panikzettel.philworld.de

Geometry Processing Panikzettel

Hans Wurst

September 14, 2021

Contents

1 (Discrete) Differential Geometry Primer

Regression Line (Normal Estimation)

1. Center of gravity (c) should lie in the origin:

$$c = \frac{1}{n}\sum p_i \quad \Rightarrow \quad p_i' = p_i - c$$

2. Determine *covariance matrix* M:

$$M = \sum p_i' p_i'^T$$

3. Compute eigenvector corresponding to *smallest* eigenvalue of M, results into normal direction:

$$(A - \lambda)v = 0, \text{ or } \det(A - \lambda I) \quad \Rightarrow \quad v =: n$$

4. Evaluate implicit representation of line:

$$L = \{x \mid n^T x - n^T c = 0\}$$

1.1 (Discrete) Differential Geometry of Curves

Parametric representation of curves, that maps a 1D parameter domain to the curve:

$$f : \Omega = [a, b] \subset \mathbb{R} \to C$$

Implicit definition is only available for *planar* curves:

$$C = \{\mathbf{x} \in \mathbb{R}^2 | F(\mathbf{x}) = 0\} \quad \text{with} \quad F : \mathbb{R}^2 \to \mathbb{R}$$

Interpretations:

- Position $f(t) = (x(t), y(t), z(t))^T$, $t \in [0, 1]$
- Tangent vector $f'(t) = (x'(t), y'(t), z'(t))^T$: orientation, length = *velocity* at time t
- "Curvature" vector $f''(t)$: *change of direction* in sec
- Normal vector $n(t) = f'(t)^\perp / \|f'(t)^\perp\|$ ($^\perp$ = turned 90°)

Curvature $\kappa = \frac{1}{r}$ can be described as an *osculating circle* with radius r touching the point $f(t)$ and two neighboring points, while being tangent to the curve at $f(t)$.

Decompose curvature vector:

$$f''(t) = r + y, \quad \text{with} \quad r \parallel f'(t), \quad y \perp f'(t)$$

where r measures the change in velocity *(non-geometric information)*, y measures the change in orientation *(geometric information)*.

1.1.1 Arc-Length Parametrization

Given a curve, we are interested in the local properties (position, orientation, curvature) of point t:

$$f(t) = f(0) + tf'(0) + \frac{1}{2}t^2 f''(0) + \cdots \text{ (Taylor expansion)}$$

with $f(0)$ being the position, $tf'(0)$ the orientation, $\frac{1}{2}t^2 f''(0)$ the curvature information.

Assume *constant unit speed*, such that $r = 0$, therefore $f''(t) = y$.

Idea: For the *arc length parametrization*, we want $\boxed{\|f'\| \stackrel{!}{=} 1,\ \|f''\| \stackrel{!}{=} \kappa}$ with κ curvature/length of y.

The length $l(c,d)$ of any curve segment defined on an interval $[c,d] \subseteq [a,b]$ can be computed as the integral of the tangent vector:

$$l(c,d) = \int_c^d \|f'(u)\| du$$

Parametric curves allow for an unique parametrization that can be defined as a length-preserving mapping *(isometry)*, between the parameter interval and the curve using the parametrization

$$s = s(u) = \int_a^u \|f'(t)\| dt$$

where s is *arc length*. This arc length parameterization is independent of the specific representation of the curve and maps the parameter interval $[a,b] \to [0,L]$, where $L = l(a,b)$ total length of curve.

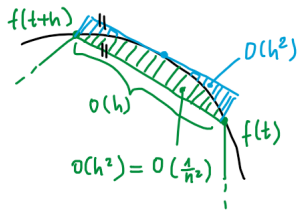
General formula for curvature κ:

$$\frac{\|f' \times f''\|}{\|f'\|^3} = \frac{\|f'\|\|f''\| \sin\alpha}{1} \stackrel{\alpha=90^\circ}{=} \boxed{\|f''\| = \kappa}$$

1.1.2 Approximation Power

Idea: Replace curve with linear segments using *h-refinement* (polynomial → piecewise polynomial).

From calculus we know that a C^∞ function g with bounded derivatives can be approximated over an interval of length h by a polynomial of degree p such that the approximation error behaves like $\mathcal{O}(h^{p+1})$, e.g., *mean value theorem* or *Taylor's theorem*:



$$f(t) = f_0 + t f_0' + \mathcal{O}(t^2)$$

Mean Value Theorem: If curve is given, and two points are connecting one linear segment, there is at least one point, which tangent is parallel to this segment.

1.1.3 Discrete Tangent, Discrete Curvature: First and Second Differences

Discrete orientation $[f'(t)]$ can be described as:

- Tangent $t_i = p_{i+1} - p_i$ or $t_i = (p_{i+1} - p_{i-1})/2$ with p_i being the points between the linear segments
- Normal $n_i = t_i^\perp$ or $n_i = t_{i+1} - t_i$

Discrete curvature $[f''(t)]$ can be described as:

- Angle (between two segments)
- 3-point osculating circle with radius $1/r$
- $\|p_{i+1} - 2p_i + p_{i-1}\| = \|(p_{i+1} - p_i) - (p_i - p_{i-1})\|$ *(2nd difference)*

1.2 (Discrete) Differential Geometry of Surfaces

Parametric surfaces are defined by a vector-valued parametrization function, that maps a 2D parameter domain to the surface:

$$f : \Omega \subseteq \mathbb{R}^2 \to S \subseteq \mathbb{R}^3, \begin{pmatrix} u \\ v \end{pmatrix} \mapsto \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}$$

An *implicit*/volumetric surface representation is defined to be the zero set of a scalar-valued function:

$$S = \{\mathbf{x} \in \mathbb{R}^3 | F(\mathbf{x}) = 0\} \quad \text{with} \quad F : \mathbb{R}^3 \to \mathbb{R}$$

Geometric Consideration:

- Position $f(u,v)$
- Tangent plane
$$f_u = \frac{\partial f}{\partial u}, \quad f_v = \frac{\partial f}{\partial v}$$
- Curvature
$$f_{uu} = \frac{\partial^2 f}{\partial u^2}, \quad f_{uv} = \frac{\partial^2 f}{\partial u \partial v}, \quad f_{vv} = \frac{\partial^2 f}{\partial v^2}$$

Analytic consideration with Taylor expansion:

$$f(u,v) = f_0 + \underbrace{[f_u, f_v]}_{\mathbb{R}^{3\times 2}} \begin{pmatrix} u \\ v \end{pmatrix} + \frac{1}{2}(u,v) \underbrace{\begin{pmatrix} f_{uu} & f_{uv} \\ f_{vu} & f_{vv} \end{pmatrix}}_{\mathbb{R}^{2\times 2\times 3}} \begin{pmatrix} u \\ v \end{pmatrix} + \cdots$$

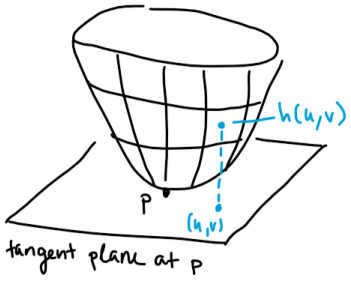
1.2.1 Height Field Parametrization

Simplify setting and assume, that surface is a *height field*, where we map the tangent plane at p to a paraboloid h at p:

$$h(u,v) = \frac{1}{2}(u,v) \underbrace{\begin{pmatrix} h_{uu} & h_{uv} \\ h_{vu} & h_{vv} \end{pmatrix}}_{h_{uv}=h_{vu}} \begin{pmatrix} u \\ v \end{pmatrix}$$

$$\overset{\substack{eigen- \\ decomposition}}{=} \frac{1}{2}(u,v) R \begin{pmatrix} \gamma & 0 \\ 0 & \Gamma \end{pmatrix} R^T \begin{pmatrix} u \\ v \end{pmatrix}$$

$$= \frac{1}{2}(\bar{u},\bar{v}) \begin{pmatrix} \gamma & 0 \\ 0 & \Gamma \end{pmatrix} \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}$$



Eigenvalues γ, Γ determine local curvature behavior, Γ and γ are perpendicular:

- $\gamma, \Gamma > 0$: convex, elliptic
- $\gamma = 0 \vee \Gamma = 0$: cylinder, parabolic
- $\gamma < 0 < \Gamma$: saddle, hyperbolic

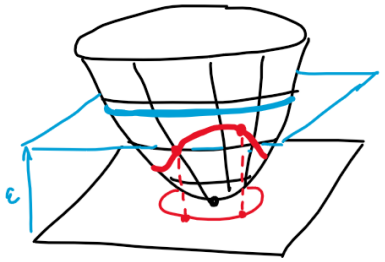
Height over the unit circle (red intersection curve):

$$(u,v) = (\cos\alpha, \sin\alpha)$$

$$\Rightarrow \quad (\cos\alpha, \sin\alpha) \begin{pmatrix} \gamma & 0 \\ 0 & \Gamma \end{pmatrix} \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix}$$

$$= \gamma \cos\alpha^2 + \Gamma \sin\alpha^2$$

Dupin indicatrix:



Constant height (blue intersection curve):

$$(u,v) \begin{pmatrix} \gamma & 0 \\ 0 & \Gamma \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \underbrace{1 = \gamma u^2 + \Gamma v^2}_{\text{quadric}}$$

1.2.2 1st Fundamental Form

Let's look at $[f_u, f_v](u,v)^T$ from the Taylor expansion, which contains the orientation information. We use the *Jacobian matrix* $[f_u, f_v]$ to encode, which *stretches* are occurring in the resulting vector, if we map from the parameter domain to the surface.

Length of resulting vector:

$$\| [f_u, f_v] \begin{pmatrix} u \\ v \end{pmatrix} \|^2 = ([f_u, f_v] \begin{pmatrix} u \\ v \end{pmatrix})^2 = (u,v) \begin{pmatrix} f_u^T \\ f_v^T \end{pmatrix} [f_u, f_v] \begin{pmatrix} u \\ v \end{pmatrix} = (u,v) \boxed{\underbrace{\begin{pmatrix} f_u^T f_u & f_u^T f_v \\ f_v^T f_u & f_v^T f_v \end{pmatrix}}_{\mathbf{I}}} \begin{pmatrix} u \\ v \end{pmatrix}$$

The *first fundamental form* $\mathbf{I}$ is a symmetric matrix, such that we can apply eigendecom-

position:

$$\mathbf{I} = \begin{pmatrix} f_u^T f_u & f_u^T f_v \\ f_v^T f_u & f_v^T f_v \end{pmatrix} = R \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} R^T$$

The expression of the eigenvalues can be found using the characteristic polynomial $\det(\mathbf{I} - \sigma Id)$.

If $\|(u, v)\| = 1 \Rightarrow (u, v) = (\cos\alpha, \sin\alpha)$, then the circle in Ω is mapped to a (stretched) ellipse in S with $\sqrt{\sigma_1}$ being the "shorter" radius and $\sqrt{\sigma_2}$ being the "longer" one. R rotates the half axes vectors of the ellipse to the coordinate axes. The half axes vectors describe the orientation of R or the stretch *(eigenvalues)*.

1. Ideal (*isometric*): $\sigma_1 = \sigma_2 = 1$
2. *Angle preservation*: $\sigma_1 = \sigma_2$
3. *Area preservation*: $\sigma_1 \cdot \sigma_2 = 1$

Surface Area Integral

We can measure the *surface area* A corresponding to the parameter region Ω:

$$A = \int_\Omega \|f_u \times f_v\| d\Omega = \int_\Omega \sqrt{\det(\mathbf{I})} d\Omega$$

Because

$$\begin{aligned} \|f_u \times f_v\|^2 &= \|f_u\|^2 \|f_v\|^2 \sin\alpha^2 \\ &= \|f_u\|^2 \|f_v\|^2 (1 - \cos\alpha^2) \\ &= \|f_u\|^2 \|f_v\|^2 - \|f_u\|^2 \|f_v\|^2 \cos\alpha^2 \\ &= f_u^2 f_v^2 - (f_u^T f_v)^2 \\ &= \det(\mathbf{I}) \end{aligned}$$

1.2.3 2nd Fundamental Form

Generally, the curvature on a surface has two component, namely the *normal curvature* and the *geodesic curvature* (bending within the surface).

Let $\bar{t} = (u_t, v_t)^T$ be the tangent vector at a surface point p in parameter space. The *normal curvature* $\kappa_n(\bar{t})$ at p is the curvature of the planar curve created by intersecting the surface at p with the plane spanned by t and surface normal $n(u, v) = \frac{f_u \times f_v}{\|f_u \times f_v\|}$:

$$\kappa_n(\bar{t}) = \frac{\bar{t}^T \mathbf{II} \bar{t}}{\bar{t}^T \mathbf{I} \bar{t}}$$

where $\mathbf{II}$ the *second fundamental form* defined as

$$\boxed{\mathbf{II} = \begin{pmatrix} n^T f_{uu} & n^T f_{uv} \\ n^T f_{vu} & n^T f_{vv} \end{pmatrix}}$$

The curvature properties of the surface can be characterized by rotating t around the surface normal. In particular, the *bending* behaviour of the surface is described with n_u, n_v.

