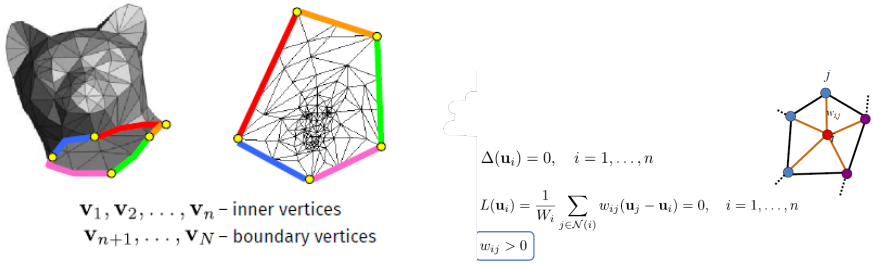
2. If the coordinates of the internal vertices are convex combination of their neighbors

3. Then the (u,v) coordinates form a valid parametrization

11.3.3 Convex Mapping

Boundary vertices are fixed.

Convex weights in the Laplacian matrix.

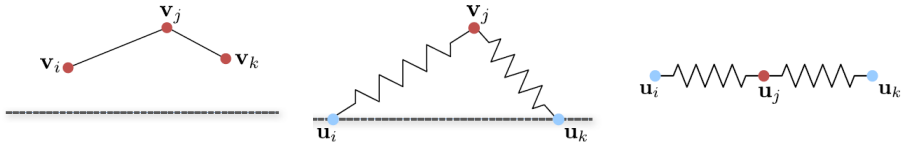


Solve the linear system $\mathrm{Lu} = 0$, the values of the boundary vertices are known and thus substituted

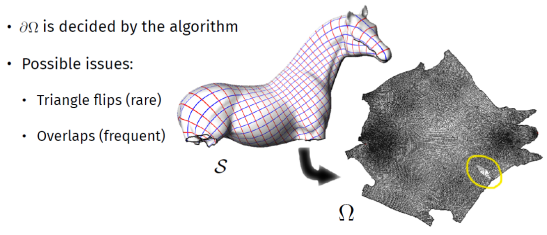
11.4 Harmonic Mapping

Inner mesh edges as springs.

Find minimum-energy state where all vertices lie in the 2D plane.



11.5 Boundary-Free Parametrization



Conformality can be imposed as a constraint on the Jacobian: the two partial derivatives are orthogonal , and they have the same length.

We define the parametrization P by defining its inverse U.

We define U as a per vertex(u,v) assignment: our variables are the (u,v) positions of the vertices in Ω.

The vectors $\partial P/\partial u$ and $\partial P/\partial v$ are linear with the variables constant inside triangles.

Such vectors can be derived triangle by triangle by inverting the derivatives of U

Conformality : $\partial P/\partial u$ x n = $\partial P/\partial v$, n normal of triangle on surface is known , $\partial P/\partial u$ x n - $\partial P/\partial v = 0$

11.6 Global Parametrization

All the algorithms that we studied until now are able to parametrize a part of a mesh homeomorphic to a disk.

To be able to parametrize arbitrary shapes we cut a mesh into parts , and we parametrize every part independently.

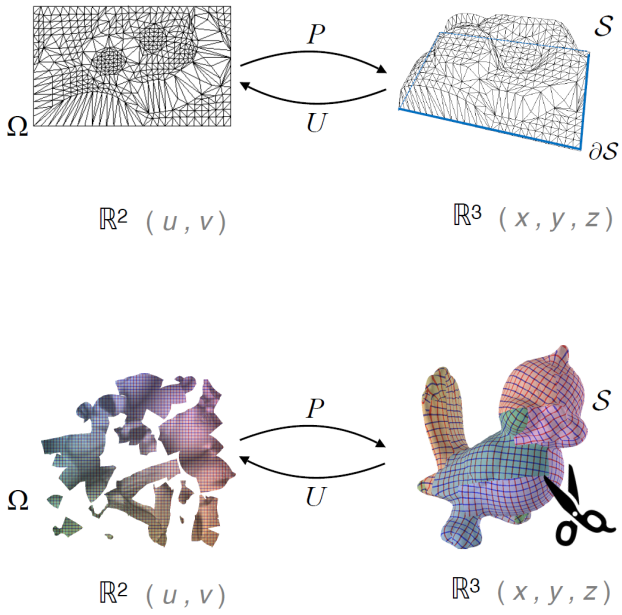


Figure 114: We have to cut the shape , in this way we can parametrixe it

11.6.1 How to handle cuts?

- We can ignore them and parametrize every chart separately. The collection of the separate chart can be grouped in a square with a packing algorithm. Segmentation.

- We want to impose continuity of the derivatives of the parametization across cuts. Packing.

12 Mesh Deformation

Shape deformation is used for animation , editing and simulation.

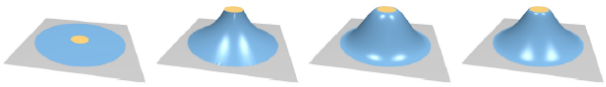
The deformation of a given surface S into the desired surface S' is described by a displacement function d that associates to each point p ∈ S a displacement vector d(p). By this map the given surface S to its deformed version S' : $S' = \{p + d(p) | p \in S\}$

For a discrete triangle mesh the displacement function d is piecewise linear, such that it is fully defined by the displacement vectors $d_i = d(p_i)$ of the original mesh vertices.

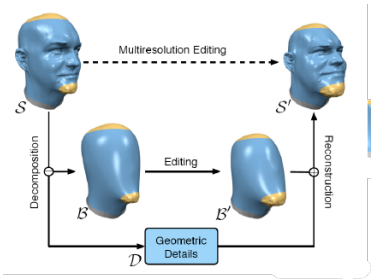
The user controls the deformation by prescribing displacements $\bar{d}_i$ for a set of so-called handle points p_i and by constraining parts which stay fixed during the deformation.

12.1 Surface-Based Deformations

A simple popular approach works by propagating the user-defined handle transformation within the deformation region. After specifying the support region R of the deformation and a handle H within it , the handle is transformed using some modelling interface.



Example of Surface-Based Deformations



Consists of three main operators : 1) the decomposition operator which separates low and high frequencies, 2)Editing operator , which deforms the low frequency components, 3)Reconstruction operator , which adds the details the details back onto the modified base surface.

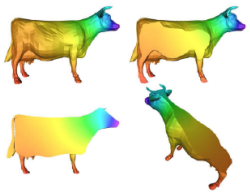
Basically we need to find a mesh that optimizes some objective functional and satisfies modeling constraints : $x_d ef = argmin E(x')$

12.2 Space Deformations

Global and local deformation of solids. F : $R^3 \rightarrow R^3$

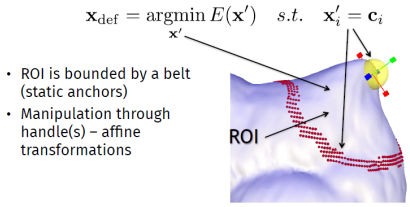
Design a set of coordinates for all points in R with respect to the "cage" vertices.

