Approximation of Surfaces by Normal Meshes

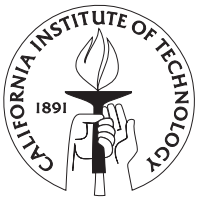
Thesis by

Ilja Friedel

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2005

(Defended May 18, 2005)

For my parents, Alexandra and Klaus Friedel.

Acknowledgements

Foremost, I have to thank my advisor Peter Schröder. He brought me to Caltech and provided intellectual guidance and support during my whole graduate student career. He believed in me even in emotionally difficult times. His encouragement and enthusiasm allowed me to find my own path toward scientific research and helped me finish my dissertation.

It has been tremendous fun to work with Mathieu Desbrun! His sharp mind and quick understanding, combined with an office door that always seemed open for me, made him invaluable for testing and developing ideas. His great passion for computer graphics, pared with seemingly infinite energy for attacking problems, provided an amazing research experience.

Al Barr and Emmanuel Candes have been great mentors! As members of my candidacy and defense committees, their discussions and advise have greatly benefited my research—and finally the writing of this thesis.

When I started graduate school Andrei Khodakovsky was already a member of the Multi-Res group. Ironically it took us until after he left for NVIDIA, to start a close collaboration. I have to thank Andrei for preparing experiments during his Christmas vacation for the variational normal meshes paper! In Andrei I also found a reliable partner for hikes and climbs throughout the southwestern USA.

I am indebted to Patrick Mullen, who collaborated with me on the fairing project. After graduation he left academia for a position at Microsoft. I am very happy to see him returning to Caltech now, to continue research as a graduate student.

Countless discussions with my colleague Nathan Litke gave me a different perspective on many methods and applications. He truly helped me to obtain a deeper understanding of computer graphics.

Interacting with members and associates of the Multi-Res modeling group provided me with a great intellectual environment! I am grateful to Burak Aksoylu, Eitan Grinspun, Sharif Elcott, Liliya Kharevych, Steven Schkolne, Zoë Wood, Yiying Tong, Fehmi Cirak, Sylvain Jaume, Fabio

Rossi, Kai Hormann and Igor Guskov for being kind colleagues.

I want to thank Alain Martin for making time for our regular chats on the current state of Europe and the rest of the world. Also, thank you Mika Nyström, for sharing your encyclopedic knowledge, for teaching me photography and driving. I am much obliged to the computer science department assistants Jeri Chittum, Betta Dawson, Louise Foucher and Kathryn Moran. They made my life easier by dealing with many tedious administrative tasks.

I found numerous friends during my six years at Caltech. They provided distractions from school that kept me happy and somewhat sane. For this I thank Paul for dragging me into Calaveras, Rocio for always teasing me, Connie for brightening many a day, Attila for being such a great cook and epicure, Kerry for chatting with me, Cedric for staying with me for three years, Helia for showing me her culture, Cici for sharing her wonderful cats, Anelia for reminding me how happy I am, Sharif for cooking with me, Alexei for many hikes, Dawn for being kind, Rafi for forgiving how tough a TA I was, Wrighton for being a model conservative, Rassul for some great dives, Dima for holding the belay, Weiwei and Josh for being crazy, Ann and Bobby for not forgetting me!

Thank you, Clint Dodd, for being one of the greatest teachers I ever had! He showed me the aquatic world by teaching me to swim, dive and scuba—all in his characteristic, humorous style.

And finally, thank you Anja!

Abstract

This thesis introduces a novel geometry processing pipeline based on unconstrained spherical parameterization and normal remeshing. We claim three main contributions:

First we show how to increase the stability of Normal Mesh construction, while speeding it up by decomposing the process into two stages: parameterization and remeshing. We show that the remeshing step can be seen as resampling under a small perturbation of the given parameterization. Based on this observation we describe a novel algorithm for efficient and stable (interpolating) normal mesh construction via parameterization perturbation.

Our second contribution is the introduction of Variational Normal Meshes. We describe a novel algorithm for encoding these meshes, and use our implementation to argue that variational normal meshes have a higher approximation quality than interpolating normal meshes, as expected. In particular we demonstrate that interpolating normal meshes have about 60 percent higher Hausdorff approximation error for the same number of vertices than our novel variational normal meshes. We also show that variational normal meshes have less aliasing artifacts than interpolative normal meshes.

The third contribution is on creating parameterizations for unstructured genus zero meshes. Previous approaches could only avoid collapses by introducing artificial constraints or continuous reprojections, which are avoided by our method. The key idea is to define *upper bound* energies that are still good approximations. We achieve this by dividing classical planar triangle energies by the minimum distance to the sphere center. We prove that these simple modifaction provides the desired upper bounds and are good approximations in the finite element sense.

We have implemented all algorithms and provide example results and statistical data supporting our theoretical observations.

Contents

List of Figures

Chapter 1

Introduction

This thesis is about representing and processing geometric surface data in the form of Riemannian manifolds. In well-chosen coordinate systems these surfaces can be described as scalar functions. Having methods for selecting these local coordinate systems in a generic way allows the reduction of vertex coordinate data from 3*d vectors to scalars. We develop novel algorithms for transforming surfaces hierarchically into* interpolating *and* variational normal meshes*. Because these algorithms require precomputed parameterizations as input we present a novel approach for computing spherical parameterizations leading to a simple modification of planar energies. This allows a stable minimization without the need for artificial constraints.*

1.1 Motivation

Geometry is an integral part of our sensory world. The shape of physical objects can be seen and experienced every day. Because of this close interaction, most people develop an intuitive understanding of geometric relationships. For this reason, geometry can be used as an efficient tool for visualizing physical data and for communicating abstract concepts. The expressive power of formerly static drawings of geometry has been greatly magnified by the increased processing speed of computers, and the development of interactive display techniques. It has been an ongoing project in computer graphics to make the interaction with geometric data as intuitive as dealing with physical objects.

The *geometry processing pipeline* is a paradigm describing how to handle a large class of geometric data. This paradigm has been adapted from the signal processing literature to address manifold surfaces, having inherent curvature and nontrivial topology. The pipeline concept introduces modularity, re-usability and simplicity into the transformation of geometry. This is achieved by decomposing the transformation into simpler steps. These steps cover all stages from creation and encoding, denoising, editing, compression, transmission and the final display.

Geometric data, which could be location, shape and spatial relationships of objects, is rarely observed directly. Indeed most ways of obtaining geometry by physical sensors involves sensing energy transports, as when structured light, radar or ultrasonic sound are used. Specialized drivers using computational models of the particular sensors produce raw geometric data. We call this data *raw*, as in most cases it will have many undesirable properties. Physical data is usually noisy — not just in the positions of surface samples, but also in the topology and connectivity. Data might be missing, which leads to holes in the surface. Or samples might not have been connected properly with their neighbors, resulting in unwanted flips or handles. The application or algorithm, for which the data was originally acquired, might have strong assumptions on its input. Making such assumptions is useful, as it usually reduces a processing method's internal complexity. But to

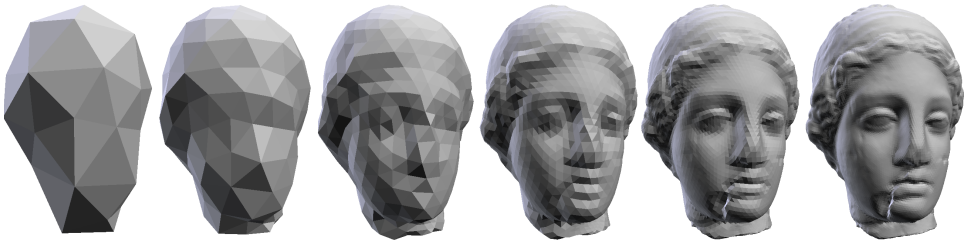


Figure 1.1: *Interpolating normal meshes are standard (semi-)regular mesh hierarchies, except that detail coefficients can be expressed as scalars using the normal directions of the coarser level mesh.*

Figure 1.2: *A 3d surface (left) is sampled by a number of points (center left). Connectivity information is added to the point cloud to form an irregularly triangulated mesh (center right). The connectivity between vertices can be simplified by resampling the geometry into a multi-resolution mesh with (semi-)regular connectivity (right).*

satisfy these assumptions, holes might have to be filled, handles removed and the overall shape of the surface smoothened. Chaining multiple such steps forms a geometry processing pipeline.

Different methods in this pipeline will work best with particular surface representations. One of the simplest ways to represent surfaces uses an unstructured collection of simple primitives, like point clouds or polygon soups (Figure 1.2). This means few assumptions are made on the coherence of the data, which is often necessary near the beginning of the geometry processing pipeline. But having no assumption nor structure to rely on, complicates many algorithms by increasing their internal complexity to handle special cases. There has been considerable effort to add minimal structure to point clouds, for instance, by organizing them using spatial trees. This permits extending a range of complex geometry processing methods to point based surface representations [AGP+04, KB04].

We will make slightly stronger assumptions about the structure of our data. In particular we decided to work with triangle meshes having either irregular or (semi-)regular connectivity. For efficiency reasons we have a strong preference toward (semi-)regular meshes. These meshes are the result of repeated application of uniform subdivision steps, as illustrated in Figures 1.2 and 1.3. The regular structure created inside of each base patch allows for dense storage and fast evaluation using simple 2d arrays. This regularity also opens the door for application of classic signal processing methods, as wavelet analysis, filtering and compression, to manifold surfaces. For a deeper understanding on multi-resolution techniques in geometric modeling, we refer the reader to to the recent survey [DFS05]. The interpolating and variational normal meshes presented in this thesis are instances of multi-resolution transformations. Both algorithms use (semi-)regular meshes for input and output. Finally, because (semi-)regular meshes are so useful, we developed for the frequent class of genus zero meshes an automatic parameterization and remeshing method.

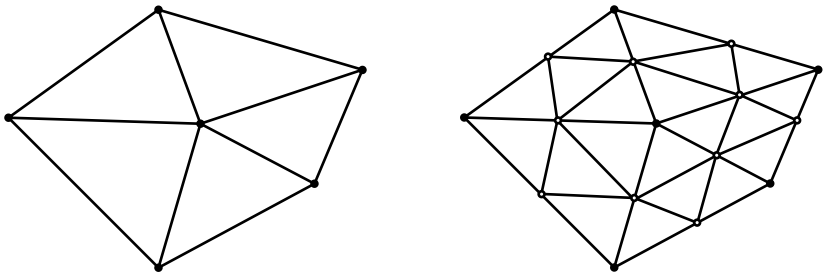


Figure 1.3: *Subdivision is a process that creates a multi-resolution hierarchy of (semi-)regular meshes. Each subdivision step replaces all coarser level triangles with* 4 *smaller triangles on the refined level.*

1.2 Overview

One of the fundamental questions of surface representation concerns the relation between approximation quality and size of the representation. Even though a full theoretical characterization is not yet available, the practical importance of efficient representations for digital geometry processing is so great that a broad variety of algorithms have been put forward. Of particular interest in the context of display, editing and compression applications are *multi-resolution* representations based on irregular [Hop96] and (semi-) regular meshes [ZSS97, GGH02]. The latter have many connections with classical functional representations such as wavelets [SS96] and Laplacian pyramids [BA83], which can be leveraged for digital geometry processing applications [SS01].

Normal Meshes Of the (semi-)regular surface representations, *normal meshes* [GVSS00]—and their non-hierarchical relatives, *displaced subdivision surfaces* [LMH00]—are of particular interest. Normal meshes are a hierarchical representation in which almost all coefficients are scalar rather than 3-vector valued. That is, level l is given as an offset from the coarser level $l-1$, with each offset being along the local normal direction on the surface. This immediate reduction in size by a factor of three can be exploited, *e.g.*, in compression representation of displacement maps [LMH00].

Unfortunately only few theoretical results, which could guide the construction of normal meshes, are known so far. For example, in [DRS04] it was shown that normal curve parameterizations possess (essentially) the same smoothness as the underlying coarse to fine predictor. The bivariate *functional* setting was studied in [JBL03] for purposes of compression.

One expects that the best results in terms of minimizing approximation error can be achieved without any constraints on the hierarchical displacement vectors. What is the penalty in terms of error if one insists on normal displacements only? What is the trade-off between allowing some non-normal coefficients and associated reduction in error? In this thesis we explore these questions

and will provide an algorithm that provides explicit control over this trade-off.

To gain the advantages of normal mesh representations, arbitrary input geometry must be remeshed so that almost all offsets are in the normal direction only. Guskov and co-workers [GVSS00] formulated this as a resampling problem using a recursive triangle quadrisection procedure based on smooth interpolating subdivision [ZSS96]. All vertices produced by this process are samples of the original surface. Since no low pass filtering is performed, this leads to aliasing artifacts (see Figure 3.2). Constraining all vertices to lie on the original mesh also increases the approximation error compared to methods, which allow a more unconstrained placement of vertices. In the method of Guskov *et al.* the parameterization needed for resampling was computed on the fly, a process which is rather expensive and numerically very delicate, in particular for large meshes.

Unconstrained Spherical Parameterization We introduce a novel approach to the construction of spherical parameterizations based on energy minimization. The energies are derived in a *general manner* from classic formulations well known in the planar parameterization setting (*e.g.*, conformal, Tutte, area, stretch energies, *etc.*), based on the following principles: the energy should (1) be a measure of spherical triangles; (2) treat energies independently of the triangle location on the sphere; and (3) converge to the continuous energy *from above* under refinement. Based on these considerations we give a very simple non-linear modification of standard formulas that fulfills all these requirements. The method avoids the often observed collapse of flat energies when they are transferred to the spherical setting without additional constraints (*e.g.*, fixing three or more points). Our *unconstrained* energy minimization problem is amenable to the use of standard solvers. Consequently the implementation effort is minimal while still achieving excellent robustness and performance through the use of widely available numerical minimization software.

