SLOVAK UNIVERSITY OF TECHNOLOGY
Faculty of Civil Engineering

Registry number: SvF-5342-63956

Optimal Discretization of 3D Objects by Evolving Surfaces

Diploma Thesis

Study program: Mathematical and Computational Modeling
Study program number: 1114
Major: 9.1.9 Applied Mathematics
Department: Department of Mathematics and Constructive Geometry
Supervisor: prof. RNDr. Karol Mikula, DrSc.

Bratislava 2020 Bc. Martin Čavarga

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Civil Engineering

Reg. No.: SvF-5343-63956

Optimal discretization of 3D objects by evolving surfaces

Master thesis

Study programme: Mathematical and Computational Modeling
Study field: Mathematics
Training workplace: Department of Mathematics and Descriptive Geometry
Thesis supervisor: prof. RNDr. Karol Mikula, DrSc.

Bratislava 2020 **Bc. Martin Čavarga**

Acknowledgments

I would like to thank my supervisor prof. RNDr. Karol Mikula, DrSc. for extensive expert assistance and consultation which he provided for my research whenever necessary. Also, considerable gratitude goes to Balázs Kósa for the assistence in the implementation of the Fast Sweeping algorithm, to Matej Medľa for insights on the proof of the volume density "divergence formula" for volume-based tangential redistribution, and to Peter Sarkoci for consultations on intrinsic differential geometry.

Čestné prehlásenie

Vyhlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry a s odbornou pomcou vedúceho práce.

v Bratislave 14.5.2020

.....

Vlastnoručný podpis

Abstract

Surface evolution has been used for various applications, such as numerical construction of minimal surfaces (Truss structure design), point cloud meshing, and quad remeshing of triangular meshes. We present a novel approach with all its current limitations to remeshing of polygonal geometries governed by evolution equation: $\partial_t F = v_N + v_T$ where F is the immersion of the evolving manifolds into $\mathbb{R}^3$, that is: the time-dependent state of the evolving surface, and v_N with v_T are the normal and tangential velocities respectively. Evolution in the normal direction is controlled by mean curvature (Laplace-Beltrami operator), and preferred gradient field $-\nabla d$ in $\mathbb{R}^3$ of the (signed) distance function d of the original mesh. Tangential velocity v_T is given by area-based, length-based, and angle-based redistribution of mesh vertices, which inherently controls the polygonal density in target regions of the surface.

Keywords - remeshing, surface evolution, immersion, Laplace-Beltrami operator, signed distance function, tangential redistribution

Abstrakt

Evolúcia plôch má už viacero aplikácii, ako napríklad numerická konštrukcia minimálnych plôch (dizajn Trussových štruktúr), triangulácia dát vo forme mračna bodov, alebo aj remeshing trojuholníkovej na optimálnu štvoruholníkovú sieť. Aj s jeho súčasnými obmedzeniami v tejto práci ukazujeme nový prístup k optimalizácii n-uholníkových geometrii ($n = 3, 4$), riadenej evolučnou rovnicou: $\partial_t F = v_N + v_T$ kde F je vnorenie vyvíjajúcej sa variety do $\mathbb{R}^3$, teda časovo-závislý stav vyvíjajúcej sa plochy v priestore a v_N s v_T sú normálová a dotyková (tangenciálna) rýchlosť. Evolúcia v normálovom smere je kontrolovaná strednou krivosťou (Laplace-Beltramiho operátor) a zvoleným gradientným poľom $-\nabla d$ v $\mathbb{R}^3$ (znamienkovej) vzdialenostnej funkcie d pôvodnej siete. Tangenciálna rýchlosť v_T je určená redistribúciou uzlových bodov podľa plôch n-uholníkov, dĺžok hrán a vnútorných uhlov, čo zabezpečuje kontrolu hustoty n-uholníkov v cieľových oblastiach plochy.

Kľúčové slová - remeshing, evolúcia plôch, vnorenie, Laplace-Beltramiho operátor, znamienková vzdialenostná funkcia, tangenciálna redistribúcia

Contents

A Word on Notation

This chapter is recommended for readers who may find the notation in this thesis in contrast with their preferred symbols. We adopt a combination of notational approaches mainly from textbooks on differential geometry, algebra, and analysis. The notation is chosen, so that symbols do not conflict, at least within chapters, and if they do, the reader is notified.

Number fields are denoted with blackboard bold symbols: $\mathbb{N}$ (natural numbers), $\mathbb{Z}$ (whole numbers), $\mathbb{Q}$ (rational numbers), $\mathbb{R}$ (real numbers), and $\mathbb{C}$ (complex numbers). Their particular restrictions are then expressed in upper and lower indices, for example $\mathbb{R}_0^+$ stands for positive real numbers including zero (non-negative). *Cartesian product* producing *pairs* (x, y) (as well as *n-tuples* $(x_1, ..., x_n)$) between sets is marked with $X \times Y$.

We make use of the standard *set-theoretic notation*: $\in$ (element of), $\notin$ (not an element of), $\subset$ (proper subset of), $\subseteq$ (subset of or equal to) etc. Including their reversed versions. Individual sets have different symbols, depending on their context. Topological spaces and manifolds are marked with capital X, Y, Z etc. while their subsets are in caligraphic capital letters $\mathcal{U}$, $\mathcal{V}$. Sets defined by elements with particular attributes are marked as $\{x \mid P(x)\}$, that is "elements x such that $P(x)$" where P is a predicate (statement about the nature of x). *Intervals* in $\mathbb{R}$ follow: $[a, b]$ (closed), $[a, b[$ (closed in a, open in b), $]a, b]$ (open in a, closed in b), and $]a, b[$ (open), to avoid confusion between a pair (a, b) and an open interval.

Mappings between sets X and Y are marked with arrows $f : X \to Y$ (f maps set X to Y) and by elements $f : x \mapsto f(x)$ (f maps $x \in X$ onto $f(x) \in Y$). Occasionally, if we require a concise note that f is a linear map, we write $f : X \overset{lin}{\to} Y$. *Unary operators* "act" on set elements from the right: fx or are marked as mapped values $f(x)$. A mapping is an object fundamentally different from its image of the particular element $f(x)$. Specific operators (especially those which process tuples of elements) can be written in their "evaluating form" $f(\cdot)$ using dots for missing arguments, for example $b = b(\cdot, \cdot) : V \times V \to \mathbb{F}$ is a viable notation for a bilinear form processing two vectors form V. The image of a set $U \subseteq X$ by f is marked with brackets: $\mathrm{Im}_f[U] = f[U]$, to denote the difference between mapping an element, and mapping a set. If we do not want to restrict ourselves to a particular subset of the domain X, we can put $f[X] = \mathrm{Im}(f)$. A composition of mappings $f : X \to Y$ and $g : Z \to W$, where $Z \subseteq \mathrm{Im}(f)$ is marked by $\circ$ operator "in order of acting", that is: if element $w = g(f(x)) \in W$ then we put $w = (g \circ f)(x)$.

As for *binary operators*, we use standard notation for operations in number fields and vector spaces, unless we intend to emphasize the difference between operations in respective spaces, for example: $u + v = u \oplus v = u \oplus_V v$ which emphasizes that the addition is carried out on the vector space structure of V (and $u, v \in V$). Whenever a "matrix approach" to algebra is required, vectors (column by default) from $\mathbb{R}^n$ are in bold: $\mathbf{x} = (x^1, x^2, x^3)^\top$, including respective matrix operators on them: $\mathbf{Ax} = \mathbf{b}$.

Vector spaces on fields are usually marked with the number field $\mathbb{F}$ they are constructed on: $\mathbb{F}^n = \underbrace{\mathbb{F} \times ... \times \mathbb{F}}_{n\text{-times}}$ where $n \in \mathbb{N}$ denotes its dimension (sometimes we use d as "dimension" while n is used for indexing). The notation for *inner products* can change depending on the setting. $\langle u, v \rangle$ denotes standard

inner product in $\mathbb{R}^n$. However, we use "dot" $\mathbf{x} \cdot \mathbf{y}$ as for *dot product* (especially in 2D and 3D spaces $\mathbb{R}^2$, $\mathbb{R}^3$), while the bracket notation is resorted to higher-dimensional abstract vector spaces, sometimes with an emphasis $\langle \cdot, \cdot \rangle_V$ on a particular vector space where this inner product is defined. Similarly, we put manifold symbols as lower indices for metric $g_X(\cdot, \cdot)$ to show that it is constructed on a manifold X.

In some instances, we follow the *Einstein summation rule* to express linear combinations of higher-dimensional objects: $v^1 \hat{e}_1 + ... + v^n \hat{e}_n = v^i \hat{e}_i$, which helps in higher order expansions, especially when deriving results based on the multi-linearity of forms on manifolds. A new index, within bounds, but generally unrelated to the original indexing, is created for each expansion. Upper and lower expansion indices often "cancel out" to produce a single object. We also use the *Kronecker delta*: $\delta_i^j = 1$ (if $i = j$) and $\delta_i^j = 0$ (otherwise).

Chapter 1

Introduction to Modern Remeshing Techniques in Computer Graphics

Almost all computer-generated surface models ever rendered on a screen are composed of triangular elements. Perhaps the only exception to this approach are implicitly generated surfaces rendered via ray-marching methods using shaders which include various procedurally generated fractals and landscapes in some animations.

Discretized versions of real-world objects are also the essence of engineering techniques which make use of various numerical methods (FEM, BEM, etc.). These areas require the underlying *mesh* representation of an object to have certain qualities, that is: the individual mesh elements need to be "well shaped" for the particular numerical solution to converge, let alone to produce results representative of reality. Since triangular elements are easy to render on a screen, computer-aided designs (CAD), on the other hand, only require the model to be composed of triangles. The quality of these triangular elements is often not guaranteed by the very process of *mesh generation*.

Mesh quality is also desired in computer graphics (CG) modeling within film-making and video-game development applications, mainly due to the following reasons:

- It is easy to perform various deformations on high-quality meshes, and such meshes are generally desired for use in character design with animated bone movements.
- A high-quality mesh is desired for optimal surface rendering with UV-mapped textures and normal (bump) maps.
- The subdivision detail (i.e.: the number of discretisation vertices) is generally easy to control for high-quality meshes.
- One needs effective control over the amount of polygons in a model, in order to save storage space.

For a combination of the above reasons, there also seems to be a preference for quadrilateral (quad) meshes over triangular or ($n \geq 5$)-gonal meshes. For example, it is easy to see why a "*subdivision surface*" operation produces better results on a quad mesh. Another example can be seen in various UV-mapping tools which are easier to work with when parametrizing a texture map over a quad mesh.

The process of generating a higher-quality mesh from a low-quality one is referred to as *remeshing*.

1.1 Mesh Generation

The most straight-forward approach to generating a surface or volume mesh is by sampling the parametrization (chart map) of the modeled object.

The sampling may be performed by directly evaluating the parametrization in $\mathbb{R}^n$ or recursively subdividing an existing mesh (for example an icosahedron, see Fig. 1.1). In both cases we speak of a *structured mesh*, although some sources [30] refer to meshes generated by recursive subdivision of a base mesh as *semi-regular*.

In almost any (even moderately complex) model the designs require an outline which is generally not easy to parametrize. Take, for example, a general polygonal boundary. Such object can be sampled by triangular elements in many ways, but only a small subset of all *tessellations* can be considered as structured. Structured meshes require a strict geometric outline, such as opposing edges for 2-parametrization, or other direct mathematical ways to sample individual points of a given surface.

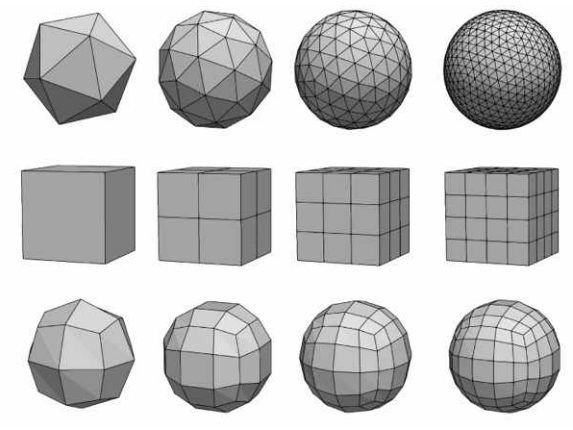

Figure 1.1: Four steps of subdivision of geometric primitives (icosahedron, cube, cube-sphere).

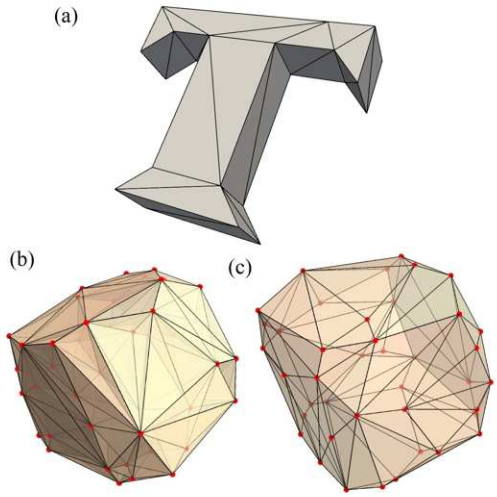

Figure 1.2: Crude approaches to unstructured mesh generation: (a) A coarse polygonal model, triangulated using the "ear-clipping" algorithm. (b) A Delaunay triangulation of a random point cloud, and (c) a modified Delaunay triangulation for convex hulls on the same random data.

The default technique for triangulating general polygonal surfaces in virtually all rendering frameworks is called an *ear-clipping algorithm*. It relies on recursive clipping of triangular cuts from the polygon, as can be seen in Fig. 1.2 (a). One can imagine cutting off triangular corners of a polygonal piece of paper until we are left with a single triangle. This procedure can be extended even to polygons with holes.

When processing point-cloud data with a significant amount of noise, the most common approach is the *Delaunay triangulation*, relying on searching for such adjacency configuration for all given vertices, that no point lies inside the circumcircle (or circumsphere) of any triangle. Running a Delaunay procedure on 3D data, however, may result in a *non-manifold geometry* which makes any form of global texture UV-mapping impractical. It can, of course, be modified to only produce a convex hull of the point set (see Fig.1.2 (b), (c)) or even a suitable surface boundary of data with given point normals.

Most FEM mesh generators use *advancing front* mesh generation to produce tessellations of desired element size and density as a basis for additional refinement. The result is a relatively homogeneous discretisation of a given region. It relies on constructing equilateral triangles (or tetrahedra in 3D) from the boundary inwards, and adjusting the each element generation according to collision with

elements advancing from the other side. Not only can such meshes properly process boundaries with a lot of detail (see Fig.1.3 (a)), but the element density can also be imposed on elements in the interior, effectively controlling the mesh quality and sampling.

The *marching cubes* algorithm generates an iso-surface of a given scalar field (implicitly or explicitly generated), by effectively subdividing a desired volume into regular cells, and triangulating only the cells whose vertex values surround a given threshold value, in other words: if we choose a particular value f_0 of a scalar field $f : \mathbb{R}^3 \to \mathbb{R}$, the algorithm renders an approximation of a level surface of f up to the detail of the sampled regular grid. The position of triangulated vertices is interpolated according to the threshold's relative position within vertex values. This is perhaps the most effective (and the easiest to implement) technique to visualize 3D scalar data, and is widely used in medical imaging.

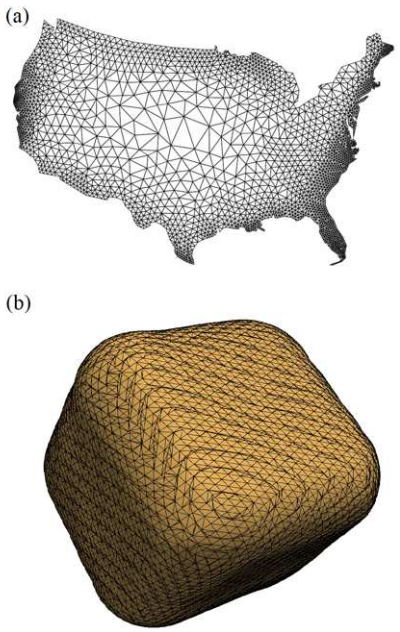

Figure 1.3: (a) Mesh of a polygonal region generated by the advancing front technique, and (b) an iso-surface generated by the marching cubes algorithm.

1.2 Mesh Quality

All of the above techniques might, under certain configurations, produce elements of low quality. However, it is by no means obvious what "mesh quality" might be a function of. FEM software demands inner angles of an element to be bounded from above as well as from below, in such way that we avoid constructing elements that would be "too thin". For optimal numerical results, individual elements ought to locally resemble regular structured grids as much as possible. ANSYS ©, for example, measures element orthogonality, aspect ratio as well as skewness.

The discrete representation of a surface also needs to be optimized for shape-related operations such as normal, tangent plane, or curvature estimations. Specific quality metrics are described in more detail in Knupp [20].

Another crucial property is *mesh fidelity*. The generated mesh has to approximate a target shape with sufficient accuracy, while keeping the mesh "coarse enough". It can be controlled by a specific error metric, or choosing between approximating or interpolating mesh nodes.

Chapter 2

Computational Geometry Fundamentals

In this work we will consider 3D models with properties specific to CG applications. This chapter deals with the elementary mathematical definitions of our objects of interest.

Any geometry rendered on scene can be a collection of approximations of manifolds. However, there is no guarantee that such collection is connected, or even form a larger manifold. Non-manifold geometries are common in CAD applications, since the design process involving a set union of two manifold geometries with the intention of producing a connected model usually results in regions where any chart-map parametrization would not have an inverse. It is usually easier to connect two geometries via non-manifold "seams", than redesign the entire model to produce a manifold geometry (see Fig.2.1).

Throughout this thesis, however, we will only consider manifold geometries.

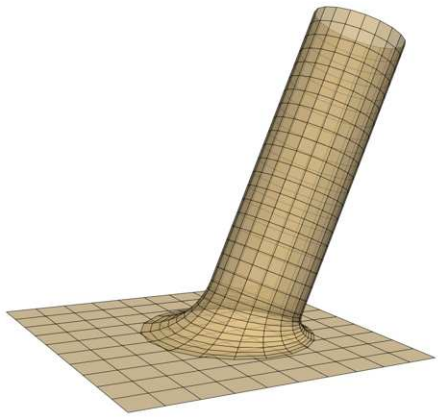

Figure 2.1: A Dupin Cyclide connecting a cylinder with a plane.

2.1 Triangulations and Polygonal Approximations

Since triangles are geometric primitives most representative of a linear 2-space, a triangulation is generally considered to be a piecewise-linear representation of a given 2-manifold (surface) immersed in $\mathbb{R}^n$. Such approximations can easily be extended to arbitrary dimension:

Definition 2.1.1. A *k-simplex* is a convex hull of $k+1$ vertices $\sigma := \{\lambda_0 x_0 + ... + \lambda_k x_k | \sum_{i=0}^{k} \lambda_i = 1, \lambda_i \geq 0\}$.

A simplex is a k-dimensional generalization of a triangle. A 0-simplex is a point, 1-simplex a line segment, 2-simplex a triangle, 3-simplex a tetrahedron, etc. By definition, individual vertices are elements of vector space $\mathbb{R}^n$, $n \geq k$, however their exact position is irrelevant, as long as there is a topological equivalence (a homeomorphism) between simplices, in other words: a triangle is topologically equivalent to any other in

$\mathbb{R}^n$. Technically, the definition of a convex hull can be extended to a set of points on a Riemannian manifold using geodesic polyhedra from given vertices, but it will be clear in the upcoming chapters that we will be working with manifolds immersed in $\mathbb{R}^n$.

A union of simplices homeomorphic to a k-sphere: $\mathbb{S}^k$ will be referred to as a *polytope*, and for $k = 2$ we recognize such objects as *polygons*.

Definition 2.1.2. A *simplical complex* K is a finite collection of vertices V and *simplices* $S \subset \mathcal{P}(V) = 2^V$ [1] such that if $\sigma \in S$ and $\tau \subset \sigma$ then $\tau \in S$.

The combinatorial data (S, V) is referred to as an *abstract simplical complex*, and due to finiteness, V can be constructed in such fashion that a simplical complex is a topological space. For example, let $K = (S, V)$ be a simplical complex such that $V = \{v_1, v_2, v_3, v_4\}$ and

$S = \{\{v_1\}, \{v_2\}, \{v_3\}, \{v_4\}, \{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_3, v_4\}, \{v_1, v_3, v_4\}\}$

A geometric realization of K can be seen in Fig. 2.2

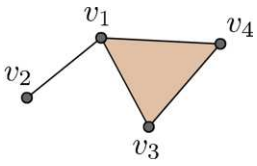

Figure 2.2: A simplical complex $K = (V, S)$

For $S' \subset S$ a set $\mathrm{Cl}(S') := \{\tau \in S \mid \tau \subseteq \sigma \in S'\}$ is called the *closure*, $\mathrm{St}(\tau) := \{\sigma \in S \mid \tau \subseteq \sigma\}$ the *star*, and $\mathrm{Lnk}(\tau) := \{\sigma \in \mathrm{Cl}(\mathrm{St}(\tau)) \mid \tau \cap \sigma = \varnothing\}$ a *link* of a simplex $\tau \in S$.

In the above example, the link of $\{v_4\}$ is edge $\{v_1, v_3\}$, and the link of $\{v_1\}$ a union of $\{v_2\}$ and edge $\{v_3, v_4\}$.

A homotopy-theoretic generalization of a simplical complex is a CW-*complex*. Given a discrete collection of vertices, and successively attaching collections of $(k+1)$-simplices homeomorphic to disks $\mathbb{D}^{k+1}$ along their boundaries $\mathbb{S}^k$, for each $k = 0, 1, ..., n$.

Definition 2.1.3. A *triangulation* of a topological space X is a homeomorphism $\mathcal{T} : X \to K$ onto a simplical complex $K = (V, S)$.

Not every CW-complex can be triangulated, but every such object obtained by gluing polyhedral cells by piecewise-linear maps is triangulable [17]. Take, for example, a triangulation of the torus $\mathbb{T} = \mathbb{S}^1 \times \mathbb{S}^1$ in Fig. 2.3 which is clearly based on a union of quad cells forming a simplical complex K.

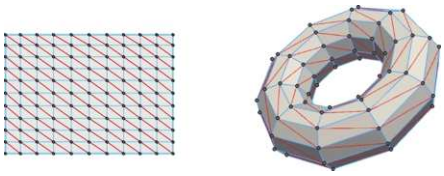

Figure 2.3: A Triangulation of a torus $\mathbb{T}$

Now considering only topological manifolds, we distinguish between:

- general topological C^0-manifolds with continuous transition maps.
- smooth manifolds with C^∞ transition maps.
- *piecewise linear* (*piecewise Euclidean*) manifolds with piecewise linear transition maps.

Links of simplices $\sigma \in S$ of a triangulation $\mathcal{T}$ which are Piecewise-linearly homeomorphic to a k-sphere are referred to as *combinatorial*. In fact, every topological space which admits a combinatorial triangulation is a (piecewise-linear) manifold. On the other hand, every piecewise-linear manifold can be shown to admit a combinatorial triangulation [22].

On the verge of the 20th century, Poincaré himself asked a question whether every smooth manifold admits a triangulation [28], until Cairns [6] (in 1935) and Whitehead [38] (in 1940) showed that every C^∞-manifold has a unique piecewise-linear structure, and therefore is triangulable. A question which followed

[1] $\mathcal{P}(V) = 2^V$ denotes a *power set*, i.e.: the set of all subsets of V.

after Poincaré's initial investigation, whether every topological manifold has a triangulation [19] was far more difficult to answer. In fact, it was one of open problems in topology, called the *triangulation conjecture*. It turns out that the answer depends on the dimension of a particular manifold. The conjecture is trivial for 0 and 1-dimensional manifolds. Radó [29] and Moise [27] proved that for $\dim X = 2, 3$ there always exists a triangulation. The circumstances change for $\dim X = 4$. In 1990, for example, Freedman's E_8 manifold has been shown to be an exception [9]. For $\dim X \geq 5$, Manolescu [22] refers to his 2013 work on Pin(2)-equivariant Steinberg-Witten Floer homology and the triangulation conjecture, and shows that general 5 and higher-dimensional manifolds do not admit a triangulation. A similar restriction to manifolds of dimension up to 3 follows for the existence of a piecewise-linear atlas.