Each point x can be represented as a weighted sum of cage points p_i : x = $\sum_{i=1}^{k} w_i(x)p_i$, when the cage changes , the coordinate stay the same but we have to substitute the new cage geometry : x' = $\sum_{i=1}^{k} w_i(x)p_i$x = $\sum_{i=1}^{k} w_i(x)p'_i$



12.3 Surface-Based Differential Deformations

12.3.1 ROI-Handle Editing Metaphor



12.4 Recap: Differential Coordinates

Detail = smooth(surface) - surface

Smoothing = averaging.

12.4.1 Simple Laplacian Editing

Preserve mean curvature normal at every point in the ROI (part of surface we are interestd in deforming)

$$\text{continuous:} \quad E(\mathcal{S}') = \int_{\mathcal{S}'} \|\Delta\mathbf{x}' - \delta\|^2 d\mathbf{x}'$$

$$\text{discrete:} \quad E(\mathbf{x}') = \sum_{i=1}^{n} A_i \|\Delta(\mathbf{x}'_i) - \delta_i\|^2$$

12.4.2 Simplifying the Laplacian Energy

$$E(\mathbf{x}') = \sum_{i=1}^{n} A_i \|\Delta(\mathbf{x}'_i) - \delta_i\|^2 = \sum_{i=1}^{n} A_i \left(\Delta(\mathbf{x}'_i)^T \Delta(\mathbf{x}'_i) - 2\Delta(\mathbf{x}'_i)^T \delta_i + \delta_i^T \delta_i\right) =$$

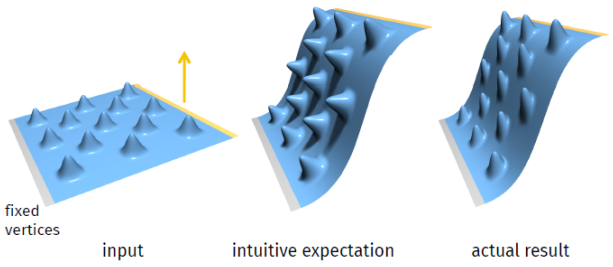
$$= \mathbf{x}'^T L^T M L \mathbf{x}' - 2\mathbf{x}'^T L^T M \delta + \text{const}$$

$$L^T M L = (M^{-1} L_w)^T M (M^{-1} L_w) = L_w M^{-1} M M^{-1} L_w =$$
$$= L_w M^{-1} L_w \longleftarrow \text{Symmetric sparse matrix!}$$

12.5 Fundamental Problem : Invariance to Transformations

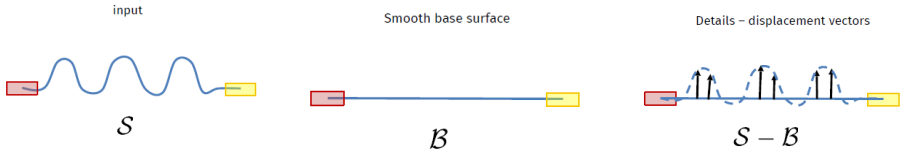
The basic Laplacian operator is translation-invariant , but not rotation-invariant.

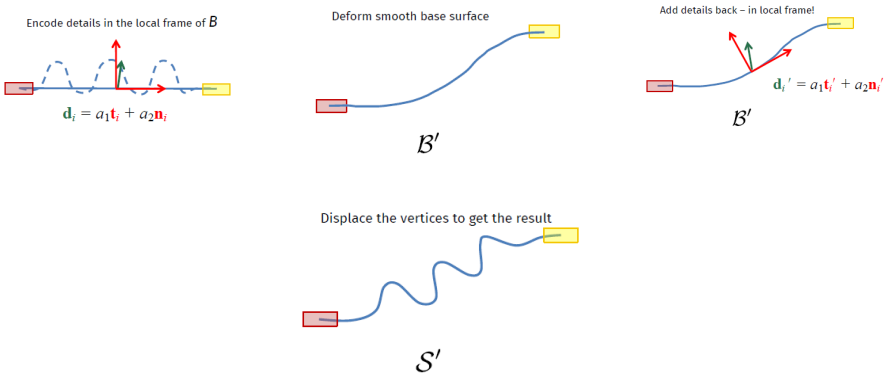
E(x') attempts to preserve the original globe orientation of the details (the normal directions).



We need a rigid-invariant energy : E(x') = $\sum_{i=1}^{n} A_i || \triangle (x'_i) - \delta_i ||^2$

12.6 Fixing Local Rotations : Multiresolution Approach





13 Splines

General idea : smooth curve/surface in parametric form.

Defined by combining two ingredients :

- Control polygon/grid built upon a small set of **control points**
- **Basis functions** to compute affine averages of point positions.

The curve/surface can be edited by moving the control points or reshaping the basis functions

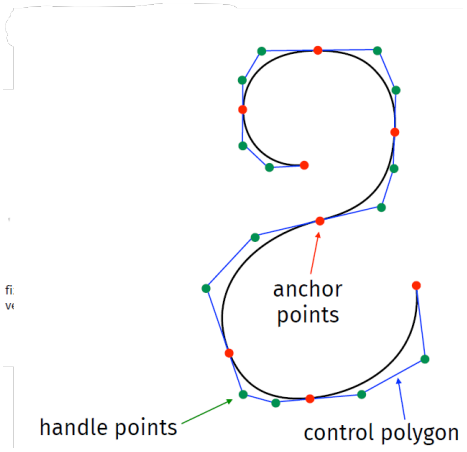
13.1 Curves

13.1.1 Spline Curves

Informal idea : control points determine the shape of the curve. **Anchor points are on the curve , Handle points pull the curve**.

Control polygon joining the control points approximates the curve.

A spline may consist of several segments of curve.



In math terms : s(t) = $\sum_i p_i B_i(t)$

The affine sum blends the contributions of the various control points. The enumeration of indices define the control polygon.

p_i are the control points , B_i is the base functions that weigh the influence of control points at each parameter value.

Basis functions characterize the type of spline.

13.1.2 Bezier Curves

Bases are polynomials of given degree n.