Thesis Overview Our goal is the construction of low error approximations of a given surface with a (semi-)regular mesh while minimizing the number of non-normal coefficients [FSK04]. We control this trade-off by controlling the *perturbation of an initial, globally smooth parameterization* during the normal mesh construction process (Chapter 2). This is in contrast to previous methods which computed a parameterization on the fly. We will demonstrate, that separating the global parameterization computation from the remeshing phase, leads to a numerically more stable, efficient and simple resampling algorithm.

The perturbation of the parameterization creates an explicit association between the original and approximating surface, which is driven by the *geometry*. Consequently, it becomes meaningful to

ask for the best approximation in the *mean squared distance* sense (Chapter 3). A simple variational problem, to be solved at each level of the hierarchy, results in an approximation which is least squares optimal *for all levels* subject to a constraint on the magnitude of the parameterization perturbation . We achieve an overall reduction in error *and* better control of aliasing—the variational normal mesh is *approximating* rather than interpolating (see Figure 3.2). The trade-off between normality and least squares optimality can be controlled explicitly, and we show in Section 3.6 that the penalty—increase in approximation error—is small compared to the gain—reduction from 3-vector coefficients to scalars.

The construction of normal meshes requires for distortion control reasons the approximate measurement of distances. We show that this can be done very efficiently, if a globally smooth parameterization is available. In Chapter 4 we discuss the computation of smooth parameterizations for objects that are topological spheres (have genus zero). We show how to derive simple, approximate formulas for spherical energies that are *upper bounds* of the exact spherical integrals. We prove that our approximation is good in the finite element sense, as its approximation quality is $O(R^2)$ if expressed in terms of the min-containment circle diameter R. Being upper bounds, we show that the minimization is well-defined and does not collapse *even in the absence of any constraints*. This is important, as constraints often cause unnecessary, additional distortion. Finally, we examine the approximation properties of our new formulas.

The methods and applications presented in this thesis make heavy use of ideas from the numerical optimization literature. We include Appendix A as a reference on available solution methods and describe some of our practical experiences with nonlinear optimization.

Chapter 2

Interpolating Normal Meshes

Hierarchical representations of surfaces have many advantages for digital geometry processing applications. Normal meshes *are particularly attractive since their level to level displacements are in the local normal direction only. Consequently, they only require scalar coefficients to specify. We will review the construction of interpolating normal curves and meshes. We show how to decompose the construction into a parameterization and a perturbation/resampling phase. We explicitly construct the reparameterization used for perturbing the input mesh. Having a fast way to evaluate parametric correspondences will allow us later to construct variational normal meshes efficiently.*

In this chapter we discuss and develop hierarchical transformations defining canonical parameterizations based on coarser level data. Because of the "normal" dependence on coarser level data these representations give up linearity and can't be written anymore as in equation 2.1. We will see nevertheless that these representations are efficient to compute and well-behaved (in particular with respect to quantization). This makes this representation attractive for instance in compression applications [KG02].

2.1 Motivation

For many applications, a continuous signal $s(t)$ can be reasonably approximated with a linear combination of basis functions. Of particular interest is the progressive case, where a sequence of progressively more detailed functions f^l is constructed, approximating s increasingly better $f^l \to s$.

$$f^l(t) = \sum_j c_j^l \phi_j^l(t) \tag{2.1}$$

It is natural to ask for transformations from the basis functions (ϕ_j^l) into another hierarchical basis, for which the coefficients $(\bar{c}_j^l)$ are decorrelated (or sparse) for a large class of "interesting" data (Figure 2.1). Answering such questions is one topic of wavelet theory. Our primary focus is

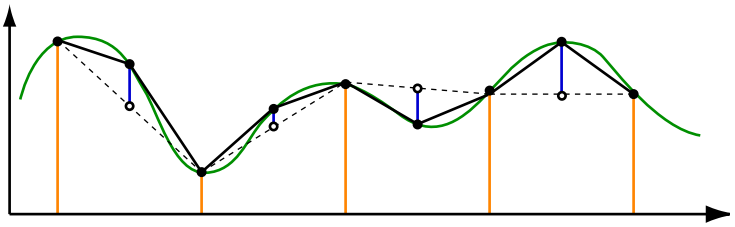


Figure 2.1: *Linear interpolating function refinement is one of the simplest basis transformations to obtain a hierarchical and sparse representation of the data.*

on extending very simple ideas from the well-understood setting of one-dimensional functions to manifolds in higher dimension (curves in the plane and surfaces in 3d). A curve c in the plane is often given in its parametric form: $c(t) = (x(t), y(t))$. Here, each of the functions $x(t)$ and $y(t)$ are one-dimensional signals, and standard wavelet theory can be applied on each signal separately. As a consequence of this approach, two sequences of coefficients c_x^l, c_x^l are obtained — which are usually interpreted as a single sequence of *vectors* $(c_x, c_y)^l$. This sequence of vectors encodes not only the geometric data, but also the chosen parameterization p. The curves $c(t)$ and $c(p(t))$ define the same set of points in the plane, as long as the parameterization $p : \mathbb{R} \to \mathbb{R}$ is a bijection. We will see that even implicitly encoding the given parameterization is costly.

2.2 Interpolating Normal Curves

It is instructive to first analyze the simple setting of curves in the plane to gain insights applicable to the more complex setting of surfaces in 3d. Guskov and coworker [GVSS00] observed that one can construct curves in the plane by specifying a hierarchy of mostly scalar offsets for the mesh vertices. In the construction of normal curves, one starts from a polyline S^0 that interpolates the reference curve R. Each segment of S^0 is divided into two smaller segments by inserting a point p, using, *e.g.*, the midpoint rule. The point insertion serves as a prediction of the missing data. A detail

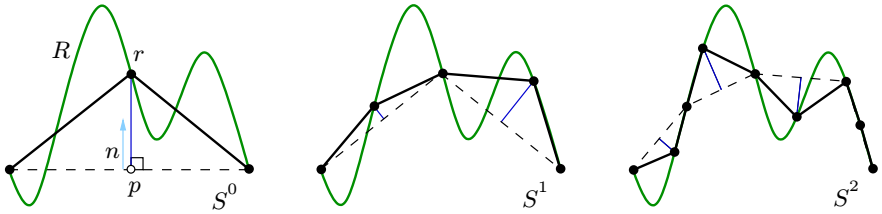


Figure 2.2: *Three levels of interpolating normal curves construction.*

coefficient expressing the difference between prediction and given data is constructed by shooting a ray from p in the normal direction n at p (see Figure 2.2). The ray intersects the reference curve R one or more times. To avoid folds in the reconstruction only intersections parametrically *between* the endpoints of the base segment are considered. One of the intersections r is picked by some heuristic—the algorithm works for a range of choices—and the scalar normal offset t is computed using $r = p + t \cdot n$.

Sometimes even the best intersection r corresponds to a parametric location on R, which is "far" from the parametric midpoint. For example, to avoid too high a distortion, one may want to reject locations r which are very close to one of the endpoints of R. As with standard tensor product refinement, the detail is encoded as a vectorial offset ("non-normal coefficient") from the prediction p to the parametric midpoint of R.[1] This decision process is typically controlled by an "aperture", defining a feasible region around the parametric midpoint covering a fraction of interval R. Having decided and encoded the detail finishes the construction of S^1, the process can now be repeated to obtain further refinements (Figure 2.2).

Higher order schemes than the midpoint rule $\frac{1}{2}[1, 1]$ are possible (and desirable for smooth input data) for predicting the positions of the newly inserted points. Examples of such schemes include the four point rule with coefficients $\frac{1}{16}[-1, 9, 9, -1]$ and the six-point rule $\frac{1}{256}[3, -25, 150, 150, -25, 3]$ [DRS04]. For predicting the normal directions, higher order rules could be used as well. In practice

[1] Using the parametric midpoint of R assumes that the given data is parameterized nicely. Instead one might be tempted to use any other point inside the interval simultaneously minimizing the approximation error and parametric distortion.

this is rarely done; the reason appears to be a too strong sensitivity to perturbations of the input data. Because of this, [GVSS00] and [DRS04] restrict themselves to predicting normals with the midpoint rule.

2.2.1 Convergence Analysis of Daubechies, Runborg and Sweldens

The theoretical behavior of interpolating normal curves was studied by Daubechies, Runborg and Sweldens in [DRS04]. Their main questions were on the decay of the normal offsets and the regularity of the resulting parameterization. We will discuss some of their results.

Midpoint predictors will always produce valid intersections as long as the input curve is continuous. If disallowing non-normal detail and aperture control, higher order predictors, such as the four-point scheme, will fail for certain input curves. Reasons for such failure include a lack of intersections with the reference curve, invalid intersections with the reference curve outside of the corresponding parametric interval (creating folds), or due to segments of the curve that are not refined by the process. Theorem 3.5 in [DRS04] states conditions on the subdivision rules and the initial spacing of coarsest normal curve points, such that the interpolating normal curve refinement converges without introducing non-normal detail. This theorem also shows that the spacing between the points declines exponentially with each refinement level. This supports the intuitive notion that non-normal offsets have less importance on finer reconstruction levels. (See also the remarks in [GVSS00] on relaxing the aperture for finer levels.)

Under the conditions of Theorem 3.5 the smoothness of the normal parameterization depends on the smoothness of the reference curve and the regularity of the subdivision scheme. For smooth input, the midpoint rule leads to $C^{1-\epsilon}$ continuous normal curves, while the four-point scheme achieves $C^{2-\epsilon}$ continuity in the limit.

Normal offsets (“wavelet” coefficients) decline with exponential rate (Theorem 3.6). Even though the normal curve construction is a nonlinear process: under small perturbations of the coefficients we can expect to observe a stable reconstruction of the original curve (Theorem 3.7 [DRS04]). This somewhat justifies using normal meshes in progressive compression applications; quantization of the coefficients has predictable effect on the approximation quality of the reconstruction.

2.3 Interpolating Normal Meshes

Guskov and coworkers extended the 2d algorithm from curves to surfaces by drawing curves onto irregular meshes [GVSS00]. The surface was pierced by rays as described before in 2d. But because

The old interpolating normal meshing algorithm

Step 0: Input.
Given an arbitrary (irregular) triangle mesh M.

Step 1: Mesh simplification.
Obtain — for instance by half-edge collapses — an irregular hierarchy $(M^0, M^1, \ldots, M^L)$ with base mesh M^0 and finest level mesh $M^L = M$.

Step 2: Building an initial net of curves.
Map the edges of the coarsest mesh M^0 to the finer meshes $M^1, \ldots, M^L$ using bijection between the levels (for instance MAPS).
(This defines a set of curves C^0 on each level M^l.)

Step 3: Fixing the global vertices.
Relax the vertex positions of C^0 to obtain a nicer base curve network.
Redraw curves on finer levels.
(In general these curves will not appear straight or smooth on finer levels.)

Step 4: Fixing the global edges.
Canonically parameterize the area of M defined by two triangular areas of C^i sharing an edge c and redraw the curve c as iso-parameter line on M.
(This has the effect of smoothening each curve piece.)

Step 5: Initial parameterization.
Parameterize the interior of each base triangle keeping the boundaries fixed.
(At this stage a smooth global parameterization of M is obtained.)

Step 6: Piercing.
Subdivide the current net of curves to obtain C^{l+1}.
Predict normals and new point positions — pierce with M.
Reject intersections according to the aperture criterion.

Step 7: Adjusting the parameterization.
Update new point positions.
Redraw curves through intersection points.

Increment level l and continue with step 4.

Figure 2.3: *The old normal mesh construction algorithm as described by Guskov and co-workers [GVSS00]. The algorithm interleaves global parameterization and normal remeshing on each refinement level.*

curves on manifolds are not necessarily flat, the rays would pierce the surface at some distance from the existing curve network. This made it necessary to extend the parameter domain from curves to the whole surface. We will recall how these problems where addressed by Guskov et al. [GVSS00] before discussing our more modular approach.

2.3.1 The Previous Normal Remeshing Algorithm

The interpolating normal meshing algorithm is stated for convenience in Figure 2.3. The algorithm starts with an irregular input mesh M. Steps 1 to 3 describe how a "nice" base network of curves C_0 is automatically obtained from M. This is achieved via mesh simplification of M to get an irregular hierarchy of meshes. The base mesh M_0 is used as an initial approximation of the curve network C_0. In step 3 the aspect ratios of the curves were improved by relaxing the network's knot positions with respect to M_0. The curve network as observed on the finest mesh M was straightened in step 4 by redrawing each curve as an iso-parameter line after parameterization. A natural candidate for angle preserving parameterizations are harmonic maps computed for instance using the discrete Dirichlet energy [PP93]. Guskov and coworkers decided on using Floater's weights [Flo97] which has the advantage of guaranteeing injective solutions. In addition to straightening the curves this parameterization was also used to define correspondences for the regions in between the curves (step 5).

The network of curves was refined in step 6 by a subdivision step and the new point positions were predicted by the interpolating Butterfly rule [DLG90]. These points and associated normals were used to intersect the original surface M in a piercing step to obtain distance scalars and corresponding intersection parameter values. An aperture criterion was used on the parameter values to determine acceptable intersection points. To keep the previously computed parameterization consistent with the newly found intersections parameters, the network of curves was redrawn in step 7 through the intersection.

This effectively meant that after each level of refinement the entire surface parameterization had to be recomputed. Generally, this is costly because the irregular input mesh and the (semi-)regular curve network overlap arbitrarily across triangle edges and faces. The enforcement of the intersection constraints during relaxation also has the potential of cutting triangles at poor aspect ratios. All of this made the original method numerically challenging.

2.3.2 Our Simplified Algorithm

Guskov and co-workers [GVSS00] decided to design an algorithms using irregular meshes as input and produce (semi-)regular normal remeshes. An option that they mentioned was creating progressive irregular normal mesh hierarchies. We follow a different approach and describe a normal remeshing algorithm using (semi-)regular meshes both for input and output. The reasoning behind this is to increase the modularity of the geometry processing pipeline and leave the conversion

from irregular triangulations to (semi-)regular meshes to one of the many available globally smooth parameterization algorithms like MAPS [LSS$^+$98], GI [GGH02], GSP [KLS03], or the spherical parameterization algorithm presented in Chapter 4.