Although there is no agreed upon term for it, for $\dim X = 2$ a homeomorphism $\mathcal{Q} : X \to K_Q$ onto a complex $K_Q = (V, S_Q)$ of quadrilateral elements $q = \{v_1, v_2, v_3, v_4\} \in S_Q$ is sometimes referred to as a *quadrangulation* of X. The 3-dimensional analogue produces a hexahedral (cuboid) tesselation of a 3-manifold, but the practical computational process is only referred to as hexahedral meshing. Theoretically, the representation can be extended to all polytopes (polygons) even with holes, but with increasing topological complexity, the piecewise-linear aproximations using such structures become impractical to handle in computational scenarios. We will, however, use a general definition of objects used throughout this work:

Definition 2.1.4. Let X be a smooth manifold, and $\mathcal{P} : X \to \overline{X}$ a (polytope) homeomorphism onto a piecewise linear manifold $\overline{X}$ then $\overline{X}$ is called a *mesh* of X.

Since $\overline{X}$ is a piecewise linear manifold, it is homeomorphic to a simplical complex $K_P = (V, F)$, where V is the set of vertices $V \subset X$, and F the set of polytope cells $f = \{v_1, ..., v_m\}, m \in \mathbb{N}$. For $\dim X = 2$ cells f are called *faces* of $\overline{X}$. Since vertices $v \in V$ lie on X we can say that $\overline{X}$ is a piecewise linear approximation of X.

In this work we deal with manifold geometries, that is: meshes of smooth manifolds. A general 3D model in CG modeling can, of course, be a collection $\bigcup_i \overline{X}_i$ of polygonal 2-dimensional meshes of not necessarily the same manifolds.

2.1.1 Piecewise-Linear Metric

Manifolds are fully abstract objects endowed with the property that their individual parts can be given Euclidean coordinates. What we see when observing a surface mesh representation in $\overline{X} \subset \mathbb{R}^3$ is a particular embedding of parts of $\mathcal{U} \subseteq X$ as piecewise Euclidean (linear) submanifolds - simplices.

Practically, a metric $g_X : TX \times TX \to \mathbb{R}$ in a discrete setting represented by a piecewise linear mesh $\overline{X}$ inherits the metric of the manifold's ambient space $\mathbb{R}^n$, but only for the interior of individual simplices. Tangent vectors on boundaries ∂f of elements $f \in F$ are restricted to tangent spaces with the dimension of boundary. For paths passing through simplex boundaries, there is no definite tangent vector. However, for increasing discretization detail, they are expected to converge to tangent vectors $v \in TX$ of smooth manifold X from both sides of the boundary ∂f.

Therefore, all distances, angles and areas will be computed using the standard Euclidean metric $g_{\mathbb{R}^n} = \langle \cdot , \cdot \rangle$ (the (*inner*) *dot product*).

2.2 Manifold and Simplex Orientation

Orientation of mesh elements is essential for computational geometry, since it determines the directions of unit normal vectors to individual polytopes, and for the entire linear approximation of a manifold. Normal

vectors to a 2-mesh $\overline{X}$ affect the colors of triangles with respect to a given lighting configuration. This is vital for path-tracing as well as for ordinary shaders which process individual image pixels in real time.

The *orientation* of a manifold X is described as an equivalence class $[\varpi]$ of *volume forms* on X. Simply put, a manifold equipped with an orientation is a manifold where volumes are computed via inner products with basis vectors in a particular order. A permutation of inputs with an odd number of inversions changes the sign of this volume. For a particular triangulation $\mathcal{T}$ of X the simplices $\sigma \in S$ inherit the manifold's orientation on their boundaries in the same way as an oriented manifold with boundary induces an orientation on its boundary.

Definition 2.2.1. An *oriented* k-simplex Σ is an ordered k-tuple of oriented $(k-1)$-simplices: $\Sigma = (\sigma_1, ..., \sigma_k)$, all the way down to 0-simplices $(v) = \{v\}$ (which do not have an orientation). Multiplying an oriented k-simplex: $\lambda\Sigma$ by a scalar value returns an oriented simplex of the opposite orientation when $\lambda < 0$.

The simplest oriented simplex we can construct is an *edge*. Take vertices $v_1, v_2 \in V$ and their corresponding 0-simplices $\{v_1\}, \{v_2\} \in S$. We can easily induce orientation on an edge $\{v_1, v_2\}$ (1-simplex) by writing it as an ordered pair: (v_1, v_2). Additional complexity requires a more elaborate tool. An *oriented edge* $(v_1, v_2) = e \in S$ can be created by "adding" 0-simplices $\{v_1\}, \{v_2\} \in S$ using *concatenation*: $\{v_1\} \oplus \{v_2\}$, but individual edges $e_1 = (v_1, v_2), e_2 = (v_2, v_3)$ can also be added using the same operator, and they will produce a connected oriented polytope - a 1-*chain* - represented as a triple (v_1, v_2, v_3), as long as they share a vertex. Adding a sequence of oriented edges: $e_1 \oplus e_2 \oplus ... \oplus e_m \oplus e_1$ will produce an *oriented boundary* ∂f of a 2-polytope (polygon) f which can also be represented as an m-tuple $(v_1, v_2, ..., v_m)$ with the additional property that the first and the last vertices v_1 and v_m are also connected by an oriented edge (v_m, v_1).

Extending the concatenation operator to polygons f, we can create 3-polytopes generated by adding polygons as long as they share edges in such way that every polygon f_1 has an oriented connection to another polygon f_2 (sharing at least one edge). The amount of possible resulting geometries matches all geometries that can be created using 2-dimensional surface meshes.

Definition 2.2.2. Let $\sigma_1, \sigma_2 \in S$ be k-simplices of a simplical complex $K = (V, S)$. A *concatenation operator* $\oplus : S \times S \to K$ returns an *oriented* k-polytope (σ_1, σ_2) if $\sigma_1 \cap \sigma_2$ is a $(k-1)$-polytope.

There is no reason to define $\oplus$ for $(k-1)$-disjoint simplices, since there is no uniqueness in inducing orientation for example from a polygon to another which only share a single vertex, let alone for fully disconnected configurations.

A boundary of an oriented k-simplex $\Sigma = (\sigma_1, ..., \sigma_k) \in S$ is usually defined as

$$\partial\Sigma = \partial(\sigma_1, ..., \sigma_k) = \bigoplus_{j=1}^{k} (\sigma_1, ..., \sigma_{j-1}, \sigma_{j+1}, ..., \sigma_k) \tag{2.1}$$

However, Desbrun et. al. [11] defines the *boundary operator* ∂ on (oriented) 2-simplices as follows:

$$\partial(v_1, ..., v_k) = \bigoplus_{j=1}^{k} (-1)^{j-1} (v_1, ..., v_{j-1}, v_{j+1}, ..., v_k)$$

to account for possible orientation errors in lower 1-simplices (edges). For example, the boundary of triangle $\{v_1, v_2, v_3\}$ with oriented edges (v_2, v_3), (v_3, v_1), and (v_2, v_1) in Fig. 2.4 will have correct orientation by using signed concatenation: $\partial\{v_1, v_2, v_3\} = (v_2, v_3) \ominus (v_1, v_3) \oplus (v_1, v_2)$.

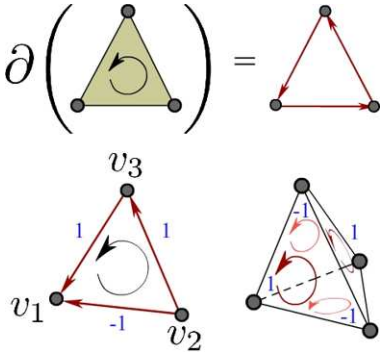

Figure 2.4: A boundary operator on simplices, modified for possible orientation errors.

2.3 Implementation Conventions

A good practice for developing a computational mesh framework involves saving as much memory and storage space, as possible. Meshes $\overline{X} \subset \mathbb{R}^n$ with large amounts of vertices already require one to store at least $N_V n$ (where N_V is the vertex count) floating point values. Individual vertices have to be stored in an array of size N_V containing vector objects, or as a flat array of $3N_V$ values, considering that vectors can be accessed individually with a stride of 3. Polygons are then represented as tuples (arrays) of indices, that is: positions in the vertex array.

It is often the case that a mesh geometry object stores overlapping triples of indices corresponding to triangles, and on top of that there is a dynamic array of "triangulations" which stores indices of triangles adjacent within polygons. Retrieving indices of boundary vertices of a polygon given by a triangulation array of triangles usually requires specialized procedures. WebGL frameworks such as `three.js` even go as far as storing all vertex coordinates in a linear array as many times as they are indexed in a triangle index buffer, in their indexing order.

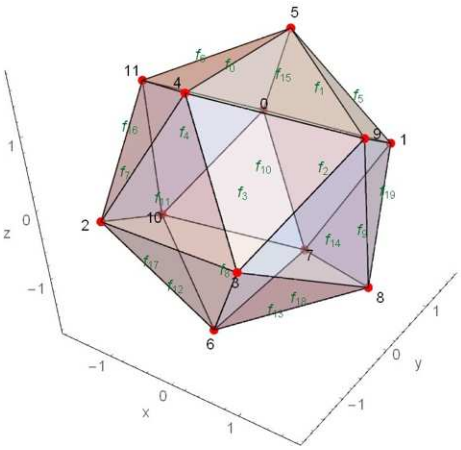

Figure 2.5: Indexing of an icosahedron mesh.

Due to the properties of the Euclidean cross product between polygonal edge vectors, the order of multiplication matters, and produces two outcomes with opposite signs. In order to simulate the light interaction properties of a real-world surface, for meshes of closed manifolds without boundary, the surface normals are supposed to point outwards, and for that reason polygons in polygonal meshes are positively oriented.

The `C++` implementation for this thesis uses the following definition of a polygonal mesh:

$$\begin{aligned}\overline{X} = (V, S, T)\ &, \text{ where } V = (\mathbf{v}_1, ..., \mathbf{v}_{N_V}),\ \mathbf{v}_i \in \mathbb{R}^3,\\ &S = \big((i_{1,1}, i_{1,2}, i_{1,3}), ..., (i_{N_T,1}, i_{N_T,2}, i_{N_T,3})\big)\\ &\text{with } T = \big((t_{1,j_1}, ..., t_{1,j_q}), ..., (t_{N_P,j_1}, ..., t_{N_P,j_q})\big),\ q = 1, 2, ...\end{aligned}$$

with additional data, such as vertex normals and edges, derived from the essential data when necessary. Array V is the unique vertex array, S the array of N_T 2-simplices (triangles), and T the array of N_P triangulation indices (polygons - triangles for $q = 1$ and quads for $q = 2$).

As an example take a regular icosahedron mesh $\overline{X}$ as an approximation of $\mathbb{S}^2$ as in Fig. 2.5. Since the length between two opposing edges is given by the golden ratio: $\varphi = \frac{1}{2}(1 + \sqrt{5})$, we can put

$$\begin{aligned}V = &\big((-1, \varphi, 0), (1, \varphi, 0), (-1, -\varphi, 0), (1, -\varphi, 0), (0, -1, \varphi), (0, 1, \varphi), (0, -1, -\varphi), (0, 1, -\varphi), (\varphi, 0, -1),\\ &(\varphi, 0, 1), (-\varphi, 0, -1), (-\varphi, 0, 1)\big)\\ T = &\big((0, 11, 5), (0, 5, 1), (0, 1, 7), (0, 7, 10), (0, 10, 11), (1, 5, 9),\\ &(5, 11, 4), (11, 10, 2), (10, 7, 6), (7, 1, 8), (3, 9, 4), (3, 4, 2),\\ &(3, 2, 6), (3, 6, 8), (3, 8, 9), (4, 9, 5), (2, 4, 11), (6, 2, 10),\\ &(8, 6, 7), (9, 8, 1)\big)\end{aligned}$$

while indexing vertices from 0.

Chapter 3

Manifold Evolution

Because surface meshes are piecewise-linear approximations of 2-manifolds, there is a possibility of acquiring an approximation of higher quality, by using a process which manifests itself in various natural phenomena. Almost all systems we describe by modern mathematics tend to minimize some form of energy functional. Elastic membranes, such as soap foam bubbles, admit the shape which minimizes the energetic costs of the material. Closed bubbles tend to fluctuate around a spherical equilibrium, while configurations fixed by Dirichlet boundary conditions admit a form which will later be described as a *minimal surface*. The transient formulation of the energy minimization problem reduces to a parabolic geometric flow of the manifold of interest.

Manifolds are entirely abstract objects whose "shape" is determined only locally by a metric (inner product on tangent vector spaces), or a connection (a generalized differential operator on tangent vector or tensor fields). Formally, the *manifold evolution* problem requires a target space Y into which our manifold of interest X is *immersed*[1] so that it acquires all the additional differential properties (for example the unit normal vector field ν), additional to intrinsic attributes (such as metric, or a connection). And while it is entirely possible to describe manifold evolution in terms of fully intrinsic properties, for the purposes of computational geometry, a formulation with some essential extrinsic features suffices.

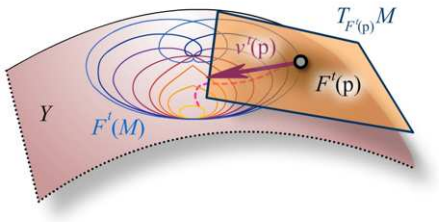

Figure 3.1: An evolving curve in manifold Y with velocity v^t at each point.

Definition 3.0.1. Consider *Riemannian manifolds* (X, g_X) (possibly with a boundary ∂X) and (Y, g_Y) such that $m = \dim X \leq \dim Y = n$. The *evolution of* X *in* Y is a smooth map:

$$F : X \times [0, t_s] \to Y \tag{3.1}$$

such that $F^t = F(\cdot, t)$ is an *immersion* for every $t \in [0, t_s]$.

(3.1) is the solution to a parabolic evolution problem $\partial_t F = v$ with additional information on the nature of velocity v including boundary and initial conditions. It should be noted that by F^t being an immersion, self-intersections in the ambient manifold Y are, for all intents and purposes, not excluded.

The velocity of immersion F^t can be decomposed into a *normal* and a *tangential* component for all points and all $t \in [0, t_s]$:

$$\partial_t F = v_N + v_T \tag{3.2}$$

[1] a map $F : X \to Y$ whose push-forward (differential) $F_* = \mathrm{d}F : TX \to TY$ is everywhere injective.

Both vector fields are smooth sections of tangent bundle TY. For every point $p \in X$ the tangential velocity v_T lies in $(F^t)_*(T_pX) \subset T_{F^t(p)}Y$ while the normal velocity $v_N \in (F^t)_*(T_pX)^\perp \subset T_{F^t(p)}Y$ (the orthocomplement of the push-forward of tangent space T_pX along F^t).

Evolution equations of form (3.2) are used in a plethora of mathematical models in physics, computer vision, image processing, and biology. Notable examples include phase interface evolution, forest fire front propagation, image segmentation, modeling of molecular surfaces, and minimal surface construction in architecture [24]. The practical approaches to solving evolution equations of this type can be divided into two categories:

- *Eulerian* (level set) approach - considers immersions $\mathrm{Im}(F^t)$ to be level sets of a function (of n variables) on Y.
- *Lagrangian* approach - evolves points of $\mathrm{Im}(F^t)$ directly.

Daniel et al. [10] use the Lagrangian approach to reconstruct surface models from point cloud data. In this work we will adopt the same method with suitable modifications for input mesh data.

3.1 Mean Curvature Flow

Diffusion is a well-known natural process with a multitude of manifestations in which it evolves a system from a lower entropy state towards a state of maximum entropy. In other words, it evolves a system into its steady state, or an equilibrium[2]. A "textbook" example is the linear heat equation $\partial_t u - \Delta u = 0$ on $\Omega \subseteq \mathbb{R}^n$ with initial condition u_0 and boundary conditions on $\partial\Omega$. Its various mutations found their use also in models of non-physical phenomena, such as image processing [18] or optimal evaluation of financial derivatives [4].

The heat equation has specific properties which apply even to more general parabolic equations:

- *The Maximum Principle*: At a point $x^* \in \Omega$ where $\partial_t u$ admits a maximum, the second derivatives are non-positive, hence the time derivative is non-positive. It follows that the maximum temperature $u_{\max}(t) = \sup_{x\in\Omega} u(x,t)$ does not increase for increasing time.
- *Gradient Flow*: The solution u is the flow of the steepest decrease of the *Dirichlet energy*:

$$E(u) = \frac{1}{2}\int_\Omega ||\nabla u||^2 \mathrm{d}x \tag{3.3}$$

The same diffusion equation can be formulated for an immersion F of a smooth manifold X. The Euclidean Laplace operator then changes into its "geometric" counterpart, namely $\Delta_{g_F} F$ - the *Laplace-Beltrami operator*.

For a manifold-valued function $f : X \to \mathbb{R}$ the Laplace-Beltrami operator can be expressed in terms of metric g_X and individual chart components as:

$$\Delta_{g_X} f = g^{ij}\nabla_i\nabla_j f = g^{ij}\left(\frac{\partial^2 f}{\partial x^i \partial x^j} - \Gamma^k_{ik}\frac{\partial f}{\partial x^k}\right) = \frac{1}{\sqrt{|\det g_{ij}|}}\frac{\partial f}{\partial x^i}\left(\sqrt{|\det g_{ij}|}g^{ij}\frac{\partial f}{\partial x^j}\right)$$

Where ∇ is the *Levi-Civita connection* and Γ^k_{ij} the *connection coefficient functions* (*Christoffel symbols*) which arise as a consequence of the product rule. However since Δ_{g_F} acts on an immersion F, it is expressed in terms of its components in Y. Specifically, if F determines a regular parametrization of a surface in

[2] from a physical standpoint, a steady state is not the same as an equilibrium, since the former still involves a flow of some physical quantity, e.g.: heat.

$Y = \mathbb{R}^3$, all of its three components are expressed as: $\Delta_{g_F} F = (\Delta_{g_F} x, \Delta_{g_F} y, \Delta_{g_F} z)$. With a given initial immersion F^0, the resulting evolution equation takes form:

$$\begin{aligned} \partial_t F - \Delta_{g_F} F &= 0 \\ F(\cdot, 0) &= F^0 \end{aligned} \tag{3.4}$$

A classical formula by Weierstrass states that an orientable hypersurface in Euclidean space yields $\Delta_{g_F} F = h$, where $h = H\nu$ is the *mean curvature vector* while ν is the outward-pointing unit normal to $F[X]$ with $H = g^{ij} h_{ij} = \mathrm{tr}(II) = \kappa_1 + ... + \kappa_m$ being the *mean curvature*. Specifically, the *second fundamental form* is defined as $II(\chi, \Upsilon) = -\langle S_\chi \Upsilon, \nu \rangle = \langle \Upsilon, \nabla_\chi \nu \rangle$ where χ and Υ are smooth vector fields from TY, and the *shape operator* $S_\chi \Upsilon = (\nabla_\chi \Upsilon)^\perp$ is an orthogonal projection of the *covariant derivative* $\nabla_\chi \Upsilon$ onto the normal bundle of X. Given a coordinate chart, the components of II are defined as:

$$h_{ij} = -\left\langle \frac{\partial^2 F}{\partial x^j \partial x^i}, \nu \right\rangle_Y = \left\langle \frac{\partial F}{\partial x^i}, \frac{\partial}{\partial x^j} \nu \right\rangle_Y$$

The matrix of the *Weingarten map* $W(\chi) = \nabla_\chi \nu : T_p X \to T_p X$ is given by $h^i_j = g^{il} h_{lj}$. Eigenvalues κ_j of h^i_j are the *principal (geodesic) curvatures* and for $m = 2$ they can be thought of as reciprocal lengths of semi-major axes of an osculating ellipsoid tangent to each point of $F[X]$.

The mean curvature H is known to be the first variation of the area functional $X \mapsto \int_X \mathrm{d}\mu_{g_F^t}$, namely:

$$\partial_t A(\mathcal{U}^t) = -\int_{\mathcal{U}^t} H^2 \mathrm{d}\mu_{g_F^t} \ , \quad \mathcal{U}^t \subseteq F^t[X]$$

In other words, the mean curvature flow is the gradient flow of the area functional. And evolving the manifold in the direction of h will maximally decrease (surface) area.

Theorem 3.1.1. *(Maximum/Comparison Principle) If two hypersurfaces immersed into a Euclidean space are initially disjoint, they remain so. Furthermore, embedded hypersurfaces remain embedded.*

Specifically, the above theorem states that if the initial hypersurface $F[X]$ is convex ($\kappa_j < 0$) then F^t is convex for all $t \in [0, t_s]$. Furthermore, Huisken [2] shows, that:

Theorem 3.1.2. *Convex embedded compact hypersurfaces converge to points. After rescaling to keep the area constant, they smoothly converge to round spheres.*

There are only a few examples of analytic solutions to (3.4). The most elementary is the *shrinking sphere evolution*. Let r_0 be its initial radius. With spherical symmetry, the evolution problem can be transformed into an ordinary differential equation:

$$\frac{\mathrm{d}r}{\mathrm{d}t} = -\frac{m}{r} \ , \quad r(0) = r_0 \tag{3.5}$$

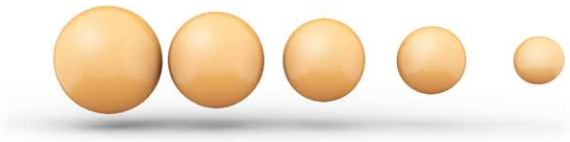

Figure 3.2: A shrinking 2-sphere solution to mean curvature flow in $\mathbb{R}^3$.

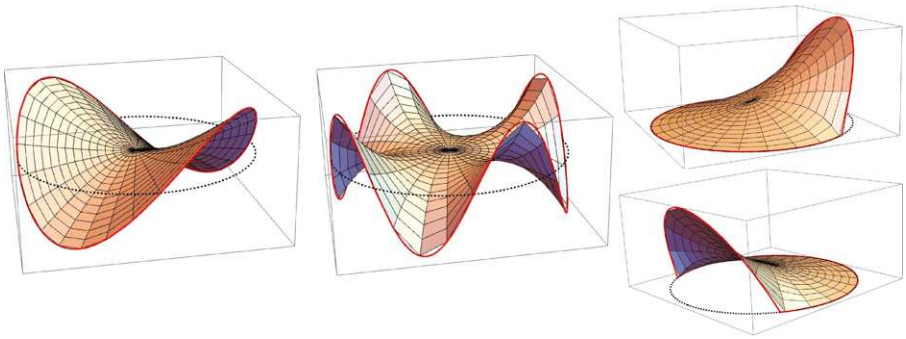

Figure 3.3: Minimal surfaces generated as solutions to a Dirichlet problem on a circle, using the Poisson integral formula in the complex plane.

Where m is the embedding dimension of the sphere. The radial solution r is then obtained by direct integration as: $r(t) = \sqrt{r_0^2 - 2mt}$. Not that the maximum time of existence of the solution r is $t_{\max} = \frac{r_0^2}{2m}$. By only considering $k < m$ radii in the shrinking sphere equation (3.5) we obtain a *shrinking cylinder* solution $r(t) = \sqrt{r_0^2 - 2kt}$. These are examples of, so called, *self-similar shrinkers*, namely solutions to the mean curvature flow (MCF) which only move by scaling.

Another class of examples are *translating solutions* which translate in direction τ, satisfying an elliptic equation $H = -\langle \tau, \nu \rangle$. These involve so called *graphical* MCF *solutions* $u : \mathbb{R}^n \times [0, \infty[\to \mathbb{R}$ to graphical MCF equation: $\partial_t u = \sqrt{1 + ||\nabla u||^2}\ \nabla \cdot \left(\frac{\nabla u}{\sqrt{1+||\nabla u||^2}}\right)$, such as the *grim reaper surface*. Unlike for the shrinking solutions, the existence of translating solutions is not bounded in time (for more details see [15] and [37]).

The behavior of immersions F of 1-manifolds into $\mathbb{R}^2$ (plane curves) has been studied analytically as well as numerically in [16] and [25].

According to numerical experiments [34, 36], given Dirichlet boundary conditions $F\big|_\Gamma = F_\Gamma$ where Γ is a finite set of Jordan curves in $\mathbb{R}^3$ evolves towards a *minimal surface*. There are multiple independent definitions of a minimal surface (an overview of which can be found in [1, 8]). Since it is the limit immersion of MCF, we view a minimal surface as a 2-manifold which minimizes its area functional by vanishing mean curvature:

Definition 3.1.1. A surface $F[X] \subset \mathbb{R}^3$ is *minimal* if $H \equiv 0$ for all $p \in F[X \setminus \partial X]$.