Formulation in *height field parametrization* we see, that **II** corresponds to $h(u,v)$, we derived previously:

$$f(u,v) = \begin{pmatrix} u \\ v \\ h(u,v) \end{pmatrix}, \quad f_{uu} = \begin{pmatrix} 0 \\ 0 \\ h_{uu} \end{pmatrix}, \quad n = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad \Rightarrow \quad n^T f_{uu} = h_{uu}$$

1.2.4 Discrete Formulae

Per Triangle Stretch

Approximation: Not $\mathcal{O}(h^2)$, but $\mathcal{O}(\frac{1}{n})$!
(Intuition: in linear setting, with cutting h in half results into two smaller segments, in triangular case, cutting h in half results into 4 smaller triangle)

Given tangent, orientation, and normal per triangle and per vertex (weighted average in angle/area/length). Measure the *stretch* by mapping a circle on the initial triangle and measuring the circles' stretch in the resulting triangle. With barycentric coordinates we get

$$p = (a,b,c)\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}, p' = (a',b',c')\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \quad \Rightarrow \quad p' = (a',b',c')(a,b,c)^{-1}p = \begin{pmatrix} L & t \\ 0^T & 1 \end{pmatrix} p$$

Get *distortion* in L using SVD: $L = U\begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} V^T$

Discrete Curvature

- *Mean Curvature*:

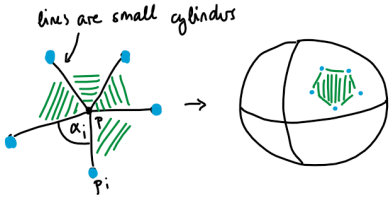
$$H = \frac{1}{2}(\kappa_1 + \kappa_2) = \frac{1}{2}(\gamma + \Gamma) = \frac{1}{2} \cdot \text{trace}(\mathbf{II}) = \frac{1}{2}(h_{uu} + h_{vv})$$

  $\rightarrow$ *Laplace operator*
  (If not in height field setting: $L(f) = \frac{1}{2}(f_{uu} + f_{vv}) = Hn$)
  Discrete version of the Laplace operator, where p_i are the neighbors of p:

$$L = Hn = \sum_i \omega_i (p_i - p)$$

- *Gaussian Curvature*:

$$K = \kappa_1\kappa_2 = \gamma\Gamma = \det(\mathbf{II})$$
$$= h_{uu}h_{vv} - h_{uv}^2 = n^T f_{uu} n^T f_{vv} - (n^T f_{uv})^2$$



Intuition: *Gauss Sphere*
  - 0 curvature: normals are mapped to one point in the Gauss Sphere
  - Cylinder curvature: normals are mapped to an arch in the Gauss Sphere

Gaussian curvature on meshes:

$$K = \frac{1}{A}(2\pi - \sum_i \alpha_i)$$

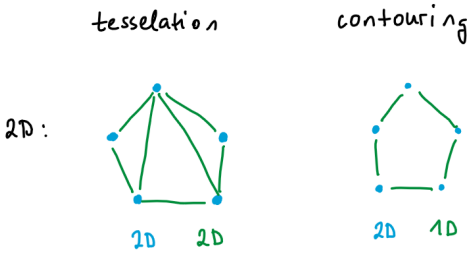
2 Mesh Generation

Let $M = (V, E, F)$ be mesh with vertices $V = \{p_i\}$, edges $E = \{(i,j)\}$, and faces $F = \{(i,j,k,..)\}$ (three tuple in case of triangular faces).
M has the manifold conditions:

1. Surface in a triangle is a manifold.
2. Every edge is shared by exactly two triangles.
3. Every point has an unique circle of neighboring vertices.

Euler: $V - E + F = 2(1-g)$, with g being the genus (number of closed cuts allowed before the surface falls apart into 2 pieces).

2.1 Tessellation and Contouring



The contour problem is ill-defined (not easy to see). (simple solution: more dense sampling)

Meshing Setup

	meshes/points	2D	3D
curves	1D	c ↘	c
surfaces	2D	t ↘	c
volumes	3D	×	t

c: contouring
t: tesselation

2.1.1 2D Tessellations

Let P be a set of points. We associate to each *site* p_i its *Voronoi region* $V(p_i)$ such that

$$V(p_i) = \{x \in \mathbb{R}^d : \|x - p_i\| \leq \|x - p_j\|, \forall_{j \neq i}\}$$

The collection of the nonempty Voronoi regions and their faces, together with their incidence relations, constitute to the *Voronoi diagram* of P.

The dual structure to the Voronoi diagram is the *Delaunay triangulation*. This triangulation is defined such that no point in P is inside the circumcircle of any other triangle in the triangulation *(empty sphere property)*. Delaunay triangulation maximizes the minimum angle of all the angles of the triangles in the triangulation. In 2D, each Delaunay triangle (p, q, r) is dual to a Voronoi vertex where $V(p)$, $V(q)$, and $V(r)$ meet (connect neighboring Voronoi sites).

Looking at two triangles ABD and BCD with the common edge BD, if $\alpha + \gamma \leq 180°$, the triangles meet the *Delaunay condition*. Otherwise, flip the edge, such that BD becomes AC and the triangles meet the Delaunay condition.

Delaunay Refinement

If triangle is too flat, add the so called *Steiner points* (points, than don't belong to the input), which make the triangulation better:

- Insert *center* of the bad triangles circumcircle as another point, thus, the triangle violates the Delaunay property and gets removed.
  $\Rightarrow$ angles $\geq 20°$
- Pick a point on the bad triangles circumcircle (*petal point*), which is the furthest away from the other sample points.
  $\Rightarrow$ angles $\geq 40°$

2.1.2 Contouring

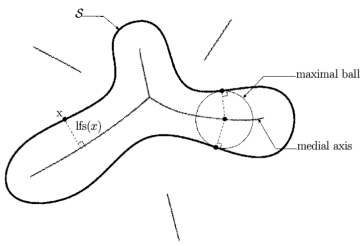
Distance Definitions

$$\underbrace{d(q,p) = \|p - q\|,}_{\text{dist between two points}} \quad \underbrace{d(p,C) = \min_{q \in C} d(p,q)\ ,}_{\text{dist between point and contour}} \quad \underbrace{d(C_1,C_2) = \max_{p \in C_1} \min_{q \in C_2} d(p,q)}_{\substack{\text{Hausdorff distance} \\ \text{symmetric: } \max\{d(C_1,C_2),d(C_2,C_2)\}}}$$

Medial Axis (ME)

The medial axis of an object is the set of all points having more than one closest point on the object's boundary C:

$$MA_C = \{q \in \mathbb{R}^n : |\{p \in C : \|q - p\| = d(q, C)\}| > 1\}$$

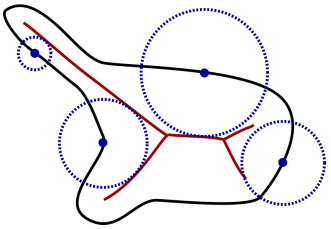


Thus, the medial axis consists of all points which have equal minimal distance to more than one point of the contour.
The medial axis characterizes the "middle" (and outside) of the object, and is used to determine, whether an edge between two vertices is following the contour or crossing the contour.

Local Feature Size (LFS)

The *LFS* is a measure that tells us, what the size of the local geometric structures in our contour is. For $p \in C$:

$$LFS(p) = d(p, MA_C)$$



Points with a small *LFS* are (probably) smaller structures, where we need more samples.

R-Sample

Let $P \subseteq C$ be samples on a contour C. P is called *R-sample* of C if the following condition holds:

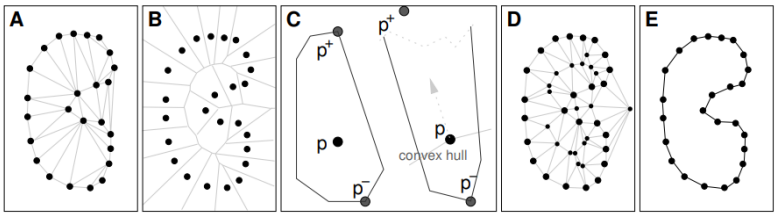
$$\forall_{c \in C} \exists_{p \in P} : LFS(c) \cdot R \geq \|c - p\|$$

2.1.3 Voronoi Filtering (2D)

Input: $\{p_i\}$ assumed to be an R-sample of some unknown contour C.

1. Compute Voronoi diagram to obtain Voronoi vertices $\{v_i\}$ (in 3D: "poles"), which are the points furthest away from sample point. This will give us points in the medial axis.
2. Compute Delaunay triangulation for $\{p_i\} \cup \{v_i\}$.
3. Remove all edges adjacent to some v_i (cave out interior).
4. *(optional)* Clean up, if data is corrupted by noise.

Algorithm guarantees correct extraction of the contour if P is a R-sample with $R \leq 0.06$, where it requires $\mathcal{O}(n^2)$, where $n = |P|$.

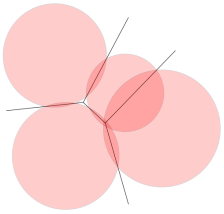


2.1.4 Power Diagram

A *power diagram* is a partition of the euclidean plane into polygonal cells defined from a set of circles. The cell for a given circle C consists of all the points for which the power distance to C is smaller than the power distance to the other circles. The power diagram is a form of generalized Voronoi diagram, and coincides with the Voronoi diagram of the circle centers in the case that all the circles have equal radii.

For samples p_i and radii r_i the distance is defined as:

$$d(p_i, q) = \underbrace{\|q - p_i\|^2}_{d^2} - r^2$$



2.2 Volumentric Mesh Generation

We want to compute an approximative *signed distance function* (SDF) for an object, where the only information required are samples P on the contour. SDF is a function f, that maps a point to its distance from an object's surface, thus, uses the implicit definition of surfaces.

For the distance functions $f(x,y,z) = dist((x,y,z)^T, S)$ we get the properties

signed *(SDF)*: $\begin{cases} f < 0, \text{ inside} \\ f = 0, \text{ on} \\ f > 0, \text{ outside} \end{cases}$ unsigned *(USDF)*: $\begin{cases} f = 0, \text{ on} \\ f \neq 0, \text{ not on} \end{cases}$

Pipelines, red are the complications:

- $\{p_i\} \to \{\pm n_i\} \to \{n_i\} \to$ SDF $\to$ extract mesh
- $\{p_i\} \to \{\pm n_i\} \to$ USDF $\to$ mesh extraction

1) $\{p_i\} \to \{\pm n_i\}$

Associate a normal with each sample. The problems is that the normal is based on the sample's tangent plane, which is not known. Hence, we need to find a way to locally

approximate this plane in order to compute the normal. A plane can be expressed in normal form

$$T_i = \{(x,y,z)^T : n_x x + n_y y + n_z z + d = (n_x, n_y, n_z, d) \cdot (x,y,z,1)^T = 0\}$$

We shift the point cloud's center of gravity into the origin, thus $d = 0$. Therefore, we require

$$(n_x, n_y, n_z) M (n_x, n_y, n_z)^T \to min$$

with tensor for a set of points $(x_i, y_i, z_i)^T$ of length k

$$M = \begin{pmatrix} \sum x_i^2 & \sum x_i y_i & \cdots \\ \sum y_i x_i & \cdots & \cdots \\ \cdots & \cdots & \sum z_i^2 \end{pmatrix}$$

M has three orthogonal eigenvectors v_1, v_2, v_3, which form a basis. Thus, for some α, β, γ the normal n can be written as

$$n = \alpha v_1 + \beta v_2 + \gamma v_3$$

1. *Least Square Plane*: Take a set of neighbors of some point and compute plane, which minimizes the least squared error with respect to the neighbors.
2. *Voronoi Cell Shape*: When a point lies on an edge, the Voronoi cells become skinny, resulting into a good approximation of the normal.