A Bezier segment of degree n has n+1 control points :

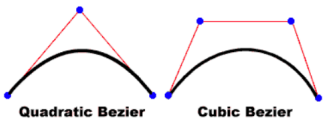
$$s(t) = \sum_{i=0}^{n} B_{i,n}(t)\mathbf{p}_i$$

- where $B_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}$

  are the Bernstein polynomials

- n = 1 : straight line segment
- n = 2 : parabolic arc

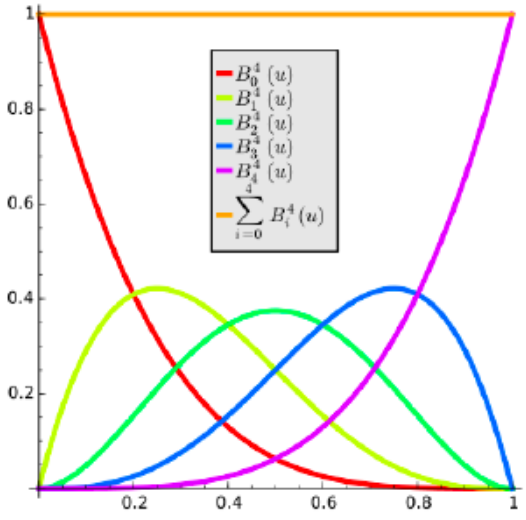
- n = 3 : cubic arc



For all n the curve : interpolates the endpoints of control polygon and is tangent to first and last segment of the control polygon at the end points

Properties

- The $B_{i,n}$ are non-negative in [0,1]
- Partition of unity: $\sum_{i=0}^{n} B_{i,n}(t) = 1$
- Interpolation at endpoints: $B_{0,n}(0) = B_{n,n}(1) = 1$
- Symmetry : $B_{i,n}(t) = B_{n-1,n}(1-t)$



Bernstein bases
of degree 5

Bernstein bases

- Recursive definition: $B_{i,0}(t) = 1 \quad i \geq 0$

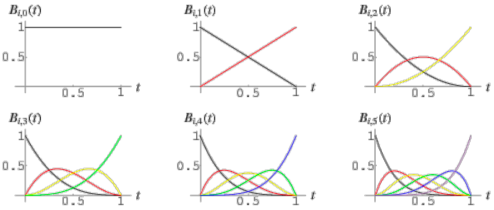
  $B_{i,n}(t) = (1-t)B_{i,n-1}(t) + tB_{i-1,n-1}(t)$

- Derivatives: $B'_{i,n}(t) = t(B_{i-1,n-1}(t) - B_{i,n-1}(t))$

  with $B_{-1,n}(t) = B_{n,n-1} = 0$

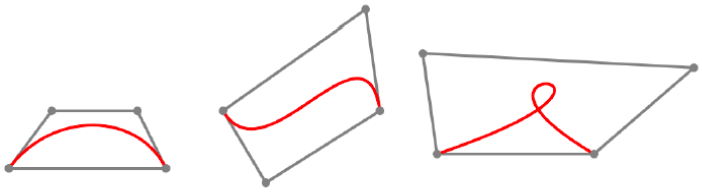
- Tangents at endpoints: derivatives at $\begin{matrix} t=0 \\ t=1 \end{matrix}$ depend only on $\begin{matrix} \mathbf{p}_0, \mathbf{p}_1 \\ \mathbf{p}_{n-1}, \mathbf{p}_n \end{matrix}$

Bernstein bases



Convex hull property : The curve is contained in the convex hull of its control points

Variation diminishing property : no straight line can have more intersection with the curve than with the control polygon.



Disadvantages :

- Global support: moving each single control point modifies the whole curve and the change is not intuitive
- Numerical instability at high degree / large number of control points
- Low expressive power : conics such as circles cannot be represented

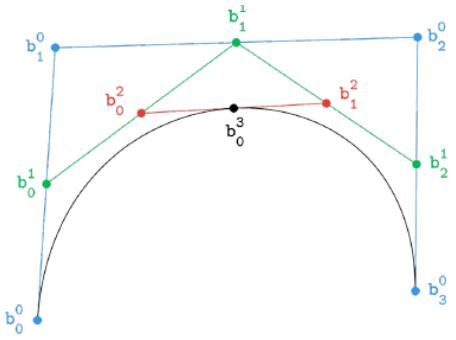
13.2 de Casteljau Algorithm

Bezier curves admit recursive definition : $b_i^0 = p_i$, $b_i^r = (1-t)b_i^{r-1}(t) + tb_{i+1}^{r-1}(t)$

$s(t) = b_0^n(t)$

This definition immediately provides a geometric construction to evaluate the curve by means of repeated affine averages.

The de Casteljau algorithm computed at any value t splits the curve into two Bezier segments, providing their control polygons b_0^0, b_0^1, b_0^2, b_0^3 and b_0^3, b_1^2, b_2^1, b_0^3

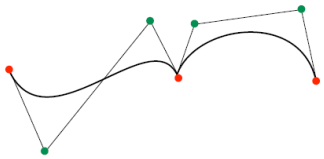


Recursive evaluation at t = 1/2 provides a sequence of control polygons that collapses onto the curve.

13.3 Bezier splines

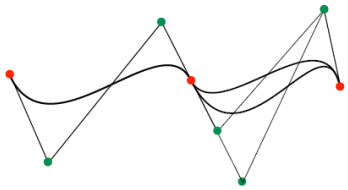
Bezier curves can be joined easily with C^0 and C

C^0: the last control point of a curve and the first control point of the next curve coincide



G^1 : the last edge of a control polygon and the first edge of the next control polygon are collinear

C^1 : the last edge of a control polygon and the first



C^2 continuity involves the mutual positions of the last three control points of a curve and the first three of the next. This may cause a loss of local control.

13.3.1 Can we do better than Bezier?

Wish list :

- Local support : each control point should influence only one small portion of curve
- Smooth splines : it should be easy to join curve segments smoothly
- Expressive power : use a more flexible set of bases

13.4 B-splines

Basis functions with compact support and flexible.
Defined upon a set of knots that subdivide an interval on R.
Knots control the amplitude and the shape of each basis function.
Basis function determine the influence of each control point.
Each segment of spline is determined by a subset of control points.

B-spline basis functions:

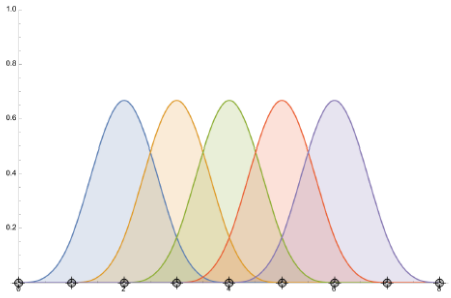
- $U = \{u_0, \cdots, u_m\}$ uniform partition of interval $[u_0, u_m] \subset \mathbb{R}$
- The i-th basis function of degree p (order p+1) on U is defined:

$$N_{i,0}(t) = \begin{cases} 1 & \text{if } t \in [u_i, u_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - u_i}{u_{i+p} - u_i} N_{i,p-1}(t) + \frac{u_{i+p+1} - t}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(t)$$