A plane is the most trivial example of a minimal surface. Other such surfaces arise as graphs of solutions to the Laplace equation $\Delta u = 0$ with Dirichlet boundary conditions (see Fig: 3.3), or more generally, solutions to the minimal surface equation

$$\nabla \cdot \left(\frac{\nabla u}{\sqrt{1 + ||\nabla u||^2}} \right) = 0$$

There is no shortage of solutions to the so called *Plateau's problem*, that is: surface immersions into $\mathbb{R}^3$ with Dirichlet boundary conditions with minimized (steady state) mean curvature. Some of which admit rather complicated (even periodic) 3-dimensional shapes [1]. In [31] minimal surfaces are used as base mesh for Riemann surfaces, and also minimal surfaces are used as an initial condition for a remeshing procedure in chapter 3 of [23].

3.2 Evolution In the Normal Direction

Consider (3.4) with an additional advection term:

$$\partial_t F = \epsilon \Delta_{g_F} F + \eta N \tag{3.6}$$

where N is the outward-pointing unit normal to $F[X] = \overline{X}$ and ϵ, η are *control functions* which weigh the effect of mean curvature flow by the Laplace-Beltrami operator Δ_{g_F} and flow in the direction of N respectively. With $\eta \equiv 0$ the evolution becomes a mean curvature flow accelerated by ϵ. The advection term ηN gives rise to more sophisticated applications of surface evolution. Most importantly, it forces a desired shape onto the surface immersion F.

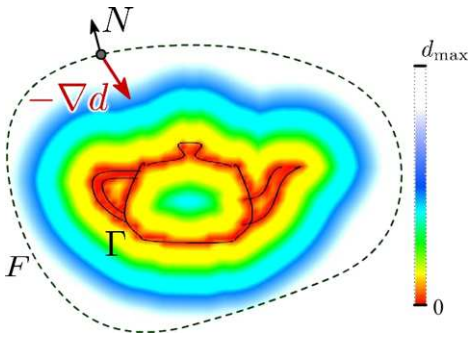

Figure 3.4: One particular form of an edge detector is the distance function.

In section 3.2 Mikula et. al [24] give a particular example from image processing previously used in [26] for the segmentation of cell nuclei images. Given an image intensity function $I : \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}^3$ and an edge detector $e : \Omega \to \mathbb{R}$ of I, in particular $e(x) = \frac{1}{1+K\|\nabla I(x)\|^2}$ for $x \in \Omega$ and $K > 0$, the resulting evolution model (without tangential redistribution) is

$$\partial_t F = b\ e\Delta_{g_F} F + a(\nabla e \cdot N)N\ , \quad a, b > 0$$

Daniel et. al [10] utilize this model for surface reconstruction from point cloud data, replacing the edge detector e with $d = G_\sigma * d^+$ and using gradient field $-\nabla d$ where G_σ is a smoothing Gaussian kernel, and d^+ is the *distance function* to the (sufficiently "dense") target point set forming a closed surface. Since d^+ is not differentiable at the boundary Γ (see Fig. 3.4) of the target surface, we consider a Gaussian-smoothed convolution d.

On the other hand, Medľa [23] (chapter 3) develops a remeshing procedure which first partitions the mesh into topologically simple partitions (a *topology skeleton*), generates minimal surfaces suspended on partition boundaries, and evolves each surface outward towards the target mesh. It turns out that the dot product $(-\nabla d \cdot N)$ is insufficient for configurations where it vanishes ($-\nabla d$ and N are perpendicular). For this reason an additional advection term is used to evolve the surface proportionally to the distance function d, namely:

$$\eta(d(F))N = d(F)\big(|-\nabla d(F)\cdot N| + \sqrt{1-(\nabla d(F)\cdot N)^2}\big)N$$

where the absolute value secures the outward direction of evolution, and subsequent backward relaxation towards the surface in case F overflows the target mesh. The mean curvature control function is constructed so that the surface evolves faster when farther away from the target mesh, in particular:

$$\epsilon(d(F)) := C_1\big(1 - \mathrm{e}^{-\frac{d(F)^2}{C_2}}\big)\ , \quad C_1, C_2 > 0 \tag{3.7}$$

In this work, however, we evolve surfaces inward, that is: we consider the advection control function:

$$\eta(d(F)) := C\ d(F)\big((-\nabla d(F)\cdot N) + D\sqrt{1-(\nabla d(F)\cdot N)^2}\big)\ , \quad C > 0,\ D \geq 0 \tag{3.8}$$

Real parameters C_1, C_2, C, and D are adjustable in the full model:

$$\begin{aligned} \partial_t F &= \epsilon(d(F))\Delta_{g_F} F + \eta(d(F))N \\ F(\cdot, 0) &= F^0 \end{aligned} \tag{3.9}$$

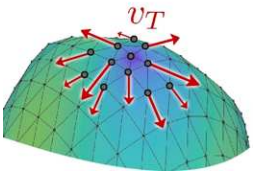

Figure 3.5: Tangential redistribution of surface points.

3.3 Evolution With Tangential Redistribution

The numerical realization of the previously mentioned models encounters an obstacle when we test different configurations and types of immersion F. Since the mean curvature flow evolves parts of the surface with highest mean curvature H fastest, it is not surprising that a discretized surface $\overline{X}$ will evolve into a degenerate configuration with accumulating numerical singularities driven by the underlying flow. Hence the tangential velocity term v_T becomes essential to ensure numerical stability.

The evolution of points $p \in X$ by the means of an immersion gives rise to the evolution of metric g_F induced by F^t. In particular $g_{F^t} = (F^t)^*(g_Y)$ (as a pull-back of g_Y along F^t). Let μ_{g_X} be the measure on the Borel subsets of X induced by g_X. For every $t \in [0, t_s]$ we can define measure $\mu^t_{g_F}$ induced by g_{F^t}, specifically for $U \subseteq X$:

$$\mu^t_{g_F}(U) = \int_U \mathrm{d}\mu^t_{g_F} = \int_U G^t \mathrm{d}\mu_{g_X} \quad \text{with} \quad G^t = \frac{\mathrm{d}\mu^t_{g_F}}{\mathrm{d}\mu_{g_X}} \tag{3.10}$$

Quantity G^t is referred to as the *volume density* of F^t expressing how the immersion shrinks or expands local volumes.

Besides the volume density, any desired metric-induced quantity can be used. We will expand upon the notion of *volume-based* tangential redistribution, as well as *length-based* and *angle-based* approach.

3.3.1 Volume-Based Tangential Redistribution

Let G be a volume density given by (3.10), and $w^t_T = (F^t)^*(v^t_T)$ a vector field given by the tangential velocity field v_T pulled back onto X along F^t. Then G satisfies the following equation:

$$\partial_t G = (-g_Y(h, v_N) + \mathrm{div}_{g_F} w_T)G \tag{3.11}$$

A generalized form of (3.11) is shown in Lemma 5.7 of Bauer, Harms, and Michor [3], yet we adequately resort to our assumption that $Y = \mathbb{R}^3$ with all necessary vector space structure.

Let $F[U]$ be a regular surface patch with chart map (local parametrization) $(\lambda, \zeta) \mapsto F(\lambda, \zeta)$. Then the time derivative of volume density can be obtained by differentiating a differential surface area element:

$$\partial_t G = \partial_t \|\partial_\lambda F \times \partial_\zeta F\| = \frac{\partial_\lambda F \times \partial_\zeta F}{\|\partial_\lambda F \times \partial_\zeta F\|} \cdot \partial_t(\partial_\lambda F \times \partial_\zeta F) \tag{3.12}$$

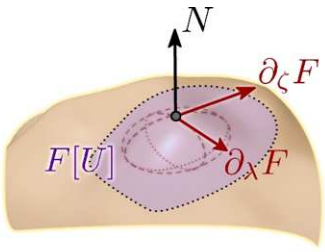

Figure 3.6: A regular surface patch $F[U]$ with a tangent ellipsoid at $p \in F[U]$ locally depicting principal curvatures.

and further expanding the derivative of the cross product:

$$\partial_t(\partial_\lambda F \times \partial_\zeta F) = \partial_{t\lambda} F \times \partial_\zeta F + \partial_\lambda F \times \partial_{t\zeta} F = \partial_\lambda(\partial_t F) \times \partial_\zeta F + \partial_\lambda F \times \partial_\zeta(\partial_t F)$$

where the last form is given by the surface patch regularity when we then apply the Schwarz rule for mixed partial derivatives. At this point, the time derivative of immersion could be a linear combination, say:

$\partial_t F = \alpha_0 N + \alpha_1 \partial_\lambda F + \alpha_2 \partial_\zeta F$ where, in general, $\alpha_j : F[U] \to \mathbb{R},\ j = 0, 1, 2$. Hence upon further expansion:

$$\begin{aligned}\partial_t(\partial_\lambda F \times \partial_\zeta F) &= \partial_\lambda(N + \alpha_1\partial_\lambda F + \alpha_2\partial_\zeta F) \times \partial_\zeta F + \partial_\lambda F \times \partial_\zeta(N + \alpha_1\partial_\lambda F + \alpha_2\partial_\zeta F) = \\ &= \partial_\lambda N \times \partial_\zeta F + \partial_\lambda(\alpha_1\partial_\lambda F + \alpha_2\partial_\zeta F) \times \partial_\lambda F + \\ &\quad \partial_\lambda F \times \partial_\zeta N + \partial_\lambda F \times \partial_\zeta(\alpha_1\partial_\lambda F + \alpha_2\partial_\zeta F)\end{aligned}$$

First, we substitute terms with derivatives of the surface normal N into (3.12) and express:

$$\begin{aligned}&\overbrace{\frac{\partial_\lambda F \times \partial_\zeta F}{\|\partial_\lambda F \times \partial_\zeta F\|}}^{=N} \cdot \big(\partial_\lambda(\alpha_0 N) \times \partial_\zeta F + \partial_\lambda F \times \partial_\zeta(\alpha_0 N)\big) = \\ &N \cdot \big((\partial_\lambda\alpha_0)\cancel{N \times \partial_\zeta F} + \alpha_0\partial_\lambda N \times \partial_\zeta F + (\partial_\zeta\alpha_0)\cancel{\partial_\lambda F \times N} + \alpha_0\partial_\lambda F \times \partial_\zeta N\big) = \\ &\alpha_0 N \cdot \big(\partial_\lambda N \times \partial_\zeta F + \partial_\lambda F \times \partial_\zeta N\big) = -\alpha_0 G(h \cdot v_N) = -\alpha_0 G\ g_Y(h, v_N)\end{aligned} \tag{3.13}$$

and afterwards, we take

$$\begin{aligned}&\frac{\partial_\lambda F \times \partial_\zeta F}{\|\partial_\lambda F \times \partial_\zeta F\|} \cdot \big(\partial_\lambda(\alpha_1\partial_\lambda F + \alpha_2\partial_\zeta F) \times \partial_\lambda F + \partial_\lambda F \times \partial_\zeta(\alpha_1\partial_\lambda F + \alpha_2\partial_\zeta F)\big) = \\ &= \frac{\partial_\lambda F \times \partial_\zeta F}{\|\partial_\lambda F \times \partial_\zeta F\|} \cdot \big((\partial_\lambda\alpha_1\partial\lambda F + \alpha_1\partial_{\lambda\lambda}F + \partial_\lambda\alpha_2\partial_\zeta F + \alpha_2\partial_{\zeta\lambda}F) \times \partial_\zeta F + \\ &\qquad \partial_\lambda F \times (\partial_\zeta\alpha_1\partial_\lambda F + \alpha_1\partial_{\lambda\zeta}F + \partial_\zeta\alpha_2\partial_\zeta F + \alpha_2\partial_{\zeta\zeta}F)\big) = \\ &= \frac{\partial_\lambda F \times \partial_\zeta F}{\|\partial_\lambda F \times \partial_\zeta F\|} \cdot \big(\partial_\lambda\alpha_1\partial_\lambda F \times \partial_\zeta F + \alpha_1\partial_{\lambda\lambda}F \times \partial_\zeta F + \partial_\lambda\alpha_2 \overbrace{\partial_\zeta F \times \partial_\zeta F}^{=0} + \alpha_2\partial_{\zeta\lambda}F \times \partial_\zeta F + \\ &\qquad \partial_\zeta\alpha_1 \overbrace{\partial_\lambda F \times \partial_\lambda F}^{=0} + \alpha_1\partial_\lambda F \times \partial_{\lambda\zeta}F + \partial_\zeta\alpha_2\partial_\lambda F \times \partial_\zeta F + \alpha_2\partial_\lambda F \times \partial_{\zeta\zeta}F\big) = \\ &\qquad \bigg(\partial_\lambda\alpha_1\|\partial_\lambda F \times \partial_\zeta F\| + \alpha_1\frac{(\partial_\lambda F \times \partial_\zeta F) \cdot (\partial_{\lambda\lambda}F \times \partial_\zeta F + \partial_\lambda F \times \partial_{\lambda\zeta}F)}{\|\partial_\lambda F \times \partial_\zeta F\|} + \\ &\qquad \partial_\zeta\alpha_2\|\partial_\lambda F \times \partial_\zeta F\| + \alpha_2\frac{(\partial_\lambda F \times \partial_\zeta F) \cdot (\partial_\lambda F \times \partial_{\zeta\zeta}F + \partial_{\zeta\lambda}F \times \partial_\zeta F)}{\|\partial_\lambda F \times \partial_\zeta F\|}\bigg) = \\ &= \bigg(\partial_\lambda\alpha_1\|\partial_\lambda F \times \partial_\zeta F\| + \alpha_1\frac{(\partial_\lambda F \times \partial_\zeta F) \cdot \partial_\lambda(\partial_\lambda F \times \partial_\zeta F)}{\|\partial_\lambda F \times \partial_\zeta F\|} + \partial_\zeta\alpha_2\|\partial_\lambda F \times \partial_\zeta F\| + \alpha_2\frac{(\partial_\lambda F \times \partial_\zeta F) \cdot \partial_\zeta(\partial_\lambda F \times \partial_\zeta F)}{\|\partial_\lambda F \times \partial_\zeta F\|}\bigg) = \\ &= \big(\partial_\lambda\alpha_1\|\partial_\lambda F \times \partial_\zeta F\| + \alpha_1\partial_\lambda\|\partial_\lambda F \times \partial_\zeta F\| + \partial_\zeta\alpha_2\|\partial_\lambda F \times \partial_\zeta F\| + \alpha_2\partial_\zeta F\|\partial_\lambda F \times \partial_\zeta F\|\big) = \\ &\quad = \frac{\|\partial_\lambda F \times \partial_\zeta F\|}{\|\partial_\lambda F \times \partial_\zeta F\|}\big(\partial_\lambda(\alpha_1\|\partial_\lambda F \times \partial_\zeta F\|) + \partial_\zeta(\alpha_2\|\partial_\lambda F \times \partial_\zeta F\|)\big) = G\ \nabla \cdot w_T = G\ \mathrm{div}_{g_F} w_T\end{aligned} \tag{3.14}$$

Taking $\alpha_0 \equiv 1$ and finally combining (3.13) with (3.14) yields $\partial_t G = G(-g_Y(h, v_N) + \mathrm{div}_{g_F} w_T)$. The dot product $(h \cdot v_N)$ becomes the mean curvature scalar: $H = g_Y(h, v_N)$ when we consider the mean curvature flow model: $\partial_t F = h$.

Now consider area measure A^t of X evolving in time. Then

$$\partial_t A = \int_X (-g_Y(h, v_N) + \mathrm{div}_{g_F} w_T)\mathrm{d}\mu^t_{g_F} = \int_X (-g_Y(h, v_N))\mathrm{d}\mu^t_{g_F} + \int_{\partial X} g_F(w_T, \nu)\mathrm{d}H_{\mu^t_F} \tag{3.15}$$

where ν is the outward-pointing unit normal to the manifold's boundary ∂X with respect to g_F and $H_{\mu^t_F}$ is the $(\dim X - 1)$-dimensional Hausdorff measure on the boundary. Clearly, when $\partial X = \varnothing$ (X is without boundary) or $w_T\big|_{\partial X}$ is tangential (that is: $g_F(w_T, \nu) = 0$), the boundary integral vanishes.

To control volume density, we require the conservation of *relative volume* of any surface patch $U \subseteq X$:

$$\frac{\mu^t_{g_F}(U)}{A^t} = \frac{\mu^0_{g_F}(U)}{A^0} \ , \quad \text{for any } t \in [0, t_s]$$

which, of course, implies the conservation of volume density:

$$\frac{G^t(U)}{A^t} = \frac{G^0(U)}{A^0} \ \Rightarrow \ \partial_t\left(\frac{G}{A}\right) = 0 \tag{3.16}$$

for almost all $p \in X$. Combining the above law with (3.11) and (3.15) we get

$$\begin{aligned}
0 = \partial_t\left(\frac{G}{A}\right) &= \frac{1}{A^2}\big(\partial_t G \ A - G \ \partial_t A\big) = \\
&= \frac{1}{A^2}\left((-g_Y(h, v_N) + \mathrm{div}_{g_F} w_T)GA - G\left[\int_X (-g_Y(h, v_N))\mathrm{d}\mu^t_{g_F} + \int_{\partial X} g_F(w_T, \nu)\mathrm{d}H_{\mu^t_F}\right]\right) = \\
&= -g_Y(h, v_N) + \mathrm{div}_{g_F} w_T + \frac{1}{A}\int_X g_Y(h, v_N)\mathrm{d}\mu^t_{g_F} - \frac{1}{A}\int_{\partial X} g_F(w_T, \nu)\mathrm{d}H_{\mu^t_F} \\
\Rightarrow \qquad & \mathrm{div}_{g_F} w_T = g_Y(h, v_N) - \frac{1}{A}\int_X g_Y(h, v_N)\mathrm{d}\mu^t_{g_F} + \frac{1}{A}\int_{\partial X} g_F(w_T, \nu)\mathrm{d}H_{\mu^t_F}
\end{aligned} \tag{3.17}$$

Condition (3.16) achieves a conservation of relative volume density based on the initial immersion $F^0[X]$ (the initial mesh $\overline{X}^0$). However if we are not satisfied with the initial state, the above redistribution will not improve the configuration. For this reason, we define an immersion F^t *volume-uniform with respect to* g_X:

$$\frac{G^t(p)}{A^t} = C \ , \quad C > 0$$

for almost all $p \in X$. An *asymptotically uniform evolution* F then satisfies

$$\frac{G^t}{A^t} \underset{t\to\infty}{\longrightarrow} C$$

which can be achieved, for instance, when

$$\partial_t\left(\frac{G}{A}\right) = \left(C - \frac{G}{A}\right)\omega \ , \quad \omega : [0, t_s] \to \mathbb{R}^+ \tag{3.18}$$

or the global volume A might converge to a given value A_∞ with:

$$\frac{\partial_t G}{A_\infty} = \left(C - \frac{G}{A_\infty}\right)\omega$$

Using the same substitution as in (3.17) with (3.18) gives

$$\mathrm{div}_{g_F} w_T = g_Y(h, v_N) - \frac{1}{A}\int_X g_Y(h, v_N)\mathrm{d}\mu^t_{g_F} + \frac{1}{A}\int_{\partial X} g_F(w_T, \nu)\mathrm{d}H_{\mu^t_F} + \left(C\frac{A}{G} - 1\right)\omega, \ C > 0$$

Now we are left with the task of finding a suitable vector field w_T on X. The problem at hand is that there are uncountably many ways of pushing an arbitrary smooth section w_T on X forward along F to obtain v_T. One possibility is to find a unique gradient field $w_T = \nabla_{g_F}\psi$ of some *redistribution potential* $\psi : X \to \mathbb{R}$ for each $t \in [0, t_s]$. More precisely, taking a smooth section $\chi \in TX$ and $g_F(\nabla_{g_F}\psi, \chi) = \frac{\partial\psi}{\partial\chi}$

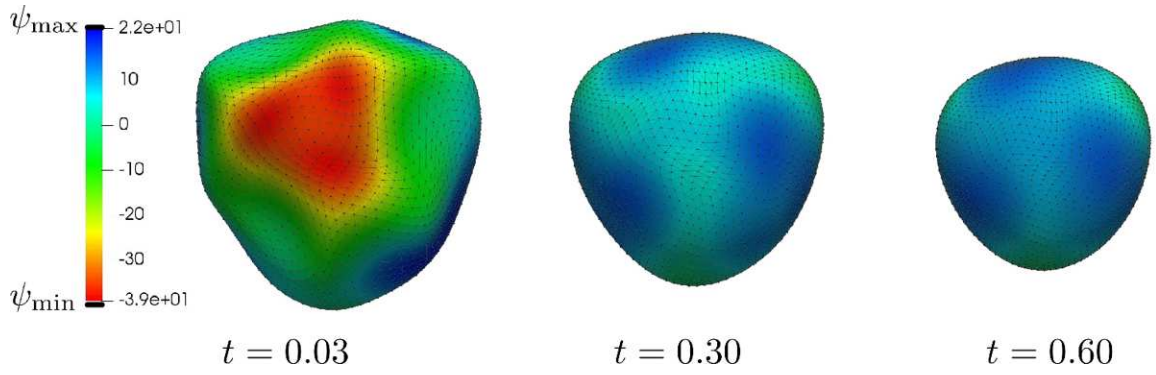

Figure 3.7: Redistribution potential ψ on a closed surface evolving under mean curvature flow with $\omega = 200$.

defines its directional derivative, and with $\mathrm{div}_{g_F} w_T = \mathrm{div}_{g_F} \nabla_{g_F} \psi = \Delta_{g_F} \psi$ we acquire an elliptic problem for the redistribution potential:

$$\Delta_{g_F} \psi = g_Y(h, v_N) - \frac{1}{A} \int_X g_Y(h, v_N) \mathrm{d}\mu_{g_F} + \frac{1}{A} \int_{\partial X} g_F(\nabla_{g_F} \psi, \nu) \mathrm{d}H_{\mu_F} + \left(C \frac{A}{G} - 1 \right) \omega \tag{3.19}$$

for each $t \in [0, t_s]$.

Furthermore, we choose $C = 1/A_X$ according to Daniel et. al [10] where $G \to A/A_X$ as $t \to \infty$ where

$$A_X = \mu_X(X) = \int_X \mathrm{d}\mu_X \qquad \text{and} \qquad A = \mu_{g_F}(X) = \int_X G \, \mathrm{d}\mu_X \tag{3.20}$$

which secures area-uniformity of the redistribution. An immersion of this type is a convenient approach to construct meshes with uniformly-sized polygonal elements, specifically for target surfaces without dominant concavities. However, to represent more intricate features of most target meshes, we might need to concentrate more mesh vertices into areas with high curvature. In particular it is desired to find a tangential velocity field v_T such that for G constant over a surface patch $U \subset X$ we have $\mu_{g_F}(U) = G\mu_X(U)$, which implies that higher value of G causes local expansion of area while lower G shrinks surface patches. One possibility is taking $G(p)f(H(p)) = \text{const}$ for every point $p \in X$ at all times, where f is a positive increasing function of mean curvature scalar H which can take forms

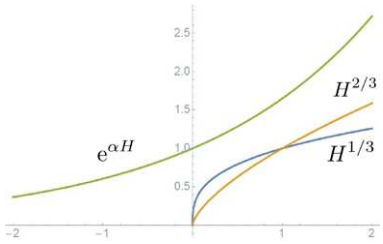

Figure 3.8: Mean curvature control function types.

$$\begin{aligned} f(H) &= H^{1/3} \\ f(H) &= H^{2/3} \\ f(H) &= \mathrm{e}^{\alpha H} \ , \quad \alpha > 0 \end{aligned} \tag{3.21}$$

and consequently

$$\frac{G^t}{A^t} \underset{t \to \infty}{\longrightarrow} C = \frac{\frac{1}{f(H)}}{\int_X \frac{1}{f(H)} \mathrm{d}\mu_X} \tag{3.22}$$

For manifolds with boundary $\psi\big|_{\partial X}$ needs to be prescribed. For example when constructing minimal surfaces as in [34] we impose zero Neumann condition $g_F(\nabla_{g_F}\psi,\nu)\big|_{\partial X}=0$. Also for closed manifolds, we need to ensure uniqueness of solution ψ by defining it at one selected point (say, for example, that $\psi(v_1)=0$ for the first mesh vertex $v_1\in\overline{X}$).