2) $\{\pm n_i\} \to \{n_i\}$

1. *Orientation Propagation*: If points lay close enough, the normal orientation of the neighbor is likely to be the same. Now we can change one normal, and the other propagate through the following relation:

   $$\langle n_i, n_j \rangle > 0, \text{ pick } n_j$$
   $$\langle n_i, n_j \rangle < 0, \text{ pick } -n_j$$

   Choose the version, which spans the smaller angle. To make the technique more reliable we want to control the propagation order:

   $$cost(p_i, p_j) = \alpha \|p_i - p_j\| + \beta(1 - |\langle n_i, n_j \rangle|)$$

   where $\alpha, \beta \in [0,1]$ with $\alpha + \beta = 1$, and *cost* is low, when the points are close to each other and the normals are aligned. Using this cost function we can construct a *minimum spanning tree*.
2. *Binary Labelling Problem*:

   $$consistency(p_i, p_j) = \frac{\langle b_i n_i, b_j n_j \rangle}{\|p_i - p_j\|}$$

   $$\Rightarrow \sum_{(p_i,p_j) \in E} consistency(p_i, p_j) \to \min$$

3.1) $\{p_i\}, \{n_i\} \rightarrow$ SDF

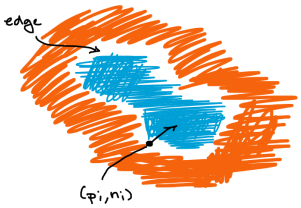
1. (weighted) *Average Distance to Local Tangent Elements*: The SDF value for p is its distance to the tangent plane at p_i:

$$f(p) = (p - p_i) \cdot n_i$$

where p_i is the nearest sample to p, and n_i the normal associated with this sample.

2. *Poisson Reconstruction*: Define characteristic function χ:

$$\chi = \begin{cases} 1, \text{ all points inside} \\ 0, \text{ all points outside} \end{cases}, \quad n_i \sim \nabla(\chi \star G)(p_i)$$



$$\Rightarrow \nabla(\chi \star G) = \int_{p \in S} G \star N(p) dp$$
$$= \sum_{p_i \in N} G(p_i - q) n_i = V$$

$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$ (gradient)

$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ (Laplace)

with N being the normal field, V the vector field, and G a smoothing filter.

Instead of $\nabla f = V$, we can use $\Delta f = div(V)$ with div being divergence (*Poisson Equation*).

3.2) USDF $\rightarrow$ mesh extraction

1. Assume matrix $Q \in \mathbb{R}^{3\times3}$ associated with every point, that somehow measures the local normal information. We want:

$$\nabla f(p_i) \sim \pm n_i$$

Compute

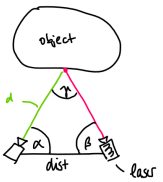
$$\|\nabla f^T \nabla f - Q\| + \lambda \|\delta f\| \rightarrow \min$$

which contains no orientation information in f. $\lambda\|\delta f\|$ for smoothness.

2. *Graph Cut*: Divide space into voxels. For each voxel we assign some probability, that the surface goes through the voxel. From these probabilities we compute the probability for the surface to pass through a voxel edge. These are the weighted edges of the graph. Pick two corners to be the source/sink, where the sink lies inside of the object and the source on the outside.

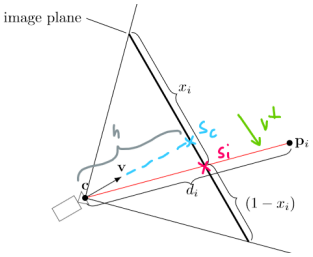
2.3 Mesh Generation from 3D Laser Scanning

Now, one pixel is given as $(r, g, b) + d$ ($d = \|P_C - P_L\| \cdot \frac{\sin\beta}{\sin(\alpha+\beta)}$ is depth information as $(x, y, z)^T$). First, we want to align both coordinate systems using *registration*. Once the scans have been converted into a common coordinate system, the overlapping regions appear more than once; To convert the mesh into a manifold by merging overlaps using *mesh stitching* and *mesh zippering*.



Compute world coordinates from Depth Camera (1D → 2D):

Let $p_i = (x_i, d_i)$, where x_i is the relative position on the image plane and d_i is the depth value, c camera center, v viewing direction, and α opening angle.



1. Define distance from camera to image plane h.
2. Intersection of v with the image plane is $s_c = c + vh$.
3. Intersection of ray from c to p_i is $s_i = s_c + (x_i - 0,5)v^\perp$, where $v^\perp$ switches x and y coordinates and negates new y, initially from v.

$\rightarrow$ $p_i = c + d_i \frac{s_i - c}{\|s_i - c\|}$

2.3.1 Scan Registration

Here we will present a semi-automatic method called *iterative closest points* (ICP). Based on a manual pre-alignment the algorithm identifies point correspondences between both meshes. Based on these correspondences, an optimal rigid transformation (translation, rotation and scaling) between the meshes is computed, which moves them closer together. This procedure is repeated until both meshes converge.

Given are scan $A = \{a_i \in \mathbb{R}^3\}$, and scan $B = \{b_i \in \mathbb{R}^3\}$ such that $a_i \leftrightarrow b_i$. Find the optimal transformation $M = RTS$:

$$\sum_i \|a_i - M(b_i)\|^2 \rightarrow \min$$

1) Translation

Centers of gravity $\bar{a}, \bar{b}$:

$$\bar{a} = \frac{1}{n}\sum_{a\in A} a, \quad \bar{b} = \frac{1}{n}\sum_{b\in B} b$$

$$\Rightarrow \boxed{T = \bar{a} - \bar{b}}$$

2) Scaling

Sizes:

$$S_a = \sum_{a\in A}(a - \bar{a})^2, \quad S_b = \sum_{b\in B}(b - \bar{b})^2$$

$$\Rightarrow \boxed{S = \frac{S_a}{S_b}}$$

3) Rotation

3.1) Quaternions q

(Complex Numbers)

$\mathbb{C} : c = a + \hat{i}b \rightarrow (a, b)^T \in \mathbb{R}^2$

$$(a_1, b_1) \cdot (a_2, b_2) = \begin{pmatrix} a_2 & -b_2 \\ b_2 & a_2 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \end{pmatrix}$$

(Unit Complex Numbers)

Kills one degree of freedom.

$$\|c\| = 1 = a^2 + b^2 \Rightarrow a = \cos\phi, \quad b = \sin\phi$$

$$\begin{pmatrix} a_2 & -b_2 \\ b_2 & a_2 \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

$\Rightarrow$ 2D rotation!

(Hyper Complex Numbers)

$\mathbb{H} : p = p_0 + \hat{i}p_1 + \hat{j}p_2 + \hat{k}p_3$, $\hat{i}^2 = \hat{j}^2 = \hat{k}^2 = -1$, $\hat{i}\hat{j}\hat{k} = -1$

$$\begin{aligned} &(p_0, p_1, p_2, p_3) \cdot (q_0, q_1, q_2, q_3) \\ &= \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} \\ &= \begin{pmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & p_3 & p_2 \\ p_2 & -p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} \end{aligned}$$

(Unit Hyper Complex Numbers)

$\|p\| = p_0^2 + p_1^2 + p_2^2 + p_3^2$, results into the rotations in 3D:

$$\boxed{R(p) = q \cdot p \cdot \bar{q} = \underbrace{M_L(q) M_R(q)}_{\text{rotation matrix}} p}$$

$$q = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2}N_x, \sin\frac{\theta}{2}N_y, \sin\frac{\theta}{2}N_z)$$
$$N_y^2 + N_x^2 + N_z^2 = 1$$

with $\bar{q} = q_0 - iq_1 - jq_2 - kq_3$ being the conjugated of q.

Interesting observation:

$$\langle a, qb\bar{q} \rangle = \langle aq, qb \rangle$$

Note, that multiplication is not symmetric:

$$p \cdot q = M_L(p) \cdot q = M_R(q) \cdot p$$

Optimal rotation q is obtained by the eigenvector of M with max. eigenvalue.

$$\sum \|a_i - R(b_i)\|^2 \rightarrow \min \iff q^T (\sum M_L(a_i)^T M_R(b_i)) q = \boxed{q^T \underbrace{M}_{\text{symmetric}} q \rightarrow \max}$$

where $M_L(p)^T = M_L(\bar{p})$ and $M_R(q)^T = M_R(\bar{q})$.

1. Determine characteristic polynomial: $\chi_M(\lambda) = \det(M - \lambda I)$.
2. Compute max. eigenvalue λ_{max}.
3. Compute q out of $(M - \lambda_{max} I)q = 0$.

Iterative Closest Point Algorithm (ICP)

```
do
    sample scan A -> {a_i}
    find closest points {b_i}
    filter correspondences
    compute optimal T, R
    transform scan B by RT(b_i)
until both meshes converge
```

2.4 Mesh Repair, Mesh Healing

2.4.1 Surface Oriented Approaches

These algorithms operate directly on the input data and try to explicitly identify and resolve artifacts on the surface.

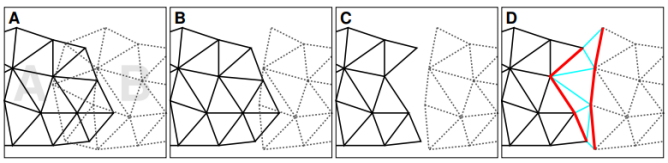
Pro and Cons

+ structure preserving, minimal modification of the input	- needs user interaction, because most of the time, input doesn't satisfy quality requirements
	- doesn't resolve intersections, large overlaps, or gaps
	- no guarantee on output quality

Input two overlapping scans A, B.

Zippering: Shoot normals from each vertex from one mesh, and remove overlapping polygons on the other, switch roles and repeat. Walk along the boundary polygons of A and B and connects their vertices with new edges

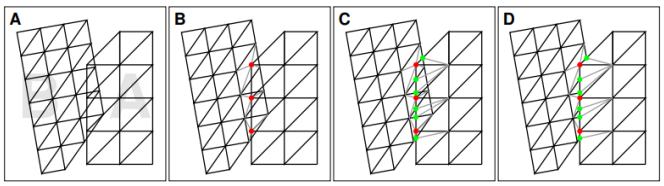
Problem: We are throwing away geometry!



Stitching: Project the boundary edges of one boundary into the other scan. Insert boundary polygon of the one mesh into the other mesh. Cut away overhanging part. (Maybe retriangualtion for refinement as last step.)

1. *Vertex Insertion*: Project the boundary vertices of A into the polygons of B.
2. *Edge Insertion*: The boundary edges of A are also integrated into the triangulation. If a boundary edge of A crosses an edge of B, an additional vertex is required at the intersection of the two edges to resolve this non-manifold configuration.
3. *Manifold Extraction*: Erase the overlapping part of B.

Only removes overlapping parts of one patch *(non-symmetrical)*



With the above two methods, some of the created triangles are badly shaped or small. The vertices of such triangles are deleted from the mesh, and the hole is filled using a triangulation scheme of choice.

These methods are difficult to implement and not very robust. Alternative:

1. Preclassify all vertices, on which site they lie.
2. Look for points on line which correspond to a sign change.
3. Triangulate with "marching" triangle.

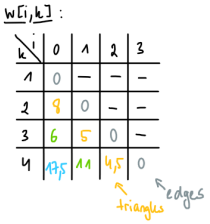
Example Algorithm for Hole Filling *(Peter Liepa)*

1. Compute a coarse triangulation T (of minimal weight $w(T)$) to fill a hole (hole has to have a clean contour).
We favour triangulations of *low area* and *low normal variation*.

Recursive formula:

a) Compute areas from the triangles: Let a, b the spanning vectors. Then $\boxed{A = \frac{1}{2}\|a \times b\|}$. If $a, b \in \mathbb{R}^2$, then last coordinate is 0, or $A = \frac{1}{2}|\det(a \quad b)|$.

b) Set up table for cost of best triangulation, as pictured on the right.



$$w[x, x+1] = 0$$
$$w[a, c] = \min_{a<b<c} w[a, b] + A(a, b, c) + w[b, c]$$

c) Use min. term from $w[0, n]$ to find triangulation.

Dynamic programming leads to an $\mathcal{O}(n^3)$ algorithm.

2. Refine the triangulation $T \to T'$ to match the vertex, densities of the surrounding area.
$\to$ remeshing

3. Smooth the triangulation T' to match the geometry of the surrounding.
$\to$ mesh smoothing

2.4.2 Volume Oriented Approaches

These algorithms convert the input model into an intermediate volumetric representation from which the output model is then extracted.