Example

B-spline basis functions with m=8, p=3



Properties

- There are n = m-p bases of degree p on m intervals

- All non-negative
- Each basis spans p+1 intervals
- Consecutive bases overlap
- Partition of unity in $[u_p, u_{m-p}]0 : \sum_{i=0}^{n-1} N_{i,p}(t) = 1$

B-spline curve :

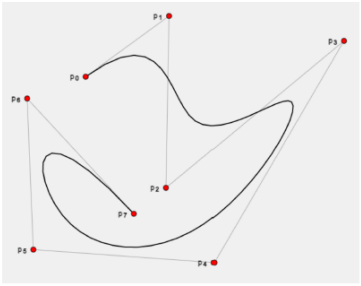
$$s(t) = \sum_{i=0}^{n-1} N_{i,p}(t)\mathbf{p}_i$$

- defined in interval $[u_p, u_{m-p}]$
- consisting of m-$2p$ segments
- the j-th segment is defined in $[u_{p+j}, u_{p+j+1}]$ and is controlled by points $\mathbf{p}_j, \ldots, \mathbf{p}_{j+p}$ for $j \in [0, m-2p-1]$

Uniform B-splines :
Completely defined by the n control points and the choice of p.
Have smoothness C^{p-1}, closed curves can be defined by using the knot interval in a periodic way.

Non-uniform B-splines : Knots can be moved to create non-uniform intervals. Non uniform intervals modify the shape of the basis function.
Different knots can be dragged to the same location to change smoothness of the curve
Each time we increase the multiplicity of a knot, we decrease smoothness by one.
A knot with multiplicity p forces the curve to interpolate one control point.

Open-uniform B-splines :
Set multiplicity p at the endpoints of the interval and uniform intervals inside.
We obtain a curve interpolating the endpoints while keeping the smoothness and control of B-splines.



13.5 de Boor Algorithm

Generalization of de Casteljau algorithm for B-splines.

Exploits the recursive definition of B-spline bases:

$$\begin{aligned}
&\mathbf{b}_i^0 = \mathbf{p}_i \\
&\mathbf{b}_i^j = (1-\alpha_i^j)\mathbf{b}_{i-1}^{j-1} + \alpha_i^j \mathbf{b}_i^{j-1} \text{ where } \alpha_i^j = \frac{t-u_i}{u_{i+p-j+1}-u_i} \\
&s(t) = \mathbf{b}_i^p
\end{aligned}$$

13.6 Knot Insertion

Purpose : represent the same curve with more knots to support more flexible editing.

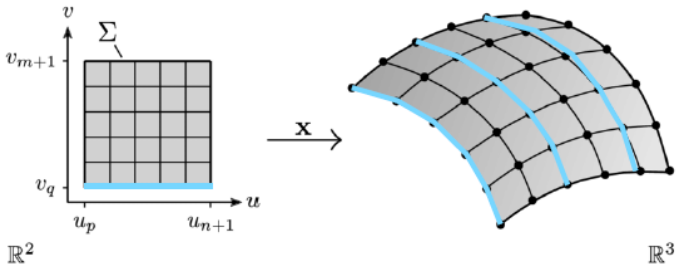
Problem: Given a B-spline with m knots, find another B-spline with m+1 knots that describes the same curve.

Boehm algorithm provides a simple formula to compute the positions of the control points upon insertion of new node.

Oslo algorithms generalize to the simultaneous insertion of more nodes.

13.7 Surfaces

Idea : sweep a curve in space by dragging its control points along other curves.



In math terms :

$$S(u,v) = \sum_{i,j} B_i(u) B_j(v) \mathbf{p}_{ij}$$

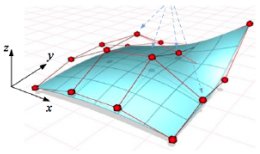
indices vary on a 2D grid — two 1D base functions — control points arranged on a grid

This scheme can be applied to obtain surfaces from all piecewise polynomial schemes of curves

13.7.1 Bezier surfaces

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{n} B_{i,n}(u) B_{j,n}(v) \mathbf{p}_{ij}$$

- the $B_{i,n}$ are Bernstein polynomals of degree n
- the control points form a square control net
- the control net is in fact a regular $n \times n$ quad mesh



13.7.2 B-spline surfaces

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,p}(v) \mathbf{p}_{ij}$$

- the $N_{i,p}$ and $N_{j,p}$ are B-spline basis functions of degree p defined on knot vectors $[u_0,\ldots,u_{n+p+1}]$ and $[v_0,\ldots,v_{m+p+1}]$, respectively
- the control net is a regular $n \times m$ quad mesh

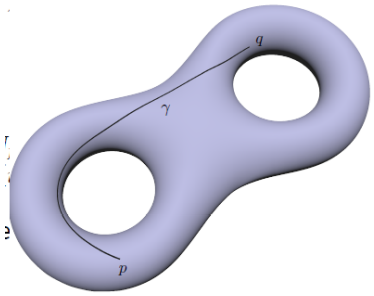
14 Geodesics on Surfaces

14.1 Curve on a surface

M surface.

p,q points on M.

γ curve on M joining p to q:



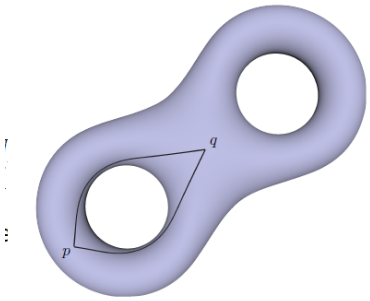
- $\gamma[0, l_\gamma \rightarrow$ M
- $\gamma[0] = p$, $\gamma[l_\gamma]$
- l_γ of γ

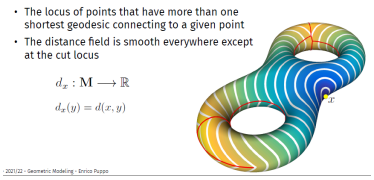
14.2 Geodesic distance on a surface

Distance between A and B : length of the shortest curve joining A to B : d(A,B) = min l_γ

If d(A,B) = $l_{\bar{\gamma}}$ then $\bar{\gamma}$ is called a **shortest geodesic (path) between A and B**

The geodesic distance is unique but the shortest geodesic path is not always unique.



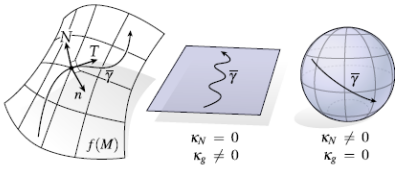


14.3 Cut Locus

14.4 Geodesic Lines

A curve γ is a geodesic line if it bends with M but it does not bend on M.

- Tangent of γ: $T(s) = \frac{d}{ds}\gamma(s)$ where s is the arc-length parameter
- Normal of γ: $n = T \times N$ where N is the normal to the surface
- Normal curvature: $\kappa_N = N \cdot \frac{d}{ds}T$
- Geodesic curvature: $\kappa_g = n \cdot \frac{d}{ds}T$
- A curve is a geodesic line iff $\kappa_g = 0$



In some sense , a geodesic line is straight on M.