Figure 3.7 shows a mean curvature flow evolution of a closed surface with the values of redistribution potential ψ. The initial distorted state has much larger gradient $\nabla_{g_F}\psi$ in some parts. The volume-based redistribution of points spreads individual vertices apart in such way that the potential ψ becomes increasingly more homogeneous.

3.3.2 Length-Based Tangential Redistribution

A natural approach to controlling distances on a manifold is to focus on the evolution of curves $\gamma:[0,1]\to X$ whose images can be thought of as 1-manifolds immersed into Y. Considering a single curve will simplify the above procedure, since we can directly obtain the tangential velocity given by $\mathrm{div}_{g_F}w_T$ and an appropriate boundary condition:

$$v_T=\|v_T\|_{g_Y}\frac{v_T}{\|v_T\|_{g_Y}}=\alpha T_Y$$

where $\alpha:\mathrm{Im}(\gamma)\times[0,t_s]\to\mathbb{R}$ and $T_Y=F_*(T_F)$ with unit vector field T_F on $\mathrm{Im}(\gamma)$ with respect to metric g_F. The divergence of w_T then becomes a directional derivative:

$$\mathrm{div}_{g_F}w_T=\frac{\partial\alpha}{\partial T_F}$$

substituting into (3.17) and adding the term for asymptotically uniform redistribution gives

$$\frac{\partial\alpha}{\partial T_F}=g_Y(v_N,h)-\frac{1}{L(\gamma)}\int_X g_Y(h,v_N)\mathrm{d}\mu^t_{g_F}+\frac{\alpha(\gamma_1)-\alpha(\gamma_0)}{L(\gamma)}+\left(C\frac{A}{G}-1\right)\omega \tag{3.23}$$

where γ_0 and γ_1 are endpoints of $\mathrm{Im}(\gamma)$ (if any). It is clear that in order to obtain a unique solution to (3.23), it suffices to define the value of α at one of the points in $\mathrm{Im}(\gamma)$.

For curves γ and σ intersecting at $p\in X$ the tangential velocity becomes

$$v_T(p,t)=\frac{1}{2}(v_{T,\gamma}(p)+v_{T,\sigma}(p))$$

with components computed individually using (3.23) for each curve. If the tangent vectors of γ and σ at p are linearly independent, then $v_T(p,\cdot)\neq 0$ if the corresponding immersions are non-uniform. We can have up to m curves intersecting at p and still obtain an asymptotically uniform immersion for all of them.

The reasoning extends to a network of intersecting curves on X. If each m-tuple of curves intersecting at p spans T_pX with their velocities, we can apply the asymptotically uniform redistribution to all curves in the network. Specifically, in the discrete case, the intersection points are the mesh vertices.

3.3.3 Angle-Based Tangential Redistribution

Let $\gamma_1,...,\gamma_M$ be a finite set of curves intersecting at $p\in U\subseteq X$ with velocities γ_j' such that no velocity is ever a positive scalar multiple of any other, namely if $\gamma_j'=C\gamma_k'$, then $C<0$. This means that the velocity vectors $\gamma_j'\in T_pX$ form a 1-ring, just like for vertices of a polygonal mesh.

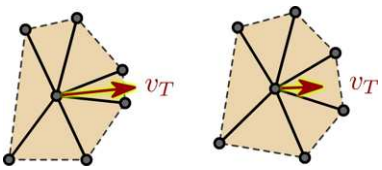

Figure 3.9: Tangential velocity transporting a vertex so that it homogenizes angles with adjacent mesh vertices.

Then we would expect an optimal tangential redistribution to make the angles between two consecutive curve velocities more homogeneous, that is: when more vectors are bundled in one particular direction, the central vertex should move towards it to increase the relative angles of those polygonal regions, and decrease the relative angles of the remaining regions. More specifically

$$\tilde{v}_T = \frac{\omega}{M} \sum_{j=1}^{M} \left(1 + \frac{\gamma_j'}{\|\gamma_j'\|} \cdot \frac{\gamma_{j+1}'}{\|\gamma_{j+1}'\|}\right)(\gamma_j' + \gamma_{j+1}') \tag{3.24}$$

With a control parameter $\omega : [0, t_s] \to \mathbb{R}^+$. Clearly, this only holds for points p in a locally flat neighborhood U. For discrete configurations we count edges e_j of a polygonal 1-ring which are not necessarily tangent to the surface at the central vertex. Thus we need to project $\tilde{v}_T$ onto the tangent plane: $v_T = \tilde{v}_T - (N \cdot \tilde{v}_T)N$.

When applied to all mesh vertices, we obtain a tangential velocity vector field.

Chapter 4

A Finite Volume Formulation

Similarly to other numerical techniques for solving partial differential equations, the *finite volume method* subdivides the computational domain into simple elements - *finite volumes*. The unique property of this approach is that its underlying mathematics simulate the differential properties of flow of an unknown quantity, say: temperature, concentration, or mean curvature, in a wide variety of conservation laws.

Note that finite volumes are not necessarily 3-dimensional partitions. They can take form of any (usually the simplest available) partition of the computational domain with a representative point inside where the unknown value is defined. Secondary quantities (for example, heat flux, velocity, etc.) are formulated as normal derivatives of the primary quantity at the boundary of each finite volume. Adjacent volumes then transmit and receive the primary scalar quantity through their boundaries. This leads to a sparse, and usually diagonally dominant, linear system whose matrix elements are non-zero only when they multiply adjacent nodes.

In section 3.3.1, the redistribution potential ψ is a single-valued unknown quantity, but the unknown variable of full surface evolution model:

$$\begin{aligned} \partial_t F &= \epsilon \Delta_{g_F} F + \eta N + v_T \\ F(\cdot, 0) &= F^0 \end{aligned} \tag{4.1}$$

with ϵ and η defined by (3.7) and (3.7), is the immersion $F : X \times [0, t_s] \to \mathbb{R}^3$ which has three real components (within the standard basis) for each representative point.

In this chapter, we formulate the fully discrete finite volume representation of (4.1), and discuss its computational properties.

4.1 Discrete Laplace Operators

Real-valued functions on polygonal meshes $\overline{X}$ are given by their values at individual vertices: $f : V \to \mathbb{R}$ where V is the (ordered) set of mesh vertices. If $N_V = |V|$ is the number of vertices, then a linear operator:

$$\Delta_{\overline{X}} : \mathbb{R}^{N_V} \to \mathbb{R}^{N_V} : f_i \mapsto \sum_{j=1}^{N_V} w_{ij}(f_i - f_j)$$

is a general form of the *discrete Laplace operator on mesh* $\overline{X} = F[X]$ given by an immersion F into $\mathbb{R}^3$. Theoretically, each mesh vertex is influenced by all other vertices via weights w_{ij}. However, due to increasing

computational costs, the influence is generally believed to vanish beyond vertices directly adjacent to i-th vertex.

To achieve satisfying results, the discrete Laplacian should possess as many of the following properties, as possible:

- *Symmetry* - The continuous Laplace operator is *self-adjoint*, and so should be $\Delta_{\overline{X}}$. Namely, it should have real eigenvalues and orthogonal eigenvectors.

- *Locality* - The support for evaluation at each vertex should be small enough to mimic the differential properties, as well as save computation time. Discrete Laplacians are thus generally restricted to Laplacians with 1-ring support, that is: $w_{ij} = 0$ if $j \notin \mathcal{N}(i)$ where $\mathcal{N}(i)$ is the index set of neighboring vertices.

- *Positive Off-Diagonal Weights* - $w_{ij} > 0$ for $i \neq j$ to ensure the maximum principle for discrete harmonic functions. Consider a discrete harmonic function f and $\tilde{w}_{ij} = w_{ij} / \big(\sum_{k \in \mathcal{N}(i)} w_{ik} \big)$. Then $f_i = \sum_{j \in \mathcal{N}(i)} \tilde{w}_{ij} f_j$ with $\tilde{w}_{ij} \in [0, 1]$, there exist $j', j'' \in \mathcal{N}(i)$ for which we have $f_{j'} \leq f_i \leq f_{j''}$, hence the value f_i is never an extremum at any vertex.

- *Positive Semi-Definiteness* - Just like for the continuous Δ operator, the positive semi-definiteness ensures the non-negativity of the Dirichlet energy $E(f) = \langle \Delta_{\overline{X}} f, f \rangle$. Clearly, the smooth Dirichlet energy vanishes for linear functions f. As a consequence: $\dim \left(\mathrm{Ker}(\Delta_{\overline{X}}) \right) = 1$ for a closed mesh.

- *Linear Precision* - The operator should be *linearly precise*, that is: $(\Delta_{\overline{X}} f)_i = 0$ whenever

  (a) all incident polygons to vertex v_i are coplanar.

  (b) f is linear on v_i as well as all $v_j,\ j \in \mathcal{N}(i)$.

  (c) v_i is not a boundary vertex.

  Note that this condition does not contradict positive semi-definiteness because a planar immersion $\overline{X}$ will have boundary vertices and linear functions are not required to vanish at $\partial \overline{X}$ since they will not appear in $\mathrm{Ker}(\Delta_{\overline{X}})$.

- *Convergence* - Clearly, as $N_V \to \infty$, all properties of $\Delta_{\overline{X}}$ should, under reasonable refinement conditions, converge to the continuous Laplacian Δ_X.

The simplest discrete Laplacian is the *umbrella operator*, with weights $w_{ij} \in \{0, 1\}$ such that $w_{ij} = 1$ whenever v_i and v_j share an edge. It satisfies all the properties except linear precision. Another possibility is to normalize the umbrella operator row-wise to produce the *Tutte Laplacian*, which however, destroys symmetry for meshes with irregular connectivity. These are examples of the so called *combinatorial Laplacians*. The compliance of the given discrete operator on meshes relies heavily on their topological properties. Additional forms of $\Delta_{\overline{X}}$ are examined in [13] where the difficulty to satisfy all properties is emphasized.

Since in this work, we utilize the finite volume method (FVM) for solving (3.2) with some initial (and boundary) condition, specifically for triangular and quadrilateral meshes, we require a discrete Laplacian of type:

$$\Delta_{\overline{X}} : \mathbb{R}^{3N_V} \to \mathbb{R}^{3N_V} : F_i \mapsto \sum_{j \in \mathcal{N}_{\triangle,\square}(i)} w_{ij} (F_i - F_j) \tag{4.2}$$

where $F_i \in \mathbb{R}^3$ are mesh vertices and $\mathcal{N}_{\triangle,\square}(i)$ are indices of the vertices neighboring in triangular $\triangle$ or quadrilateral $\square$ mesh elements coincident with vertex F_i. Clearly, linear systems produced by (4.2) can be divided into three independent systems of dimension N_V. Their weights w_{ij} will satisfy the discrete Laplacian properties in such manner that the resulting operator matrix is symmetric, and diagonally dominant.

4.2 The Cotangent Scheme for Triangular Meshes

Also known as the *Desbrun cotan operator*, named by one of the co-authors of [12] - Mathieu Desbrun - is considered to be the most widespread mesh Laplacian as well as mean curvature estimator for triangulated meshes $\overline{X}$. Let V_i be the finite volume generated by the *voronoi region*[1] around vertex F_i restricted to the piecewise-linear metric of $\overline{X}$ (see Fig. 4.1). Desbrun [12] suggests voronoi regions to be computed using a circumcenter of each triangle. This approach has to be corrected for triangles with obtuse angles. Desbrun et. al. point out that using barycenters $B_{i,p}$ of triangles $\mathcal{T}_{i,p}$ is a suitable substitute for fully voronoi finite volume scheme, which is utilized by [24].

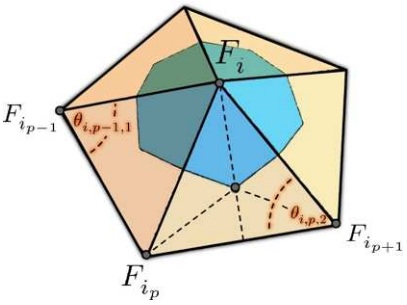

Figure 4.1: The outline of the Desbrun cotan operator on mesh interior vertices and on boundary vertices F_i.

Let F_{i_p} and $F_{i_{p+1}}$ be adjacent mesh vertices forming a triangle $\mathcal{T}_{i,p} = (F_i, F_{i_p}, F_{i_{p+1}})$, and let $F_{i_{p-1}}$ be another vertex adjacent to F_i which forms a triangle $(F_i, F_{i_{p-1}}, F_{i_p})$ neighboring with the former triangle from the left (clockwise direction) along edge (F_i, F_{i_p}).

Our goal is to approximate the integral formulation:

$$\iint_{V_i} h \, \mathrm{d}\mu_{g_F} = \iint_{V_i} \Delta_{g_F} F \, \mathrm{d}\mu_{g_F} = \sum_{p=1}^{m} \iint_{\mathcal{T}_i^p \cap V_i} \Delta_{\lambda,\zeta} F \, \mathrm{d}\lambda \, \mathrm{d}\zeta \qquad (4.3)$$

where μ_F is the measure induced by metric g_F (of immersion F) and (λ, ζ) is the local parametrization of each triangle $\mathcal{T}_i^p$ forming a 1-ring neighborhood (of m triangles), given by $\overline{F}(\lambda, \zeta) = (1-\lambda-\zeta)F_i + \lambda F_{i_p} + \zeta F_{i_{p+1}}$ with $\lambda \in [0,1]$ and $\zeta \in [0, 1-\lambda]$ (see Fig. 4.2).

By the divergence theorem we write (4.3) as

$$\sum_{p=1}^{m} \iint_{\mathcal{T}_i^p \cap V_i} \Delta_{\lambda,\zeta} F \, \mathrm{d}\lambda \, \mathrm{d}\zeta = \sum_{p=1}^{m} \int_{\partial(\mathcal{T}_i^p \cap V_i)} g_F(\nabla_{\lambda,\zeta} F, \nu_{\lambda,\zeta}) \mathrm{d}H_{\mu_{g_F}} = \sum_{p=1}^{m} \int_{\partial(\mathcal{T}_i^p \cap V_i)} \langle \nabla_{\lambda,\zeta} F, \nu_{\lambda,\zeta} \rangle_{\mathbb{R}^3} \mathrm{d}s \qquad (4.4)$$

where $\nu_{\lambda,\zeta}$ is the outward-pointing normal of the finite volume V_i given by (λ, ζ)-parametrization and $H_{\mu_{g_F}}$ is the 1-dimensional Hausdorff measure induced by μ_F which just yields a line integral of an inner product $\langle \nabla_{\lambda,\zeta} F, \nu_{\lambda,\zeta} \rangle$ in $\mathbb{R}^3$.

Regardless of the type of finite volume tessellation of $\overline{X}$, the integral along the boundary $\partial(\mathcal{T}_i^p \cap V_i)$ within a triangle $\mathcal{T}_i^p$ will yield the same result as integrating along an edge between midpoints $M_{i,p,1}$ and $M_{i,p,2}$. The reason for this is mentioned in Appendix A of [12] and [35].

Denote Ω the region enclosed by curve $\partial(\mathcal{T}_i^p \cap V_i)$ and edge $(M_{i,p,1}, M_{i,p,2})$ with a positively oriented boundary. By linearity of the immersion F on each triangle we have

$$0 = \iint_{\Omega} \Delta_{g_F} F \mathrm{d}\mu_{g_F} = \int_{\partial\Omega} g_F(\nabla_{g_F} F, \nu) \mathrm{d}H_{\mu_{g_F}} \qquad (4.5)$$

The boundary integral can then be expanded into

$$\int_{\sigma_p} g_F(\nabla_{g_F} F, -\nu_p) \mathrm{d}H_{\mu_{g_F}} + \sum_{s=1,2} \int_{\sigma_{p,s}} g_F(\nabla_{g_F} F, \nu_{p,s}) \mathrm{d}H_{\mu_{g_F}} \qquad (4.6)$$

Where σ_p is the edge $(M_{i,p,1}, M_{i,p,2})$ and $-\nu_p$ its outward-pointing normal to region Ω, while the boundary partition $\partial(\mathcal{T}_i^p \cap V_i)$ of finite volume V_i is composed of edges $\sigma_{p,1}$ and $\sigma_{p,2}$ with outward-pointing normals $\nu_{p,1}$ and $\nu_{p,2}$.

[1] the set of points whose closest mesh vertex is F_i and no other.

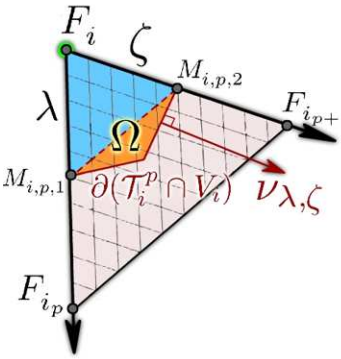

Figure 4.2: A local parametrization of triangle $\mathcal{T}_i^p = (F_i, F_{i_p}, F_{i_{p+1}})$

Since (4.6) given by (4.5) amounts to zero, we move the integral along σ_p to the left-hand side and obtain

$$\int_{\sigma_p} g_F(\nabla_{g_F} F, \nu_p) \mathrm{d}H_{\mu_{g_F}} = \sum_{s=1,2} \int_{\sigma_{p,s}} g_F(\nabla_{g_F} F, \nu_{p,s}) \mathrm{d}H_{\mu_{g_F}} \tag{4.7}$$

The result would hold even if we replaced σ_p by an arbitrary curve contained in the finite volume V_i. Hence (4.4) is path-independent and the "interior sampling point" can either be a barycenter $B_{i,p} = \frac{1}{3}(F_i + F_{i_p} + F_{i_{p+1}})$ or a circumcenter as in [12].

Denote $\nu_p = \left[M_{i,p,1} - M_{i,p,2}\right]^{\perp}_{\lambda,\zeta}$ with respect to parametrization (λ, ζ). Note that operator $[\cdot]_{\lambda,\zeta}$ binds the argument vector onto the plane of triangle $\mathcal{T}_{i,p}$ and $\cdot^{\perp}$ rotates a vector by $\pi/2$ in counter-clockwise direction to obtain a perpendicular vector of the same length. Then by path independence (4.7):

$$\int_{\partial(\mathcal{T}_i^p \cap V_i)} \langle \nabla_{\lambda,\zeta} F, \nu_{\lambda,\zeta} \rangle \mathrm{d}s \approx \nabla_{\lambda,\zeta} F \cdot \left[M_{i,p,1} - M_{i,p,2}\right]^{\perp}_{\lambda,\zeta} = \frac{1}{2} \nabla_{\lambda,\zeta} F \cdot \left[F_{i_p} - F_{i_{p+1}}\right]^{\perp}_{\lambda,\zeta} \tag{4.8}$$

To obtain gradient $\nabla_{g_F} F$ we compute the gradient of parametrization

$$\nabla_{g_F} F = \nabla_{\lambda,\zeta} \overline{F} = \left(\partial_\lambda \overline{F}, \partial_\zeta \overline{F}\right) = \begin{pmatrix} \partial_\lambda \overline{F}_1 & \partial_\zeta \overline{F}_1 \\ \partial_\lambda \overline{F}_2 & \partial_\zeta \overline{F}_2 \\ \partial_\lambda \overline{F}_3 & \partial_\zeta \overline{F}_3 \end{pmatrix} = \begin{pmatrix} F_{i_p,1} - F_{i,1} & F_{i_{p+1},1} - F_{i,1} \\ F_{i_p,2} - F_{i,2} & F_{i_{p+1},2} - F_{i,2} \\ F_{i_p,3} - F_{i,3} & F_{i_{p+1},3} - F_{i,3} \end{pmatrix} = \left(F_{i_p} - F_i, F_{i_{p+1}} - F_i\right) \tag{4.9}$$

which is a differential (push-forward) map $\mathrm{d}\overline{F}(\lambda, \zeta) = F_*\big|_{\mathcal{T}_{i,p}}$ of the immersion on triangle $\mathcal{T}_{i,p}$. The dot product with the gradient (3×2) matrix in (4.8) is then given by

$$\left(F_{i_p} - F_i, F_{i_{p+1}} - F_i\right)^{\top} [F_{i_p} - F_{i_{p+1}}]^{\perp}_{\lambda,\zeta} = \begin{pmatrix} (F_{i_p} - F_i)^{\top} [F_{i_p} - F_{i_{p+1}}]^{\perp}_{\lambda,\zeta} \\ (F_{i_{p+1}} - F_i)^{\top} [F_{i_p} - F_{i_{p+1}}]^{\perp}_{\lambda,\zeta} \end{pmatrix} \tag{4.10}$$

whose two components are the linear combination coefficients such that area-normalized vectors

$$\begin{aligned} \overline{u} &= \frac{1}{\mathcal{A}_{\mathcal{T}_{i,p}}} (M_{i,1,p} - F_i) = \frac{1}{2\mathcal{A}_{\mathcal{T}_{i,p}}} (F_{i_p} - F_i) \\ \overline{v} &= \frac{1}{\mathcal{A}_{\mathcal{T}_{i,p}}} (M_{i,2,p} - F_i) = \frac{1}{2\mathcal{A}_{\mathcal{T}_{i,p}}} (F_{i_{p+1}} - F_i) \end{aligned} \tag{4.11}$$

span points $\overline{F}(\lambda, \zeta) \in \mathcal{T}_{i,p}$. Note that triangle area can be equivalently expressed as

$$\begin{aligned} \mathcal{A}_{\mathcal{T}_{i,p}} &= \frac{1}{2} \|(F_i - F_{i_p}) \times (F_{i_p} - F_{i_{p+1}})\| = \frac{1}{2} \|F_i - F_{i_p}\| \, \|F_{i_p} - F_{i_{p+1}}\| \sin \theta_{i,p,1} = \\ &= \frac{1}{2} \|(F_i - F_{i_{p+1}}) \times (F_{i_p} - F_{i_{p+1}})\| = \frac{1}{2} \|F_i - F_{i_{p+1}}\| \, \|F_{i_p} - F_{i_{p+1}}\| \sin \theta_{i,p,2} \end{aligned} \tag{4.12}$$

Where $\theta_{i,p,1}$ and $\theta_{i,p,2}$ are internal angles at vertices F_{i_p} and $F_{i_{p+1}}$ respectively.

Hence combining (4.9) with (4.11) in approximation (4.8), the integral along finite volume boundary in

(4.4) is given by:

$$\sum_{p=1}^{m} \int_{\partial(\mathcal{T}_i^p \cap V_i)} \langle \nabla_{\lambda,\zeta} F, \nu_{\lambda,\zeta} \rangle \mathrm{d}s \approx$$
$$\approx \frac{1}{2} \sum_{p=1}^{m} \left(\frac{1}{2\mathcal{A}_{\mathcal{T}_{i,p}}} \big[(F_{i_p} - F_i)^\top [F_{i_p} - F_{i_{p+1}}]^\perp_{\lambda,\zeta} \big] (F_{i_p} - F_i) + \frac{1}{2\mathcal{A}_{\mathcal{T}_{i,p}}} \big[(F_{i_{p+1}} - F_i)^\top [F_{i_p} - F_{i_{p+1}}]^\perp_{\lambda,\zeta} \big] (F_{i_{p+1}} - F_i) \right) =$$
$$= \frac{1}{2} \sum_{p=1}^{m} \left(\frac{\|F_{i_p} - F_i\| \, \|F_{i_p} - F_{i_{p+1}}\| \cos\theta_{i,p,1}}{\|F_i - F_{i_p}\| \, \|F_{i_p} - F_{i_{p+1}}\| \sin\theta_{i,p,1}} (F_{i_p} - F_i) + \frac{\|F_i - F_{i_{p+1}}\| \, \|F_{i_{p+1}} - F_{i_p}\| \cos\theta_{i,p,2}}{\|F_i - F_{i_{p+1}}\| \, \|F_{i_p} - F_{i_{p+1}}\| \sin\theta_{i,p,2}} (F_{i_{p+1}} - F_i) \right) =$$
$$= \frac{1}{2} \sum_{p=1}^{m} \big(\cot\theta_{i,p,1} (F_{i_p} - F_i) + \cot\theta_{i,p,2} (F_{i_{p+1}} - F_i) \big) \tag{4.13}$$

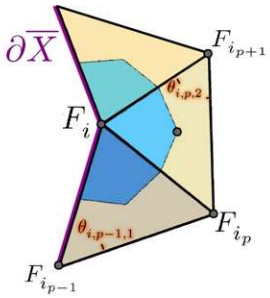

Figure 4.3: Finite volume scheme for boundary vertices.