Pro and Cons

+ fully automatic
+ produces guaranteed *watertight* models

- volumetric representation don't allow for artifacts like intersections, holes, gaps, overlaps, or inconsistent normal orientations
- resampling often introduces aliasing artifacts and loss of model features
- memory-intensive, hard to run at very high resolutions

Example Algorithm

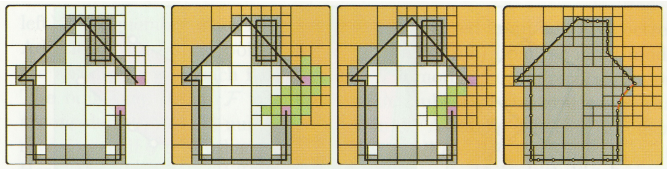
Given: input model M, maximum approx. tolerance ε, maximum hole/gap size ρ

Find: watertight, *manifold* model R with

- $dist(M, R) < \varepsilon$ *(error tolerance)*
- $dist(R, M) < \rho$ *(maximum diameter up to which gaps should be closed)*
- $dist(R, M) > \varepsilon \Rightarrow$ boundary of M
- faithful normal reconstruction

Algorithm:

1. Rasterize input on an *adaptive octree*. Every triangle is voxelized by throwing it into an octree, where each cell stores the triangles intersecting with it. From these triangles a feature-sensitive sample point can be computed for each cell.
2. *Morphological Operations*:
   a) Set maximum depth of the octree cells such that the diameter of the finest-level cells is smaller than ε. Cells that are not yet maximum depth are recursively split if they contain a boundary edge.
   b) *Dilate* all leaf cells that contain a boundary, which fills the gaps. *Erode* the dilated extra geometry from the boundary edge.
3. Smooth out the surface.



2.4.3 Hybrid Representations

voxel grid... simple topology, triangle mesh... best available geometry.

3 Geometric Mesh Optimization

We have two options to modify/improve a mesh: On the one hand we can change the positions of the vertices p_i *(geometry)*. On the other hand we can change the mesh connectivity through triangles T_j *(topology)*.

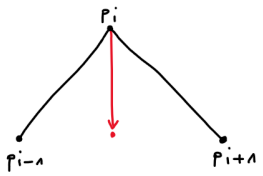
Our first optimization goal is to remove measurement noise while preserving important features of the meshes like sharp edges and corners or fine detail.

3.0.1 Smoothing (2D)

We distinguish two different smoothing directions: Smoothing in *normal direction* is generally of most interest when the goal is to remove noise. The vertex distribution, i.e., the density of vertices on the surface is not changed. Smoothing in *tangent direction* behaves the opposite way. Its main application is to change the vertex distribution to, e.g., improve the shape of triangles.

Update $p_i \to p_i + \Delta p_i$ with computed *update vectors* for some α, β, such that it moves points to the c.o.g. of their neighbors:

$$\begin{aligned}\Delta p_i &= \alpha(p_{i+1} - p_i) + \beta(p_{i-1} - p_i) \\ &= \frac{1}{2}(p_{i+1} - p_i) + \frac{1}{2}(p_{i-1} - p_i) \\ &= \frac{1}{2}(p_{i+1} - p_i) - \frac{1}{2}(p_i - p_{i-1})\end{aligned}$$



$\to$ second derivative of a discrete function
With the Taylor expansion one can show, that the smoothing operator moves p_i in direction which *reduces* curvature.

Approach for Drawing:

- For λ, draw center of gravity between the two neighboring points p_{i+1}, p_{i-1}.
- Cut the vector from point p_i to cog (Δp_i) by λ.
- For restriction on normal/tangential direction, project Δp_i on normal with $n^T \Delta p_i n$ or on tangent with $\Delta p_i - n^T \Delta p_i n$.

Smoothing as Low-pass Filtering of Polygons

All vertices of the polygon are stacked upon each other to form a vector denoted by $[p_i]$:

$$p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \rightarrow [p_i] = \begin{pmatrix} x_0 & y_0 \\ x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}$$

Using this notation, write the smoothing operator as

$$[p_i] \leftarrow [p_i] + \frac{\lambda}{2} \underbrace{\begin{pmatrix} -2 & 1 & & & & \\ & \ddots & & & & \\ & 1 & -2 & 1 & & \\ & & 1 & -2 & 1 & \\ & & & \ddots & & \\ & & & & 1 & -2 \end{pmatrix}}_{U} [p_i]$$

$$= (I + \lambda U)[p_i]$$

with *damping factor* λ to avoid oscillating behaviour.

The smoothing operator U approximates second derivatives, i.e., $\boxed{U \approx \frac{d^2}{dt^2}}$ *(Laplace operator)*.

To gain insight into the properties of the linear transformation, we compute eigenvectors e_i and eigenvalues λ_i of U. $[p_i]$ then can be expressed as a basis of eigenvectors:

$$[p_i] = \sum_j e_j c_j$$

Describe frequency f as a combination of sin, cos to get coefficients c_i:

$$f(x) = \sum c_i \cdot \cos(ix)$$

Apply smoothing operator U:

$$U[p_i] = \sum_j U e_j c_j = \sum_j \lambda_i e_j c_j$$
$$U^k[p_i] = \sum_j \lambda_i^k e_j c_j$$

The *eigenfunctions* of U are sin and cos:

$$[sin(\omega x)]'' = -\omega^2 sin(\omega x)$$
$$[cos(\omega x)]'' = -\omega^2 cos(\omega x)$$

$\rightarrow$ corresponds to transformation into frequency domain *(Fourier transform)*

Formulation into frequency domain:

$$[p_i] \circ\!\!-\!\!\bullet [c_i], \quad U[p_i] \circ\!\!-\!\!\bullet [\lambda_i c_i], \quad (I + \lambda U)[p_i] \circ\!\!-\!\!\bullet [(1 + \lambda\lambda_i)c_i]$$

A single application of the operator corresponds to a simple linear filter which reduces high frequencies and preserves low frequencies. Hence, this smoothing operator $(1 + \lambda U)$ corresponds to a *low-pass filter*.

Design a Smoothing Operator

Construct an extended filter that not only preserves the very low, but also the mid frequencies. Given $[p_i]$, apply a simple smoothing operator $[q_i] := (I + \lambda U)[p_i]$. Now, $[q_i]$ contains just the low frequencies of the original input. We obtain *mid frequencies* by first get the high frequencies by subtraction, and then filter the high frequencies:

$$[\bar{q}_i] = (I + \lambda U)([p_i] - [q_i]) = (I + \lambda U)(-\lambda U)[p_i]$$

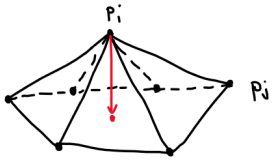
To get *low and mid frequencies*:

$$\begin{aligned}[q_i] + [\bar{q}_i] &= ((I + \lambda U) + (I + \lambda U)(-\lambda U))\,[p_i] \\ &= (I - \lambda^2 U^2)[p_i] \\ &= (I - \lambda U)(I + \lambda U)[p_i]\end{aligned}$$

3.0.2 Smoothing (3D)

Extend the intuitive smoothing operator by moving vertex p_i to the c.o.g. of its neighbors to triangle meshes:

$$\begin{aligned}p_i \leftarrow p_i + \Delta p_i &= p_i + \underbrace{\frac{1}{\sum_j w_{ij}} \sum_{j \in N_1(i)} w_{ij}(p_j - p_i)}_{\text{umbrella operator}} \\ &= \frac{1}{\sum_j w_{ij}} \sum_{j \in N_1(i)} w_{ij} p_j - p_i\end{aligned}$$



where N_1 is 1-ring neighborhood and w edge weights.

Properties for new weights:

- *Linear Precision*: Operator doesn't move the center vertex of a planar (i.e. linear) 1-ring configuration.
- *Locality*: Updating a vertex only depends on the neighboring vertices.
- *Symmetric*: $w_{ij} = w_{ji}$
- *Positive*: $w_{ij} \geq 0$

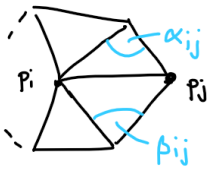
Uniform weights:

$$\boxed{w_{ij} = \frac{1}{|N_1(i)|}}$$

Properties: locality, symmetric, positive

cot**-weights:**

The smoothing operator with cot-weights performs normal smoothing by construction and has almost no tangent component.



$$\boxed{w_{ij} = \cot(\alpha_{ij}) + \cot(\beta_{ij})}$$

Properties: linear precision, locality, symmetric, not always positive (for $\alpha_{ij} + \beta_{ij} < \pi$)

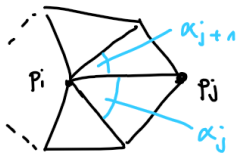
Delaunay cot-weights:

When using Delaunay triangulation, $\alpha_{ij} + \beta_{ij} < \pi$, which makes the cot-weights *negative*, never happens. Therefore, flip edge, if we come across a negative weight.

Properties: linear precision, symmetric, positive

Mean value weights:

"mean value coords" (MVC):



$$\boxed{w_{ij} = \frac{\tan(\alpha_j) - \tan(\alpha_{j+1})}{\|p_j - p_i\|}}$$

Properties: linear precision, locality, positive

Physical Interpretation

The application of the smoothing operator $(I + \lambda U)$ is defined as an iterative process, which converges when the length of the update vectors Δp_i approaches zero ($U[p_i] \to 0$). Because U corresponds to the Laplace of a function $f(u, v)$ interpolating p_i and its 1-ring vertices, we obtain the *Laplace equation*

$$\Delta f = f_{uu} + f_{vv} = 0$$

The physical interpretation of the Laplace equation is that of a *membrane surface* with a C^0 boundary condition (i.e. it only interpolates the boundary).

For triangle meshes and a bi-variate surface function $f(u, v)$, the operator $(I + \lambda U)(I - \lambda U)$ corresponds to squared Laplace $\Delta^2 f$. In case of convergence of the update vectors $U^2[p_i] \to 0$ this yields

$$\Delta^2 f = f_{uuuu} + 2f_{uuvv} + f_{vvvv} = 0$$

Now the physical interpretation of the above equation is that of a *thin plate* with a C^1 boundary condition (i.e. it interpolates the boundary and the tangents at the boundary).

Error Control

ε-Balls (isotropic method)
Laser scanners usually have a certain known error tolerance ε. To remove the measurement noise, each vertex has to be moved by a distance of at most ε, resulting into *balls* of radius ε. Then the smoothing process is allowed to move each vertex in its respective ball only.

$$p_i \leftarrow p_i^* + \frac{\varepsilon}{\|p_i - p_i^*\|}(p_i - p_i^*) \quad \text{if} \quad \|p_i - p_i^*\| > \varepsilon$$

where p_i^* is the original position. It doesn't allow for a regularization of the vertex distribution by tangent smoothing since this usually requires larger movements than the ε-balls permit.

Error-Slabs (anisotropic method)
Allow to move by a distance of ε in the normal direction of the surface, but by an arbitrary distance on tangent direction. This leads to the definition of slabs around each vertex, constructed around the original positions p_i^* and perpendicular to the known surface normal n_i^*:

$$p_i \leftarrow p_i^* + \frac{\varepsilon}{|{n_i^*}^T(p_i - p_i^*)|}(p_i - p_i^*) \quad \text{if} \quad |{n_i^*}^T(p_i - p_i^*)| > \varepsilon$$

3.1 Surface Modeling

3.1.1 Free-from Modeling

Assume we have a mesh with a hole, which is filled used the linear system $\Delta^2 p = 0$, and a *handle* in the filled hole. Now, if we move/rotate the handle, the filled surface is smoothed accordingly. The boundary condition of the handle and the mesh can differ. This method is used for user manipulation of the surface in real time.

The user controls the deformation by prescribing displacements for a set of *handle points* $h_i \in H \subset S$, and by constraining certain parts $F \subset S$ (e.g., outer boundaries) to stay *fixed* during the deformation.

Let update vector be a Laplacian smoothing operation with $\Delta p_i = Lp$. For bi-Laplacian $L^2 p = 0$, p free/unknown points, f fixed points, and h handle points given in similar form as seen on page 21, where the diagonal entry for fixed and handle points is 1:

$$\underbrace{\left(\begin{array}{c|c} L^2 & \\ \hline 0 & I \end{array}\right)}_{M} \begin{pmatrix} p \\ f \\ h \end{pmatrix} = \begin{pmatrix} 0 \\ f \\ h \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} p \\ f \\ h \end{pmatrix} = M^{-1} \begin{pmatrix} 0 \\ f \\ h \end{pmatrix} = M^{-1} \begin{pmatrix} 0 \\ f \\ 0 \end{pmatrix} + M^{-1} \begin{pmatrix} 0 \\ 0 \\ h \end{pmatrix}$$

All the vertices in the handle can be represented as an affine combination consisting of 4 points a, b, c, d and a matrix $Q \in \mathbb{R}^{k\times 4}$:

$$h = Q(a, b, c, d)^T$$

Restrict handle control to *affine transform* T:

$$T(h) = QT((a, b, c, d)^T)$$

$$\Rightarrow \boxed{p = \tilde{M}^{-1} f + \tilde{\tilde{M}}^{-1} Q(a, b, c, d)^T}$$

Computation of detail preserving free-form modeling:

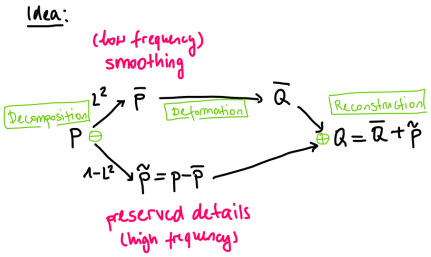
1. Smooth points without handle transformation *(not updated smoothing)*, and smooth points with handle transformation *(updated smoothing)*.
2. Go over each point and compute output points:
   feature = point - not updated smooth point
   output point = updated smooth point + feature

3.1.2 Multi-Resolution Modeling

Classical Multi-Resolution Modeling

We want to preserve details, if applying global modification.