Increasing modularity To achieve increased modularity, we will show how to separate the intermingled parameterization and normal remeshing steps 4 to 7 of the old algorithm in Figure 2.3, such that the geometry processing pipeline appears as:

1. Compute a global parameterization for the irregular input mesh R.
2. Remesh R into a (semi-)regular mesh $\bar{R}$.
3. Remesh $\bar{R}$ into a (semi)-regular interpolating normal mesh.

As a benefit to this, we will see that using precomputed (semi-)regular remeshes simplifies the implementation, *greatly* improving the numerical stability of the normal mesh construction. It also opens the algorithm for extension to variational normal meshes as discussed in Chapter 3.

Observations We are now going to make a few observations on the old algorithm [GVSS00]. Extending the 2d case to 3d surfaces via straight line drawings might appear natural at first sight. Drawing straight lines on surfaces corresponds to computing geodesics. These can be obtained for instance by using the angle preserving properties of some parameterization schemes [PP93, Flo97], or alternatively the direct computation of shortest paths on surfaces [KGH04]. There has been great progress in speeding these computations up [AKS, KGH04], but because of their frequent use as an elementary operation by Guskov's normal remeshing algorithm (steps 4 and 5 in Figure 2.3) the accumulative cost is still relatively high. Our main insight is that encoding interpolating normal meshes only needs two metric operations: finding "midpoints" between two points and measuring distances. These operations needed are well supported by (semi-)regular meshes:

- ◇ Evaluating a surface R for a given base patch of S at arbitrary barycentric coordinates is easily realized through a logarithmic time traversal of the (semi-)regular hierarchy.
- ◇ The inverse operation, *e.g.*, turning a ray intersection at the finest level into a coordinate value with respect to the base patch, is similarly easy to implement and efficient to run.

This allows the computation of parametric distances *within* a base patch. Using a (semi-) regular parameterization also reduces the complexity of flattening R locally, which is needed if distances are to be computed *across* base patch boundaries. Finally, using a (semi-)regular remesh as input places

no greater restriction than using a parameterized irregular mesh, because parameterized meshes can be converted very efficiently into other representations [AMD02].

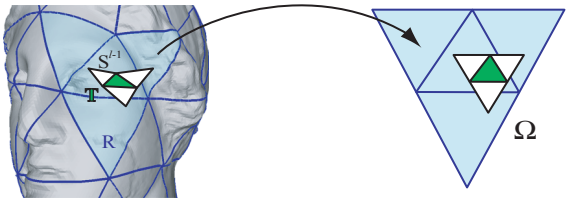


Figure 2.4: *Flattening of a region around remesh triangle T defined by a base patch and its three neighbors.*

The curves drawn on the surface by the old algorithm implicitly established a correspondence between the remesh and the input data. In the new algorithm we will encode this correspondence explicitly by using piecewise linear *reparameterizations* p^i. As in the old algorithm we will need to compute distances. Sometimes this is required across base domains as shown in Figure 2.4. Using (semi-)regular meshes we will attempt to create a larger, *flat* domain of the input mesh R that includes the triangle T in question and all of its 3 neighbors. In [KLS03] exactly this problem was solved (iteratively) by expressing the barycentric coordinates of one base domain triangle with respect to a selected neighboring base domain triangle. Doing so is somewhat involved, because one has to select a specific sequence of domain crossings. We avoid the problem of selecting this sequence of crossings by performing only one step of the process, *e.g.*, by flattening the three neighbors of a base domain triangle only (Figure 2.4). This is done using the *hinge map* of [LSS$^+$98, KLS03], which simply extends the barycentric coordinates of a triangle to its three neighbors. In the very rare case that an even larger flattened domain is needed, the algorithm creates a non-normal vertex. We have not observed any negative impact of this restriction in our experiments. (Larger parametric displacements are rare and in any event are better dealt with through a non-normal coefficient.) Thus the worst case requires flattening a base mesh triangle patch of R and its three patch neighbors.

Now, associate the new vertices of S^l with the parametric values of the intersections, as in the curve case, to build the new piecewise linear p^l. Because the topology of p^l is the same as of S^l, one does not need to construct a new mesh for p^l. Instead, we store the parameter values as attributes of the vertices in S^l. Figure 2.6 (right) shows the new S^l with one intersection rejected and replaced with a point on R, which corresponds to the parametric midpoint (red dot) analogous to the curve case.

Simplified interpolating normal meshing algorithm

Step 0: Initialization.
Given a (semi-)regular triangle mesh hierarchy $R = (R^0, R^1, \ldots, R^L)$.
Set remesh $S^0 = R^0$ and initialize reparameterization $p^0 = id$.
Set multi-resolution level $l = 1$.

Step 1: Subdivide and predict.
Subdivide connectivity of normal mesh S^{l-1} to obtain connectivity of S^l.
Predict new positions and normals of odd vertices (midpoint or butterfly rule).

Step 2: Piercing.
Intersect input mesh R^L at finest level with normal lines.
Evaluate vertex positions and flattened parameter values.

Step 3: Select intersections.
Reject intersections
if new triangle is flipped in flattened domain.
if intersection deviates too much from parametric midpoint (aperture).
Of remaining intersections select the intersection closest to parametric midpoint.
If all intersections rejected create non-normal offset (parametric midpoint).

Step 4: Update normal mesh and reparameterization.
Update new vertices with
intersection coordinates to obtain S^l.
intersection parameter values to obtain p^l.

Increment level l and continue with step 1.

Figure 2.5: *Our simplified construction of interpolating normal meshes from a semi-regular mesh hierarchy. The most obvious differences compared to the old algorithm of Figure 2.3 is the removal of any curve drawings. This is achieved by indexing into the precomputed global parameterization using the piecewise linear (semi-)regular reparameterization p^l and local flattening of the input domain.*

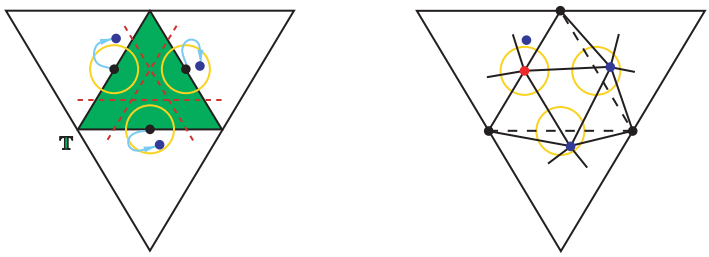


Figure 2.6: *Piercing and reparameterization in the parameter domain: One of the new details (blue) pierces the mesh outside of its aperture (yellow circles). This causes the creation of a non-normal vertex, whose parameter value (red dot) remains as predicted. The parameter values of normal vertices on the other hand are slightly perturbed.*

Our new algorithm In terms of the above assumption on the input, our algorithm (Figure 2.5) starts with a hierarchy of meshes $R^0, R^1, \ldots, R^L = R$. With $S^0 := R^0$ as the base domain, the parameterization perturbation starts with the identity, $p^0 := id$. Note that if vertex insertion were always performed at parametric midpoints of R, all offsets would (in general) be vectorial and for all i, $p^l := id$.

Let T be some triangle of the normal remesh S^{l-1}. This triangle (green in Figure 2.4) and its neighbors (white) are in most cases completely contained inside a base domain triangle (blue boundaries). In this case one can compute midpoints and distances within the parametric domain as described. (For a remesh triangle that is not completely contained within a single base domain patch see below.)

Once we have flattened R in a neighborhood of T, we can make decisions on the piercing points. The triangle T and its three neighbors (see Figure 2.6 which shows the parametric domain) are associated with R via p^{l-1}. The piercing procedure begins by shooting rays from the midpoints of the edges in the normal direction to S^{l-1}. The normal direction at the midpoint of an edge is set to bisect the dihedral angle of the two incident triangles. These rays will generate intersections with R (otherwise the distance to the intersection is set to ∞ and a non-normal offset is created). Given the current parameterization these intersection points correspond to blue dots in Figure 2.6 (left). Not all intersection points can be accepted, as flipped triangles and unacceptable parametric distortion of the remesh might occur.

Triangle flips could be detected using the orientation of the vertices in the parameter domain. This test only provides the information that a flip occurred, but not which of the vertices was responsible. Consistency of the orientation is guaranteed, if it is possible to separate the intersection points onto different half-planes (red dotted lines in Figure 2.6). This conservative test is simplified further by the idea of apertures [GVSS00]. Aperture regions are circular areas (yellow) drawn

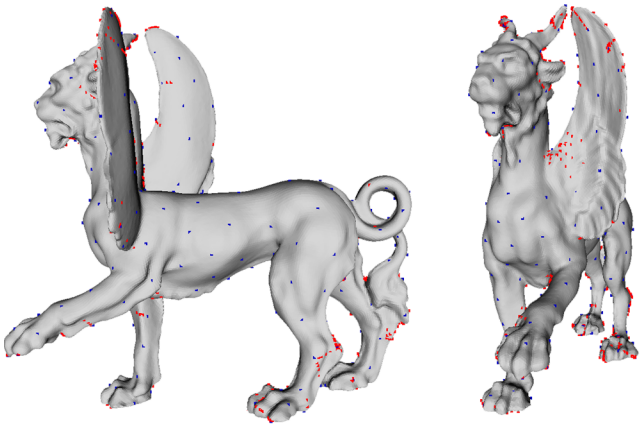


Figure 2.7: *An interpolating normal mesh (INM) of the feline dataset. Vertices of the base mesh S^0 are shown in blue while non-normal displacements (relative aperture size of 0.2) are colored red. Most non-normal displacements are due to severe geometric distortion (paws, edge of wing,* etc.*). However, there also some non-normal coefficients in geometrically "flat" regions. These are due to parametric distortion (see Fig. 2.8) causing essentially tangential displacements. The location of non-normal coefficients for VNM are very similar for this geometry.*

around the parametric midpoints of the edges in T. These regions are separated by lines, if their radius is at most one quarter of the height of the equilateral parametric triangle T. This corresponds to an aperture of about 0.43. Choosing smaller apertures reduces the deviation of the remesh from the input parameterization. This permits the user to control the parametric distortion.

2.4 Discussion

2.4.1 Efficiency of Computing the INM Transformation

We are comparing our method with standard (semi-)regular mesh refinement. Parameterization overhead is not considered.

Construction The most expensive operation in our new interpolating normal mesh algorithm is the piercing with normal directions. This operation is the same as shooting a ray in a global illumination algorithm. In our experiments, we did not use a hierarchical data structure (BSP, octree etc.) for speeding these operations up. Instead, we searched and tested intersections directly in the aperture region. (A similar search has been proposed in [LKK03] using the barycentric coordinate values of the intersection points for guidance.) This is less efficient for coarse levels in the hierarchy but performs very well for finer ones. A combined approach with a global data structure could improve the encoding times in table 3.1 further.

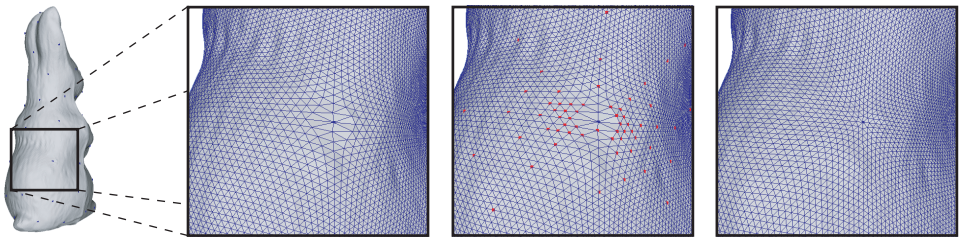


Figure 2.8: *A closeup of the neighborhood of a base mesh vertex (blue) of high valence. The distortion in the input parameterization is clearly visible (left). Because the geometry is simple, a nice remesh is achieved if we do not interfere with the normal remesh (aperture* 0.3*, right box). A small aperture (*0.05*) allows for only a small perturbation of the input mesh (middle) and results in more non-normal coefficients due to tangential displacement (red dots).*

Reconstruction Reconstruction (semi-)regular meshes from normal coefficients can be done at a cost of a single normal computation per vertex, which can be obtained as the cross product of the differences of the 4 neighbors positions. (There is no memory access overhead — at least for higher order schemes — because these vertices are needed for the point position prediction anyways.)

2.4.2 Non-Normal Coefficients

The algorithms discussed in this chapter occasionally produce vertices with non-normal displacement (red dots in Figure 2.7). On first thought, the existence of these vertices is undesirable: aren't *purely* normal meshes preferable over (even so slightly) hybridized ones? The answer depends on the application. One might argue, that displaced subdivision surfaces [LMH00] encode surface data in purely normal fashion. But this comes at the cost of a fairly large base mesh.

Running the algorithm on the same model with different apertures, we obtained normal meshes with a wide range of vectorial coefficients. Unfortunately, remeshes with less non-normal coefficients typically have a larger parametric distortion. This means that some regions are under-sampled while others are oversampled in comparison with the input parameterization. Undersampling often leads to a steep increase of the approximation error (for the same reconstruction level).[2] Undersampling can be countered by increasing the reconstruction level. This increases the number of triangles *dramatically*. Adaptive reconstruction counters this problem, and, for some applications (especially for compression applications in combination with a zero-tree coder [KG02]), such an approach could be feasible. In most cases, it is highly undesirable.

Still, we would like to argue that currently too many non-normal offsets are computed. Some non-normal coefficients are created in flat regions only to reproduce the input parameterization (Figure 2.8). If this is undesirable one could attempt to get rid of them by using an adaptive aperture.