For adjacent edges in the triangle 1-ring we can combine the terms using $F_{i_p} = F_{i_{p+1}}$ and $F_{i_m} = F_{i_1}$ to obtain the final result:

$$\iint_{V_i} h \, \mathrm{d}\mu_{g_F} \approx \frac{1}{2} \sum_{p=1}^{m} \big(\cot\theta_{i,p-1,1} + \cot\theta_{i,p,2} \big) (F_{i_p} - F_i) \tag{4.14}$$

For vertices of the mesh boundary $\partial\overline{X}$ the mean curvature vector is not defined. For example for minimal surface construction, boundary vertices have fixed support (Dirichlet conditions) for all time steps, which is implemented in the linear system. For scalar or vector fields on $\overline{X}$ there is no longer a closed loop of finite volume partitions $\mathcal{T}_i^p \cap V_i$, so a "raw" version of the mean curvature integral (4.13) has to be used, and boundary conditions have to be imposed on adjacent boundary vertices on the leftmost and rightmost triangle (see Fig. 4.1).

The specific formulas for geometric entities are shown in (4.21), (4.22), and (4.23).

4.2.1 Implementation of the 1-Ring Neighborhood

In sections (2.2) and (2.3) we discuss the practical development of piecewise-linear approximations of manifolds. It is clear that unless the constructed mesh surface is fully structured, the vertex and polygon adjacency relations have to be obtained a posteriori. This can be automated using suitable data structures.

Not only are the 1-ring vertex neighborhoods vital for the cotangent finite volume scheme, but they are also essential for computing unit normals N_i to mesh vertices. In this section we elaborate on another set of conventions, widely used in computational geometry. More specifically, we establish the following rules

- Assume that the set of mesh vertices $(v_1, ..., v_{N_V})$ is ordered randomly, and the polygonal indices are the only topological information available for the object.
- All polygons are tuples of indices to vertices which form positively-oriented polytopes[2]
- The adjacent 1-ring vertices form a positively-oriented (counter-clockwise) polytope when the central vertex is not a boundary vertex.

[2] a general requirement for most 3D models to be properly rendered on the scene.

- When the central vertex is a boundary vertex, the neighbors follow the same (counter-clockwise) orientation from the first (leftmost) boundary neighbor to the last (rightmost).

Since the 1-ring elements overlap, it might be desirable to save as much storage space as possible by only storing tuples of indices to adjacent vertices, indexed by the index of each central vertex. First step is to acquire the adjacency information from the polygonal vertex index set:

Algorithm 1: Get vertex-to-polygon multi-map

Result: A multimap $M_{\overline{X}} : v \to \mathcal{P}_v$ indexing adjacent polygons $\mathcal{P}_v$ to each vertex v
foreach $\mathcal{P} \in \overline{X}$ **do**
  foreach $v_j \in \mathcal{P}$ **do**
    $M_{\overline{X}}$.insert$(v_j \to \mathcal{P}_v)$;
  end
end

Note that $\mathcal{P}_v$ is a polygon with marked vertex v because the central vertex is not necessarily the first in each triangulation, which means that we need to iterate through polygon vertices until we find the marked vertex, and set its index to 0 (or 1). The `insert()` operation implemented, for example, for `std::multimap` object in the `C++` standard template library inserts a new key-value pair element for each polygon, but allows easy search by vertex afterwards.

The obvious application of the vertex-to-polygon multi-map is to compute a vertex normal N_i as a superposition of adjacent triangle normals. Simply adding and normalizing the triangle normal vectors might produce incorrect results, for vertices with high valence. For this reason, we use *angle-weighted pseudonormals* for each vertex:

Algorithm 2: Get angle-weighted pseudonormals

Result: A set of normal vectors N_i to each vertex
Data: a vertex-to-triangle multimap $M_{\overline{X}}$
foreach $v \in V$ **do**
  $N \leftarrow \mathbf{0}$
  while $M_{\overline{X}}$ *contains triangles* $\mathcal{T}_v$ *marked by* v **do**
    $\mathcal{T}_v \leftarrow M_{\overline{X}}$.find$(v)$;
    obtain adjacent vertices $v_0, v_1 \neq v$;
    $N_{\mathcal{T}} \leftarrow \frac{e_0 \times e_1}{\|e_0 \times e_1\|}$; `// compute triangle normal from adjacent edges`
    $\alpha \leftarrow \arccos\left(\frac{e_0}{\|e_0\|} \cdot \frac{e_1}{\|e_1\|}\right)$;
    $N \leftarrow N + \alpha N_{\mathcal{T}}$;
    $M_{\overline{X}}$.erase$(v \to \mathcal{T}_v)$;
  end
  $N \leftarrow \frac{1}{2\pi} N$; $N \leftarrow N/\|N\|$;
end

The angle-weighted pseudonormals are widely used in computer graphics, specifically for correct computation of the *signed distance function* to meshes which could contain holes [5].

Constructing *co-volume* (finite volume) polytopes would be relatively straight-forward if the values of $M_{\overline{X}}$ for each vertex v were ordered in counter-clockwise direction. Orientation of a 1-ring neighborhood inevitably conflicts with the orientation of a 1-ring neighborhood of a neighboring vertex. Again, we assume a random ordering of vertices v which means, the polygons $\mathcal{P}_v$ with marked vertex v first need to be sorted by adjacency.

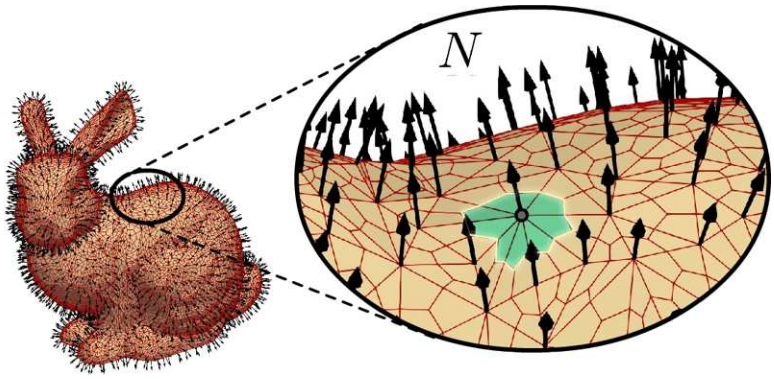

Figure 4.4: A finite volume network and angle-weighted pseudonormals on the *Stanford bunny* model.

This requires a specific case of a *topological sort* without any connectivity information for individual nodes (adjacent polygons), so the simpler version of our algorithm assumes v is not a boundary vertex, which means that adjacent polygons form a cycle.

Algorithm 3: Sort 1-ring polygons by adjacency

Result: an array of polygons ($\mathcal{P}_v$) with marked vertex sorted by edge adjacency in counter-clockwise direction.
Data: an (unordered) array of polygons $\{\mathcal{P}_v\}$ with marked vertex
$\mathcal{P}_{\text{current}} \leftarrow \{\mathcal{P}_v\}$.pop(); `// take out the first polygon`
while $\{\mathcal{P}_v\}$ *contains polygons* **do**
  $(\mathcal{P}_v)$.push($\mathcal{P}_{\text{current}}$);
  $v \leftarrow \mathcal{P}_{\text{current}}.v$;
  $v_1 \leftarrow \mathcal{P}_{\text{current}}$.previousVertex($v$); `// get right edge` (v, v_1)
  foreach $\mathcal{P}_v \in \{\mathcal{P}_v\}$ **do**
    $\tilde{v}_0 \leftarrow \mathcal{P}_v$.nextVertex($v$);
    if $v_1 == \tilde{v}_0$ **then**
      $\mathcal{P}_{\text{current}} \leftarrow \mathcal{P}_v$ `// next for processing`
      break
    end
  end
end

The above algorighm requires a modification for processing 1-rings for boundary vertices. Since the first processed polygon is taken from the top of container $\{\mathcal{P}_v\}$, and has a random position in the 1-ring, we must first iterate in one direction (counter-clockwise) until we reach a triangle with no edge, and then search through left neighbors of the first processed polygon towards the boundary again.

When we reach the end of container $\{\mathcal{P}_v\}$ when searching for right neighbor candidates, we exit the search and return to the first processed polygon, completing the adjacency from the left. Note that for iterating in the clockwise direction we compare the left edge vertices of current polygon $\mathcal{P}_{\text{current}}$ with right edge of the left neighbor candidate $\mathcal{P}_v$.

The computation of the co-volume network is straight-forward when given an array ($\mathcal{P}_v$) sorted by adjacency for each vertex v. It should be noted that topologies with multiple boundary vertices joined at a single vertex are not supported, nor should they be used as initial conditions for surface evolution.

4.3 A Discrete Formulation of the Evolution Model

Consider now model (3.9) without tangential redistribution. In this work, we utilize a *semi-implicit* finite volume scheme, that is: we use forward time difference, and all terms containing geometric information such as angles and areas, will be considered from the previous time step. Let V_i be a co-volume with F_i as representative vertex, and let $\tau > 0$ be the length of a time step, then from (3.9) the following integral formulation holds:

$$\iint_{V_i} \frac{F_i^{t+\tau} - F_i^t}{\tau} \mathrm{d}\mu_{g_F^t} = \iint_{V_i} \epsilon^t \Delta_{g_F} F_i^{t+\tau} \mathrm{d}\mu_{g_F^t} + \iint_{V_i} \eta_i^t N_i^t \, \mathrm{d}\mu_{g_F^t}$$

where control functions ϵ_i^t and η_i^t are obtained for each vertex from the current geometry state from (3.7) and (3.8) for example[3]. From an assumption of the finite volume formulation that integrands (except for the mesh Laplacian) are constant on V_i we obtain an approximation

$$\mu_{g_F^t}(V_i) \frac{F_i^{t+\tau} - F_i^t}{\tau} = \frac{\epsilon_i^t}{2} \sum_{p=1}^{m} \big(\cot \theta_{i,p-1,1}^t + \cot \theta_{i,p,2}^t \big) (F_{i_p}^{t+\tau} - F_i^{t+\tau}) + \eta_i^t N_i^t \mu_{g_F^t}(V_i) \tag{4.15}$$

Factoring the unknown vertex immersion positions from (4.15) for the next time step $F_i^{t+\tau}$ and $F_{i_p}^{t+\tau}$ we acquire a linear system

$$\mathrm{A}_{ii}^t F_i^{t+\tau} + \sum_{p=1}^{m} \mathrm{A}_{ii_p}^t F_{i_p}^{t+\tau} = F_i^t + \tau \eta_i^t N_i^t \tag{4.16}$$

with coefficients

$$\begin{aligned} \mathrm{A}_{ii}^t &= \left(1 + \frac{\tau \epsilon_i^t}{2 \mu_{g_F^t}(V_i)} \sum_{p=1}^{m} \big(\cot \theta_{i,p,1}^t + \cot \theta_{i,p,2}^t \big) \right) \\ \mathrm{A}_{ii_p}^t &= -\frac{\tau \epsilon_i^t}{2 \mu_{g_F^t}(V_i)} \big(\cot \theta_{i,p-1,1}^t + \cot \theta_{i,p,2}^t \big) \end{aligned} \tag{4.17}$$

Given a Dirichlet boundary condition $F^t\big|_{\partial X} = F^{\partial X}$ for all $t \in [0, t_s]$ reduces to trivially putting $\mathrm{A}_{ii}^t = 1$ and $\mathrm{A}_{ii_p}^t = 0$ with the right-hand side containing the fixed positions of the boundary vertices $F_i^{\partial X}$ [34].

Clearly, matrix $\mathbf{A}^t$ is diagonally dominant and the resulting linear system $\mathbf{A}^t \mathbf{F}^{t+\tau} = \mathbf{b}^t$ with right-hand side $\mathbf{b}^t$ given by (4.16) for each vertex coordinate is solved using the stabilized bi-conjugate gradient method (BiCGStab). However, the BiCGStab method (especially with good preconditioning) is a tool which is often stronger than required for most meshes (with low vertex valence). Based on the system's diagonal dominance [24] had successfully tested convergence for an SOR method as well.

4.4 Discretization of Tangential Velocity

The tangential velocity term adds additional information to the right-hand side of the linear system. For angle-based redistribution, the computation of $v_{T,i}^t$ for each time step is relatively straight-forward using formula (3.24) projected, of course, onto the tangent plane at vertex F_i^t by $v_{T,i}^t = \tilde{v}_{T,i}^t - (N_i^t \cdot \tilde{v}_{T,i}^t) N_i^t$.

[3] for scalar grids with discrete sampling, the values of the distance function, for example, need to be interpolated using trilinear interpolation.

On the other hand, the asymptotically uniform volume-based tangential redistribution requires one to solve a single linear system based on (3.19) integrated in the same way along each co-volume:

$$\begin{aligned}&\int_{V_i} \Delta_{F^t}\psi^t \ \mathrm{d}\mu_{g_F^t} = \\ &= \int_{V_i} (v_{N,i}^t \cdot h_i^t)\mathrm{d}\mu_{g_F^t} - \int_{V_i} \frac{1}{A^t}\bigg(\int_X (v_{N,i}^t \cdot h_i^t)\mathrm{d}\mu_{g_F^t}\bigg)\mathrm{d}\mu_{g_F^t} + \int_{V_i} \omega\bigg(\frac{A}{GA_X} - 1\bigg)\mathrm{d}\mu_{F^t}\end{aligned}$$

The area density G is approximated using the fact that

$$A^t = \int_X G^t \mathrm{d}\mu_X \approx \sum_{i=1}^{N_V} G_i^t \mu_X(V_i) \quad \Longleftrightarrow \quad A^t \approx \sum_{i=1}^{N_V} \mu_{g_F^t}(V_i)$$

and using formulas (3.20) we put

$$G_i^t = \frac{N_V}{A_X}\ , \quad \text{with} \ \ A_X = \mu_X(X) = 1 \quad \text{and} \quad \mu_X(V_i) = \frac{1}{N_V} \tag{4.18}$$

for convenience, which gives rise to discrete formulation:

$$\begin{aligned}&\frac{1}{2}\sum_{p=1}^{m}(\cot\theta_{i,p-1,1}^t + \cot\theta_{i,p,2}^t)(\psi_i^t - \psi_{i_p}^t) = \\ &= \mu_{g_F^t}(V_i)(v_{N,i}^t \cdot h_i^t) - \frac{\mu_{g_F^t}(V_i)}{A^t}\sum_{j=1}^{N_V}\mu_{g_F^t}(V_j)(v_{N,j}^t \cdot h_j^t) + \mu_{g_F^t}(V_i)\omega\bigg(\frac{A^t}{N_V\mu_{g_F^t}(V_i)} - 1\bigg)\end{aligned} \tag{4.19}$$

or alternatively:

$$\begin{aligned}&\frac{1}{2}\sum_{p=1}^{m}(\cot\theta_{i,p-1,1}^t + \cot\theta_{i,p,2}^t)(\psi_i^t - \psi_{i_p}^t) = \\ &= \mu_{g_F^t}(V_i)(v_{N,i}^t \cdot h_i^t) - \frac{\mu_{g_F^t}(V_i)}{A^t}\sum_{j=1}^{N_V}\mu_{g_F^t}(V_j)(v_{N,j}^t \cdot h_j^t) + \mu_{g_F^t}(V_i)\omega\bigg(\frac{A^t}{N_V\mu_{g_F^t}(V_i)}\frac{\frac{1}{f(H_i^t)}}{\sum_{j=1}^{N_V}\frac{\mu_X(V_j)}{f(H_j^t)}} - 1\bigg)\end{aligned} \tag{4.20}$$

where mean curvature and normal velocity are approximated as:

$$\begin{aligned}h_i^t &\approx \frac{1}{2\mu_{F^t}(V_i)}\sum_{p=1}^{m}(\cot\theta_{i,p-1,1}^t + \cot\theta_{i,p,2}^t)(F_i^t - F_{i_p}^t) \\ v_{N,i}^t &:= \epsilon(d(F_i^t))h_i^t + \eta(d(F_i^t))N_i^t\end{aligned}$$

The cotangents of angles opposing to each edge as well as all auxiliary geometric quantities can be calculated directly from the derivation of scheme:

$$\begin{aligned}\cot\theta_{i,p-1,1}^t &= \frac{(F_{i_{p-1}}^t - F_i^t)\cdot(F_{i_{p-1}}^t - F_{i_p}^t)}{||(F_{i_{p-1}}^t - F_i^t)\times(F_{i_{p-1}}^t - F_{i_p}^t)||} \\ \cot\theta_{i,p,2}^t &= \frac{(F_i^t - F_{i_{p+1}}^t)\cdot(F_{i_p}^t - F_{i_{p+1}}^t)}{||(F_i^t - F_{i_{p+1}}^t)\times(F_{i_p}^t - F_{i_{p+1}}^t)||}\end{aligned} \tag{4.21}$$

$$\mu_{g_F^t}(V_i) = \frac{1}{2}\sum_{p=1}^{m}\left(||(M_{i,p,1}^t - F_i^t)\times(B_{i,p}^t - F_i^t)|| + ||(B_{i,p}^t - F_i^t)\times(M_{i,p,2}^t - F_i^t)||\right) \tag{4.22}$$

$$\begin{aligned} M_{i,p,1}^t &= \frac{1}{2}(F_{i_p}^t + F_i^t)\ , \quad M_{i,p,2}^t = \frac{1}{2}(F_{i_{p+1}}^t + F_i^t)\ , \quad B_{i,p}^t = \frac{1}{3}(F_i^t + F_{i_p}^t + F_{i_{p+1}}^t) \\ \tilde{M}_{i,p,1}^t &= \frac{1}{2}(M_{i,p,1}^t + B_{i,p}^t)\ , \quad \tilde{M}_{i,p,2}^t = \frac{1}{2}(M_{i,p,2}^t + B_{i,p}^t) \end{aligned} \tag{4.23}$$

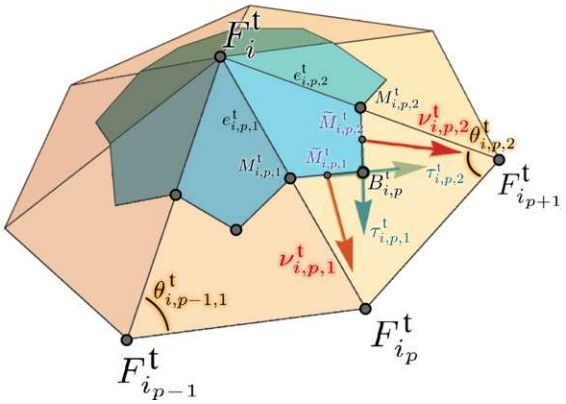

Figure 4.5

Afterwards a linear system is composed from (4.19):

$$\mathrm{A}_{i,i}^{\psi,t}\psi_i^t + \mathrm{A}_{i,p}^{\psi,t}\psi_{i_p}^t = \mathrm{b}_i^{\psi,t} \tag{4.24}$$

such that:

$$\begin{aligned} &\mathrm{A}_{1,1}^{\psi,t} = 1 \\ &\mathrm{A}_{i,i}^{\psi,t} = \frac{1}{2}\sum_{p=1}^{m}(\cot\theta_{i,p-1,1}^t + \cot\theta_{i,p,2}^t)\ , i = 2,...,N_V \\ &\mathrm{A}_{i,p}^{\psi,t} = -\frac{1}{2}(\cot\theta_{i,p-1,1}^t + \cot\theta_{i,p,2}^t)\ , \quad i = 2,...,N_V \\ &\mathrm{A}_{i,j}^{\psi,t} = 0 \quad \text{(everywhere else)} \end{aligned}$$

because potential ψ requires a reference value which we chose to be $\psi_0^t = 0$. The right-hand side components are then computed by:

$$\begin{aligned} &\mathrm{b}_1^{\psi,t} = 0\ , \text{ and for } i = 2,...,N_V \quad \text{with } \tilde{A}^t = \sum_{i=1}^{N_V-1}\mu_{g_F^t}(V_i)\ : \\ &\mathrm{b}_i^{\psi,t} = \mu_{g_F^t}(V_i)(v_{N,i}^t\cdot h_i^t) - \frac{\mu_{g_F^t}(V_i)}{\tilde{A}^t}\sum_{j=1}^{N_V-1}\mu_{g_F^t}(V_j)(v_{N,j}^t\cdot h_j^t) + \mu_{g_F^t}(V_i)\omega\left(\frac{\tilde{A}^t}{\mu_{g_F^t}(V_i)(N_V-1)} - 1\right) \end{aligned} \tag{4.25}$$

or, for curvature-controlled redistribution:

$$\mathrm{b}_i^{\psi,t} = \mu_{g_F^t}(V_i)(v_{N,i}^t\cdot h_i^t) - \frac{\mu_{g_F^t}(V_i)}{\tilde{A}^t}\sum_{j=1}^{N_V-1}\mu_{g_F^t}(V_j)(v_{N,j}^t\cdot h_j^t) + \mu_{g_F^t}(V_i)\omega\left(\frac{\tilde{A}^t}{\mu_{g_F^t}(V_i)}\frac{\frac{1}{f(H_i^t)}}{\sum_{j=1}^{N_V-1}\frac{1}{f(H_j^t)}} - 1\right) \tag{4.26}$$

A convenient way to verify the underlying geometric calculations is to take a sum of the right-hand side over all vertices:

$$\begin{aligned} &\sum_{i=1}^{N_V}\mathrm{b}_i^{\psi,t} = \sum_{i=1}^{N_V-1}\left[\mu_{g_F^t}(V_i)(v_{N,i}^t\cdot h_i^t) - \frac{\mu_{g_F^t}(V_i)}{\tilde{A}^t}\sum_{j=1}^{N_V-1}\mu_{g_F^t}(V_j)(v_{N,j}^t\cdot h_j^t) + \omega\left(\frac{\tilde{A}^t}{N_V-1} - \mu_{g_F^t}(V_i)\right)\right] = \\ &= \sum_{i=1}^{N_V-1}\mu_{g_F^t}(V_i)(v_{N,i}^t\cdot h_i^t) - \frac{1}{\tilde{A}^t}\left(\sum_{i=1}^{N_V-1}\mu_{g_F^t}(V_i)\right)\sum_{j=1}^{N_V-1}\mu_{g_F^t}(V_j)(v_{N,j}^t\cdot h_j^t) + \omega\left((N_V-1)\frac{\tilde{A}^t}{N_V-1} - \sum_{i=1}^{N_V-1}\mu_{g_F^t}(V_i)\right) = \\ &= \omega\left(\tilde{A}^t - \tilde{A}^t\right) = 0 \end{aligned}$$

which, of course, holds for (4.26) as well. Summing up $\mathrm{b}_i^{\psi,t}$ numerically should therefore be "almost zero" when considering round-off errors.