The main idea is to decompose the object into two frequency bands, where low frequencies correspond to the smooth global shape, while the high frequencies correspond to fine-scale details.



Restrict detail representation to *normal displacements* only. Represent the high frequency details in regard to a local plane (*local frames*) to achieve the wanted behaviour of simultaneously bended normals. Then a vector h_i is a weighted sum of a normal vector and two tangent vectors of the base surface $\bar{P}$ regarding $\bar{p}_i$.

Formulate as simple bi-Laplacian smoothing problem on the encoded smoothed mesh P and the deformed mesh Q:

$$\Delta^2 \bar{P} = 0, \quad \boxed{\Delta^2 \bar{Q} = 0}$$

Let's look at how the shapes differ after modification:

$$\Delta^2(Q - P) = \Delta^2((\bar{Q} + \tilde{P}) - (\tilde{P} + \bar{P})) = \Delta^2(\bar{Q} - \bar{P}) = \boxed{\Delta^2 \bar{Q} - \Delta^2 \bar{P} = 0} \text{ (Laplace system)}$$

Therefore, we are no longer generate smooth surfaces, but we generate *smooth deformations*. The key difficulty is here to implement the $+$ operator in $P = \bar{P} + \tilde{P}$ and $Q = \bar{Q} + \tilde{P}$.

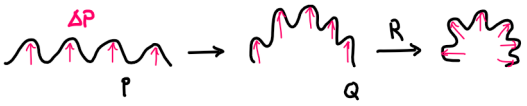
Poisson Editing

Avoid decomposition into the separate components. Think of ΔP to encode detail information. Modify surface P into a surface Q. If $\Delta Q \approx \Delta P$, then we have some sort of *detail preservation*.

$$\begin{aligned} & \Delta Q \approx \Delta P \\ \Rightarrow \quad & \|\Delta Q - \Delta P\|^2 \to \min, \text{ subject to } constr(Q) = 0 \\ \Rightarrow \quad & \|\Delta(Q - P)\|^2 \to \min \\ \Rightarrow \quad & \Delta^2(Q - P) = 0 \\ \Rightarrow \quad & \boxed{\Delta^2 Q = \Delta^2 P} \textit{ (Poisson system)} \end{aligned}$$

For a more natural bending we need to minimize

$$\|\Delta Q - R\Delta P\|^2 \to \min$$



with R being a rotation.

4 Topological Mesh Optimization

Given $M = (V, E, F)$ with n vertices, find $M' = (V', E', F')$ with n' vertices.

Parameters in this decimation:

a) Prescribe target resolution n' *(Target Complexity)*:

$$dist(M', M) \to \min$$

b) Prescribe error tolerance $dist(M', M) < \varepsilon$ *(Global Error)*:

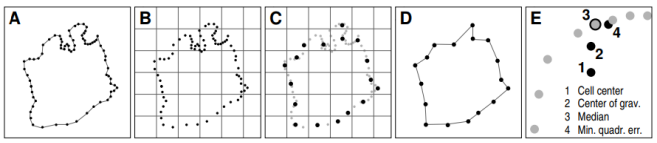
$$n' \to \min$$

Overview:

	Global Error	Target Complexity
Vertex Clustering	✓	×
Remeshing	×	✓
Incremental Decimation	✓	✓

4.1 Vertex Clustering

The idea behind vertex clustering is to cluster vertices in a mesh based on proximity, and to replace them with a single *representative*.



The clustering is done using a regular 3D voxel grid.

There are several choices for the representative:

- *Voxel center*
  - \+ computationally efficient
  - \- high decimation error, poor quality
- *Center of gravity*
  - \+ based on actual positions
  - \- low-pass filter: low quality
- *Medoid/Median*
  - \+ less pronounced low-pass filter
  - \- not optimal method
- ***Error Quadric***
  - \+ features in the mesh are retained: high quality

Iff all vertices of a triangle are located in three *different* cells, a triangle record is created which links to these cells.

Pros and Cons

\+ computationally efficient
\- uniform sampling density: one representative regardless of how many vertices fall into a voxel
\- is not guaranteed to produce manifold meshes

4.1.1 Error Quadric

Using *error quadrics*, a position for the representative minimizing the squared distances to all planes spanned by the triangles in the cell is chosen.

Recall plane equation:

$$P = \{(x,y,z)^T : (n_x, n_y, n_z, d) \cdot (x,y,z,1)^T = 0\}$$

with d being the perpendicular distance between plane and coordinate origin, and $n_x^2 + n_y^2 + n_x^2 = 1$. The plane containing a triangle $\triangle(a,b,c)$ can be constructed as $n = (b-a) \times (c-a)$ and $d = -n^T a$. For a 2D plane equation, use $d = n^T p$ for some point p, which lies on the plane. Then $N = (n, -d)$.

The distance between a point and a plane is $dist(p,N) = (n_x, n_y, n_z, d) \cdot (x,y,z,1)^T$.

With this, we can express the sum of squared distances of a point p to a set of planes Φ:

$$d(p,\Phi) = \sum_{N\in\Phi} (N^T p)^2 = \sum_{N\in\Phi} p^T (NN^T) p = p^T \cdot \left(\sum_{N\in\Phi} NN^T \right) \cdot p$$
$$= \boxed{p^T \cdot \left(\sum_{N\in\Phi} Q_N \right) \cdot p} = p^T \cdot Q_\Phi \cdot p$$

The symmetric matrix Q_N is called *fundamental error quadric* of plane N. Find a point x_{opt} that minimizes the squared distances to all planes:

$$x_{opt}^T Q_\Phi x_{opt} \to \min$$

x_{opt} is the eigenvector to the minimal eigenvalue of Q_Φ. A more direct method of solving this optimization problem:

$$Q = \begin{pmatrix} A & l \\ l^T & m \end{pmatrix} \quad \Rightarrow \quad \boxed{Ax_{opt} = -l}$$

with $A \in \mathbb{R}^{3\times3}, l \in \mathbb{R}^3, m \in \mathbb{R}$.

N doesn't have full rang in the most cases. Rather use the following method, than using the 4×4 equation system: Decompose $N = U\Sigma V^T$ with SVD, and use it to compute $N^* = V\Sigma^* U^T$, which is the (pseudo-)inverse of matrix N, if inverse Σ^* exists (it does, if N has full rank, otherwise take over zeros in the diagonal of Σ to Σ^*), and use this N^* for NN^T.

4.2 Incremental Decimation

Incremental decimation describes a class of decimation algorithms which share the same greedy approach: iteratively select and delete two triangles at a time, starting with the deletion causing the smallest error. Such approaches are based on two components: a *priority queue* of entities (i.e. vertices or edges) that can potentially be deleted from the mesh, and a *local decimation operator* which performs the deletion.

General Setup:

```
repeat:
    pick mesh region
    apply decimation operator
until no further reduction possible
```

Greedy Optimization:

```
for each region
    evaluate quality after decimation
    enqueue(quality, region)
repeat:
    pick best mesh region
    apply decimation operator
    update queue
until not further reduction possible
```

Global Error Control:

```
for each region
    evaluate quality after decimation
    enqueue(quality, region)
repeat:
    pick best mesh region
    if error < epsilon
        apply decimation operator
        update queue
until no further reduction possible
```

Multiple Choice Approximation:

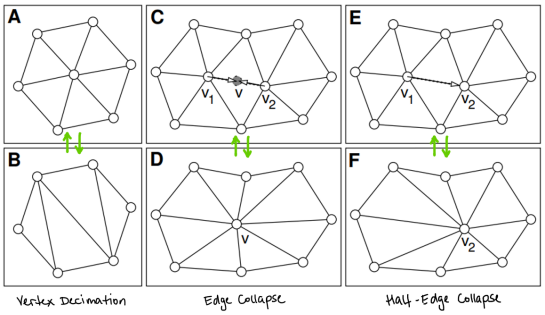
```
repeat:
    pick best mesh region (among k candidates)
    if error < epsilon
        apply decimation operator
until no further reduction possible
```

The priority queue is created in such a way that operations which cause the smallest error are performed first. In order to measure the error introduced by a certain decimation operation, we will discuss several *global* and *local error metrics* on triangle meshes. In addition to the priority queue which orders all possible decimation operations according to the error they cause, one often uses one or more binary validity checks called *fairness criteria*.

4.2.1 Decimation Operators

Vertex Decimation: remove vertex and retriangulate its one ring (combinatorial DOF: *subsampling* approach)

Edge Collapse: merge two adjacent triangles into a new position (continuous DOF: *filtering* approach)



Half-Edge Collapse: merge two adjacent vertices into one of them (special vertex removal, special edge collapse) (no DOFs: *subsampling* approach)

4.2.2 Error Metrics

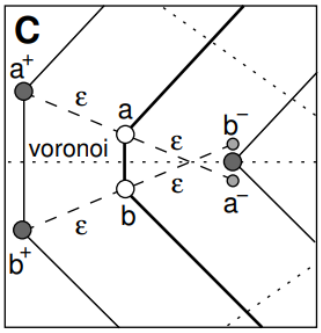
In the specific case of incremental decimation, we use *error metrics* to rate decimation operations in order to find the one which causes the smallest error.

Global Error Metrics

Global error metrics measure the error between two meshes. In the context of mesh decimation, one of the meshes is the original mesh, and one the decimated mesh.

1. *(directed) Hausdorff distance* (see page 10)

2. *Simplification Envelopes*:
   - A given vertex v of the mesh M is offset into normal direction by an amount of ε (yielding v^+) and $-\varepsilon$ (yielding v^-). ($\rightarrow$ *ε-envelope*)
   - To avoid self intersections of the envelope, check if either v^+ or v^- lie within the Voronoi cell of a different vertex than v. If so, correct the position of the corresponding new vertex to be the closest point in the own Voronoi cell.
   - Volume between the meshes is the *simplification envelope* of M.



3. *Error Quadrics* (see page 27): The basic idea is to associate one quadric with each vertex and initialize them with the tangent planes of the incident triangles. To evaluate the (edge- or halfedge-) contraction of v_1 and v_2, the corresponding quadrics Q_1 and Q_2 are added to yield the error quadric $Q = Q_1 + Q_2$ of the resulting configuration after the operation.

   \+ very efficient
   \- measures distance to *tangent planes* of triangles: tangent planes can still lie arbitrarily far away from the corresponding triangles of the mesh

Local Error Metrics

1. *Local Plane Distance*:
   - To measure the error of a decimation operation, we first fit a plane to the involved vertices in least squares sense.
   - Then, the error of the new vertex is computed as its distance to the least squares plane.

2. *Volume Preservation*:
   - Consider triangle $\triangle(a,b,c)$. Move a to a', then the tetrahedron $\triangle_3(a,a',b,c)$ encloses a certain volume.
   - Edge collapse $v_1, v_2 \rightarrow v$: All triangles incident to either v_1 or v_2 span a tetrahedron. If v_1 or v_2 are moved outward (pos. normal direction), the volumes are *positive*. If one is moved inward (neg. normal direction) the volumes are *negative*.
   - Sum up all volume changes caused by a decimation step

   \- not able to measure surface distances

4.2.3 Fairness Criteria

These criteria simply check if a certain decimation operation is valid or not, e.g., if one of the angles of a resulting triangle will be too small, using binary checks:

- *Triangle shape:* Ideally, a decimation operation should lead to *equilateral triangles*, hence, use minimal maximum triangle angle as metric (relation between $\frac{r_1}{l_1}$ and $\frac{r_2}{l_2}$).
- *Normal approximation* Visual quality of a mesh strongly depends on the alignment of the triangle normals with the normals of the underlying surface.
- *Valence balance:* Ideally, the decimated mesh should be *regular* (vertex valence of 6 for all vertices).
- *Texture distortion:* If the mesh is equipped with a texture, the distortion introduced by the decimation should be as small as possible. The amount of distortion can be measured by the difference between surface triangle and associated texture triangle.
- approx. error
- dihedral angles
- color difference

4.3 Progressive Meshes

In most applications it is much better to have a preview of the mesh very early during the transmission, when the transmission takes a noticeably time interval to be fully transmitted. *Progressive Transmission* is designed for this purpose:

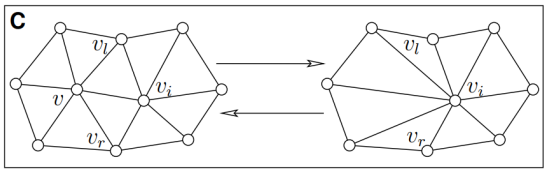
1. A very coarse version of the mesh is displayed consisting of only very few triangles.
2. Refinement steps are performed, starting with those adding the most detail, until finally all detail in the original mesh has been reconstructed

The sender decimates the mesh one step at a time. All parameters determining a step are stored in a list $D = \{d_0, \ldots, d_{n-1}\}$. Let M' denote a very coarse version of the mesh obtained after n decimation steps. The data is transmitted in *reverse order*, where M can be reconstructed by applying the inverse decimation operation d^{-1}:

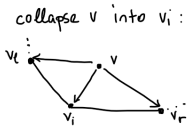
$$M' \rightarrow d_{n-1} \rightarrow \ldots \rightarrow d_0$$

When performing inverse of the half-edge collapse, the *vertex split*, store

$$v : (v_i, v_l, v_r) \rightarrow [x, y, z]$$



where v_i is the vertex we collapsed to, v_l and v_r the vertices opposite to the edge $\overline{vv_i}$, and $[x, y, z]$ the original position of v.