Shortest paths are geodesic lines.

Not all geodesic lines are shortest paths.

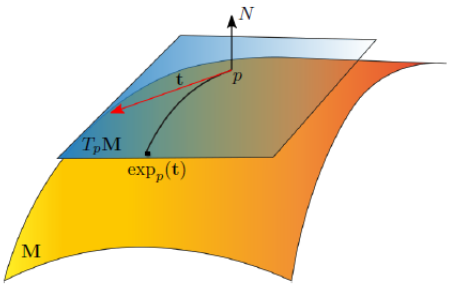
Geodesic lines are locally shortest.

A geodesic line is fully characterized with : its starting point p , its tangent direction at p and its length.

Exponential map: $exp_p : T_pM -> M$

It maps vectors from the tangent plane at a given point p into points on M.

The exp map of a vector t is the endpoint of a geodesic cast from p in the direction and for the length of t.



14.5 Normal and Convex sets

A submanifold B contained in M is a **Normal neighborhood** of point x if for each t ∈ B there exists a unique shortest ath on M from x to y.

A **totally normally set** if it is an open set that is a normal neighborhood of all its points.

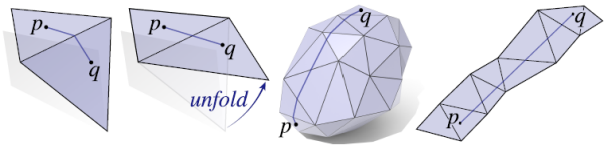
Strongly convex if for each pair x and y of its points there exists a unique shortest path on M connecting x to y and it is contained in B

14.6 Polyhedral setting

On a polygonal mesh , geodesic lines are polylines.

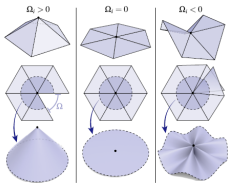
One straight-line segment per face crossed.

The geodesic line is a straight line once the crossed strip of faces is unfolded to the plane.

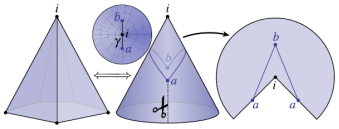


Classification of vertices by total angle: $\Omega_i = 2\pi - \sum_{ijk} \theta_i^{jk}$

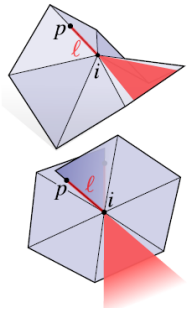
- Elliptic: $\Omega_i > 0$
- Euclidean: $\Omega_i = 0$
- Hyperbolic (saddle): $\Omega_i < 0$



- An elliptic vertex is locally like the apex of a cone
- No geodesic shortest path through elliptic vertices
- It is always shorter to go around the apex than going through it



- Infinitely many geodesics coming from a direction and through a saddle vertex
- Such geodesics form a *funnel*:
  - Cut 1-ring along incoming direction ℓ
  - Unfolding 1-ring to the plane creates overlap
  - Extending the two copies of ℓ with straight lines forms a wedge
  - All directions within wedge extend the geodesic ℓ



15 Computational Geodesics

15.1 Geodesic Tracing

We assume that the starting point is a vertex v_i (points on edges or inside triangles are easier to handle).

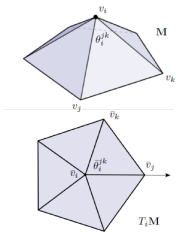
Tangent direction t is given on the tangent plane T_iM at v_i

Basic steps :

1. Project t on M and find the triangle f crossed by such projection
2. Propagate the triangle adjacent to f by crossed edge
3. Repeat step 2 until reaching distance $|t|$

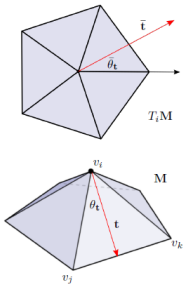
Mapping the 1-ring to the tangent plane :

- Compute the total angle Θ_i about vertex v_i
- Rescale all incident angles by $2\pi/\Theta_i$
- Map one edge to the x axis and distribute the others by rescaled angles.



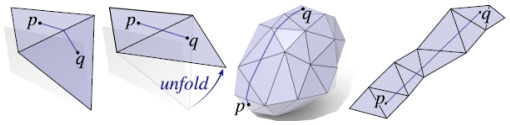
Mapping the tangent direction to the mesh:

- Find the triangle in tangent plane containing direction $\bar{t}$
- Rescale angle by $\Theta_i/2pi$
- Map angle to the corresponding triangle of M



Propagating the direction :

- Find the outgoing edge of the direction in the current triangle
- Unfold the neighboring triangle on the same plane and propagate a straight line through it.
- Repeated unfolding gives a triangle strip



What to do if the geodesic crosses a vertex?
Straightest geodesic : Extend with line that halves the total angle

15.1.1 Single source Geodesic Distance

Three classes of algorithms:

- **Graph-based**: restrict possible routes to paths in a graph; compute distance function on the graph
- **PDE-based**: resolve a global PDE whose result approximates the distance function
- **Polyhedral wavefront propagation**: propagate a front through the mesh while building a data structure that supports computing the exact polyhedral distance from any point.

15.2 Graph-Based Techniques

15.2.1 Shortest path on graphs

- $G=(V,E)$ graph; V set of nodes, E set of arcs
- $w(e)$ weight associated to any edge (for us: geodesic length)
- Weight of a path $p=<v_0,v_1,\ldots,v_k>$ is the sum of weights on the arcs

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Shortest path between u and v:

$$\bar{p}_{uv} = \underset{p}{\operatorname{argmin}}\{w(p) : p \text{ path connecting } u \text{ to } v\}$$

- Length of shortest path: $\delta(u, v) = w(\bar{p}_{uv})$
- Warning: the shortest path is not necessarily unique!

Assume all weights are non-negative. Let s $\in$ V , it is well defined the tree of shortest paths rooted at s , $G_s = (V_s, E_s)$ where V_s is the set of vertices that can be reached from s and for each v in V_s the simple path from s to v in G_s is a shortest path in G.

15.2.2 Optimal substructure of shortest paths

- For each shortest path $p=<v_0,v_1,\ldots,v_k>$ and for each pair v_i,v_j with $i<j$ the sub-path $p_{ij}=<v_i,v_{i+1},\ldots,v_j>$ is a shortest path between v_i and v_j
- Let s be a source and u, v be two nodes connected with an arc in G, then we have

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

- We can find shortest distances by expanding known shortest paths to neighbor nodes
- Since one node can be reached from different neighbors, we select the shortest among possible paths - all candidate paths must be visited

15.2.3 Graph Relaxation

For each node v let us define :

- d[v] candidate length of shortest path , initialized at inf
- π[v] node preceding v in a candidate path , initialized empty

- Relaxation of an arc(u,v) : if d[v] > d[u] + w(u,v) then d[v] = d[u] + w(w,v) , $\pi(v) = (\pi(u), (u,v))$

If we repeatedly relax all arcs of E the value of d[v] is always greater or equal than $\delta(s,v)$ and if d[v] reaches the value $\delta(s,v)$ then it becomes stable.