[2] Similar observations have been made by Guskov et al. [GVSS00]) and are supported by the analysis in [DRS04].

One could analyze which how the approximation error develops with and without introduction of a particular non-normal vertex. While being a natural criterion — if not done locally and monitored over multiple refinement levels — it could be expensive.

Parametric distortion needs to be battled only in regions of highly varying curvature: it is here that the piercing procedure can produce arbitrarily bad intersections compared with the input parameterization (Figure 2.7). For this reason, we propose to analyze the reference surface using inexpensive discrete differential geometry operators [Mey04]. This could allow the development of a heuristics to scale the aperture according to the variations encountered.

2.4.3 Treatment of Boundaries

Normal mesh construction does not naturally handle boundaries. One way to process meshes with nontrivial boundaries is to tag them and encode/transmit their coefficients interleaved on each refinement level with the surface data. This approach has been demonstrated in [LKK03].

2.4.4 Extension to Higher Dimension

It has been remarked in [GVSS00] that one should be able to encode m-dimensional manifolds embedded in n-dimensional space with $n - m$ scalars per vertex.

The extension of curves ($m = 1$) to n-dimensions does not appear too hard — we know how to measure distances and find midpoints of curves. All one needs to do is to define $n - 1$ locally orthogonal directions (for instance using a Gram-Schmidt orthogonalization) and intersect the curve with the affine space defined by the prediction point and the $n-1$ directions (Figure 2.9). The higher

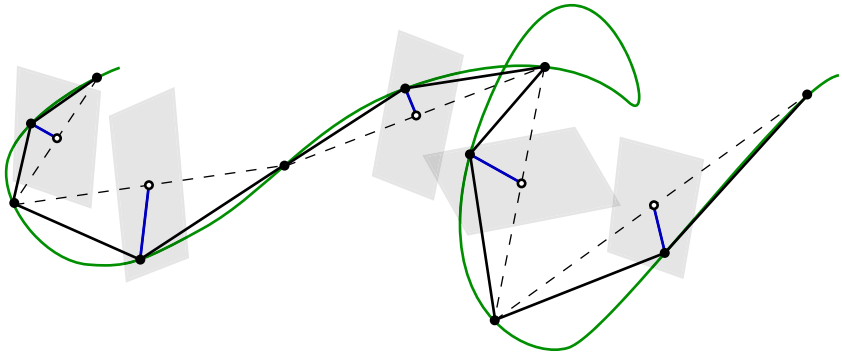


Figure 2.9: *One possible approach to generalize the normal curve construction to* 3 *dimensions is to intersect with orthogonal planes. This would reduce the data from* 3*-vectors to* $3-1=2$*-vectors.*

the dimension of the curve, the less the relative benefit: a curve in 10 dimensions is described by 9 normal scalars. This is hardly something to care about! But on the other extreme, an $n - 1$

dimensional manifold could be encoded like a surface in 3d *with scalar only*! Examples of higher dimensional data are animations and evolving iso-surfaces over time (4d).

Chapter 3

Variational Normal Meshes

The interpolating normal mesh construction creates a geometry driven *surface approximation which implicitly defines a reparameterization. Using this reparameterization we define a* L_2 *norm in normal direction. This allows us to replace the difficult geometric distance minimization problem with a much simpler least squares problem leading to a Laplacian pyramid transformation. This variational approach reduces magnitude* and *structure (aliasing) of the surface approximation error.*

3.1 Motivation

The previous chapter's interpolating normal mesh construction can be seen as a very simple wavelet scheme. Interpolating schemes are often very simple, but they come with some serious limitations. One of these limitations is that most higher order interpolative basis functions do not define (positive) partitions of unity. Interpolating basis functions tend to oscillate between data points. This often leads to undesirable artifacts in the reconstructed signal and was for instance observed by [KG02] (Figure 3.1). Using approximating scaling functions, such as cubic B-splines or Loop subdivision in the wavelet reconstruction [LDW97], leads often to a reduction of oscillations artifacts. When measured by a continuous metric approximating basis functions can provide more efficient function representations than their interpolating counterparts. For these reason it is natural to ask if it is possible to extend the definition of normal meshes to the approximating setting.

In this chapter we will introduce a *scalar* version of the Laplacian pyramid *for surfaces*. Laplacian pyramids were introduced for 2d functional setting in [BA83] where they provide the best L_2 approximations on *each* level of the refinement hierarchy. To obtain the scalar Laplacian pyramid we will derive a natural L_2 measure for normal meshes and use it to define an inner product. Having these tools we will show how to modify the interpolating normal remeshing algorithm to obtain approximating, or variational normal meshes (VNM).

Figure 3.1: *Comparison of partial reconstructions of interpolating normal meshes in a compression application using butterfly (left) and Loop (right) wavelets. Reconstructions are shown for approximately the same Hausdorff error (the compressed files are around 10KB). Note the bumps on the butterfly surface. (Figure used with permission [KG02].)*

3.2 Parametric Correspondence

We make two observations that will be important for us:

1. The result of the naïve piercing algorithm [GVSS00], which converges under mild technical conditions [DRS04], depends only on the geometries of R and S^0. Any decisions to interfere with this process—treating an intersection as "too far off the middle"—are based on the ability to measure distances and find midpoints in the parametric domain Ω of R.

2. If the interpolating normal curve refinement converges for inputs S^0 and R, then a reparameterization p^∞ is naturally defined by $S^\infty(t) = R(p^\infty(t))$ for all $t \in \Omega$. This p^∞ can be approximated on each level by a piecewise linear p^l such that for all vertices s_i of S^l we have interpolation $S^l(t_i) = s_i = R(u_i)$. Because s_i is attached to R at parameter value u_i we can construct $p^l(t_i) = u_i$. Now $S^l(t) \approx R(p^l(t))$—*implying that the difference between the two functions is a good approximation of their geometric distance.*

The latter observation is the starting point for our variational approach.

3.3 Distances and Scalar Products

Given a parameterized curve or surface R and an approximation S^{l-1} on level $l-1$, we are interested in finding the coefficients of a refined approximation S^l such that distance decreases: $d(R, S^l) < d(R, S^{l-1})$. Ideally, this distance should be measured using the symmetric Hausdorff metric [CRS98]. Unfortunately this is costly, leading to the common use of the L_2 norm of the

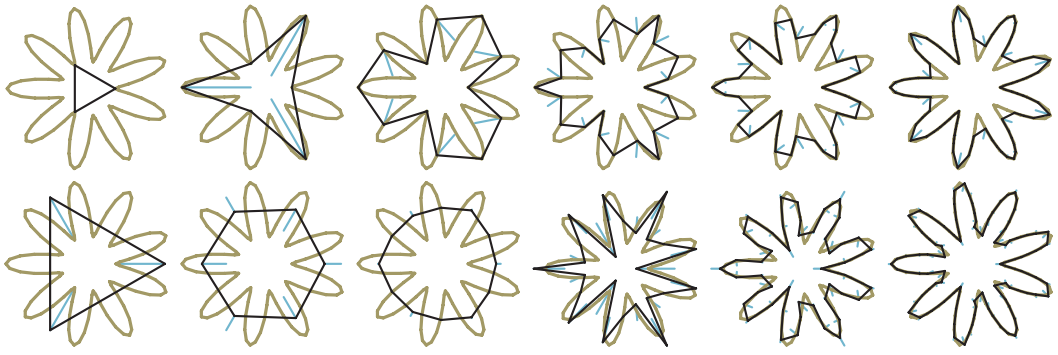


Figure 3.2: *The approximation errors of* interpolating *normal curves (top) are typically larger than for their variational counterparts (bottom). Note how the latter are low pass approximations until there are enough vertices to resolve the radial frequency avoiding aliasing artifacts (top). All coefficients (light blue) are scalar.*

distance function

$$\|d_R\| := \Big(\int_\Omega (d_R(\omega))^2 \, d\omega \Big)^{\frac{1}{2}}$$

as a way to evaluate the approximation error. Here $d_R(\omega)$ is defined on S and gives the distance to the nearest point on R for all parameter values ω in the domain Ω.

Since a parameterization of the surface gives a functional description of the surface, an even simpler norm involves parameterizations of either surface

$$\|R - S\| := \Big(\int_\Omega (R(\omega) - S(\omega))^2 \, d\omega \Big)^{\frac{1}{2}}. \tag{3.1}$$

This expression, unlike the L_2 norm of the distance function, depends on the parameterization chosen for R and S. To make it geometrically meaningful, one needs to ensure that similar parameter values describe similar regions of R and S. This can be achieved by carefully selecting a suitable reparameterization $p : \Omega \to \Omega$ for one of the surfaces. The main insight of our work is that this type of reparameterization is precisely what the "piercing" procedure in the normal mesh construction produces. Using $\|R \circ p - S\|$ as a distance measure one can then hope for a behavior that resembles the L_2 norm measure of the distance function. A consequence of using $\|R \circ p - S\|$ is that one can easily solve the variational problem

$$\arg\min \|R \circ p - (S^{l-1} + \sum_i c_i^l \phi_i^l)\|^2 \tag{3.2}$$

to obtain detail vectors c_i^l describing S^l relative to S^{l-1}. The ϕ_i^l are the basis functions of S^l—piecewise linear hats in the case of meshes. The critical advantage of Eq. (3.2) is that it defines a positive semidefinite quadratic form. Finding optimal detail vectors $c_i^l \in \mathbb{R}^3$ (or $\mathbb{R}^2$ for curves) requires only the solution of a linear system. Note that we have not yet restricted the c_i^l to be scalars.

Repeating this process at each level of refinement results in a hierarchy of coefficients c_i^l giving the best L_2 approximation *at each level*. For surfaces, these coefficients can be arranged in a Laplacian pyramid [BA83]. Letting N be the number of coefficients in the finest level L, the total number of pyramid coefficients is $(1 + \frac{1}{4} + \frac{1}{16} + \ldots)N \leq 4/3N$, a modest overhead for the flexibility afforded. An *orthogonal* wavelet hierarchy could reduce this to N coefficients; to our knowledge no such construction is available for general surfaces.

Figure 3.3: *Construction of approximating normal curves: correspondence of parameter values (left) and position of vertices after minimization (right).*

3.4 Variational Normal Curves

To turn the above ideas into a practical algorithm we need to make some specific choices:

- ◇ *Scalar* detail coefficients are allowed for *odd* (new) and *even* (old) vertices anywhere in the hierarchy.
- ◇ *Vectorial* details are only allowed for odd vertices and will be used sparingly.
- ◇ No flags, except whether an odd coefficient is scalar or vectorial, are created.

The last choice is motivated largely by limiting the side information needed to inverse transform the hierarchical surface representation.

In the standard interpolating construction, normal directions are used only once when moving a newly created (odd) vertex to its position on the reference curve R. In the variational algorithm, we need to keep directions fixed, but allow vertices to slide along their *normal line*. A normal line corresponding to a vertex s_i of S is defined by its position and normal vector *at insertion time*. Vertices are free to slide along their normal lines, but are never allowed to leave them. We must allow such motion to ensure that the vertex s_i can converge for $l \to \infty$ to the intersection point of its normal line with R. Directions of normal lines are held fixed once they have been created, though.

The variational refinement algorithm for curves consists of the following steps:

1. Refine mesh S^{l-1} by predicting odd points;
2. Find intersections of predicted normal lines with R;
3. Accept an intersection or select a vectorial offset;
4. Update the parameter perturbation p^l from p^{l-1};
5. Define (tangentially displaced) normal lines for vectorial offsets;
6. Minimize the variational functional restricted to normal lines to obtain coefficients describing S^l.

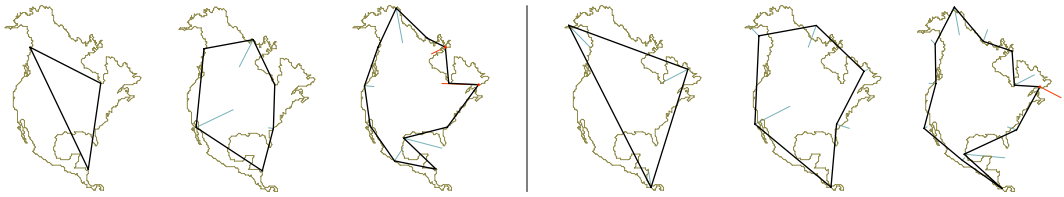


Figure 3.4: *The North American coast line represented using interpolating (left) and variational normal curves (right).*

The first three steps are essentially the same as in the interpolating curve construction. Here we focus on the remaining steps.

- ◇ The perturbation p^l is constructed by keeping $p^l(t_{2i}) := p^{l-1}(t_i)$ for even vertices (see Figure 3.3 for the various parameter locations and values). Let $R(u_{2i+1})$ be the intersection of the normal line with the reference surface and set $p^l(t_{2i+1}) := u_{2i+1}$ (blue dot on R). For a non-normal coefficient, inserted at the parametric midpoint of R, we would use a parameter value of $(u_{2i} + u_{2i+2})/2$ (red dot on R). This is all we need to define the piecewise linear reparameterization on the new level. Note that once a parameter value u is associated with t through the perturbation p, it will never change.
- ◇ Having defined the new parameterization, Eq. (3.2) is well defined at level l, and we may minimize it to determine the c_i^l. In Figure 3.3, the coefficients c_i^l move vertices along the normal lines, but in general do not interpolate R.
- ◇ Non-normal offsets should be allowed to participate in the minimization scheme. For this purpose we assign such coefficients a (translated) normal line anchored at $R(u_{2i+1})$, parallel to the originally predicted normal direction n_{2i+1} of S. Instead of recording the vectorial offset to $R(u_{2i+1})$ plus the scalar coefficient c_{2i+1} resulting from the minimization, we only record the final positions $R(u_{2i+1}) + c_{2i+1} \cdot n_{2i+1}$ of these vertices and use these as the origin of the associated normal lines.