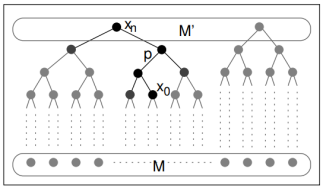
In the case of the more general edge collapse also the target vertex v_i is moved: We need to store an additional offset vector which moves v_i back to its original position during the vertex split.

Selective Refinement

In some cases one does not want to reconstruct the whole mesh but just a specific region (e.g., zoomed in). It can be advantageous to only render those parts directly visible in full detail and the rest in a much coarser resolution.

The main question therefore is which vertices v_a and v_b of the current mesh are used to generate two triangles with the newly generated vertices v_1 and v_2.

Algorithm is based on the *binary forest* of decimation operations. The vertices of the coarse mesh M' are the *roots* in the forest, whereas the vertices of the original mesh M are the *leafs*. For a vertex v represented as a node in the tree, its parent is the vertex into which v contracts, and its children are the vertices which contract into v.



Store collapses

$$v : (v_i, v_l^*, v_r^*) \rightarrow [x, y, z]$$

with *fundamental cut vertices* v_l^*, v_r^*, which are vertices of M.

To find the vertices v_a and v_b the algorithm traverses the binary forests of decimation operations from v_l^* and v_r^* (which are both leaf nodes) up towards the root nodes until the first *active* vertices is found. These vertices are then used to complete the vertex split.

To find the fundamental cut vertices for a certain edge collapse, track ancestors for the vertices of each face during decimation phase. For each face we store 3 ancestor vertices, one for each of the 3 face vertices. The faces of the original mesh are initialized with the original vertices. When performing an edge collapse, the required fundamental cut vertices can be looked up in the two triangles that are about to vanish.

4.4 Remeshing

Remeshing is the most general approach to modify the topology of a mesh. We distinguish two different variants of remeshing techniques, *isotropic* versus *anisotropic* remeshing.

+ ability to generate the meshes of the highest quality among all algorithms discussed previously
- normally require the most effort in comparison with other decimation techniques

4.4.1 Isotropic Remeshing

The goal of *isotropic* techniques is to generate triangulation with as equilateral triangles as possible:

- equal edge length
  - remove too short edges ($\rightarrow$ *edge collapse*)
  - remove too long edges ($\rightarrow$ *2-4 edge split*)
- regular valences: valence balance ($\rightarrow$ *edge flip*)
- uniform vertex distribution: tangential smoothing ($\rightarrow$ *Laplace operator*)

Re-tiling

Approach is a simpler extension of the planar Delaunay triangulation and the dual Voronoi diagram to surfaces in 3D.

Algorithm:

1. Evenly distribute a specified number of vertices v_k on the original surface.
   - *Lloyd's algorithm*: Grow Voronoi cells around each vertex v_k. Move each vertex v_k to the center of its Voronoi cell
2. Compute Voronoi cells on the surface around the vertices v_k.
   - The cell of a vertex v_k is supposed to contain all vertices p_j which have a smaller distance to v_k than to any other vertex. Use *geodesic distance*.
   - Starting with v_k let the cells grow until every point is assigned to exactly one cell.
3. Connect vertices of neighboring cells to obtain the new triangulation.

Incremental Remeshing

Algorithm:

1. Specify target edge length L
2. ***Edge split:*** Split all edges longer than $L_{max} := L + \varepsilon = \frac{4}{3}L$. This erases 2 old triangles and creates 4 new ones ($\rightarrow$ *2-to-4 split*).
3. ***Edge collapse:*** Collapse all edges shorter than $L_{min} := L - \varepsilon = \frac{4}{5}L$.
4. ***Edge flip:*** Flip edges to promote valence of 6

$$\sum_i |v_i - 6|^2 \rightarrow min$$

5. ***Tangential smoothing:*** Relax vertex positions by tangential smoothing:

$$\tilde{\Delta} p_i = \Delta p_i - (n_i n_i^T)\Delta p_i \text{ with } \Delta p_i = \frac{1}{|N_1(i)|}(p_j - p_i)$$

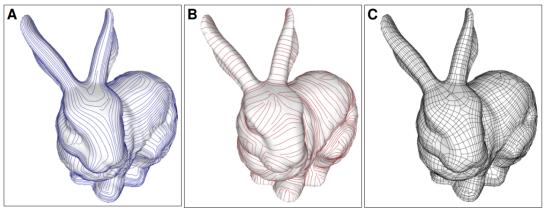
6. Project back to original surface (into the closest triangle).
7. Iterate.

4.4.2 Anisotropic Remeshing

Anisotropic remeshing tries to adapt the shape of the triangles to some property of the surface (e.g., adapt triangles to the direction of the principal curvature). Stretch faces along the direction of small absolute curvature and keeping them short along the direction of large curvature allows for minimal approximation errors and best normal approximation while spending as few faces as possible.

To generate a quad mesh we apply a method called *line tracing*. The algorithm starts by selecting a random seed point on the mesh. For this point the curvature directions are computed and then a line is traced over the surface in both directions.

In the direction of the *maximal curvature* the seed points have to be close to each other, while they can have a greater distance in the direction of the *minimal curvature*. A *quadrangulation* of the surface is then obtained by placing vertices on the intersections of the curvature lines and by connecting them with edges.



4.5 Subdivision and Refinement

A subdivision scheme always consists of two components: a *refinement rule* and a *smoothing rule*.

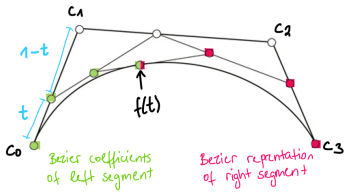
4.5.1 Curve Subdivision

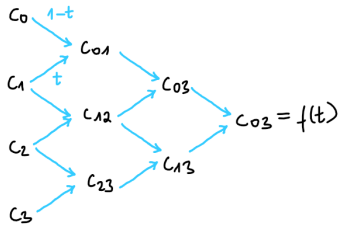
Start with rough approximation, converge to a smooth shape.

With control points c_0, c_1, c_2, c_3, we use the *De Casteljau algorithm* to compute the *Bezier curve* $f(t)$ for $n+1$ control points:

$$f(t) = c_0(1-t)^3 + c_1 3t(1-t)^2 + c_2 3t^2(1-t) + c_3 t^3$$
$$= \sum_{i=0}^{n} c_i B_i^n(t)$$

using the *Bernstein polynomials* $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$, which has the *partition of unity property*, which makes the coefficient the control points: $\sum_{i=0}^{n} B_i^n(t) = 1$.





Polar Forms

Given a polynomial $f(t)$ of degree n. Construct *polar form* $F : (t_1, ..., t_n) \to \mathbb{R}^d$, where we reduce the degree of $f(t)$ to 1, but the complexity is translated to having more arguments in the function. The polar form has to have 3 properties:

1. *Multi-affinity*:
$$F(t_1, ..., (1-\alpha)t + \alpha t', ..., t_n) = (1-\alpha)F(t_1, ..., t, ..., t_n) + \alpha F(t_1, ..., t', ..., t_n)$$
2. *Symmetry*:
$$F(t_1, ..., t_n) = F(t_{\pi(1)}, ..., t_{\pi(n)})$$
3. *Diagonal*:
$$F(t, ..., t) = f(t)$$

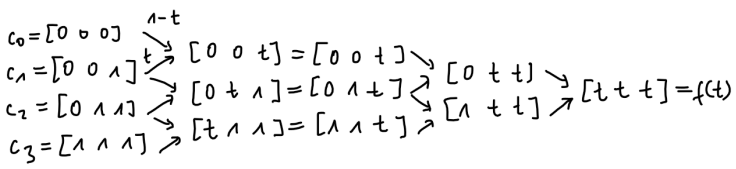
Example:

$$f(t) = a + bt + ct^2 \quad \to \quad F(t_1, t_2) = a + \frac{t_1 + t_2}{2} b + t_1 t_2 c$$
$$f(t) = a + bt + ct^2 + dt^3 \quad \to \quad F(t_1, t_2, t_3) = a + \frac{t_1 + t_2 + t_3}{2} b + \frac{t_1 t_2 + t_2 t_3 + t_1 t_3}{3} c + t_1 t_2 t_3 d$$

De Boor algorithm

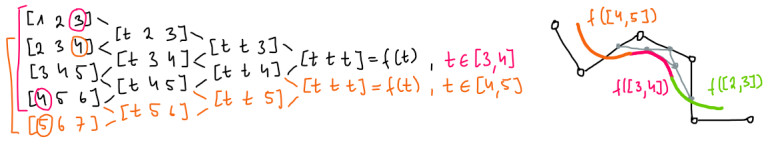
The subdivision of a curve $f([0,1])$ in polar form is defined as splitting it into two part $f([0, \frac{1}{2}])$ and $f([\frac{1}{2}, 1])$.

Notation: $[t_1 \ \dots \ t_n] =: F(t_1, \dots, t_n)$

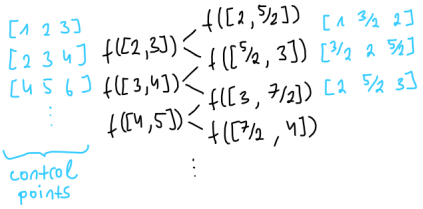


The arguments can be arbitrary, the only requirement is that neighboring positions have to differ in exactly one argument, the result is the same.

We can extend this computation to more positions by simply evaluating 4 neighboring positions at the time, generating curve segments $f([i, i+1])$. Where ever the segment meet, we have C^2 continuity property.



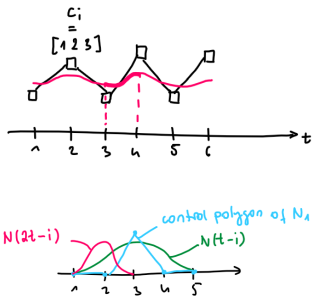
Following subdivisions simply divide the space between the point by 2, resulting into the *half-integer* patterns $[1 \quad 1,5 \quad 2], [1,5 \quad 2 \quad 2,5], [2 \quad 2,5 \quad 3] \ldots$ for initial *integer* patterns $[1 \quad 2 \quad 3], [2 \quad 3 \quad 4], [4 \quad 5 \quad 6] \ldots$.



To get the control points, use (uniform) *B-Spline Subdivision*.

Derivation of De Boor Algorithm: Scalar Functions

For some basis (piece-wise) function $N_i(t)$ and control point c_i we get the control polygon, where N_is are shifted versions of each other (*translation invariant*):



$$f(t) = \sum c_i N_i(t) = \sum c_i N(t-i)$$

The subdivision algorithm computes $\hat{c}_i$ from c_i:

$$f(t) = \sum c_i N(t-i) = \sum \hat{c}_i N(2t-i)$$

2-scale relation:

$$\boxed{N(t) = \sum_j \alpha_j N(2t-j)}$$

$$\begin{aligned} \Rightarrow f(t) &= \sum c_i \sum_j \alpha_j N(2t-2i-j), \quad k = 2i+j \\ &= \sum_i c_i \sum_k \alpha_{k-2i} N(2t-k) = \sum_k \underbrace{(\sum_i \alpha_{k-2i} c_i)}_{\hat{c}_i} N(2t-k) \end{aligned}$$

If $N(t)$ is the degree n B-spline, then $\alpha_j = \frac{1}{2^n}\binom{n+1}{j}$.

Algorithm:

- If i is even (with 0), then the new control points are computed with $\boxed{\hat{c}_i = \frac{1}{2}c_i + \frac{1}{2}c_{i+1}}$.
- If i is odd, then $\boxed{\hat{c}_i = \frac{1}{8}c_{i-1} + \frac{6}{8}c_i + \frac{1}{8}c_{i+1}}$.