To obtain the tangential velocity field from ψ assume $w_T^t = \nabla_{g_F^t}\psi^t$ and $(F^t)^*(v_T^t(p)) = w_T^t(p) = (\nabla_{g_F^t}\psi^t)(p)$ for points $p \in X$ implies

$$v_T^t(p) = (F^t)_*(w_T^t(p)) = (F^t)_*(\nabla_{g_F^t}\psi^t(p))$$

which means that $\mathrm{Im}(v_T^t) = \nabla_{\mathrm{Im}(F^t)}((F^t)_*\psi^t)$ with the specialized push-forward gradient $\nabla_{\mathrm{Im}(F^t)}$ is defined so that diagram

$$\begin{array}{ccc} \mathcal{F}(\mathrm{Im}(F^t)) & \xrightarrow{\nabla_{\mathrm{Im}(F^t)}} & \mathcal{V}(\mathrm{Im}(F^t)) \\ {\scriptstyle (F^t)_*}\uparrow & & {\scriptstyle (F^t)_*}\uparrow \\ \mathcal{F}(X) & \xrightarrow{\nabla_{g_F^t}} & \mathcal{V}(X) \end{array}$$

commutes with $\mathcal{F}(X)$ and $\mathcal{V}(X)$ spaces of functions and vector fields[4] respectively on manifold X. Note that $\mathrm{Im}(F^t) = F^t[X]$ by the previous notation. Clearly, $\nabla_{\mathrm{Im}(F^t)}$ represents the gradient on $F^t[X]$. Given a scalar field $\varphi : F^t[X] \to \mathbb{R}$, its gradient is then projected into the tangent space of the image $F^t[X]$ of immersion F^t:

$$\nabla_{\mathrm{Im}(F^t)}\varphi = \nabla_Y\overline{\varphi} - g_Y(\nabla_Y\overline{\varphi}, N)N$$

with $\overline{\varphi}$ being a smooth extension of φ to the neighborhood of $F^t[X]$. Thus the finite volume integral formulation assumes form [24]:

$$\iint_{V_i} v_T^t \,\mathrm{d}\mu_{g_F^t} = \int_{\partial V_i} \psi^t \nu_i^t \,\mathrm{d}H_{\mu_{g_F^t}} - \iint_{V_i} \psi^t h^t \,\mathrm{d}\mu_{g_F^t} \tag{4.27}$$

with co-volume approximation:

$$\int_{V_i} v_T^t \,\mathrm{d}\mu_{F^t} \approx \sum_{p=1}^{m} \left(||\sigma_{i,p,1}^t||\psi_{i,p,1}^t\nu_{i,p,1}^t + ||\sigma_{i,p,2}^t||\psi_{i,p,2}^t\nu_{i,p,2}^t\right) - \mu_{F^t}(V_i)\psi_i^t h_i^t$$

where according to Fig.4.5 we define:

$$\begin{aligned}
&\sigma_{i,p,1}^t = B_{i,p}^t - M_{i,p,1}^t \ , \quad \sigma_{i,p,2}^t = B_{i,p}^t - M_{i,p,2}^t \text{ (co-volume edges)} \\
&e_{i,p,1}^t = M_{i,p,1}^t - F_i^t \ , \quad e_{i,p,2}^t = M_{i,p,2}^t - F_i^t \text{ (triangle half-edges)} \\
&\tau_{i,p,1}^t = B_{i,p}^t - M_{i,p,1}^t \ , \quad \tau_{i,p,2}^t = B_{i,p}^t - M_{i,p,2}^t \text{ (co-volume tangents)} \\
&\nu_{i,p,1}^t = \frac{e_{i,p,1}^t - \frac{(e_{i,p,1}^t\cdot\tau_{i,p,1}^t)}{(\tau_{i,p,1}^t\cdot\tau_{i,p,1}^t)}\tau_{i,p,1}^t}{\left\| e_{i,p,1}^t - \frac{(e_{i,p,1}^t\cdot\tau_{i,p,1}^t)}{(\tau_{i,p,1}^t\cdot\tau_{i,p,1}^t)}\tau_{i,p,1}^t \right\|} \ , \quad \nu_{i,p,2}^t = \frac{e_{i,p,2}^t - \frac{(e_{i,p,2}^t\cdot\tau_{i,p,2}^t)}{(\tau_{i,p,2}^t\cdot\tau_{i,p,2}^t)}\tau_{i,p,2}^t}{\left\| e_{i,p,2}^t - \frac{(e_{i,p,2}^t\cdot\tau_{i,p,2}^t)}{(\tau_{i,p,2}^t\cdot\tau_{i,p,2}^t)}\tau_{i,p,2}^t \right\|} \text{ (co-volume edge normals)}
\end{aligned}$$

and

$$\psi_{i,p,1}^t = \frac{5\psi_i^t + 5\psi_{i,p}^t + 2\psi_{i_{p+1}}^t}{12} \ , \quad \psi_{i,p,2}^t = \frac{5\psi_i^t + 2\psi_{i,p}^t + 5\psi_{i_{p+1}}^t}{12}$$

are interpolated values of ψ^t in co-volume edge midpoints $\tilde{M}_{i,p,1}^t$, $\tilde{M}_{i,p,2}^t$.

[4] commutative and associative algebras

Chapter 5

Signed Distance Computation

The advection term controlled by (3.8) contains a scalar field which determines the shape of target geometry. The *distance function to a set*, say $\overline{X} \subset \mathbb{R}^3$ is defined as:

$$d^+ : \mathbb{R}^3 \to \mathbb{R}^+ : \ \mathbf{x} \mapsto \inf_{\mathbf{p} \in \overline{X}} \left\{ \|\mathbf{p} - \mathbf{x}\| \ \middle| \ \mathbf{x} \in \mathbb{R}^3 \right\}$$

and its modification:

$$d^\pm : \mathbb{R}^3 \to \mathbb{R} : \ \mathbf{x} \mapsto \mathrm{sgn}_{\overline{X}}(\mathbf{x}) \inf_{\mathbf{p} \in \overline{X}} \left\{ \|\mathbf{p} - \mathbf{x}\| \ \middle| \ \mathbf{x} \in \mathbb{R}^3 \right\}$$

with

$$\mathrm{sgn}_{\overline{X}}(\mathbf{x}) = \begin{cases} 1, & \text{for } \ \mathbf{x} \in \mathrm{Int}(\overline{X}) \\ \\ -1, & \text{for } \ \mathbf{x} \in \mathrm{Ext}(\overline{X}) \end{cases}$$

where $\mathrm{Int}(\overline{X})$ and $\mathrm{Ext}(\overline{X})$ are the interior and exterior of a closed surface mesh $\overline{X}$, is called the *signed distance function*.

Sampling d^+ (or $d^\pm$) in a regular grid allows us to triangulate individual level sets (points which are the same distance from mesh) as seen in Fig. 5.1, or to interpolate between grid node values using trilinear interpolation. The sampling process, however, can turn out to be computationally demanding for practical purposes.

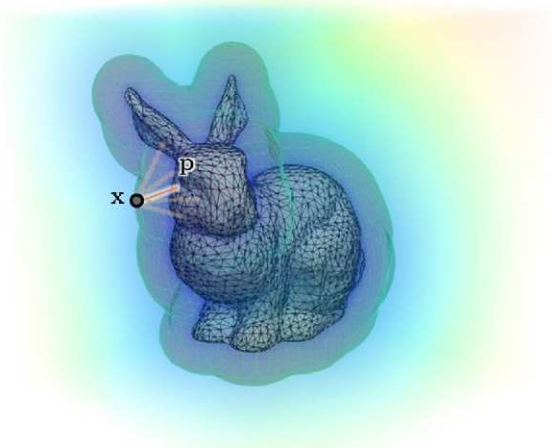

Figure 5.1: Distance function to a mesh using the brute force approach requires us to compare distances to all mesh triangles for each sampled point $\mathbf{x}$.

Consider a regular 3D grid with $100 \times 100 \times 100 = 10^6$ voxels and a mesh with $\approx 5 \times 10^3$ triangles, as it is in case of the Stanford bunny model in Fig.5.1. Sampling d^+ for all 10^6 grid nodes requires us to make 10^3 calculations of 3D distance to a mesh triangle, after which each resulting new distance is compared with the previous iteration and kept for the grid node when smaller. This makes 5×10^9 calculations total for what will be referred to as the *brute force approach*, taking several minutes even when computed on multiple threads.

The problem of accelerating the, so called, *distance query* for 3D models has been approached in a large number of publications dedicated to implementing the procedure for their particular purposes. A thesis by Sanchez [33], for example, tests multiple possibilities and compares their computational efficiency.

In this chapter, we discuss our approach to accelerating distance function in three steps:

1. construct a *binary (bounding volume) search tree hierarchy* to accelerate *voxel-triangle intersection query.*

2. subdivide the bounding space with an *octree* up to the desired resolution, and set its leaf nodes to the exact mesh distance.

3. with the mesh distance outline computed, perform 8 iterations of the *fast sweeping algorithm* for Eikonal equation $\|\nabla d(\mathbf{x})\| = 1$ to solve for d.

which require no parallel processing, and can be implemented using just the standard `C++` template library.

5.1 Axis-Aligned Bounding-Box (AABB) Trees

Given a scene with many 3D models, as is usually the case in 3D computer games for example, testing whether individual models intersect for *collision detection* is an operation which is in its brute-force form of $\mathcal{O}(mn)$ complexity, where m and n are the numbers of triangles of respective models. Similarly, sampling the distance grid has the same computational complexity. The trick used by virtually all game engines is to only test parts of each model for intersection, say only a few triangles which occupy the same space.

The key is to construct a *bounding volume hierarchy* (BVH) with the root volume enveloping an entire mesh model, and remaining nodes containing increasingly smaller volumes up to leaves which contain small numbers of triangles. In contrast to the use of an *octree*, it is generally more convenient to partition the space into a *binary space partition* (BSP) tree. The intersection queries then follow logarithmic complexity $\mathcal{O}(\log n)$ for each model, vastly accelerating the procedure.

The simplest approach is to use bounding boxes which intersect sets of non-interacting triangles (triangle soup)[1]. The, so called, *axis-aligned bounding boxes* will be the key feature of search tree nodes. Only leaf nodes will contain triangles, so the search algorithm need to recursively look for finer bounding boxes to get to the leaf node and perform the subsequent calculations. The binary tree constructed from such nodes is referred to as an *axis-aligned bounding box tree* (AABB tree).

[1] topological information such as adjacency is irrelevant for distance computation to individual triangles.

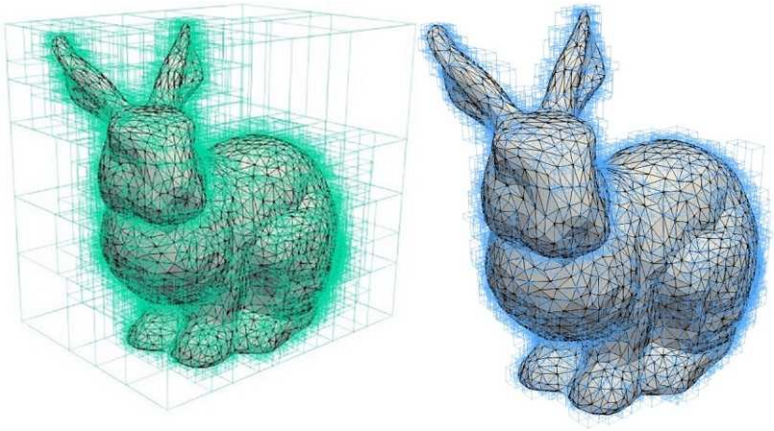

Figure 5.2: (left) all node boxes of an AABB tree, and (left) only AABB leaf nodes containing triangles of the Stanford bunny model.

There are many variants of this data structure. Specifically, similar search trees which store point cloud data are referred to as *KD-trees*[2]. The distinction between terms AABB tree and KD-tree is often non-existent.

Our method from constructing an AABB tree from a set of individual triangles can be summarized in the following algorithm:

Algorithm 4: AABB Tree (Axis-Aligned Bounding Box Tree)

Data: A "triangle soup" $\{\mathcal{T}\}$
`// this procedure is called recursively for each node` $\mathcal{N}$
$\mathcal{B}_0 \leftarrow$generate a bounding box around $\{\mathcal{T}\}$ and inflate it by a small amount;
set $\mathcal{B}_0$ as the box of the root node $\mathcal{N}_{\mathcal{B}_0}$;
if $\mathcal{B}_0$ *does not meet the leaf conditions* **then**
  $x_{\text{split}} \leftarrow$ find optimal split position for $\mathcal{B}_0$ (by SAH and longest axis);
  split $\mathcal{B}_0$ into $\mathcal{B}_1$ and $\mathcal{B}_2$ in x_{split};
  distribute triangles in $\{\mathcal{T}\}$ into two (possibly overlapping) buckets $\mathcal{N}_{\mathcal{B}_1}$ and $\mathcal{N}_{\mathcal{B}_2}$ when they intersect boxes $\mathcal{B}_1$ and $\mathcal{B}_2$;
  break if max depth is reached;
end

The initial inflation of box $\mathcal{B}_0$ serves as a correction for possible numerical errors when computing intersections. If intersection subroutine fails to find an intersection which should exist for $\mathcal{B}_0$ due to round-off errors, a larger box $\mathcal{B}_0^{\pm\varepsilon} \supset \mathcal{B}_0$ with $\varepsilon > 0$ added in all directions will not fail this test.

The "*leaf conditions*" we use can be summarized with inequality

$$N_L + N_R < \frac{3}{2} N_{\text{total}}$$

which quantifies the balance between the sum of the number of *left node triangles* N_L and the *right node triangles* N_R and a multiple of the total number of triangles N_{total}. Note that since triangles can "share boxes" $N_L + N_R \geq N_{\text{total}}$ in general.

The search for optimal split position is the most significant "bottleneck" for the construction of an AABB tree. Splitting $\mathcal{B}_0$ in half would be insufficient for meshes with large concentrations of triangles in some areas because it would take much longer for AABB nodes $\mathcal{N}$ to reach leaf conditions and when the do, the search would have to process large amounts of triangles in some nodes. Finding the longest axis of $\mathcal{B}_0$ is trivial, but finding x_{split} requires a measure of triangle concentration for both child candidates.

We use a variant of a technique called *surface area heuristic* (SAH) which minimizes a cost function

$$f_{\text{cost}}(x) = C_L(x)\frac{SA_L(x)}{SA} + C_R(x)\frac{SA_R(x)}{SA}$$

where SA is the surface area of current box $\mathcal{B}_0$, $SA_L(x)$ and $SA_R(x)$ are surface areas of left and right child boxes respectively, and cost weights $C_L(x)$ and $C_R(x)$ evaluate the cost of

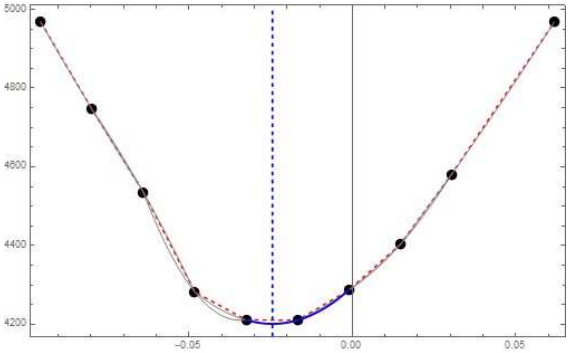

Figure 5.3: Split position sampling and minimization of f_{cost} on parabolic segments (blue) for the root node of the Stanford bunny model.

[2] as in k-dimensional trees.

traversing the left and right node candidates respectively, all while considering split position x. The traversing costs C_L and C_R are evaluated by counting triangle intersected by the left and right box candidates. Clearly C_L is non-decreasing with respect to x, since by moving the split position further to the right will result in more triangles contained in the left node. Similarly C_R is non-increasing for the same reason.

We used an optimized adaptive procedure for minimizing f_{cost} from [14] on 32-bit float SIMD registers for storing quadruples of split values x and processing them simultaneously, the result of which can be seen in Fig.5.3.

5.2 Octree-Based Mesh-Cell Generation

Octree data structures are often used when processing 3D image data. They provide a way to find adaptively finer regular subdivisions of 3D voxels (axis-aligned cubes). Our goal is to construct a scalar grid (a 3D image), and smallest octree nodes (leaves) will serve as basis for grid voxels "activated" by the triangle mesh. The computational cost of extracting leaf cells and transforming their bounds to grid index space with distance information is again $\mathcal{O}(\log n)$.

Algorithm 5: Mesh Distance Octree

Data: An AABB tree $\mathbb{T}_{\text{AABB}}$ of mesh $\overline{X}$
`// this procedure is called recursively for each node` $\mathcal{O}_N$
$\mathcal{C}_0 \leftarrow$generate a bounding cube around $\overline{X}$;
set $\mathcal{C}_0$ as the cube of the root node $\mathcal{O}_{N,0}$;
if $\mathcal{C}_0$*.size()* $<$ *minCellSize* **then**
  subdivide $\mathcal{C}_0$ into 8 subcells $\mathcal{C}_k,\ k = 1, ..., 8$;
  foreach $\mathcal{C}_k,\ k = 1, ..., 8$ **do**
    if $\mathcal{C}_k$ *intersects mesh* $\overline{X}$ *using* $\mathbb{T}_{AABB}$ **then**
      open node $\mathcal{C}_k$;
    else
      discard $\mathcal{C}_k$;
    end
  end
  break if max depth is reached;
else
  $d_{min} \leftarrow \min\{$ squared distance of the centroid of $\mathcal{C}_0$ to $\mathcal{T} \in \mathbb{T}_{\text{AABB}}$.`intersectsWith(`$\mathcal{C}_0$`)`$\}$;
end

Where `intersectsWith(`$\mathcal{C}_0$`)` finds a leaf node of $\mathbb{T}_{\text{AABB}}$ which intersects $\mathcal{C}_0$. In fact, the cell-mesh intersection in the previous **if** block can be performed by the same routine.

Because the square root operation is computationally expensive, we compare squared distances to find d_{min}. Using the octree leaves we assign the stored values d_{min} to cells g_{ijk} of a regular $(N_x \times N_y \times N_z)$-grid G using indices:

$$i_x = \left\lfloor \left(\frac{1}{2}\left(\mathcal{C}_{\mathtt{min},x} + \mathcal{C}_{\mathtt{max},x}\right) - G_{x,\min} \right) \frac{N_x}{s_x} \right\rfloor \tag{5.1}$$

and analogously for i_y and i_z, where $\mathcal{C}_{\mathtt{min},x}$ ans $\mathcal{C}_{\mathtt{max},x}$ are the minimal and maximal vertices of leaf cell $\mathcal{C}$, $G_{x,\min}$ is the minimal bound of grid G, N_x the number of cells and s_x its dimension in the x-direction.

Given the grid's index dimensions N_x, N_y, and N_z we can access grid cells with linear indexing $i_{\text{pos}} = N_x\ N_y\ i_z + N_x\ i_y + i_x$ and store all scalar data in a linear array.

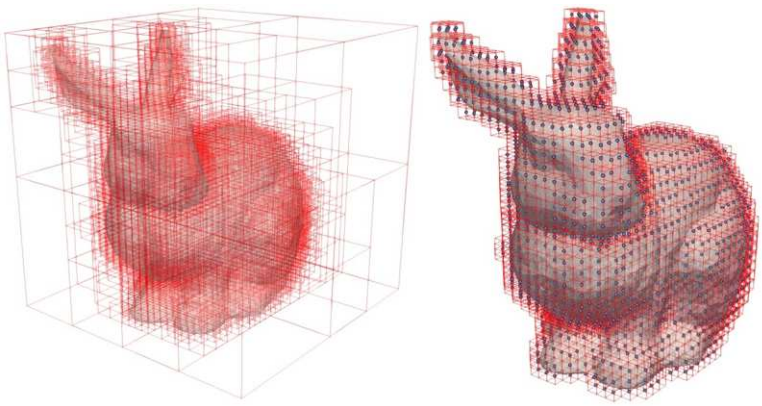

Figure 5.4: (left) full octree with all its cells, and (left) the octree leaves with centroids which serve as sampled grids points for the Stanford bunny model.

Note that the mean between the minimal an maximal cell vertex in (5.1) implies that we sample the space in the centroids of the octree leaves (see Fig. 5.4).

5.3 The Fast Sweeping Algorithm

The remaining (inactive) grid cells should, by default, contain a value larger than any value a distance function d^+ can admit in the sampled volume. This serves as an initial condition for the *fast sweeping algorithm* for non-linear *Eikonal*[3] *equations* which take general form $\|\nabla u(\mathbf{x})\| = f(\mathbf{x})\ ,\quad \mathbf{x} \in \mathbb{R}^n$ with $u(\mathbf{x})\big|_\Gamma = \phi(\mathbf{x})\ ,\quad \Gamma \subset \mathbb{R}^n$. The distance function d^+ is a specific form of Eikonal problem, namely:

$$\|\nabla d^+(\mathbf{x})\| = 1\ ,\quad d^+(\mathbf{x})\big|_\Gamma \equiv 0 \tag{5.2}$$

Clearly, the distance function d^+ to a set $\Gamma \subset \mathbb{R}^n$ is linear with slope 1 in the normal direction from Γ. According to Zhao [39], given a set of grid points $\mathbf{x}_{i,j,k} \in [G_{x,\min}, G_{x,\max}] \times [G_{y,\min}, G_{y,\max}] \times [G_{z,min}, G_{z,max}]$ with the initial condition approximated, or as in our case already computed for a subset of grid points $\mathbf{x}^\Gamma_{i,j,k}$ near $\Gamma = \overline{X}$ (the triangle mesh). We will perform $2^{n=3} = 8$ iterations (*sweeps*) in alternating index directions:

$$\begin{aligned}
&i_x = 1, ..., N_x\ , \quad i_y = 1, ..., N_y\ , \quad i_z = 1, ..., N_z\\
&i_x = N_x, ..., 1\ , \quad i_y = 1, ..., N_y\ , \quad i_z = 1, ..., N_z\\
&i_x = N_x, ..., 1\ , \quad i_y = N_y, ..., 1\ , \quad i_z = 1, ..., N_z\\
&\vdots\\
&i_x = 1, ..., N_x\ , \quad i_y = N_y, ..., 1\ , \quad i_z = N_z, ..., 1\\
&i_x = 1, ..., N_x\ , \quad i_y = N_y, ..., 1\ , \quad i_z = 1, ..., N_z
\end{aligned}$$

depicted in Fig. 5.5.

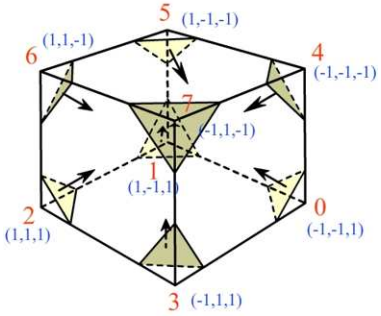

Figure 5.5: 8 sweep directions of the fast sweeping algorithm.

[3] a German form of Greek word $\varepsilon\iota\kappa\omega\nu$ (eikon) which stands for "likeness", "icon", or "image"

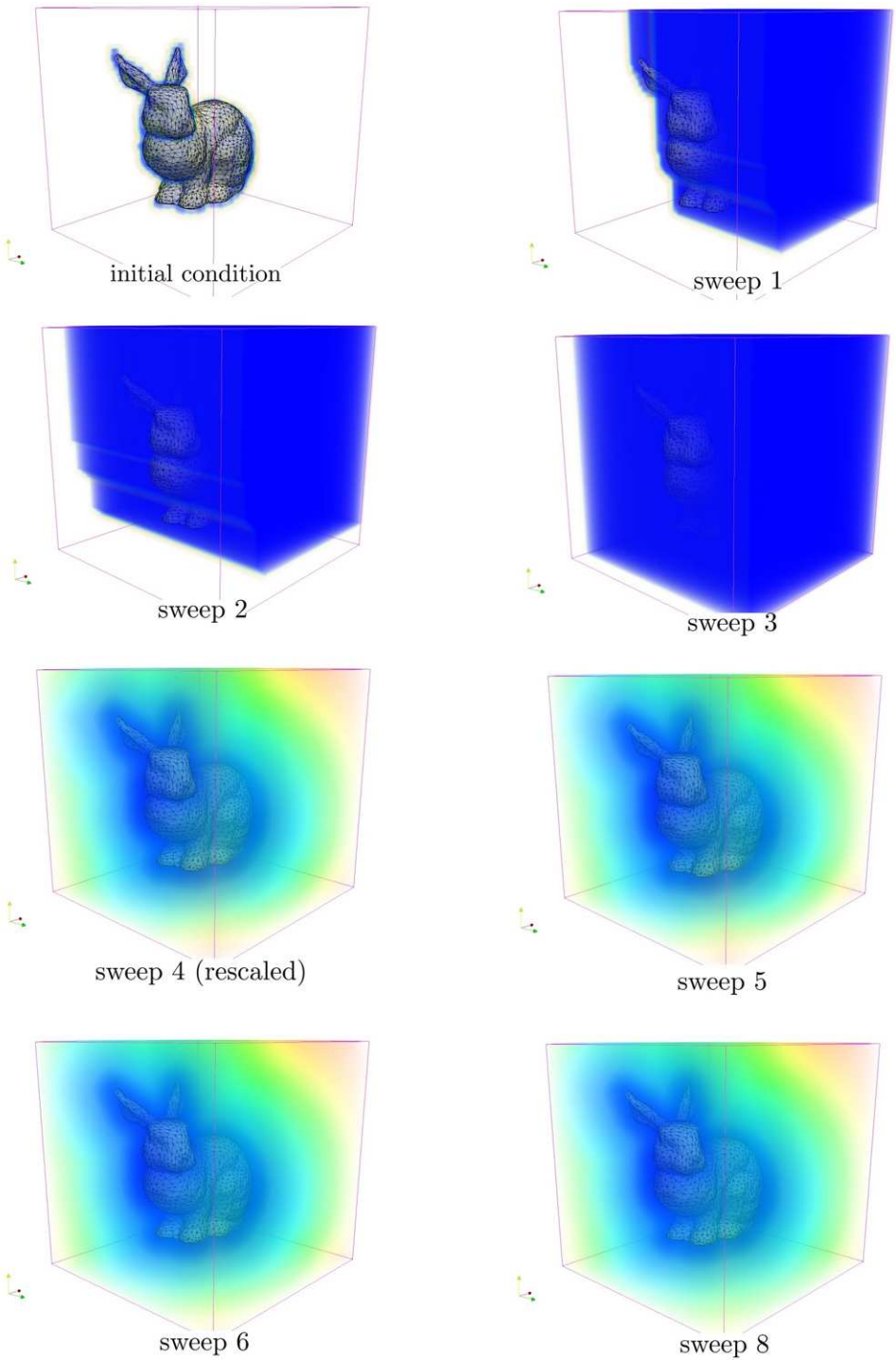

Figure 5.6: A volume rendering of the distance function computation with the fast sweeping algorithm. First 3 iterations have high contrast (infinities are transparent) and the following are rescaled to view the full range of the solution. The image of sweep 7 is omitted.