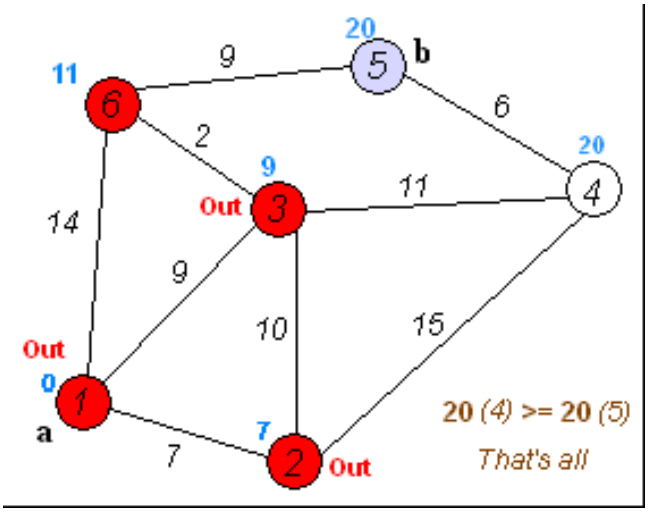
If $< s, ..., u, v >$ is the shortest path between s and v and we have d[u] = $\delta(s,u)$ then after relaxing edge (u,v) we have d[v] = $\delta(x,v)$

15.2.4 Dijkstra Expansion

Input : graph G = (V,E) weight w , source s.

Output : graph G = (V,E) where for each v we have d[v] = $\delta(s,v)$ and π(v) points to his predecessor on the shortest path from s.

Shortest paths can be extracted next in optimal time by backtracking from destination to source following $\pi(v)$



(a) Example of dijkstra

15.2.5 SLF-LLL Heuristics

The priority queue of Dijkstra warrants optimal worst-case time , but it is expensive to maintain in practice.

For planar graphs and graphs used in geodesic algorithms , an algorithm using a double ended queue achieves better practical performances, while being not optimal in the worst case :

- **Small Label First (SLF)** : a new node is added either to the front or to the back of the queue , depending on its relative cost with respect to the first node of the queue.

- **Large Label Last (LLL)**: nodes from the front of the queue that have a cost higher than the average are moved to the back of the queue.

15.2.6 Which graph to use?

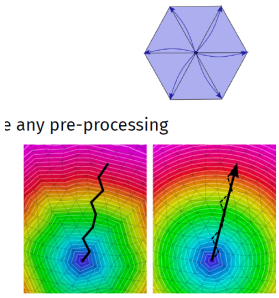
The graph restricts the possible sources to its nodes , and the possible paths to graph paths.

Trade-off : find a graph that achieves a good accuracy with a small number of nodes/arcs

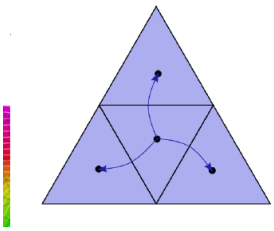
In most cases , defining the distance field just at the vertices of a mesh is sufficient.

15.2.7 Which graph to use?

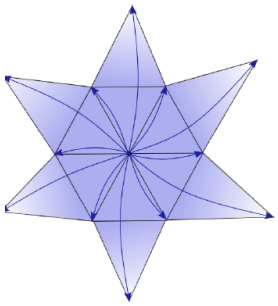
Graph of edges: V vertices of the mesh , E edges of the mesh. Easy and cheap but inaccurate.



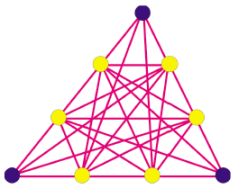
Graph of adjancencies: V centroids of triangles of the mesh, E dual edges = adjacencies between triangles of the mesh. Easy and cheep but inaccurate

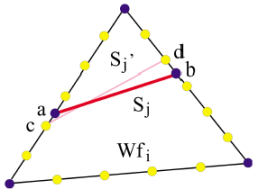


Graph of primal and dual edges: V vertices of the mesh , E edges of the mesh and dual edges. Each vertex is connected with its neighbors and with the vertices opposite to it in the triangles adjacent to its 1-ring

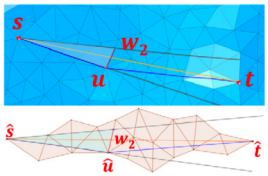


Graph with Steiner points on edges : edges of the mesh are split by distributing Steiner points along them , within each triangle , arcs joining mutually visible points on its boundary are added. Easy but add extra nodes.





Discrete Geodesic Graph: V vertices of the mesh, E subset of arcs of the total graph. Each arc is an exact shortest path between its endpoints and any exact path can be approximated with a path in the graph within a given tolerance ϵ



15.3 Exact Polyhedral Methods

Input : Mesh M and source s

Output : An explicit representation of the geodesic distance function d_s : M $\rightarrow$ R computable at any point p on M . An explicit representation of geodesics shortest paths $\delta(s, p)$

15.3.1 Polygonal Wavefront Expansion

Algorithm MPP and many subsequent improvements.

Basic Idea : expand a front of edges starting at the neighborhood of source s until covering the whole surface. Encode distance function d_s for points on each edge e as a set of windows splitting e

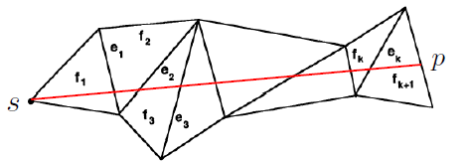
15.3.2 Geodesics through a triangle strip

Assume s is a vertex.

Consider a triangle strip starting at s and flatten it to the plane.

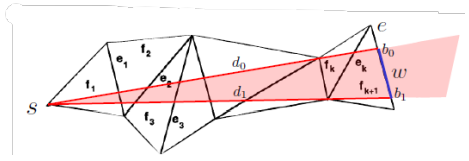
Assume there exists at least one straight line from s contained inside the strip and crossing all its triangles (if not we are not interested in that strip).

The folding of such line to M is a shortest geodesic connecting s to its other endpoint.