Lane-Riesenfeld Algorithm

Define two operators:

- *Doubling Operator* $D : \{\dots c_i \dots\} \mapsto \{...c_i c_i...\}$ with stencil $[1 \quad 1]$
- *Averaging Operator* $A : \{\dots c_i c_{i+1} \dots\} \mapsto \{\dots \frac{c_{i-1}+c_i}{2} \frac{c_i+c_{i+1}}{2} \dots\}$ with stencil $\frac{1}{2}[1 \quad 1]$

Stencil applied as *convolutional operators*: with means

$$\boxed{A^3D = \frac{1}{8}[1 \quad 4 \quad 6 \quad 4 \quad 1]}$$

$$q_{2i} = \frac{1}{8}(p_{i-1} + 6p_i + p_{i+1})$$
$$q_{2i+1} = \frac{1}{8}(4p_i + 4p_{i+1})$$

For smooth curve, which interpolates the initial control points, use *Interpolatory 4-Point-Scheme*. Stencil: with means

$$\boxed{\frac{1}{16}[-1 \quad 0 \quad 9 \quad 16 \quad 9 \quad 0 \quad -1]}$$

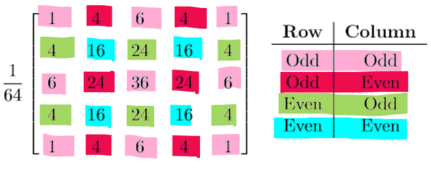
$$q_{2i} = \frac{1}{16}(16p_i)$$
$$q_{2i+1} = \frac{1}{16}(-p_{i-1} + 9p_i + 9p_{i+1} - p_{i+2})$$

4.5.2 Surface Subdivision

Quad Mesh Subdivision

We want to subdivide a quad mesh into finer quads, that converges into a smooth surface. Move the corners of each quad closer together, connect them and throw away the old quads:

$$A_V^3 A_H D = \frac{1}{64}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

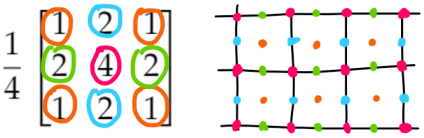


with

$$D = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad A_H = \frac{1}{2}[1 \quad 1] \quad A_V = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Or compute vertices between two existing ones lying on a edge and a vertex in the middle of the quad, connect them with the old vertices. Using D, A_V, A_H we get

$$A_V A_H D = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



(*Catmull-Clark Subdivision*)

Triangle Mesh Subdivision

For a triangle mesh, compute vertices between two existing ones lying on a edge, connect them with the old vertices. The number of triangles increases by 4. Letting m_a, m_b, m_c be the new vertices, the connectivity is changed as follows:

$$\triangle(a,b,c) \to \triangle(a,m_c,m_b), \triangle(b,m_a,m_c), \triangle(c,m_b,m_a), \triangle(m_a,m_b,m_c)$$

(*uniform refinement*)

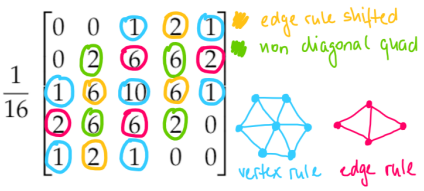
Or insert new vertex in the center of each triangle, connect them with the old vertices (*1-2-3 split*) and flip old edges. The number of triangles increases by 3. For $g = \frac{1}{3}(a+b+b)$ the connectivity is changed as follows:

$$\triangle(a,b,c) \to \triangle(g,a,b), \triangle(g,b,c), \triangle(g,b,a)$$

($\sqrt{3}$-*Refinement*)

Assume a triangle mesh consisting of quad mesh with diagonals *(regular mesh)*. Now, we additionally need a diagonal averaging operator A_D. Then we get

$$A_D^2 A_V A_H D = \frac{1}{16}\begin{bmatrix} 0 & 0 & 1 & 2 & 1 \\ 0 & 2 & 6 & 6 & 2 \\ 1 & 6 & 10 & 6 & 1 \\ 2 & 6 & 6 & 2 & 0 \\ 1 & 2 & 1 & 0 & 0 \end{bmatrix}$$



with

$$A_D = \frac{1}{2}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

4.5.3 Loop Subdivision

If we have a vertex with valance 5, then we cannot apply the rules from above. Generalize the subdivision operator: For resulting vertices $q_i, 0 \leq i \leq n$ lying in the middle of the connecting edge, where q_0 being the center vertex itself, and for neighboring points $p_i, 0 \leq i \leq n$, where p_0 is the center vertex itself, we get for $n = 5$:

$$\begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_n \end{pmatrix} = \frac{1}{16} \underbrace{\begin{pmatrix} 16-\alpha & \alpha/n & \alpha/n & \alpha/n & \alpha/n & \alpha/n \\ 6 & 6 & 2 & 0 & 0 & 2 \\ 6 & 2 & 6 & 2 & 0 & 0 \\ 6 & 0 & 2 & 6 & 2 & 0 \\ 6 & 0 & 0 & 2 & 6 & 2 \\ 6 & 2 & 0 & 0 & 2 & 6 \end{pmatrix}}_{S} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{pmatrix}$$

(Catmull-Clark Subdivision Surfaces)

Take subdivision matrix $S \in \mathbb{R}^{6\times 6}$ and assume, we compute a basis of eigenvector $v_i \in \mathbb{R}^6$ with eigenvalues λ_i.

$$p = \sum \boxed{c_i} v_i$$
$$q = Sp = S\sum c_i v_i = \sum c_i S v_i = \sum \boxed{c_i \lambda_i} v_i$$

Derive conditions for smooth convergence:

- C^0: $\lambda_0 = 1, \lambda_1, ..., \lambda_n < 1$
- C^1: $\lambda_0 = 1, \lambda_1, \lambda_2 < 1$ and $\lambda_3, ..., \lambda_n < \lambda_1$ (easy)
- C^2: $\lambda_0 = 1, \lambda_1, \lambda_2 < 1$ and $\lambda_3, \lambda_4, \lambda_5 = \lambda_1^2$ and $\lambda_6, ..., \lambda_n < \lambda_3$ (hard)

Choose suitable α:

- $\alpha = \frac{5}{8} - \frac{(3+2\cos(\frac{2\pi}{n}))^2}{64}$, then we can guarantee, that the mesh converges to a surface, that is almost curvature continuous.
- $\alpha = \frac{4}{9} - \frac{2\cos(\frac{2\pi}{n})}{9}$ for $\sqrt{3}$-subdivision

5 Parametrization

The process of finding a *parameterization* of a triangle mesh $F : \Omega \in \mathbb{R}^2 \to S \subset \mathbb{R}^3$ is strongly related to the above problem of finding an interpolating surface function $f : \mathbb{R}^2 \to \mathbb{R}^3$. This can be seen as *flattening* a 3D surface onto a 2D surface, which allows to reduce complex 3D problems onto the 2D domain.

F is *piecewise linear* and *bijective*, hence the process of finding a suitable function F for a mesh reduces to the simpler problem of finding a set of vertices $\{q_i\}, q_i \in \mathbb{R}^2$ with a one-to-one relation $F(q_i) = p_i$ to the respective vertices p_i of the mesh.

We are usually interested in a parametrization with minimal *distortion*.

Applications:

- texturing
- interpolatory smoothing (smooth in 2D, transfer to 3D)
- remeshing
- 3D-morphing (linearly interpolate between to surfaces in 2D first)
- Delaunay triangulation

Univariate Parametrization

Given n 2D points p_i from polygon $P = [p_i]$, find n parameters t_i corresponding to vertices p_i under the constraint that $F(t_i) = p_i$:

$$F(t) = \frac{t_i - t}{t_i - t_{i-1}} p_{i-1} + \frac{t - t_{i-1}}{t_i - t_{i-1}} p_i \quad \forall_{t \in [t_{i-1}, t_i]}$$

If we find a triangle in 2D, which corresponds to a triangle in 3D, the parametrization is fully defined. Define a point in barycentric coordinates, which lies in the 2D triangle (A, B, C) or (A', B', C') respectively.

For $p \in \mathbb{R}^2$:

$$\begin{aligned} p &= A + \beta(B - A) + \gamma(C - A) \\ &= \beta B + \gamma C = (B \quad C) \begin{pmatrix} \beta \\ \gamma \end{pmatrix} \end{aligned}$$

Point $p' \in \mathbb{R}^3$:

$$\begin{aligned} p' &= A' + \beta(B' - A') + \gamma(C' - A') \\ &= \beta B' - \gamma C' = (B' \quad C') \begin{pmatrix} \beta \\ \gamma \end{pmatrix} \end{aligned}$$

Then mapping is defined as

$$F(p) = (B' \quad C')(B \quad C)^{-1} p = Mp$$

Measure the distortion in scaling term, apply SVD on M per triangle:

$$F(p) = U \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{pmatrix} V^T p$$

Distortion measures:

- $s_\triangle = \sigma_1 = \sigma_2 = 1$: *isometric* mapping, just rotation
- $s_\triangle = \sigma_1 = \sigma_2$: *conformal*, angle preserving

$$s_\triangle = \sqrt{(\sigma_1^2 + \sigma_2^2)/2} \begin{cases} < 1 & \text{compression} \\ = 1 & \\ > 1 & \text{expansion} \end{cases}$$

Compute an area-weighted average of the triangle distortions per vertex p_i:

$$\boxed{s_{p_i} = \frac{1}{\sum_{\triangle \in N(p_i)} \text{area}(\triangle)} \sum_{\triangle \in N(p_i)} s_{\triangle} \cdot \text{area}(\triangle)}$$

5.1 Disk Topology

5.1.1 Harmonic Parametrization (fixed convex boundary, one patch)

A function f is *harmonic* iff $\Delta f = 0$.

Input a triangle mesh with disk topology. First, map boundary polygon in 3D to a boundary polygon in 2D.

If points p_i in the 1-ring neighborhood of p are given, we just imagine *springs* as connections to movable p and fixed p_i. The optimal solution is when the springs are in *equilibrium*.

With spring constant w_{ij}, weights with *linear precision*, we get a sum of forces between two vertices:

$$\forall_{p_i \notin \partial\Omega} : \sum_{p_j \in N(p_i)} w_{ij}(p_j - p_i) = 0 = \Delta p \quad \Rightarrow \boxed{\Delta F = 0}$$

where $\partial\Omega$ denotes the set of vertices on the boundary of the parameter domain.

Theorem: If $w_{ij} > 0$ and boundary is convex, then there exists a consistent harmonic parametrization.

This form is identical to the computation of update vectors to smooth a mesh, hence, we can re-use the same weights we have evaluated for smoothing already (see page 22f).

Stretch Minimizing Parametrization

Weights are adjusted so that compression and expansion is compensated locally.

1. Compute harmonic parametrization.
2. Compute a stretch measure s_{p_i} per vertex p_i.
3. Update "spring forces" $w_{ij} \leftarrow w_{ij} / \frac{s_{p_i} + s_{p_j}}{2}$.

5.1.2 Least Squares Conformal Maps LSCM (free boundary, one patch)

A parametrization F is *conformal* if it *preserves angles*. That is, for every pair of intersecting curves in Ω, the angle between the curve in the point of intersection $q \in \Omega$ is the same as the angle between the images of the curves in $F(q) \in S$. A parametrization is conformal iff for the matrix M $\sigma_1 / \sigma_2 = 1$.

The *least squares conformal maps* do not require a convex parameter domain and allow for relaxed boundary conditions with a minimum of only two fixed vertices.

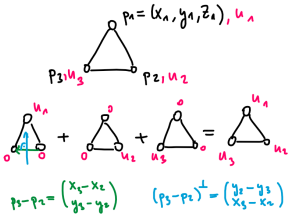
If we map u, v coordinate system to 3D with mapping F, then the v axis becomes partial derivative F_v, which is perpendicular to the surface, and u becomes F_u. Formulation of low distortion is then having $F_u \perp F_v$ and $\|F_u\| = \|F_v\|$ for every triangle:

$$\sum_{\triangle} \|F_v - R_{90^\circ} F_u\|^2$$

We observe $F_v \sim \nabla v, F_u \sim \nabla u$. Rewrite above equation to

$$\boxed{\sum_{\triangle} \|\nabla v - R_{90^\circ} \nabla u\|^2 \to min}$$

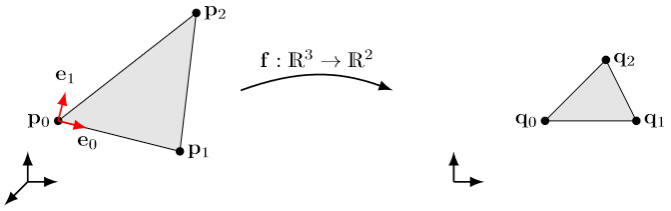
Computation of the gradients:



With magnitude $\frac{u_1}{h} = \frac{u_1 l}{2area}$ we get the resulting gradients:

$$\begin{aligned} & \frac{u_1}{2area}(p_3 - p_2)^\perp \\ \Rightarrow \quad & \nabla u = \frac{1}{2area} \begin{pmatrix} y_2 - y_3 & \dots & \cdots \\ x_3 - x_2 & \dots & \cdots \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \\ \Rightarrow \quad & \nabla v - R_{90^\circ} \nabla u = G_{\triangle} (u_1, u_2, u_3, v_1, v_2, v_3)^T \end{aligned}$$

where $G \in \mathbb{R}^{2 \times 6}$ gradient matrix.