Computing the minimum of a quadratic form requires the solution of a linear system $b = K \cdot c$ of normal equations. The load vector is defined by

$$b_i = \langle R, \phi_i^l \rangle = \int R(p(t)) \cdot \phi_i^l(t) \cdot \|(R \circ p)'(t)\| \, dt, \tag{3.3}$$

and the mass matrix is defined by

$$K_{ij} = \langle \phi_i^l, \phi_j^l \rangle = \int \phi_i^l(t) \cdot \phi_j^l(t) \cdot \|(R \circ p)'(t)\| \, dt. \tag{3.4}$$

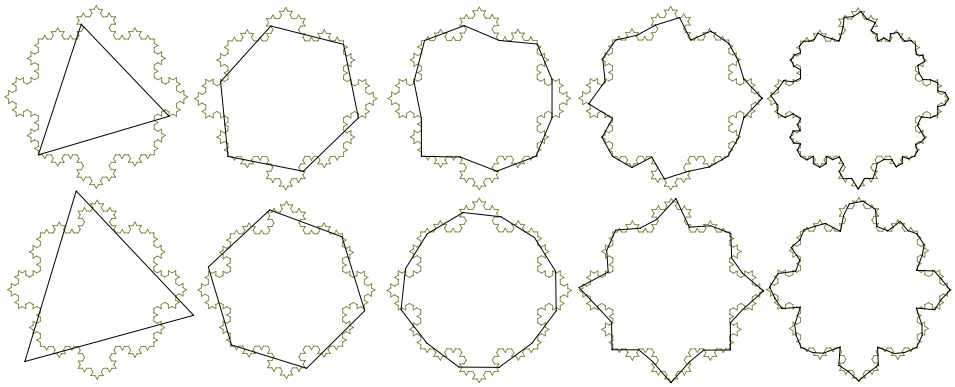


Figure 3.5: *As shown in [GVSS00] interpolating normal curves can describe complicated data like Koch curves exactly. This requires perfectly picked base mesh and parameter values. If the initial sampling is chosen without sensitivity to the underlying structure as in this figure, aliasing artifacts develop (top row). Variational normal curves handle this situation more gracefully (bottom row) with lower approximation error.*

If no area weighting is used, the entries of K can be computed offline. In practice though it is more appropriate to take the actual triangle sizes into account. Computation of the entries of b requires online quadrature because of their dependence on R. For simple B-spline basis functions this quadrature, can be performed exactly using algebraic formulas. In any case, the setup of the linear system is straightforward.

Even though the approximation is not interpolating, we are using the fact that the *basis functions* are interpolating. Consider two neighboring basis functions ϕ_i, ϕ_j and suppose that ϕ_i is nonzero at the normal line of ϕ_j. Changing the coefficient of c_i could then "push" c_j off its normal line unless the two normal lines happen to be parallel. Because interpolating basis functions, such as piecewise linear hats, are zero at the normal lines of all other vertices, we do not need to worry about this effect.

3.5 Variational Normal Meshes

As during interpolating normal mesh construction, we do not compute a parameterization of the mesh on the fly but rather rely on a pre-existing parameterization. Please recall that this could be produced with any of the algorithms now available for the construction of low distortion, globally smooth parameterizations, *e.g.*, MAPS [LSS$^+$98], GI [GGH02], or GSP [KLS03]. Our only desire is that the parameterization be *locally* close to an isometry to simplify finding reasonable "midpoints" between two vertices on R in the non-normal case. Figure 3.6 gives a summary of the variational normal remeshing algorithm. Notice that Steps 1 to 3 are identical to the interpolating algorithm in

Variational normal meshing algorithm

Step 0: Initialization.
Given a (semi-)regular triangle mesh hierarchy $R = (R^0, R^1, \ldots, R^L)$.
Set remesh $S^0 = R^0$ and initialize reparameterization $p^0 = id$.
Set multi-resolution level $l = 1$.

Step 1: Subdivide and predict.
Subdivide connectivity of normal mesh S^{l-1} to obtain connectivity of S^l.
Predict new positions and normals of odd vertices (midpoint or butterfly rule).

Step 2: Piercing.
Intersect input mesh R^L at finest level with normal lines.
Evaluate vertex positions and flattened parameter values.

Step 3: Select intersections.
Reject intersections
if new triangle is flipped in flattened domain.
if intersection deviates too much from parametric midpoint (aperture).
Of remaining intersections select the intersection closest to parametric midpoint.
If all intersections rejected create non-normal offset (parametric midpoint).

Step 4: Update reparameterization and define normal lines.
Update new vertices with
intersection parameter values to obtain p^l.
predicted normal directions.

Step 5: Compute inner products.
Use the new reparameterization p^l to compute
the load vector $b_i = \langle R, \phi_i^l \rangle$ and
the mass matrix $K_{ij} = \langle \phi_i^l, \phi_j^l \rangle$.

Step 6: Minimize variational functional.
Solve the linear equation system $b = K \cdot c$ for the coefficients c.

Increment level l and continue with step 1.

Figure 3.6: *The variational normal mesh algorithm is an extension of the interpolating normal algorithm as steps 1 to 4 are essentially the same. During the new steps 5 and 6 a least mean square problem along the normal lines is solved to minimize the L_2 approximation error.*

Figure 2.5. Step 4 minimally differs as normal directions are explicitly stored for updates in finer levels. For this reason we restrict our further discussion to Steps 5 and 6.

Semi-regular meshes efficiently support the numerical evaluation of the surface at arbitrary parameter values. This comes handy when setting up the least square system $b = K \cdot c$ for solving for the c_i^l, *i.e.*, the updated location of the vertices of S^l along their normal lines. The basis functions of R and S may overlap arbitrarily in the parametric domain. This makes the exact evaluation of the 2d integrals highly impractical. For hat basis functions, without taking account of the surface element on R, the mass matrix has entries $K_{ii} = valence_i/12$ and $K_{ij} = 1/12$ if i and j are connected by an edge. This matrix, for example, was used in [LDW97] for the construction of wavelets over (semi-)regular meshes. Since triangles are generally not uniform in size, we use numerical integration to compute the entries of K and take the actual surface area into account. For this we employ the midpoint quadrature rule with between 30 and 150 samples per triangle of S^l to evaluate the load vector and mass integrals in equations 3.3 and 3.4. The numerical evaluation of the basis elements of S^l is a trivial operation. The evaluation of the corresponding $R \circ p$ can be performed by direct access in $O(1)$ time, if the coefficients of the (semi-)regular mesh R are organized for each base patch as arrays. Otherwise a logarithmic time traversal is necessary.

Finally, we minimize the quadratic L_2 approximation error by solving the linear equation system restricted to the normal lines and obtain the refinement coefficients of S^l for the new subdivision level. This concludes the description of the algorithm.

3.6 Implementation and Results

Most of the components needed for the implementation of a variational normal remesher — mesh library, ray-surface intersection, and linear solver—were taken off the shelf. The only custom implementation was the code for flattening of base triangles of R. For the variational normal mesh (VNM) code, a simple numerical integrator (midpoint) was added. We did not explore the trade-offs due to numerical integration accuracy and final approximation error (we use between 30 and 150 integration points per triangle).

For both INM and VNM, the *observed* runtime is linear in the number of triangles. (Note that while individual point locations are $O(\log n)$ their expected cost is $O(1)$ explaining the observed behavior.) The runtime of the VNM remesher is completely dominated by the integration code (see the representative data in Table 3.1). The timing differences between INM and VNM are due to linear equation system setup and solution. The linear solver time is on the order of a second,

data set	input param	normal method	input base mesh size (♯ levels)	remesh size (vertices)	non-normal vertices	percent B-box L_2 Error	Time (sec)
skull	MAPS	INM	4(8)	32770	368	0.0392	2.5
	MAPS	VNM	4(8)	32770	494	0.0282	15.2
fandisk	MAPS	INM	73(4)	4546	103	0.0573*	0.2
	MAPS	VNM	73(4)	4546	104	0.0345*	1.5
dino	MAPS	INM	128(4)	8066	228	0.0893*	0.3
	MAPS	VNM	128(4)	8066	294	0.0576*	2.5
igea	MAPS	INM	196(5)	49666	136	0.0148	2.3
	MAPS	VNM	196(5)	49666	121	0.0096	14.9
	GSP	INM	40(6)	38914	24	0.0156	2.1
	GSP	VNM	40(6)	38914	38	0.0099	15.1
feline	GSP	INM	280(5)	72190	589	0.0156	3.4
	GSP	VNM	280(5)	72190	845	0.0096	25.3
horse	GSP	INM	140(5)	35330	256	0.0117	1.6
	GSP	VNM	140(5)	35330	317	0.0081	11.7
rabbit	GSP	INM	100(5)	25090	20	0.0107	1.1
	GSP	VNM	100(5)	25090	24	0.0067	8.6
zone-sphere	Loop	INM	12(7)	40962	570	0.0611	2.8
	Loop	VNM	12(7)	40962	146	0.0327	17.2

Table 3.1: *Using MAPS parameterizations as input to our algorithm gives us similar remeshing errors as when using GSP. But typically the number of non-normal vertices is higher for MAPS, reflecting the fact that MAPS parameterizations are not globally smooth. Variational normal meshes (VNM) typically outperform their interpolating (INM) counterparts. Errors where computed using METRO with respect to the original, irregular mesh. An exception are the fandisk and dino models, which where compared against the finest level MAPS remesh. Hence the MAPS remeshing errors need to be added to these numbers. (We discovered that the MAPS remeshes are scaled/rotated versions of the irregular models publicly available.)*

hence the difference is essentially the cost of integration. The fact that the INM code is now so fast is partially due to the simpler flattening procedure, but also to having replaced the on the fly repeated reparameterization [GVSS00] with an up-front parameterization. Even for the variational remeshing our results compare favorably with Guskov *et al.* (accounting for our timings being taken on a 2.2 GHz P4). As we relied on available remeshes [LSS$^+$98, KLS03] the time for the initial parameterization is not reflected in our numbers. Some models are not readily available as remeshes. Here one has to take the parameterization time into account. Remeshing algorithms have evolved significantly over the past few years (see for instance [LSS$^+$98, KLS03, AKS, SAPH] for timings). The best results so far where obtained by [AKS] who report solver timings of under 40 seconds for a model containing 580k vertices (David head).

We have run experiments with a range of MAPS and GSP input parameterizations. The remesh-

ing errors of our INM algorithm are about the same as in [GVSS00] (see Table 3.1 for our results).

The anti-aliasing properties of variational normal meshes are clearly visible in the "zone" sphere example of Figure 3.7.

In terms of error, VNM give us a fairly consistent improvements over INM. Typically INM have up to 60% larger remeshing error (on any level) relative to VNM. Figure 3.8 shows comparisons between different normal mesh types for different models and the GSP input parameterization. In particular we compare against the vectorial variational mesh (VVN), where detail vectors are not direction constrained. All errors where computed with METRO [CRS98]. For the feline and igea models we compared against the *original, irregular meshes* from which the GSP where derived; while the dino and zone-sphere models are compared against a finer, (semi-)regular mesh. The only difference observed is the GSP remeshing error on the finest level of feline and igea graphs.

We observe, that both interpolating methods (INM, GSP) and also both approximating methods (VNM and VVN) show roughly identical convergence behavior. Variational meshes (VNM and VVM) also preserve volumes equally well - much better than the interpolating hierarchies (INM, GSP). This behavior is illustrated by the skull series in Figure 3.9 and the error graphs in Figure 3.10.

The number of non-normal coefficients we achieved is typically a little less than in [GVSS00]. This is even though we are using an aperture of 0.2 which we keep independent of the current refinement level, while in [GVSS00] the aperture was relaxed from 0.2 on coarse to 0.6 on finer levels. The variability in these numbers is not surprising, because the construction of normal meshes depends on the base mesh and the parameterization chosen for the metric.

As in the original paper [GVSS00] we have used a spatially invariant aperture to remesh from one level to the next. This works well in regions with simple geometry and "nice" input parameterization. In those settings, no non-normal coefficients are inserted (see the feline trunk in Figure 2.7). In regions of high curvature, non-normal coefficients *are* inserted, preventing mesh degeneration. Interestingly, flat regions sometimes produce non-normal coefficients due to excessive distortion in the original input parameterization (see the feline wing attachment and tips). Increasing the aperture locally eliminates this problem resulting in a nice reparameterization (see Figure 2.8).

The VNM algorithm samples the geometry of the input mesh fairly densely (as part of the integration routine). Thus one could hope to find a strategy that adopts the aperture locally based on this information at no extra cost. We did not run experiments to examine such strategies.