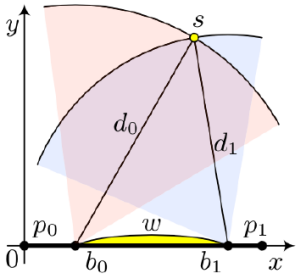
The set of all lines from s through the strip , and piercing the same edge e on the opposite end of it , form a wedge , encoded as window w on e :

- parametric coordinates b_0 and b_1 of endpoints of w on e
- Distances d_0 and d_1 of endpoints of w from s
- Binary direction τ specifying the side of e on which s lies



15.3.3 Distance function in a window

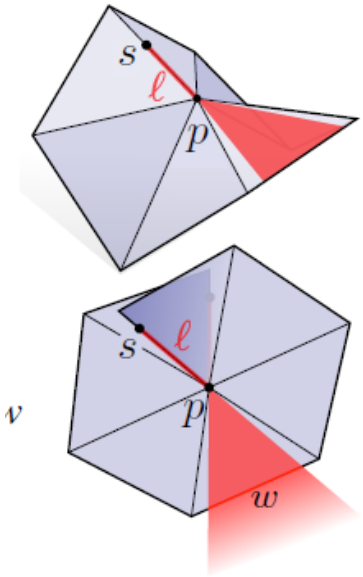
- Coordinates (x_s, y_s) of s are computed by intersecting the two circles centered at endpoints b_0 and b_1:
  - $(x_s-b_0)^2+y_s^2=d_0^2$
  - $(x_s-b_1)^2+y_s^2=d_1^2$
- $d_s(x)=\text{sqrt}((x_s-x)^2+y_s^2)$ for $x \in [b0,b1]$



15.3.4 General Case

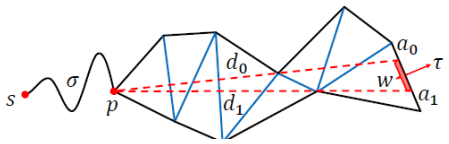
Not all points can be reached from straight lines through strips.

What happen at saddle vertices? The unfolding of the 1-ring overlaps , points in the red wedge cannot be reached by straight lines from s , so we need to turn about p to reach the red window w. Line l is a mandatory route to all points of w.



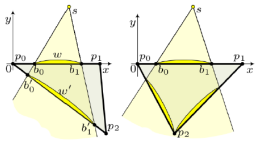
Then we decompose the paths to a generic window w into a **funnel** consisting of two parts : a polygonal path of length σ from source s to pseude-source p and a wedge from p to w.

The distance function for all x in w is encoded as : $(b_0, b_1, d_0, d_1, \sigma, \tau)$



15.3.5 Window Propagation

Given a window w on an edge e_1 propagate its distance field across an adjacent triangle t to define new potential windows on the opposing edges e_2, e_3.



To define the distance field over w' : extend the rays from the pseudo-source s through the endpoints of w and intersect them wit the new edge , to obtain the new interval $[b'_0, b'_1]$. Compute the new distances d'_0, d'_1 from pseudo-source s. Pseudo-source distace $\sigma' = \sigma$ is unchanged. Direction τ is assigned to point inside triangle (refers to the image above).

Special case : if one of the endpoints of w is a saddle vertex p_0 : Consider the other edge $\bar{e}$ of face t incident at p_0. If $\bar{e}$ outside the funnel , the generate the two new red windows in the figure, having p_0 as pseudo-source. Distance σ must be updated $\sigma = \sigma + d_0$

15.3.6 Continuous Dijkstra propagation

Maintain a priority queue Q of windows. Prioriry is minimum distance from source.

Initialization of Q :

- If is inside triangle t, then compute one window per edge of t and insert them in the queue.
- If s is on an edge e , then compute one window per edge of the two faces incident at e and insert them in the queue. Also compute two windows on e.
- If s is a vertex , then every triangle t in the star of s compute one window per edge of t opposite to s and insert them in the queue. Also compute one window for each edge e incident at s.

Loop while Q becomes empty :

- Pop a window w from Q
- Propagate window w
- Insert each non-empty window generated by propagation into Q

Propagation may:

- Add a new window
- Modify existing windows
- Delete existing windows

15.3.7 Geodesic path Construction

Once all edges are covered by windows representing distance, it is easy to trace a shortest path from any surface point p back to the source.

For a point p inside a face r: minimize $|p-p'|+d_s(p')$ for all p' on the edges of t. Jump to the window containing the point that minimizes that distance.

For a point p on a window: Find the adjacent triangle t according to direction τ. Reconstruct the position of the pseudo-source s in the plane of t and intersect the line to s with the other two edges of t. Jump to the intersection point , which is on new window , and repeat. When reaching a pseudo-source s : explore the windows incident at s until a window with a pseudo-source different from s is found.

Repeat until reaching the source.

15.4 PDE-Based Methods

15.4.1 The Eikonal Equation

The SSGDD problem can be formalized as a PDE:

$|\bigtriangledown \phi|^2 = 1$, $\phi(s) = 0$

The change in distance per unit distance along the direction of greatest increase should be 1.

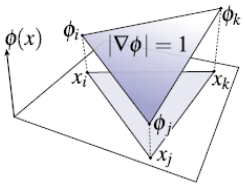
Geodesic lines are integral curves of the gradient of the distance field.

15.4.2 The Fast Marching Method

Following a Dijkstra-like traversal of the mesh, starting at the source.

Distances are not updated according to paths along edges, but by solving for the linear function that satisfies the eikonal equation.

If the values ϕ_i, ϕ_j at two corners are known , find a value ϕ_k so that the slope $|\bigtriangledown \phi|$ of a triangle passing through all three values equals $+1$.



15.4.3 The Varadhan Formula

- The distance field ϕ_x is connected with the *heat kernel* $k_{x,t}$

$$\phi_x(y) = \lim_{t \to 0} \sqrt{-4t \log k_{x,t}(y)}$$

where $k_{x,t}$ is the solution at time t to the heat equation

$$\frac{d}{dt} k_t = \Delta k_t$$
$$k_0 = \delta_x$$

with δ_x a Dirac delta centered at x

- "Start with heat concentrated at x and let it evolve for a short time"

The heat equation is linear and can be resolved easily.