Now for each sweep, use a Godunov upwind difference scheme:

$$[(d^h_{i,j,k} - d^h_{x,min})^+]^2 + [(d^h_{i,j,k} - d^h_{y,min})^+]^2 + [(d^h_{i,j,k} - d^h_{z,min})^+]^2 = h^2$$

where $h > 0$ is the grid's cell size and

$$d^h_{x,min} = \min\{d^h_{i-1,j,k}, d^h_{i+1,j,k}\}\ , \quad d^h_{y,min} = \min\{d^h_{i,j-1,k}, d^h_{i,j+1,k}\}\ , \quad d^h_{z,min} = \min\{d^h_{i,j,k-1}, d^h_{i,j,k+1}\} \tag{5.3}$$

$$(x)^+ = \begin{cases} x, & x > 0 \\ \\ 0 & x \leq 0 \end{cases}$$

For each grid cell $\mathbf{x}_{i,j,k}$ we sort the neighboring min values given by (5.3) in ascending order: $a_1 \leq a_2 \leq a_3$ and then find a solution $\overline{x}$ to

$$\begin{aligned} x - a_1 &= h \\ (x - a_1)^2 + (x - a_2)^2 &= h^2 \\ (x - a_1)^2 + (x - a_2)^2 + (x - a_3)^2 &= h^2 \end{aligned}$$

until $\overline{x} \leq a_p\ ,\ p = 1, 2, 3$.

As a result, the numerical solution $\tilde{d}$ to (5.2) will be completed in the directions of each sweep with increasing accuracy, all of which is completed in $\mathcal{O}(N)$ time (where $N = \dim G$). The grid G, of course, has to be expanded to the user's desired dimension prior to running the fast sweeping algorithm.

As we can see in Fig. 5.6, the most significant changes in the numerical solution occur up to sweep 4. The resulting solution, of course, deviates from the exact (brute-force) distance function (see Fig. 5.7). We can compute numerical error estimates by taking

$$G^\epsilon = |G_{\text{FS}} - G_{\text{exact}}|\ , \quad L_2\text{-error} = \frac{1}{N_x N_y N_z} \sum_{i,j,k=1}^{N_x, N_y, N_z} {G^\epsilon_{i,j,k}}^2$$

The L_2-error is reduced by the ratio of the number of new grid cells over the number of old grid cells when increasing grid resolution.

The algorithm has been used, for example in [21] for reconstructing surfaces from point cloud data using level sets, and can be extended to an arbitrary number of dimensions (although increasing computational demands).

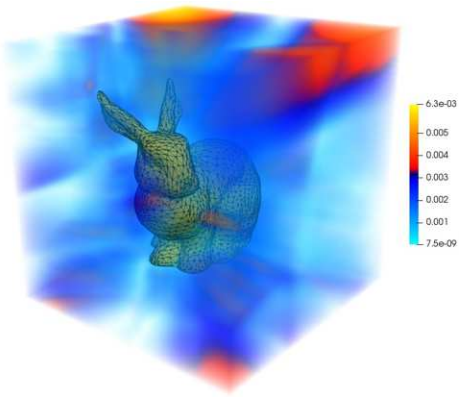

Figure 5.7: The numerical error G^ϵ of the fast sweeping algorithm on the Stanford bunny with grid size 45^3.

5.4 Mesh SDF Algorithm and Results

Combining the above approaches, we are able to obtain results within computationally reasonable time. All of this was implemented without any parallel processing. Using multiple CPU cores for demanding operations like octree subdivision, the fast sweeping algorithm, or even the construction of an AABB tree could make even the finest distance scalar grids fully interactive on changing the source mesh $\overline{X}$.

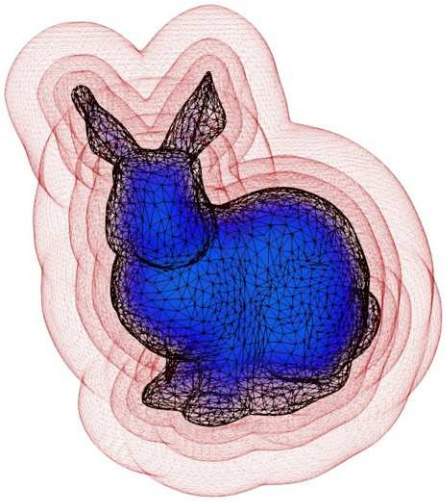

Figure 5.8: A sign of the distance function inside the bunny mesh (blue) and the outer iso-contours rendered via the marching cubes algorithm.

For our goals, we only need to compute the distance function once to use it in control function (3.8), thus a few seconds of computation is not too much of a drawback. The largest bottleneck is the computation of the sign $\text{sgn}_{\overline{X}}$ of $d^{\pm}$. Medľa [23] uses an approach of flood filling the entire voxel grid. Prior to running the fast sweeping algorithm, voxels which intersect $\Gamma = \overline{X}$ are marked as *frozen* since they are no longer subject to processing. When $\overline{X}$ is a closed surface or a surface with holes smaller than grid's cell size, the *frozen* voxels create a closed discrete representation of Γ. Negating all values after fast-sweeping and seeding a flood fill outside of Γ will result in all outside cells to get their respective positive values, while the interior of the mesh contains negative distance values.

As was previously mentioned, the construction of an AABB tree $\mathbb{T}^{\overline{X}}_{\text{AABB}}$ relies on using `float32` SIMD registers for parallel processing of multiple split positions. Using 64-bit registers would allow even better sampling at the same speed, but such framework is not supported on most machines. The construction of both $\mathbb{T}^{\overline{X}}_{\text{AABB}}$ and octree $\mathbb{O}_{\overline{X}}$ can be accelerated using a linear array of nodes with topological information stored in the form of indices to array positions.

Our procedure is summarized in the following algorithm:

Algorithm 6: A distance function scalar grid G

Data: A mesh $\overline{X}$ with extractable triangle soup $\{\mathcal{T}\}$, offset value o
$\mathbb{T}^{\overline{X}}_{\text{AABB}} \leftarrow$ generate an AABB tree from $\{\mathcal{T}\}$;
generate $\mathbb{O}_{\overline{X}}$ given `minCellSize` > 0; `// octree`
create grid G with dimensions of the bounding box of $\overline{X}$ and given cell resolution;
expand G by given offset o;
set exact distance values $g^{\text{exact}}_{i,j,k}$ to grid cells that are centroids of octree $\mathbb{O}_{\overline{X}}$'s leaf cells;
set $g_{i,j,k} \leftarrow \infty$ everywhere else;
`fastSweep(`G`); // 8 sweeps`
compute sign of G using voxel flood fill;

Testing method from algorithm (6) on the Stanford bunny with increasing grid resolution gives the following CPU time measurement results (in seconds):

grid resolution	AABB tree	Octree	Fast-Sweeping	Sign computation	TOTAL
30^3	0.053611	0.027909	0.009959	0.021141	0.112620
60^3	0.049847	0.039661	0.042242	0.099734	0.231385
120^3	0.047344	0.117273	0.338058	0.789032	1.289707
240^3	0.043543	0.291580	2.325632	6.931647	9.590404

Chapter 6

Surface Evolution as Remeshing

6.1 Numerical Experiments

Having implemented all procedures from previous chapters, we proceed to test our models. The first step is to verify the numerical accuracy of the evolution model (3.4) on an available exact solution. Since there are only a few examples of exact solutions to the mean curvature flow problem (3.4), and modeling infinite or translating solutions (for example, the shrinking cylinder or grim reaper) we choose the shrinking sphere solution for testing, as in [24].

For computing the error given by difference between the solution of (3.5), namely $r(t) = \sqrt{r_0 - 4t}$, and the numerical sphere immersion F^t for all mesh vertices as well as all time steps, we use

$$\epsilon = \sqrt{\sum_{k=1}^{N_t} \tau \Big(\sum_{i=1}^{N_V} \big(\|F_i^t\| - r(t) \big)^2 \mu_{g_F^t}(V_i) \Big)^2} \tag{6.1}$$

where $N_t = t_s/\tau$ and $t = k\tau,\ k = 1, ..., N_t$. We put $r_0 = 1$ to perform the test on a unit sphere $\mathbb{S}^2$. The sphere tessellation we use is given by recursive subdivision of an icosahedron (as seen in Fig.1.1). Each subdivision step reduces geodesic edge length between vertices on $\mathbb{S}^2$ to a half of the previous step, which means we can compute experimental order of convergence:

$$\text{EOC} = \log_2 \left(\frac{\epsilon_l}{\epsilon_{l/2}} \right)$$

where ϵ_l and $\epsilon_{l/2}$ are given by (6.1) for discretizations with geodesic edge lengths l and $l/2$. Note that we also reduce the length of time step τ by half. The results are summarized in tables (6.1), (6.2), and (6.3).

N_V	τ	N_t	L_2-error	EOC	CPU time [s]
42	0.01	6	0.00306693		0.015400
162	0.0025	24	0.000867587	1.82171	0.109411
642	0.000625	96	0.000210629	2.04231	2.879498
2562	0.00015625	384	5.11243e-05	2.04262	66.822671

Table 6.1: the EOC for the shrinking sphere test without tangential redistibution.

N_V	τ	N_t	L_2-error	EOC	CPU time [s]
42	0.01	6	0.00284543		0.054055
162	0.0025	24	0.000859596	1.72692	0.147276
642	0.000625	96	0.000209863	2.03421	3.467786
2562	0.00015625	384	5.07569e-05	2.04777	113.075791

Table 6.2: the EOC for the shrinking sphere test with asymptotically uniform tangential redistibution with $\omega = 1.0$.

N_V	τ	N_t	L_2-error	EOC	CPU time [s]
42	0.01	6	0.00491455		0.062172
162	0.0025	24	0.00106504	2.20615	0.182325
642	0.000625	96	0.000238864	2.15665	3.122358
2562	0.00015625	384	5.71001e-05	2.06462	146.78901

Table 6.3: the EOC for the shrinking sphere test with asymptotically uniform tangential redistibution with $\omega = 100.0$.

6.2 Mean Curvature Flow Evolution

As shown in the previous section, the validity of our model is verified on behalf of experimental order of convergence (EOC). The general MCF model can be implemented as:

Algorithm 7: MCF of an input mesh

Data: A mesh $\overline{X}$ (preferrably of higher quality), τ, N_t;
Result: Meshes $\overline{X}_t$ for each time step, and the result $\overline{X}_{t_s}$
`// initialization (get bounds in case the solution explodes)`
$\mathcal{B}_{\overline{X}} \leftarrow \overline{X}$`.computeBoundingBox().expandByFactor(2.0);`
`// evolution`
for $k = 1; k < N_t; k++$ **do**
  $\overline{X}$`.computeNormalsAndCoVolumes();`
  if *do volume-based tangential redistribution* **then**
    $\overline{X}$`.computeCurvatures();`
    compose and solve linear system $\mathbf{A}^{\psi,t}\psi^t = \mathbf{b}^{\psi,t}$;
  end
  compose and solve linear systems $\mathbf{A}^t\boldsymbol{F}^t = \mathbf{b}^t + \tau\boldsymbol{v}_T^t$;
  $\overline{X}$`.updateVerticesFrom(` $\boldsymbol{F}^t$, $\mathcal{B}_{\overline{X}}$ `) ;`
end

When the volume-based redistribution flag is turned off, the tangential velocity vector $\boldsymbol{v}_T^t$ is either set to the angle-based redistribution velocity vector for each vertex, or set to zero for no tangential redistribution. Besides updating mesh vertices, the `updateVerticesFrom(` $\cdot,\cdot$ `)` method also checks whether new vertices $\boldsymbol{F}^t$ are contained by inflated bounding box $\mathcal{B}_{\overline{X}}$.

The lack of tangential redistribution becomes apparent (see Fig. 6.1 (top)), since vertices with higher mean curvature tend to accumulate. Imposing asymptotically uniform volume-based redistribution results in a decrease of cumulative effects on vertices, yet also the rate of contraction is no longer driven by mean

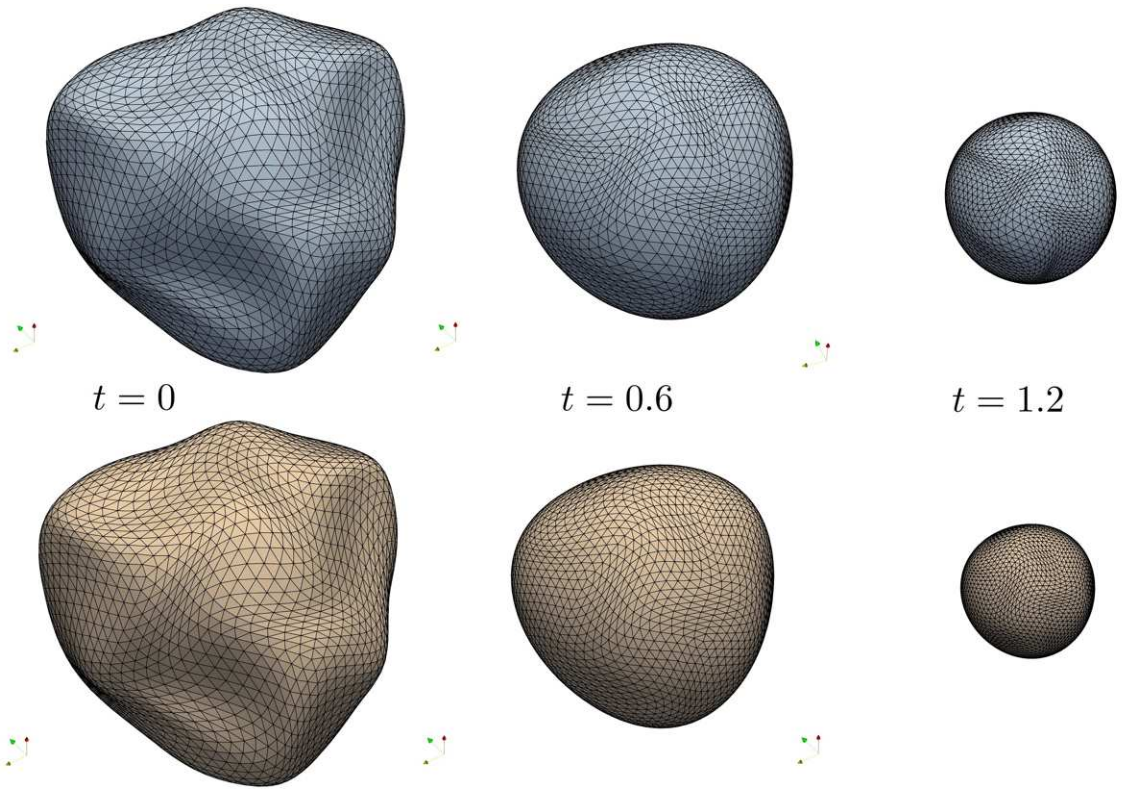

Figure 6.1: A comparison of MCF evolution with $\tau = 0.03$ of an irregular shape without tangential redistribution (top) and with volume-oriented redistribution with $\omega = 200$.

curvature. This may result in an elongated shape of the ellipsoid mesh shown in the second row of Fig.6.3, or an accelerated shrinking evolution as in Fig.6.1 (bottom).

It should be noted that when time step τ differs significantly from $\langle \mu_{g_F^t}(V_i) \rangle$ (where $\langle \cdot \rangle$ stands for mean value over N_V co-volume elements) the semi-implicit method given by system (4.16) becomes unstable. Despite being a characteristic feature of evolution with $v_T = 0$, the angle-based redistribution velocity

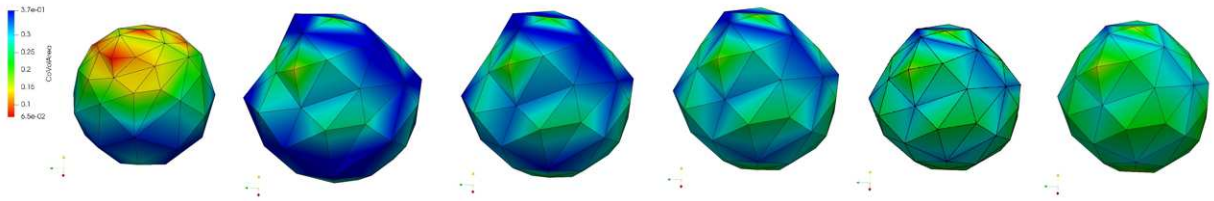

Figure 6.2: Mean curvature flow of a sphere-like initial mesh with uneven distribution of vertices. The angle-based velocity rapidly expands triangles with "bad" angles. Tangential velocity v_t also has a volume-based component with $\omega = 200$. The time steps shown are 0, 5, 10, 15, 20, and 25.

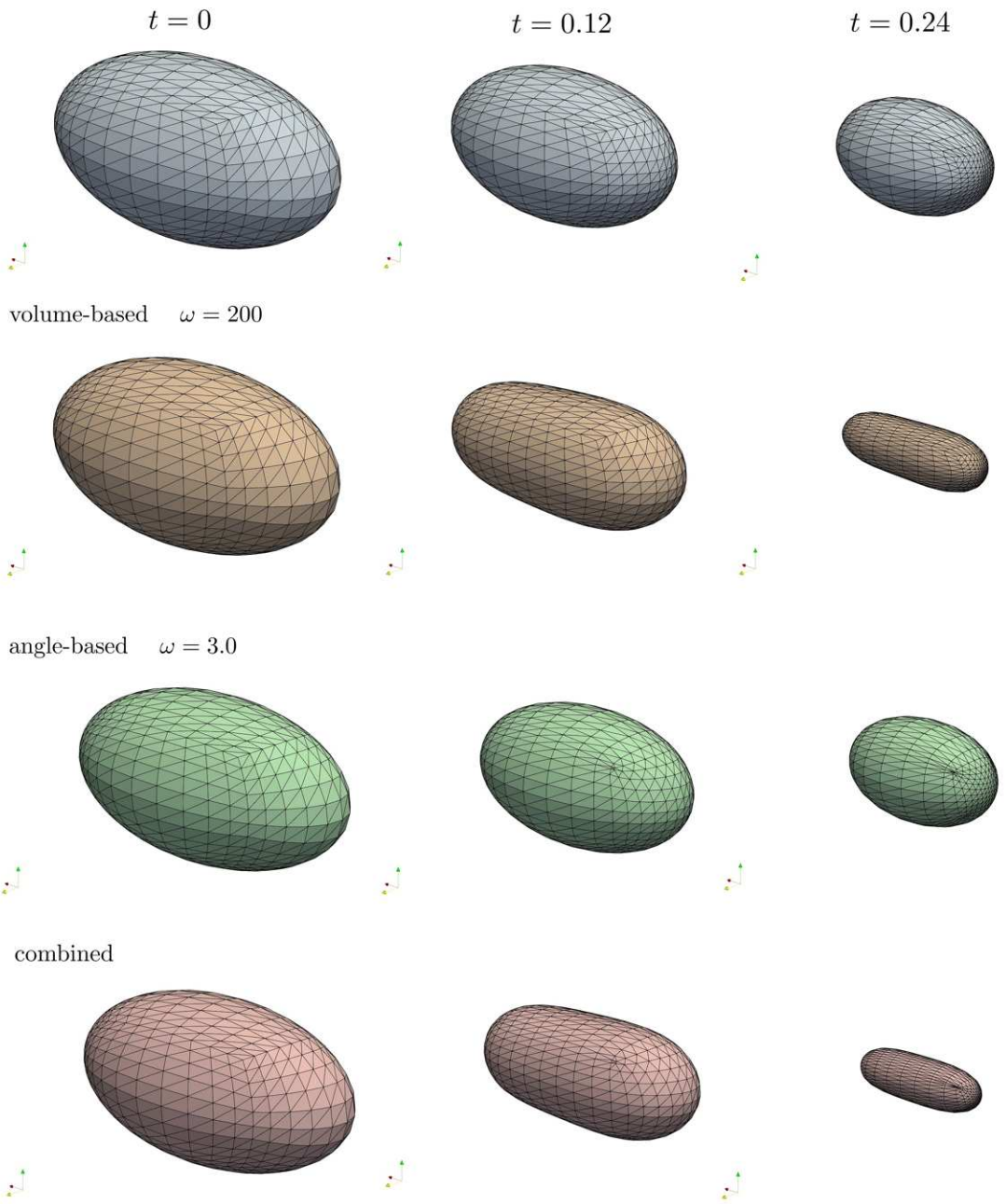

Figure 6.3: A comparison of the effects of volume and angle based tangential redistribution. The top evolution is without tangential redistribution and bottom evolution has a combination (sum) of tangential velocities from the previous two.

formula (3.24) is only weighed by internal angles, not by lengths of vertex edges.

As a result, not using the angle-based parameter carefully may cause the solution to explode for some configurations, especially for meshes with high concentrations of vertices in some parts more then other. An example of the power of the tangential velocity is shown in Fig.6.2 with $\omega_{angle} = 3.0$. The distribution of vertices diffuses, but the transient state involves rapid expansion of vertices with "bad" angles whose tangential velocity is of high magnitude for a short period of time.

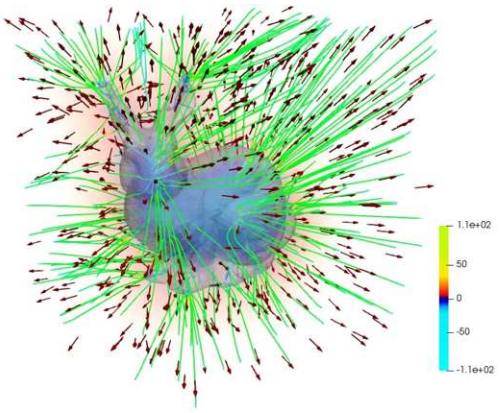

Figure 6.4: A gradient field with trajectories for the signed distance function to the Stanford bunny model.

It is also possible to combine tangential velocities which arise from volume-based and angle-based, and taking their superposition: $v_T = v_{T,vol} + v_{T,angle}$. The resulting shape is characterized by a combination of the traits given by both velocities. That being said, the effect of $v_{T,angle}$ can still be overwhelmingly more powerful than the gradient of redistribution potential ψ, the result of which is that points tend to get more concentrated in some parts of the mesh, specifically around *extraordinary vertices* with valence different from their neighborhood. These artifacts can be accounted for by making the effect of $v_{T,angle}$ inversely proportional to co-volume area or by weighing this component by a scalar function such as $d = G_\sigma * d^{\pm}$;

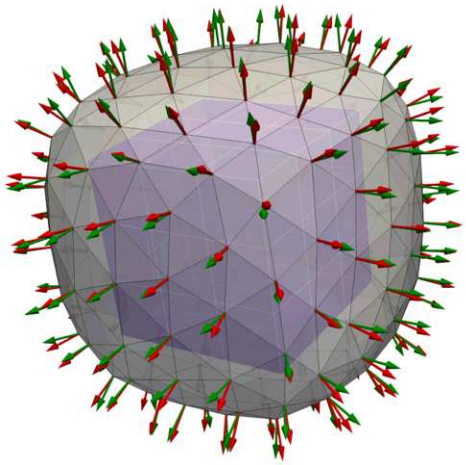

Figure 6.5: Interpolated values of ∇d (green) and surface normals (red) at vertices F_i^t

6.3 Gradient-Based Evolution

Extending our model to (4.1) we modify Algorithm 7 by introducing an advection term given by d. The gradient of the distance function is, however, defined only on grid points $\mathbf{x}_{ijk} \in G$. Using trilinear interpolation with respect to surrounding vertices for any (vertex) position F^t within the bounding box of G solves this issue straightforwardly. Trajectories solving initial value problem $F'(t) = \nabla d,\ F(0) = F^0$ can be seen in Fig. 6.4. In section 3.2 we also mention that since we are starting from an immersion F^0 the surface should evolve against ∇d towards the mesh from outside. A simulation of the vectors acting in η given by (3.8) can be found in Fig. 6.5, where we show ∇d vectors (in green) instead of $-\nabla d$ for visual purposes.