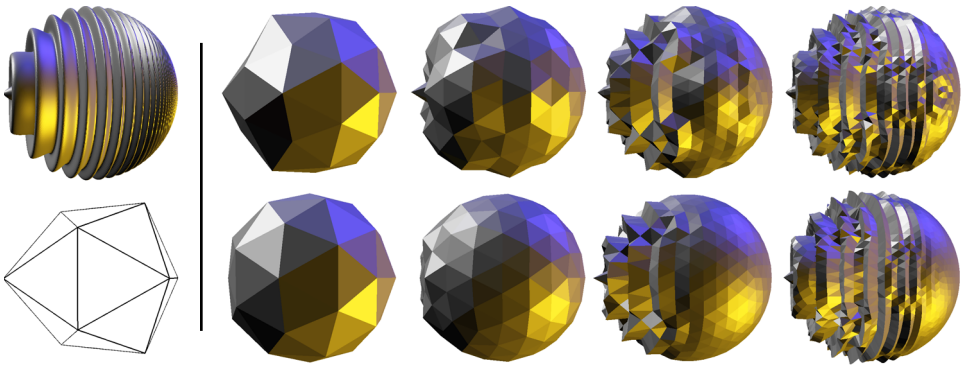


Figure 3.7: *A fine sampling of a "zone" sphere with a displacement field of increasing frequency (moving along the equator) is used to test for aliasing properties (leftmost image, also showing the icosahedral base mesh). On the right the upper row shows levels* 1 *to* 4 *of the interpolating normal mesh refinement. The right hemispheres, which contain high frequencies in the original geometry, exhibit aliasing artifacts in the interpolating construction. The corresponding variational normal meshes (bottom row) correctly low pass filter frequencies which cannot be represented at the current resolution. On the finest level both IMN and VNM show a disturbance caused by a valence* 5 *vertex (right hemisphere, center). Again, this effect is much less pronounced for the variational approximation.*

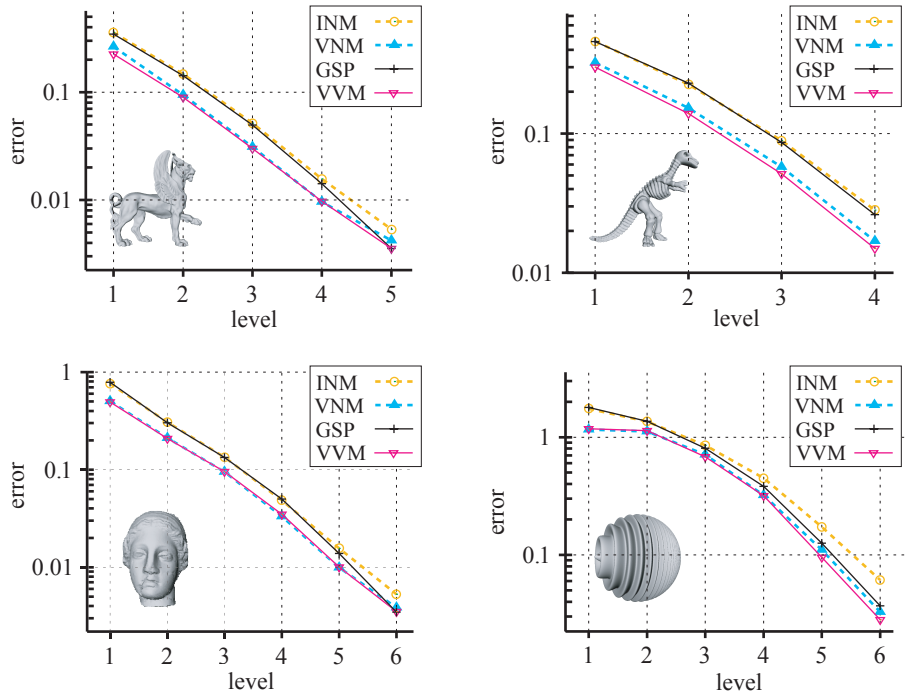


Figure 3.8: *The METRO mean squared distance errors (percent B-box) are plotted for four different models using the GSP input, normal remeshes which are: interpolating (INM), variational (VNM); and also for unconstrained variational solutions (VVM). These examples illustrates how close the constrained variational normal meshes are to the unconstrained variational meshes. Note that for the feline and igea models the errors are measured with respect to the original irregular triangle mesh, while the dino and the zone sphere meshes are compared against the finest level (semi-)regular mesh available.*

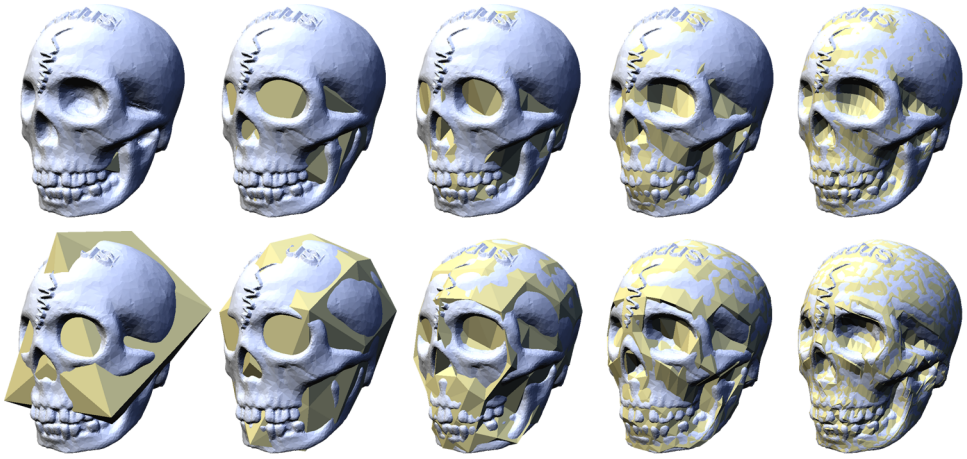


Figure 3.9: *Interpolating normal meshes are completely contained inside of convex regions of objects (top row, levels* 1 *to* 5*). This causes large errors for the volume of the reconstruction. Variational normal meshes place vertices at optimized positions (bottom row) and preserve the volumes better.*

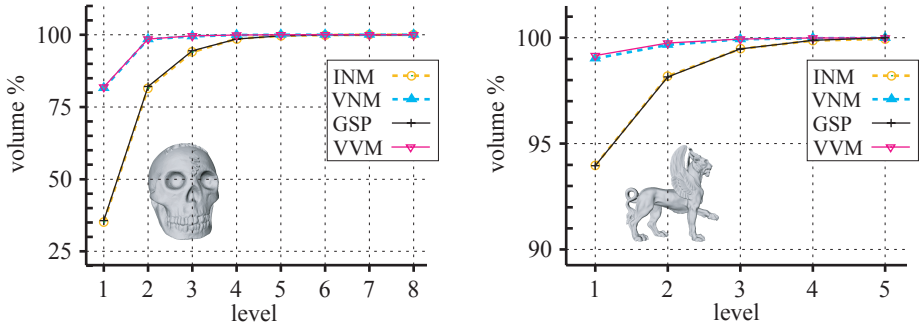


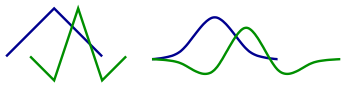
Figure 3.10: *These graphs are typical plots displaying the relative volumes (in percent of the original irregular mesh) of the* 4 *mesh hierarchies. Variational meshes consistently preserve volumes better than interpolating hierarchies. Still they are slightly biased towards underestimating the true volume. The skull base mesh is a tetrahedron, hence the graphs show a much larger volume defect than other meshes with more detailed base meshes. The relative behavior of interpolating to variational errors nevertheless is very similar.*

3.7 Future Directions

Variational normal meshes as Laplacian pyramids are data over-representations. We have shown the existence and good quality of variational normal remeshes. This result is encouraging from a theoretical view, because it might show a path towards a critically sampled normal wavelet theory. We would like to use this space to discuss some challenges that are waiting on the way.

3.7.1 A Lifting Experiment

The lifting scheme, as introduced by W. Sweldens in [Swe96], is an approach to define (biorthogonal) wavelets on complex domains. Without going into too much detail, one observes that the difference between signals on a coarse level $l-1$ and a finer level l are expressible as a combination of basis functions on the finer level l. There is some freedom in picking these linear combinations. A hat function on the coarser level can be expressed as a linear combination of $\frac{1}{2}[1,2,1]$ basis functions on the finer level [Gri03]. One way of updating detail is by using linear combinations $\frac{1}{4}[-1,2,-1]$ of hat functions on the finer level [SS95]. To solve the problem of pushing neighbor-



ing coefficients parametrically away from the normal lines (as discussed at the end of Section 3.4), one could require *the coefficients of the linear combinations to be restricted to each normal line*. We will refer to these detail basis functions as briefly as "sombreros".

The lifting (and other wavelet) schemes operate on the given data from fine to coarse levels. This approach is the opposite of normal meshing, where important data like the normal directions and *even the fundamentally important correspondence* p^l are not defined a priori and have to be discovered (non-linearly) during the coarse to fine construction. The result of adding coefficients from coarse to fine using least mean square minimization of sombreros is described in Figures 3.11 and 3.12. This can be achieved by plugging the basis refinement equation $\frac{1}{2}[-1,2,-1]$ into the linear equation system $b = K \cdot c$ used in the variational normal curve construction.

We see that sombreros predict the variational normal coefficients with high correlation. But repeated application detoriates very fast after the first level when compared to the INC and VNC convergence rates. One obvious reason for this failure is the lack of orthogonality between hats and sombreros $\langle \psi_i, \phi_j \rangle \neq 0$. Even in the classic (purely vectorial) setting, the coarse to fine least mean square computation will only converge to the original data set when adding *orthogonal* detail! The situation is further complicated, because *locally supported* orthogonal wavelet schemes for

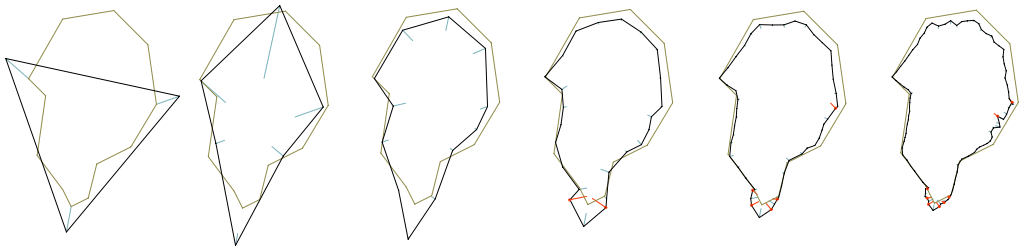


Figure 3.11: *Using sombrero functions along each levels newly inserted normal lines we attempt to reverse the lifting scheme (from coarse to fine). This approach does not converge and fails primarily because updates on finer levels are not orthogonal to data on coarser levels.*

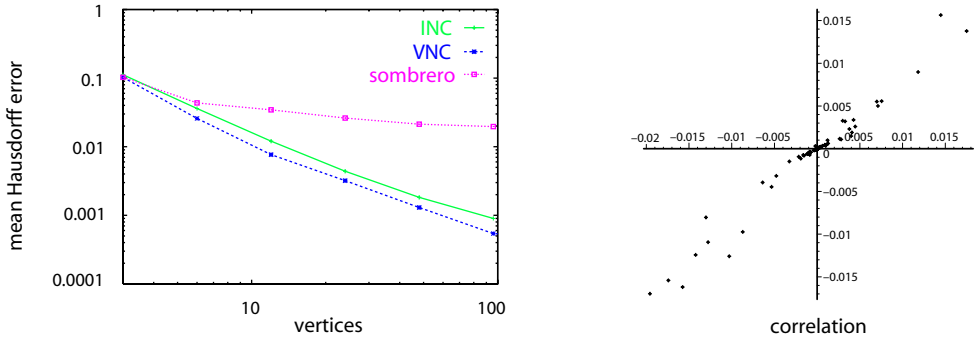


Figure 3.12: *Refinement using normal sombreros did not converge for any of our examples (graph left). Plotting coefficients obtained by normal sombrero refinement and variational normal mesh refinement we observe a strong correlation between them (right).*

subdivision surfaces are not available to our knowledge.

A potential remedy for this situation might be to relax the requirement that lifted normal meshes be best L_2 approximations *on each level* of subdivision, as long as the detoriation is "minor" in some sense. This requires studying the restrictions placed on biorthogonal wavelets by being a Riesz basis.

Chapter 4

Unconstrained Spherical Parameterization

We introduce a novel approach to the construction of spherical parameterizations based on energy minimization. The energies are derived in a general manner *from classic formulations well known in the planar parameterization setting (*e.g.*, conformal, Tutte, area, stretch energies,* etc.*), based on the following principles: the energy should (1) be a measure of spherical triangles; (2) treat energies independently of the triangle location on the sphere; and (3) converge* from above *to the continuous energy under refinement. Based on these considerations, we give a very simple non-linear modification of standard formulas that fulfills all these requirements. The method avoids the often observed collapse of flat energies when they are transferred to the spherical setting without additional constraints (*e.g.*, fixing three or more points). Our* unconstrained *energy minimization problem is amenable to the use of standard solvers. Consequently, the implementation effort is minimal while still achieving excellent robustness and performance through the use of widely available numerical minimization software.*

In this chapter we are going to develop methods for computing spherical parameterizations. By observing the planar case, where efficient algorithms and a theoretical foundation exist, we will show how to transfer ideas from planar patches to spherical domains.

4.1 Related Work

There is by now a rich literature on the construction of parameterizations for surface meshes (for an excellent recent survey we refer the reader to [FH05]). A large class of approaches is based on quadratic energy formulations which only require the solution of a linear system. Proofs of the bijectivity of the resulting mapping are available in certain cases. There are also many non-linear approaches, but their analysis is considerably more involved.

While much of this work has focused on the planar case, *i.e.*, the mapping of a topological disk region of a given mesh to the plane, spherical parameterizations have been singled out as a special case occurring frequently enough in practice to warrant their own methods [GGS03, GWC+04, HAT+00, HBS+99, SGD03, PH03, BF01]. Most of these approaches are based on applying a specific method known in the planar setting to the sphere; Praun and Hoppe [PH03], for example, use the method of Sander *et al.* [SSGH01].

One set of methods is based on puncturing the spherical topology and solving a discrete harmonic mapping functional in the plane under stereographic projection [HAT+00, HBS+99]. While the continuous conformal setting is invariant under stereographic projection this does not hold true for the *discrete* problem where the sphere is decomposed into simplicial cells. Images of spherical triangles are not straight-edge triangles in the plane and vice versa (see Figure 4.2). In fact, even on the sphere itself, the continuous Möbius degrees of freedom are lost, and only the rotations remain as symmetry group (spherical triangles are only invariant under rotations). To avoid these issues, it is therefore more natural to consider the problem directly on the sphere. For example, Gu and co-workers [GWC+04] solve the nonlinear discrete harmonic energy functional directly on an inscribed polyhedron combined with periodic centering and reprojection. It has been observed though that such approaches tend to *slip* toward degenerate solutions (see for example the comments in [GWC+04] on the use of relaxation procedures). Thus, additional constraints are imposed, for the *sole* purpose of rendering the numerics robust. Other methods, such as the approach of Gotsman and co-workers [GGS03] and Sheffer *et al.* [SGD03], are highly non-linear and numerically subtle, making them as yet unsuited for the robust parameterization of large meshes.