1. $g : \mathbb{R}^3 \to \mathbb{R}^2, p \mapsto g_0 + Gp, G \in \mathbb{R}^{2 \times 3}$: maps points from the global coordinate system into the local coordinate system (e_0, e_1) of the triangle.

$$\begin{aligned} & e_0 = \frac{v_{01}}{\|v_{01}\|}, e_1 = \frac{(v_{01} \times v_{02}) \times e_0}{\|(v_{01} \times v_{02}) \times e_0\|}, v_{01} = p_1 - p_0, v_{02} = p_2 - p_0 \\ \Rightarrow \quad & G = (e_0 \quad e_1)^T, g_0 = -Gp_0 \end{aligned}$$

2. $h : \mathbb{R}^2 \to \mathbb{R}^2, p \mapsto h_0 + Hp, H \in \mathbb{R}^{2 \times 2}$: maps points from the local coordinate system of the triangle into the parameter domain.

$$H = (q_1 - q_0 \quad q_2 - q_0)(Gp_1 - Gp_0 \quad Gp_2 - Gp_0)^{-1}, h_0 = q_0$$

3. $f : \mathbb{R}^3 \to \mathbb{R}^2, p \mapsto f_0 + Fp = h(g(p)), F \in \mathbb{R}^{2\times 3}$: maps points from the global coordinate system into the parameter domain.

$$F = HG,\ f_0 = h_0 + Hg_0$$

5.1.3 Angle-Based Flattening (free boundary, one patch)

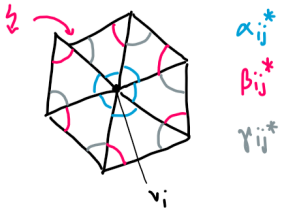
Input mesh in $\mathbb{R}^3$ has vertices v_i, triangles t_j, and inner angles α_{ij}. Output mesh in $\mathbb{R}^2$ has angles α^*_{ij}.

$$\boxed{\sum_{i,j} \frac{1}{\alpha_{ij}^2}(\alpha^*_{ij} - \alpha_{ij})^2 \to \min}$$

Conditions:

1. *Vertex condition*:

$$\forall_{\text{inner vertices } v_i} : \sum_j \alpha^*_{ij} = 2\pi$$



2. *Triangle condition*:

$$\forall_{t_j} : \alpha^*_{i_1 j} + \alpha^*_{i_2 j} + \alpha^*_{i_3 j} = \pi$$

3. *Star condition*:

$$\forall_{v_i} : \prod_j \frac{\sin(\beta^*_{ij})}{\sin(\gamma^*_{ij})} = 1$$

Third condition is not linear, therefore, change variables: $\alpha^*_{ij} := \alpha_{ij} + e_{ij}$. Simplify boxed equation from before:

$$\boxed{\sum_{i,j} \frac{1}{\alpha_{ij}^2} e_{ij}^2 \to \min}$$

Change conditions accordingly:

1. $\sum e_{ij} = 2\pi - \sum \alpha_{ij}$
2. $\sum_i e_{ij} = 0$
3. $\prod_j \frac{\sin(\beta_{ij} + e_{\beta_{ij}})}{\sin(\gamma_{ij} + e_{\gamma_{ij}})} = 1 \quad \Rightarrow \quad \sum \log(\sin(\beta_{ij})) - \log(\sin(\gamma_{ij})) = const$

5.2 General Topology

5.2.1 Globally Smooth Parametrization (fixed boundary, multiple patches)

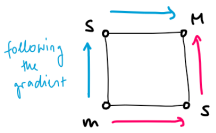
Steps:

1. Segment the surface into "square" patches.
2. Compute global parametrization by *coupling* local parametrizations.
3. (Iterate.)

1) Morse-Smale Complex

The *Morse-Smale Complex* is a topological complex that uses all the stationary points of a scalar field to define a topology. Used for segmentation into quads.

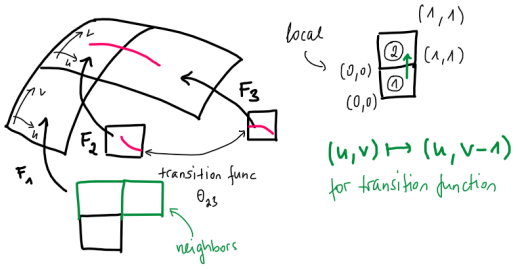
Assume scalar field $S(u,v) : \mathbb{R}^n \to \mathbb{R}$. This one has extrema $M_1, ..., M_k, m_1, ..., m_l$ and other stationary points s ("saddles"). We use these points to define a complex, as seen on the right.



Use particular scalar fields, that lead to fairly reasonable Morse-Smale complexes: *eigenvector of the Laplace operator*, which is equivalent to a scalar field, when the Laplace operator is written as a matrix multiplication (see page 21). This matrix is symmetric, so we expect to have a basis of orthogonal eigenvectors. One eigenvector can be considered as a discrete scalar function on the surface. The magic is to choose one eigenvector as a representative for the scalar field.

2) Local Parametrization

Make *local* parametrization of patches *harmonic*: $\Delta F_i = 0$ ($\to$ low distortion for every individual map). To obtain neighborhood relations, define *transition functions* θ_{ij}. With the relation between the squared that parametrizes the patches we establish a *global smoothness criterion*.



Reverse formulation (optimize parameter domain), right is the formulation if some neighbors are in another patch:

$$\Delta u_i = \sum w_{ij}(u_j - u_i) = 0, \quad ... \sum w_{ij}(\theta_{ji}(u_j) - u_i) = 0$$
$$\Delta v_i = \sum w_{ij}(v_j - v_i) = 0$$

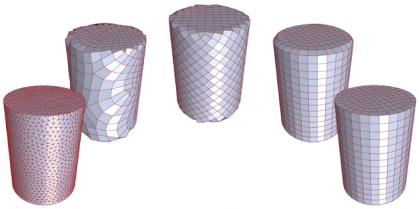
To avoid inconsistencies, we require, that the corners of patches are constrained to (0,0), (0,1), (1,0), or (1,1).

5.2.2 Mixed-Integer Quadrangulation / Integer Grid Maps

Process of converting a triangle mesh to a *quad mesh*.

Criteria for *good* quadrangulations:

- topological consistency
- individual element quality
- element orientation (in direction of principal curvature)
- element alignment



1) Compute smooth direction field

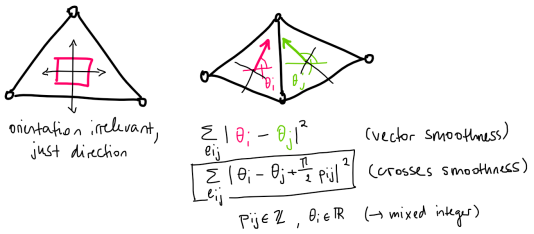
Compute for every triangle a coordinate system for orientation, resulting into a *cross field*.

$\frac{\pi}{2}$ is necessary, since every 90° rotation of the direction vector is equivalent. For n vertices, we have $5n$ degrees of freedom.

In 3D, we need a correction term. The angles θ are then computed in respect to one reference edge:

$$|\theta_i - \theta_j + \kappa_{ij}|^2 \to \min$$

Mixed-Integer Quadrangulation:



Express the *smoothness energy* of a cross field:

$$E_{smooth} = \sum_{e_{ij} \in E} (\theta_i + \kappa_{ij} + \frac{\pi}{2} p_{ij} - \theta_j)^2 \to \min$$

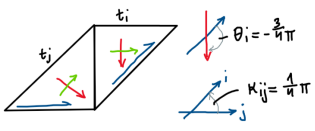
with

e_{ij}: edge between triangles t_i and t_j

θ_i: signed angle between primary direction/u-direction (red) and respective reference direction (blue)

κ_{ij}: transition angle $\kappa_{ij} \in (-\pi, \pi]$, which is the rotation of one reference direction j to another neighboring reference direction i

p_{ij}: period jumps

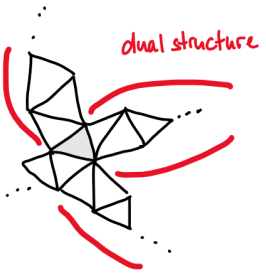


- *Regular* vertex, if $\sum p_{ij} = 0$ (valance 4)
- *Singular* vertex, if $\sum p_{ij} \neq 0$ (valance 3, 5, 6)

To minimize energy for θ_k for example, set gradient (in this case, the partial derivative) of the energy to zero: $\frac{\partial E}{\partial \theta_k} = 0$.

2) Surface cutting

Pick random triangle in your mesh. Grow a (minimal) spanning tree of triangles: Add triangles across every (free) edge, without closing a one ring. Then, glue together the gaps of the dual tree, resulting into the (intuitive) cuts according to the genus of the object, and especially into a disk topology.



Then we need $2g$ cuts to establish disk topology with g being the genus. Probably more, if we constraint, that all singularities need to lie on the boundaries of the disk topology.

For every singularity, we need additional cuts (correct angle sums). On cuts, every edge appears twice.

3) Global Parametrization

Direction field aligned parametrization:

$$\sum_{\triangle}(\|\nabla u - u^*\|^2 + \|\nabla v - v^*\|^2)A \to \min$$

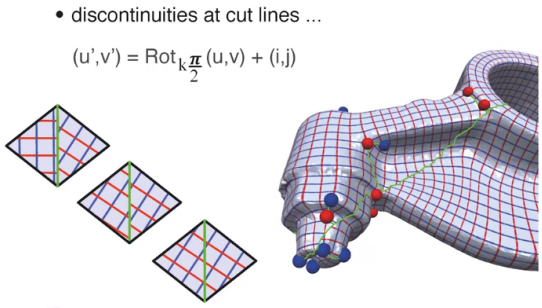
where u^* and ∇u are vectors in the plane of a triangle.

Then the quality of the u, v parametrization that we computing is as similar as possible to the prescribed orientation field u^*, v^*.

But we still have visible seams, i.e. discontinuities at cut lines. We perform some rotation and translation of the parametrization domain to match the cutting edges:

- Rotate by $k\pi/2$ to make the lines parallel.
- Translate by $T = (i, j)$.

$\to$ *Grid Automorphism* with seemingly continuous grid on the seams



This is again a mixed-integer optimization, which we can solve.

Local injectivity (no triangles are flipping) can be inforced by looking at every parameter of triangle $\triangle(A, B, C)$, and enforce

$$\det(B - A, C - A) > 0$$

5.3 Surface Layouts

When we have a *layout*, which divides the surface into quad *patches*, we can refine the resulting patches into a quad mesh. If you want a *layout*, you want to guarantee that

the singularities are connected by sequences of edges. Only then you can *localize* your local parametrization.

Given the singularities, we want to dualize the problem of finding a layout, compute dual *loops*, and then do another dualization to get back to the primer layout.

To do these, we need to satisfy some requirements for the loops:

- Intersections are transversal, orthogonal, and only *two* loops.
- Separated regions need to have *disk topology*.
- Loops should follow the *principal directions*.
- Valence 4 in flat regions.
- Valence 5 at negative Gauss curvature.
- Valence 3 at positive Gauss curvature.
- Few and short loops.

Generate a Loop

Take direction field, now trace a loop that follows this direction field. This loop will be spiraling due to numerical imprecision.

Instead of a loop, trace an *ellipse*. This ellipse is growing in the direction of the direction field fast, and slow in the perpendicular direction.

Loop Selection

We have to produce so many loops, so that every region has disk topology and contains only one singularity.

Use *"separation indicators"*: Connect every singularity with each other using geodesic curves (direction filed alignment, principal curvature). As long a separation indicator exists, we still have singularities, which are still connected. When an indicator crosses a loop, it will be removed. Perform the loop computation as long as there are some indicators left.

Then, where the loops cross each other, there will be a face, and where the loop lies will be edges of the layout.

Node Relocation

To further lower distortion in the parametrization, before the parametrization computation, we can move the singularities to their optimal positions.