At first we implement a Lagrangian evolution with advection, but without additional tangential velocity. The results follow the properties we discussed in the previous section. Apparently, the advection term influences co-volume density to a large extent, especially for convex meshes (Fig. 6.6 and 6.8).

Areas with considerably high mean curvature, with the most rapid change in $-\nabla d$, such as edges, corners and other key features of the target mesh, seem to accumulate vertices in much the same way as areas with

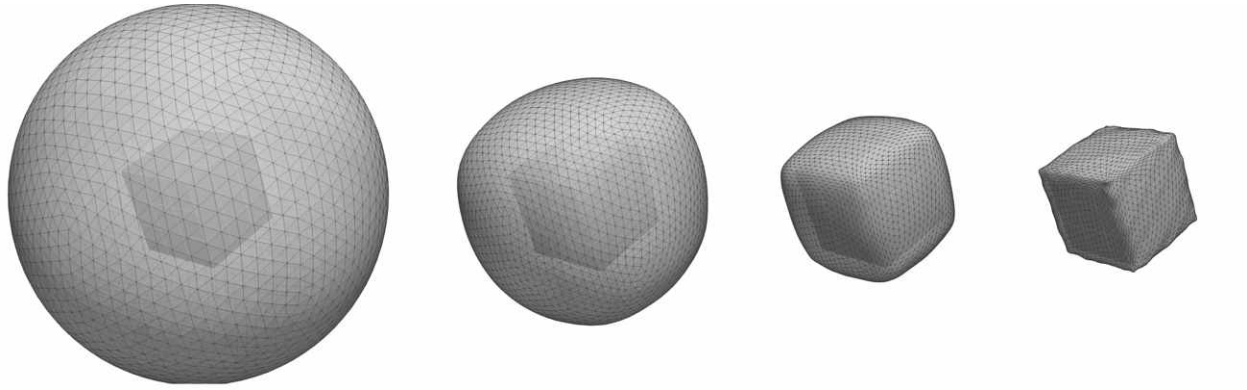

Figure 6.6: Evolution towards a cube model.

higher curvature for continuous models (ellipsoid in Fig.6.3, for example). As a result the triangles in these areas seem to have disproportionately small angles. Which is visible in colored co-volume area in Fig. 6.7.

Moreover, the distance grid resolution influences the final product to a certain extent as well. As the surface relaxes onto a region Γ which approximates the mesh, it follows the information provided by the distance gradient. Low resolution voxel grids have a specific form of discontinuity which manifests itself as roughness which can be seen in Figures 6.7 and 6.8.

We deal with the "roughness" problem by applying additional smoothing steps via mean curvature flow. To avoid the shrinking of key mesh features, we exponentially slow the mean curvature term by a modified

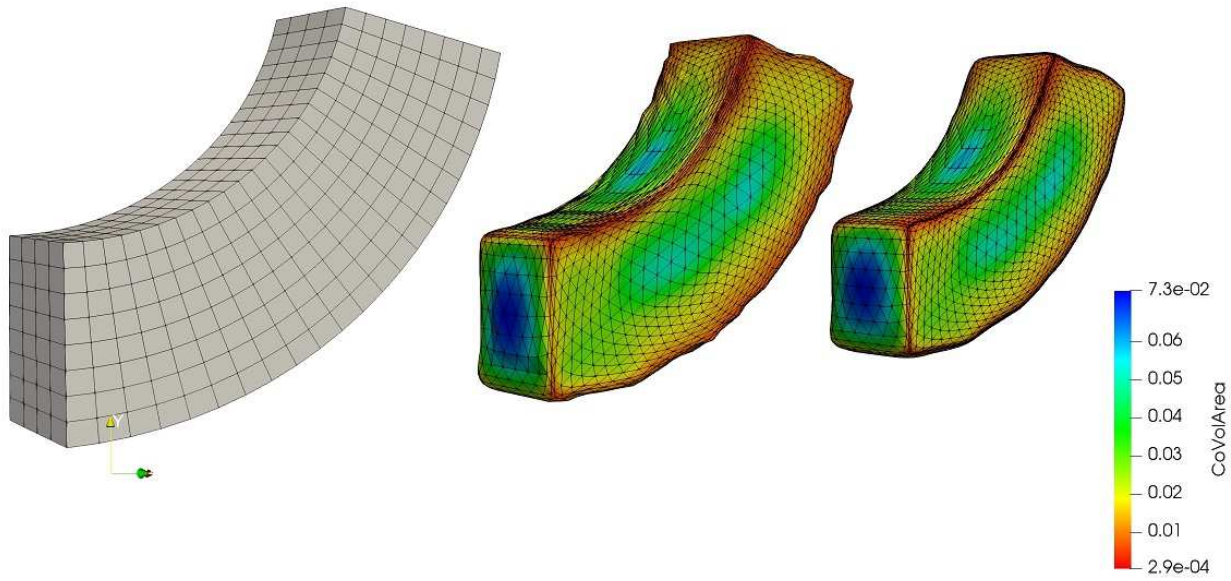

Figure 6.7: Evolution result after $N_t = 150$ steps with $\tau = 0.03$ and 10 additional smoothing steps. The distance field grid resolution is $120 \times 115 \times 96$.

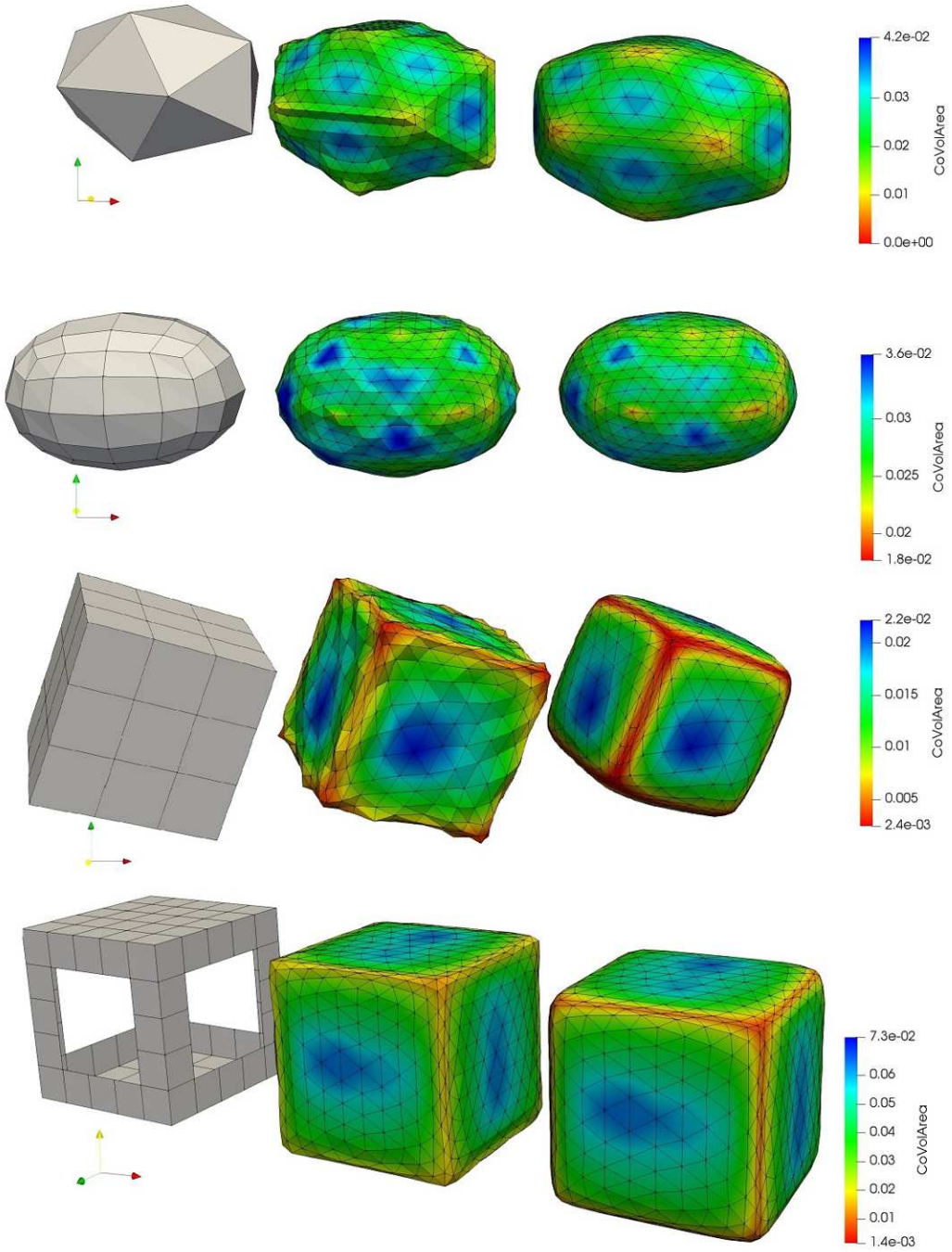

Figure 6.8: Convex meshes after $N_t = 150$ steps with $\tau = 0.03$ and 10 additional steps of smoothing. The distance grid resolution is geometry-dependent with the largest being 120^3.

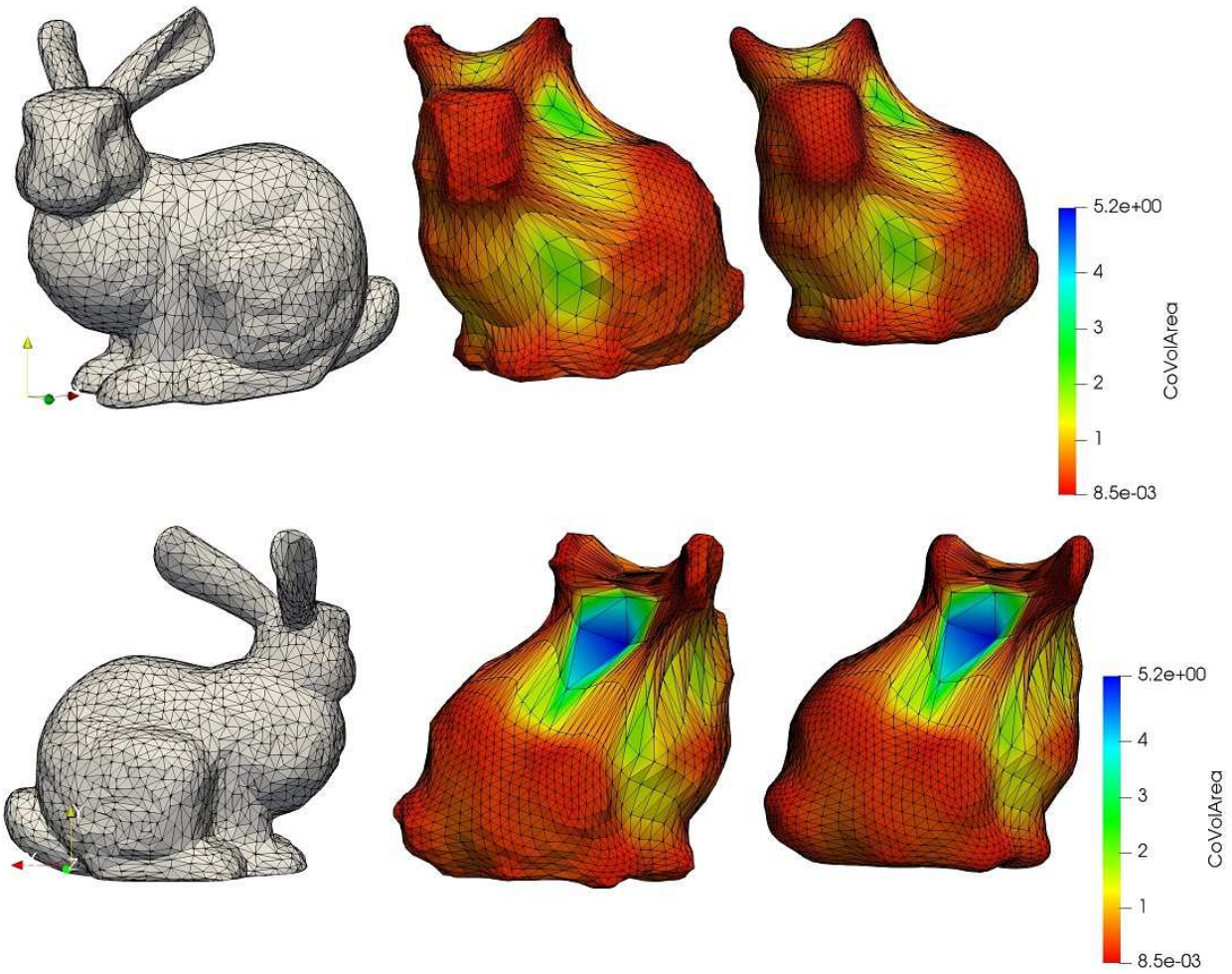

Figure 6.9: Evolution towards the Stanford bunny model without tangential redistribution. The configuration is the same as in previous examples, but with higher IcoSphere subdivision.

control function

$$\epsilon(t) := S_0 \mathrm{e}^{-\lambda t}, \quad S_0, \lambda > 0$$

Convex meshes have a significant advantage over non-convex target manifolds mainly because the regular tesselation of the initial immersion F^0 contains no information about possible redistribution to concave areas prior to successively reaching a point when the triangles are already "too stretched" for further calculations. This can be clearly seen in the "arc" example (Fig. 6.7), and in an extreme case with the Stanford bunny model (in Fig.6.9)

Applying tangential redistribution has its limitations as well. If not controlled, $v_{T,angle}$ can cause an explosion of the solution. On the other hand the advection term and possibly an asymptotically uniform volume-based redistribution velocity can still strongly influence the interior angles of triangular elements. Higher amount of redistribution results in the mesh shrinking under a form of smoothing which is a combination of the redistribution terms and mean curvature flow.

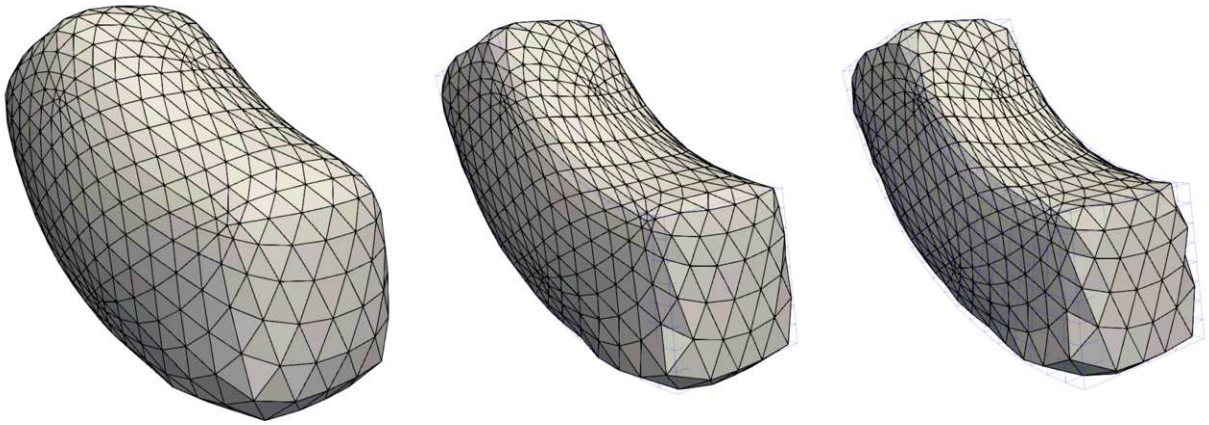

Figure 6.10: Evolution towards the arc mesh with angle-based tangential redistribution with $\omega = 3.0$.

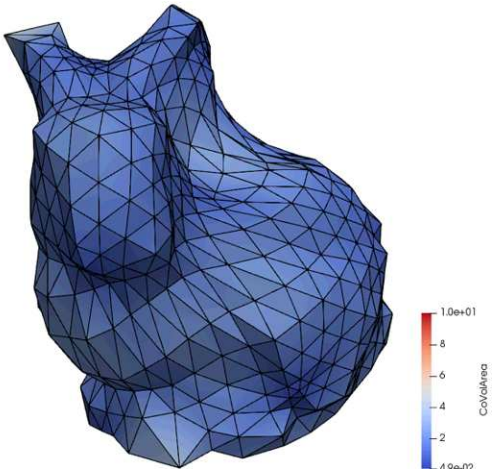

Figure 6.11: The highest quality attempt to remesh the Stanford bunny model.

The redistribution terms, however, increase the mesh quality even for simple models, such as the arc in Fig.6.10. The accumulation around extraordinary vertices and triangle stretching is, however, still visible.

For models with large concavities, like the Stanford bunny, a more robust approach is required.

6.4 Results and Discussion

Based on the results of this chapter, we conclude that although Lagrangian surface evolution models with advection can serve as a remesing tool for a small subset of (predominantly convex) meshes, an extension of this approach to general geometries demands further inquiry.

Furthermore, we would like to point out that only a small sample of possible approaches to tangential redistribution have been tried. Perhaps a form of conformal map could prevent triangle stretching with concave target meshes. The Lagrangian evolution model fills all holes of models which have higher genus with minimal surface membranes. Perhaps using iso-surfaces generated by the marching cubes algorithm could adapt more efficiently to model's intricate features. All of this will be an abundant area for further research and development.

Bibliography

[1] J. Pérez, *A New Golden Age of Minimal Surfaces*, Notices of the AMS (2017).

[2] G. Huisken, *Asymptotic Behavior for Singularities of the Mean Curvature Flow*. Journal of Differential Geometry (1990).

[3] M. Bauer, P. Harms, P. W. Michor, *Sobolev Metrics on Shape Space of Surfaces*. Journal of Geometric Mechanics 3, 4 (2011), p. 389-438

[4] F. Black, M. Scholes, *The Pricing of Options and Corporate Liabilities*, J. of Political Economy (1973)

[5] J. A. Baerentzen, H. Anaes, *Signed Distance Computation Using the Angle Weighted Pseudonormal*. IEEE Transactions on Visualization and Computer Graphics, vol. 11, no. 3,(2005)

[6] S. S. Cairns, *Triangulation of the manifold of class one*, Bulletin of the American Mathematical Society, 41(1935)

[7] Y. Canzani, *Analysis on Manifolds via the Laplacian* - Lecture Notes, Harvard University (2013).

[8] M. P. Carmo, *Differential Geometry of Curves and Surfaces*, Prentice-Hall (1976)

[9] S. Akbulut, J. D. McCarthy, *Casson's invariant for oriented homology 3-spheres*, volume 36 of Mathematical Notes, Princeton University Press (1990)

[10] P. Daniel, M. Medľa, K. Mikula, M. Remešíková, *Reconstruction of Surfaces from Point Clouds Using a Lagrangian Surface Evolution Model*. Springer International Publishing Switzerland (2015)

[11] M. Desbrun, E. Kanso, Y. Tong, *Discrete Differential Forms for Computational Modeling*, Applied Geometry Lab, Caltech (2008)

[12] M. Meyer, M. Desbrun, P. Schröder, A. H. Barr, *Discrete Differential-Geometry Operators for Triangulated 2-Manifolds*, Visualization and Mathematics III, H.-C. Hege and K. Polthier, eds. Springer-Verlag, Berlin (2003)

[13] M. Wardetzky, S. Mathur, F. Kälberer, E. Grinspun, *Discrete Laplace Operators: No Free Lunch*, Eurographics Symposium on Geometry Processing (2007).

[14] W. Hunt, W. R. Mark, G. Stoll, *Fast KD-Tree Construction with an Adaptive Error-Bounded Heuristic*, IEEE Symposium on Interactive Ray Tracing (2006).

[15] J. Clutterbuck, O. C. Schnürer, F. Schulze, *Stability of Translating Solutions to Mean Curvature Flow*, Freie Universität Berlin (2005).

[16] S. J. Alstschuler, M. A. Grayson, *Shortening Space Curves and Flow Through Singularities*, Journal of Differential Geometry (1992).

[17] A. Hatcher, *Algebraic Topology*, Cambridge University Press (2002).

[18] J. Kačur, K. Mikula, *Slow and Fast Diffusion Effects in Image Processing*, Computing and Visualization in Science (2001)

[19] H. Kneser, *Die Topologie der Mannigfaltigkeiten*, Jahresbericht Deutschen Math.-Verein (1926)

[20] P. M. Knupp, *Algebraic Mesh Quality Metrics*, Society for Industrial and Applied Mathematics (SIAM) 2001.

[21] B. Kósa, *3D point cloud surface reconstruction by using level set methods*, Bachelor thesis, Department of Mathematics and Constructive Geometry, Slovak University of Technology (2014)

[22] C. Manolescu, *Lectures on the Triangulation Conjecture*, UCLA (2016)

[23] M. Medľa, *Solving Partial Differential Equations using Finite Volume Method on Non-Uniform Grids.* Dissertation thesis, Slovak University of Technology, Bratislava (2018).

[24] K. Mikula, M. Remešíková, P. Sarkoci, D. Ševčovič, *Manifold Evolution With Tangential Redistribution of Points.* Society for Industrial and Applied Mathematics (SIAM) 2014.

[25] K. Mikula, D. Sevčovič, *Evolution of Plane Curves Driven By a Nonlinear Function of Curvature and Anisotropy*, Society for Industrial and Applied Mathematics (SIAM) 2001.

[26] M. O. Uba, K. Mikula, Z. Krivá (in collaboration with H. Nguyen, T. Savy, E. Kardash, N. Peyriéras), *3D Cell Image Segmentation By Modified Subjective Surface Method* Tatra Mountains Mathematical Publications (2014).

[27] E. E. Moise, *Affine structures in 3-manifolds: V. the triangulation theorem and Hauptvermutung*, Annals of mathematics, (2) (1952)

[28] M. H. Poincaré,(1899). *Complément a l'analysis situs*, Rendiconti del Circolo Matematico di Palermo (1899)

[29] T. Radó, *Über den Begriff der Riemannschen Fläche*, Acta Litt. Sci. Szeged, 2 (1925)

[30] P. Alliez, G. Ucelli, C. Gotsman, M. Attene, *Recent Advances in Remeshing of Surfaces*, Shape Analysis and Structuring (2008).

[31] K. Polthier, *Automatic Generation of Riemann Surface Meshes*, Freie Universität Berlin (2010).

[32] S. Rosenberg, *The Laplacian on A Riemannian Manifold*: An Introduction to Analysis on Manifolds, Cambridge University Press (1997).

[33] M. Sanchez, *Continuous Signed Distance Field Representation of Polygonal Meshes*, Bournemouth University, National Centre for Computer Animation (2011).

[34] L. Tomek, M. Remešíková, K. Mikula, *Computing Minimal Surfaces By Mean Curvature Flow With Area-Oriented Tangential Redistribution*, Acta Math. Univ. Comenianae (2017)

[35] L. Tomek, K. Mikula, *Discrete Duality Finite Volume Method With Tangential Redistribution of Points For Surfaces Evolving By Mean Curvature* ESAIM: Mathematical Modelling and Numerical Analysis, M2AN 53 (2019).

[36] K. Mikula, M. Remešíková, P. Novysedlak, *Truss Structrue Design Using a Length-Oriented Surface Remeshing Technique*, Discrete and Continuous Dynamical Systems - Series S (DCDS-S) Volume 8, Number 5 (2015) pp. 933 - 951 (2015).

[37] D. Hoffman, T. Ilmanen, M. Martín, B. White, *Notes on Translating Solitons for Mean Curvature Flow* Stanford, Zürich, Granada(2020)

[38] J. H. C. Whitehead, *On C^1-complexes*, Annals of Mathematics, (2)(1940)

[39] H. Zhao, *A Fast Sweeping Method for Eikonal Equations*, Mathematics of Computation. 74 (2005).