The discussion of the spherical setting in [GGS03] is noteworthy as it starts with a general obser-

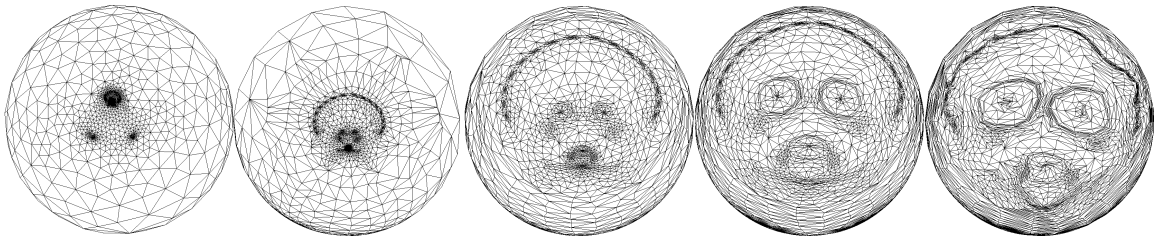


Figure 4.1: *Unconstrained solutions for our new, stable spherical parameterization operators: Tutte, Discrete Conformal Map,* $(1, 1)$ *and* $(0.2, 1)$ *weighted Dirichlet to area-distortion combinations; and pure area distortion minimization (left to right).*

vation about barycentric coordinate approaches well known from the planar setting and analyzes the implied matrix problems with the help of a special class of eigenvectors. So far little is known about specific procedures to construct matrices which satisfy the conditions necessary for the theorems to apply though.

Our approach also begins with the observation that it would be desirable to construct a general procedure to take approaches from the planar parameterization case and adapt them to the sphere. As such we are deliberately agnostic as to the particular weights being used. We will however assume that the energy derives from *area integrals* (a broad number of approaches satisfy this requirement). We will argue that many of the numerical difficulties associated with the lack of constraints in the spherical setting—there is no boundary and no canonical constraints—can be traced back to an unsuitable approximation of the underlying energy when adapting it to spherical domains (see Section 4.3). Assuming only rotation invariance, central projection, and convergence under refinement, we will give a simple modification of flat energy functionals based on a rapidly-converging *upper* bound on the corresponding integrals over spherical triangles. One of the important upshots of our new formulation is the creation of an infinite energy barrier for equatorial triangles (a single triangle covering an entire hemisphere) which gracefully prevents degeneracies. More importantly, our approach permits a robust minimization of the resulting non-linear energy *without* any artificial constraints or custom solvers. This allows us to use standard minimization software (see Section 4.4). We demonstrate the practicality of our technique with a number of examples employing different energies on various meshes of significant size.

4.2 Two Approaches to Parameterization

For patches with disk topology two approaches are popular leading to parameterizations by solving linear equation systems The first approach is variational and based on a *quadratic energies*. A famous example in this category is the Dirichlet energy for piecewise affine maps [PP93]. The min-

imum of quadratic energies can be obtained by solving positive (semi-)definite equation systems. The second parameterization approach is via *barycentric coordinates*. Here the position of each vertex is expressed as the weighted average of its one-ring neighbors [Flo03, MBLD02]. Barycentric coordinates lead naturally to linear equation systems. The main difference of the barycentric to the variational approach consists in barycentric coordinates having often no natural associations with continuous energies. Unfortunately this has consequences! Finding low-distortion spherical parameterizations is a nonlinear problem. For this reason it is not surprising that nonlinear generalizations of barycentric coordinates lead to systems of nonlinear *equations*; while the generalizations of simple quadratic energies leads to general nonlinear *energies* (often also referred to as objective functions).

When we started working on spherical parameterizations our first ideas lead to different generalizations of barycentric coordinates on the unit sphere. We still have all reason to believe that the theoretical derivations were reasonable. Finding practical solutions for them proved very difficult. The nonlinear equation system solver was quick to obtain parameterizations for simple meshes with about 100 vertices. With some user input and patience solutions could be obtained after lengthy computations for meshes with a few hundred vertices. Nevertheless it was impossible to solve anything involving more than about 1000 variables. At this point we switched from barycentric coordinates to the variational approach. Suddenly it was possible to solve parameterizations involving tens of thousands of variables. While it is possible that other program code could change this result, we do not believe this to be likely. The libraries we used (PETSc [BBG$^+$01] for solving nonlinear equation systems and TAO [BMMS04] for optimization) are well tested and share major portions of code. Instead we believe now, that solving nonlinear equation systems is more difficult than solving similar optimization problems. For a detailed discussion we refer the reader to Section A.3.4.

We will describe now the results of computing parameterizations with the variational approach.

4.3 Variational Sphere Mappings

It is an intrinsically non-linear problem to find the embedding of a genus-0 mesh on the unit sphere with minimal distortion, regardless of the choice of distortion measure. Most previous approaches have approached this issue by extending distortion measures well-known for planar parameterizations, leveraging the extensive literature on this topic. As we discuss next, these extensions assume an (often implicit) mapping between spherical and planar triangulations. If this is not properly accounted for in the final energy expression, numerical degeneracies occur.

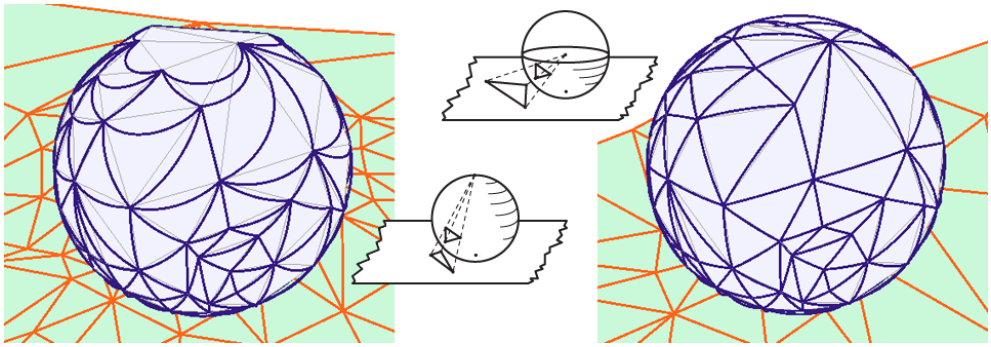


Figure 4.2: *The stereographic projection identifies straight edged triangles in the plane with banana-shaped triangles on the sphere (left). The gnomonic projection G maps spherical to planar triangles (right).*

4.3.1 Classical Parameterization Energies

Let f be (for now) a piecewise-linear map from a given (topological disk) surface patch $\mathcal{M}$ to a flat parameterization range $\mathcal{N}$ (see Figure 4.3). Many classical techniques [FH05] can be formulated as a minimization of spring-like energies of the form:

$$E(f) = \sum_{(i,j)\in directed\ \ edges} w_{ij} \cdot \|x_i^{\mathcal{N}} - x_j^{\mathcal{N}}\|^2 \tag{4.1}$$

where the coefficients w_{ij} depend on the given patch, and can often be understood as arising from area integrals. The distances $\|x_i^{\mathcal{N}} - x_j^{\mathcal{N}}\|$ are measured in the range of the parameterization. For example, the celebrated discrete conformal map corresponds to a specific w_{ij} involving only the cotangents of angles in $\mathcal{M}$ [PP93]. Many other techniques are also defined through area integrations, though the resulting (often non-linear) expressions in the unknowns $x^{\mathcal{N}}$ can be considerably more complicated [HG00, SSGH01].

It is natural to seek ways to extend these well-studied energies to the spherical setting by composition with a map from a flat triangle to a spherical triangle. Cartographers have long studied such maps and a long list of candidates exist (for an exhaustive survey see [SV89]). Unfortunately there appear to be none among these which lead to manageable expressions (see for example the set of mappings reviewed in [PH03]). Consequently authors have proposed simpler expressions which, in the continuous limit, converge to their spherical counterpart. Instead of dealing directly with a spherical triangle $T^{\mathcal{S}}_{ABC}$ for which A, B, and C are three vertices on the sphere, the most common approach considers the *secant* flat triangle (*i.e.*, the Euclidean triangle—see Fig. 4.3) $T^{\mathcal{N}}_{ABC}$ supported by the same vertices but considered in the embedding space. Since this secant triangle is flat, energies from the planar case immediately apply. Even though one may approximate a spherical tri-

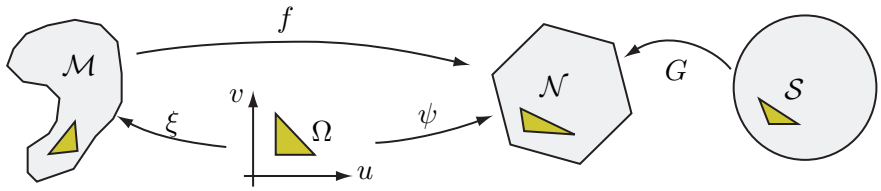


Figure 4.3: *For an input triangle mesh $\mathcal{M}$ we want to find an injective function mapping to the sphere $\mathcal{S}$ (such that the inverse of this function is a valid parameterization). Computing this function directly on $\mathcal{S}$ is very difficult. It is generally preferred to obtain an approximation f by mapping the input mesh only to the spherically inscribed mesh $\mathcal{N}$. Not accounting for the influence of G creates a bias in the approximation error, which often leads to degenerate solutions. We analyze this problem and propose a simple remedy for a range of existing methods.*

angle with a sequence of ever finer secant triangulations, the approach consistently underestimates the spherical energies: in effect the map between $T^{\mathcal{S}}_{ABC}$ and $T^{\mathcal{N}}_{ABC}$ is disregarded.

To understand the consequences of this omission we first study the projection from $T^{\mathcal{S}}_{ABC}$ to $T^{\mathcal{N}}_{ABC}$ more carefully.

Gnomonic Map: Flattening Spherical Triangles The underlying mapping mentioned above is part of a more general map type called a *gnomonic map* (or central projection). This projection maps spherical triangles to planar triangles (see [McC02] and Figure 4.2). To be more precise, a *spherical triangle* is defined by the intersection of three hemispheres.[1] If the three hemispheres are identical we call it a *hemispherical* triangle. A gnomonic projection maps a hemisphere (centered around the so-called *standard point* P) to a plane. By choosing the *specific* hemisphere and the *specific* projection plane, one can flatten any given spherical triangle, although flattening the whole sphere means picking multiple hemispheres to flatten *each* spherical triangle.

The first choice then is to pick a projection plane and a specific hemisphere to fix the gnomonic projection. A natural choice for the plane is the supporting plane of the secant triangle. For the choice of hemisphere we pick as standard point P_{ABC} the *circumcenter of the secant triangle*. This choice is symmetric and provides us with the aforementioned implicit mapping, that we will now denote by $G : \mathcal{S} \to \mathcal{N}$ (where $\mathcal{S}$ denotes the sphere; see Figure 4.3). For finer and finer triangulations G becomes the identity as expected. For this reason the mapping G has been exploited in the past when simple star maps and world globes had to be built in the shape of polyhedra [SV89]. In practice we will actually need only the inverse of this map, which has a particularly simple formula $G_P^{-1} : \mathbb{R}^3 \to \mathcal{S}, G_P^{-1}(x) = x/\|x\|$, where we assumed a unit sphere centered at the origin in $\mathbb{R}^3$.

[1] Sometimes our spherical triangle is called the *inner* spherical triangle and its larger complement the *outer* spherical triangle. This ambiguity creates much confusion and most authors assume only the smaller triangle.

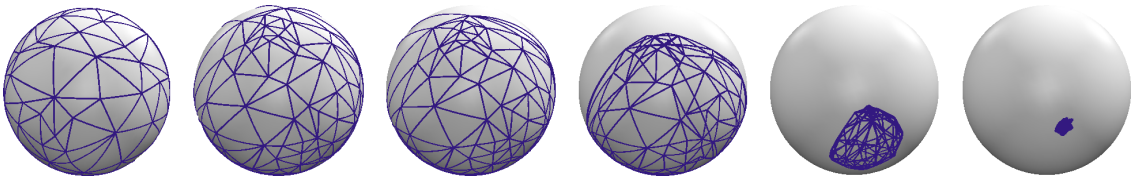


Figure 4.4: *As the iterations proceed in the solver a triangle starts growing and finally slips over the equator eventually shrinking the entire mesh to a point.*

Ignoring the Gnomonic Map Considered Harmful Consider a classic distortion energy based on maps from flat triangles (in $\mathcal{M}$) to flat triangles (in $\mathcal{N}$), *e.g.*, the Dirichlet energy. Assume that all vertices of $\mathcal{N}$ are on the sphere $\mathcal{S}$, producing a spherically inscribed mesh. Treating each of these range triangles as flat replaces the desired energy $E(G^{-1} \circ f)$ with $E(f)$. A finite element error analysis would typically give us an error estimate of the form $E(f) + O(R^k) = E(G^{-1} \circ f)$, where R is a measure of the largest triangle size (and k depends on the particulars of the energy). This type of analysis appears to confirm the validity of the approximation. However, the typical unconstrained nonlinear minimization often fails as illustrated in Figure 4.4. Considering that we are minimizing $E(f) = E(G^{-1} \circ f) \; - \; O(R^k)$, it is easy to see that the minimizer simply found a way to decrease the energy by steadily *increasing* the size of the triangle with the largest error—to the point where the triangle covers more than a hemisphere and the solution collapses. It may be possible to avoid this degeneracy by adding additional constraints, such as point or moment constraints, for example. However, this is not necessary for energies $\hat{E}(f)$ which *bound* $E(G^{-1} \circ f)$ *from above*. In the next section we will derive a very simple modification of standard weights that has this property and is tight in the sense that the error is $O(R^2)$. No additional constraints will be needed to avoid collapse of the solution.