Heat decays exponentially with distance, hence the norm of the result becomes soon unreliable : the smaller t the faster the decay , the bigger t the worse approximation of Varadhan formula

15.4.4 The Heat Method

- Compute the heat kernel by resolving

$$\frac{d}{dt} k_t = \Delta k_t, \quad k_0 = \delta_x$$

for fixed t

- Compute the normalized gradient $X = -\nabla k / |\nabla k|$
- Resolve the Poisson equation $\Delta\phi = \nabla \cdot X$

- We retain just the gradient direction of the heat kernel
- From the eikonal equation, we know that the gradient of the distance function must have norm 1

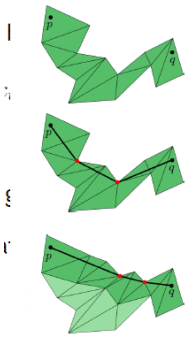
- We integrate the gradient to obtain the distance field

15.5 Local Methods for PPGP

15.5.1 Point to Point Geodesic Path

1. Start with a triangle strip joining p to q, usually found with a graph-based method

2. Unfold strip to the plane, one triangle at a time by using intersecting circles
3. Find shortest path within the strip , Funnel algorithm
4. Straighten the strip to remove turns, Straightening the strip
5. Repeat 3 and 4 until convergence

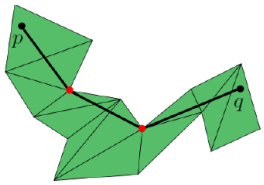


15.5.2 Shortest path in a triangle strip

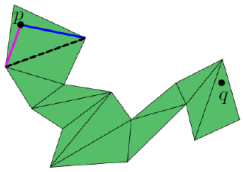
Input: triangle strip in the plane, containing p and q in its end triangles , respectively.

Output: Shortest polyline connecting p to q inside the triangle strip.

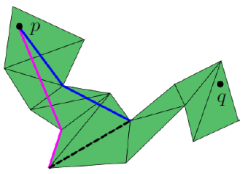
The path can turn only at reflex vertices



Initialize funnel : connect p to the endpoints of first transversal edge of the strip; the mouth of the funnel spans all points that can be reached from p

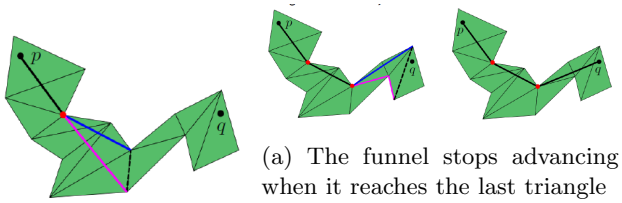


Expand funnel : move one side of the funnel to the next transversal edge; the sides of the funnel may become concave chains when they overcome reflex edges.



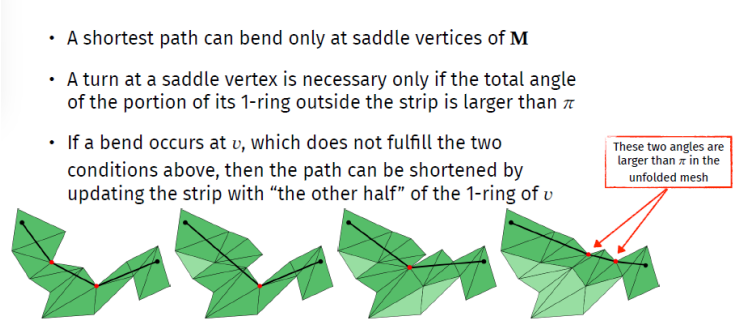
While advancing , the sides of funnel they may partially collapse to form the beak.

Collapses happen when updating one side overcomes the other side.



(a) The funnel stops advancing when it reaches the last triangle

15.5.3 Straightening the strip



16 Vector Graphics

16.1 Vector Graphics

Specify graphics drawings as composed of **vector primitives**.

A vector primitive is defined with:

- A set of control points in a reference system
- Possible additional parameters
- Math to uniquely identify the primitive from the control points and the parameters.

Examples

A straight line segment is defined by two control points p and q and parametric equation p+α(p-q).

A circle is defined by point c radius r and equation $|p - c|^2 = r^2$

16.1.1 Vector Graphics on a mesh

Can we extend vector graphics to a manifold domain, like a surface mesh?

We should redefine everything under the geodesic metric: straight line can be replaced with geodesic lines. What about angles? Ellipses?

16.1.2 Referencing points on a mesh

We can assign mesh coordinates to all points of a mesh M.

Point p has coordinates (t_p, a_p, b_p) where :

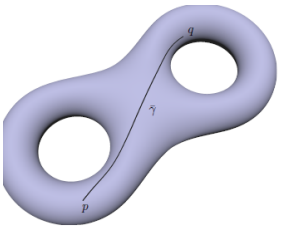
- t_p is the triangle of M containing p
- $a_p.b_p$, are the barycentric coordinates of p in t. A tuple of coordinates uniquely identifies a point of M

16.2 Simple Vector Primitives

16.2.1 Geodesic line segment

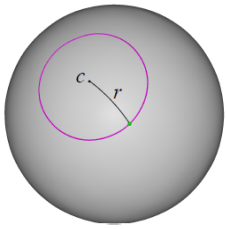
Defined by a pair of points p,q.

We assume this is the shortest geodesic joining them and it is unique.



16.2.2 Geodesic Circle

Defined by center c and radius r : SSGD from c gives distance field d_c at all vertices of M.



It is defined by three points p_1, p_2, p_3 : we need to compute the distance fields from these points with SSGD , find the point c where the three distance fields have the same value and the corresponding value r and find the circle centered at c of radius r with previous method.

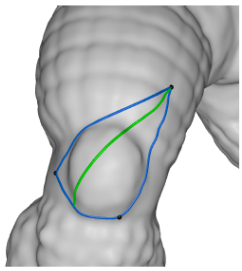
16.2.3 Triangle

Defined by its vertices p_1, p_2, p_3 :

1. Connect the vertices with shortest geodesics with PPGP
2. Find the interior of triangle by flooding the surface from such geodesics.

16.2.4 Isosceles triangle

Defined by basis and height or defined by basis and angle at basis or defined by basis and length of diagonal edges



16.2.5 Equilateral triangle

Defined by its basis. Different constructions are possible , but only one of the following properties can be guaranteed : all edges have equal length or all angles are equal.

16.2.6 Rectangle

Several possible definitions.

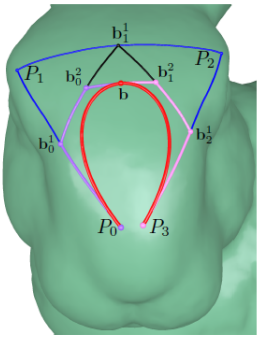
If edges are geodesics , the basic properties of Euclidean triangles cannot be guaranteed

16.3 Bezier Curves

Directly translating the Bernstein or De Casteljau formulas does not work in general case.

A recursive application of the De Casteljau construction defines a subdivision scheme that converges to a C^1 curve

Example for a cubic curve



Given the initial control polygon (blue):

1. Build the De Casteljau construction for evaluating the midpoint of the curve (the control polygon is substituted with a chain of two control polygons joining at the midpoint

2. Apply Step 1 recursively to each sub-polygon
3. Stop when the control chain is straight enough

Each recursion step requires:

1. finding the midpoint of three existing geodesics b_0^1, b_2^1, b_1^1
2. Computing geodesics b_0^1, b_1^1 and b_2^1, b_1^1 and their midpoints b_0^1, b_2^1
3. Computing geodesics b_0^2, b_1^2 and its midpoint b
4. Halving five existing geodesics