4.3.2 From Flat to Spherical Energies

Since we assume that the energy measuring distortion arises from area integrals we consider the energy of an individual triangle with the total energy being a sum over all triangles. We derive our argument in detail for the spherical Dirichlet energy and then apply the argument to other example energies.

Spherical Dirichlet Energy Pinkall and Polthier [PP93] wrote the Dirichlet energy for discrete conformal mappings between triangles as

$$E_{\mathbf{D}}(h|_{T^{\mathcal{M}}}) = \int_{T^{\mathcal{M}}} \mathbf{tr}(Dh^T Dh). \tag{4.2}$$

For a spherical triangle the map is $h = G_P^{-1} \circ f_{ABC}$ and hence

$$\begin{aligned} E_{\mathbf{D}}(T^{\mathcal{S}}_{ABC}) &= \int_{T^{\mathcal{M}}} \mathbf{tr}(D(G_P^{-1} \circ f_{ABC})^T D(G_P^{-1} \circ f_{ABC})) \\ &\leq \int \mathbf{tr}\left(\frac{Df^T_{ABC}}{d_{min}} \cdot \frac{Df_{ABC}}{d_{min}} \right) = \frac{E_{\mathbf{D}}(f_{ABC})}{d^2_{min}} = \frac{E_{\mathbf{D}}(T^{\mathcal{N}}_{ABC})}{d^2_{min}}. \end{aligned} \tag{4.3}$$

Here we used the monotonicity of the Dirichlet operator to obtain the inequality, with $d_{min} = \min \|f_{ABC}\|$ the minimum distance of triangle $T^{\mathcal{N}}_{ABC}$ to the center of the unit sphere. For acute triangles and our choice of G_P this minimum is achieved at P_{ABC}, *i.e.*, the circumcenter of the triangle. For obtuse triangles the minimum distance is achieved at the midpoint of the longest edge. Note that d_{min} is linked to the radius R_{mc} of the min-containment circle of the flat triangle by $d^2_{min} = 1 - R^2_{mc}$. Finally rewriting Eq. 4.3 as

$$\hat{E}_{\mathbf{D}}(T^{\mathcal{N}}_{ABC}) = \frac{1}{d^2_{min}} \cdot \sum_{(i,j)\ edge\ of\ T_{ABC}} \cot \alpha_{i,j} \frac{\|x^{\mathcal{N}}_i - x^{\mathcal{N}}_j\|^2}{4} \tag{4.4}$$

shows the familiar cotangent weights in our upper bound $\hat{E}_{\mathbf{D}}$. Notice that we now have both lower *and* upper bounds on the spherical Dirichlet energy $E_{\mathbf{D}}(T^{\mathcal{N}}) \leq E_{\mathbf{D}}(T^{\mathcal{S}}) \leq \hat{E}_{\mathbf{D}}(T^{\mathcal{N}})$. Using Taylor series expansion we get $d^{-2}_{min} = 1 + O(R^2_{mc})$ with a *non-negative* error term, and thus $E_{\mathbf{D}}(T^{\mathcal{S}}) + O(R^2_{mc}) = \hat{E}_{\mathbf{D}}(T^{\mathcal{N}})$.

The implication is that the approximation error is equivalent to methods using a secant-triangle approximation; but $\hat{E}$ will approach the real spherical energy *from above* and simultaneously keep min-containment circles sizes under control: as a triangle approaches the entire hemisphere $d^{-2}_{min} \to \infty$. This makes it impossible to collapse to the trivial solution, even in the absence of any constraints.

The argument giving rise to the d^{-2}_{min} factor, which makes the Dirichlet energy of the secant triangle suitable for a spherical parameterization, applies to many other integral based energy formulations as well. We give a few examples next (see Figures 4.1 and 4.5 for a comparison).

Spherical Tutte Energy The canonical use of unit stiffness springs on edges to derive a planar parameterization (for details on Tutte's embedding see [FH05]) trivially extends to the spherical setting:

$$\hat{E}_{\mathbf{T}}(T^{\mathcal{N}}_{ABC}) = d^{-2}_{min} \cdot ((x_A - x_B)^2 + (x_B - x_C)^2 + (x_A - x_C)^2). \tag{4.5}$$

Spherical Squared Area Energy The Dirichlet energy is based on angles of the input mesh only, with no notion of preserving areas, and thus performs poorly in (re-)sampling applications. Area

dependent energies of various flavors have been considered [FH05, SSGH01] to address this issue. Using a Taylor expansion of the area formula for spherical caps, we can prove that $area(T^{\mathcal{S}}) \leq d_{min}^{-1} \cdot area(T^{\mathcal{N}})$. For our experiments we will use:

$$\hat{E}_{\mathbf{A}}(T^{\mathcal{N}}_{ABC}) = \frac{(A^{\mathcal{N}}_{ABC})^2}{d^2_{min} \cdot A^{\mathcal{M}}_{ABC}}.$$

Spherical Stretch Energy The stretch energy of [SSGH01] can be written out as a combination of the last two energies:

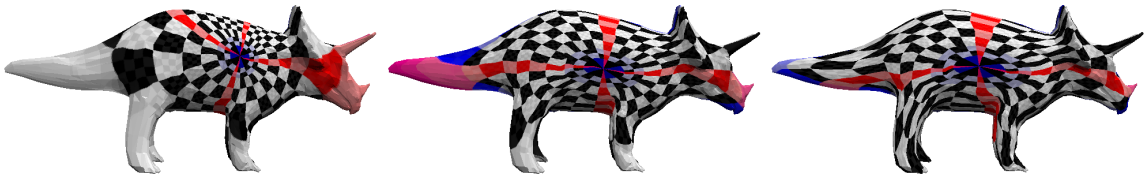
$$\begin{aligned} E_{\mathbf{S}}(f) &= \mathbf{tr}(Df^{-T}Df^{-1}) \cdot A^{\mathcal{M}} = (A^{\mathcal{M}}/A^{\mathcal{N}})^2 \cdot \mathbf{tr}(Df^{T}Df) \cdot A^{\mathcal{M}} \\ &= (A^{\mathcal{M}}/A^{\mathcal{N}})^2 \cdot E_{\mathbf{D}}(f). \end{aligned}$$

This expression explains how the stretch based parameterization strikes a balance between area and angle distortion. More importantly, we can apply our approach to obtain the upper-bound energy $\hat{E}_{\mathbf{S}}$ simply by substituting $\hat{E}_{\mathbf{D}}$ for $E_{\mathbf{D}}$. Thus one can avoid the mesh refinement step used in [PH03] and minimize the energy without requiring *additional* degrees of freedom. Of course approximation quality questions may still favor refinement for some meshes.

Combined energy We have also experimented with a combined energy $\hat{E}_{\mathbf{C}}$ that combines the area and Dirichlet energies additively:

$$\hat{E}_{\mathbf{C}}(f) = w_{\mathbf{D}} \cdot \hat{E}_{\mathbf{D}}(f) + w_{\mathbf{A}} \cdot \hat{E}_{\mathbf{A}}(f).$$

Choosing appropriate constants $(w_{\mathbf{D}}, w_{\mathbf{A}})$, the user can trade off angle and area preservation.



The left parameterization, obtained with coefficients $(1, 0)$, was optimized for angle distortion only. Consequently head and tail are under-sampled. In the center, using coefficients $(1, 1)$, a reasonable balance between under sampling and angle distortion is achieved. On the right using $(0.2, 1)$, the areas on the extremities are well-preserved - but curves do not intersect at right angles.

Compare with Figure 4.1 for views of the corresponding spherical domains.

4.3.3 Discussion

A major ingredient in any implementation of an energy minimization method is the numerical treatment of the non-linear equations that arise. The most powerful methods for general smooth non-linear problems are global Newton or Trust-Region methods. These are designed to ensure global convergence, which is very important if one starts far from the minimum of the energy. Given that there is in general no immediately valid embedding of a genus-0 mesh onto the sphere (for example, without overlap) using solvers that can robustly find a (local) minimum, no matter where one starts, is critical. It is of course possible to use custom methods, for example, a non-linear hierarchical approach which might employ a progressive mesh hierarchy to affect the solution coarse to fine. Unfortunately very little is known in terms of convergence and stability of such methods. Instead we prefer to rely on proven methods, in particular since powerful libraries implementing sophisticated black box Newton Trust-Region solvers are freely available [BBE$^+$04, BMMS04], *greatly* decreasing implementation effort. Coupling these with symbolic methods to compute gradients and Hessians of energies, rapid experimentation becomes possible.

This convenience comes at a cost. To use a black-box solver and be sure of its guarantees, the energies themselves must satisfy certain criteria. Chief among these is that Hessians provide good local (quadratic) views of the energy landscape. In our case, the division by d_{min}^2 creates poles near hemispherical triangles. Luckily, these poles are only reached by exceedingly large perturbations of the variables and are, in our experience, not a concern. The transition of the energies between acute and obtuse triangles is more subtle. Clearly it is continuous. According to our experiments we believe that even the gradient changes smoothly when deforming acute and obtuse triangles into each other. More precise statements depend on the particulars of the flat energy itself. Spring energies transition smoothly between opposite triangles orientations—and so do our modifications $\hat{E}$. We have not had any problems achieving embeddings without flipped triangles even for large and convoluted meshes (see Figure 4.5). Only E_{S}—by its very nature of assigning infinite stretch to degenerate triangles—has poles whenever triangles invert (division by $A^{\mathcal{N}}$). This gives rise to energy landscapes with many poles, which makes the use of globally perturbing black-box solvers extremely challenging.

4.4 Implementation and Results

Recall that we are optimizing a mapping from $\mathcal{M}$ to $\mathcal{N}$ with vertices $x^{\mathcal{N}}$ confined to the sphere $\mathcal{S}$, subject to a chosen upper bound energy $\hat{E}$ evaluated over flat triangles in $\mathcal{M}$. The target variables $x^{\mathcal{N}}$ are parameterized in terms of longitudes and latitudes $(x, y, z) = (\cos(\theta) \cdot \sin(\phi), \sin(\theta) \cdot \sin(\phi), \cos(\phi))$. The expression for the energy as a function of the coordinates of a given triangle $T^{\mathcal{M}}$ in the domain and a target triangle $T^{\mathcal{N}}$ is implemented in Maple. This Maple function, together with its gradient and Hessian are automatically translated to C++ code (removing a major source of errors in implementations of complicated energy expressions). Invoking symbolic differentiation might appear inefficient. We timed our codes and found that for the highly tuned PETSc [BBE+04] and TAO [BMMS04] optimization kernels, bus bandwidth is the limiting factor for problems which do not fit into the processor cache.

When using the cotangent weights appearing in the Dirichlet energy, care must be taken to avoid negative weights. These can cause fold overs in the planar setting, and we saw the same effect in the spherical case. To avoid this issue we clip all angles to $5^\circ < \alpha_{i,j} < 85^\circ$ and found this to be quite effective. Limiting the angle size from below avoids edges with very large weights (relative to their neighbors). While this tends to occur only for few angles, the resulting systems are typically better conditioned.

To initialize the optimization problem, we either start with a linear solution via the stereographic map [HAT+00], or by simply centering the model at the origin and normalizing all vertices to unit length (similar to [GWC+04]). We did not experience significant speedups using the former approach, *e.g.*, [HAT+00] method does not provide a good initial guess for our setting.

Using the initialization by normalization typically leads to many folds. After just a few iterations, models with small dynamic range in their edge lengths produce valid embeddings without folds (others take longer, but we always achieved fold free embeddings). At this point we already have a parameterization but it is visually far from being as smooth as the solution achieved at the energy minimum. For this reason, we continue our efforts until the L_2 norm of the gradient drops below $10^{-7} \ldots 10^{-9}$. This may seem excessive, but once the energy is close to the solution, the accuracy typically improves super-linearly from magnitudes such as 10^{-5} to 10^{-9} in less than 5 iterations due to the fact that a Newton method is applied.

The entire process is performed without any constraints such as fixed vertices. Even the 3 rotational degrees of freedom do not impact the solution process. Since the energy is rotationally invariant, so is the residual *magnitude*, and stopping criteria in the solver work as expected.

In terms of time performance, we found that the Trust-Region solver finds solutions for small models, such as triceratops and cow (both just under 6k triangles), in just a few seconds. For the igea model (67k triangles), the timings on a 3GHz Xeon processor are:

$(w_{\mathbf{D}}, w_{\mathbf{A}})$	$(1.0, 0.0)$	$(1.0, 0.1)$	$(1.0, 1.0)$	$(0.1, 1.0)$	$(0.0, 1.0)$
time	34m	5m	4m	12m	366m

We clearly observe a sweet spot from combining area and angle preservation. Our largest model, the 400k triangle lion vase was parameterized in just over one hour for $(w_D, w_A) = (1.0, 0.01)$. We speculate that these performance timings could be further improved by using a hierarchical preconditioning technique [AKS].

Figure 4.5 shows a number of examples computed with our solver using the $\hat{E}_{\mathbf{C}}(.)$ energy. Note the extreme texture distortion for the harmonic map on the armadillo. The entire upper body was mapped into a very small region on the sphere robustly. Figure 4.1 shows a comparison between different classical energies all lifted to the spherical setting with our d^{-2} modification. The relative effects in terms of mesh shape are qualitatively identical to the results seen in planar parameterizations. Figures 4.5 and 4.5 show remeshes of the skull and vase-lion geometry. These were obtained by concatenating a (semi-)regular spherical mesh (with icosahedral base connectivity) and the respective unconstrained spherical parameterization.

4.5 Conclusion and Future Work

We presented a simple approach to modify energies used in planar parameterization, making them directly usable for spherical parameterization. Thanks to the upper-bound derivation we used, minimizing these novel energies does *not* require the addition of constraints simply to avoid degenerate solutions (and in the process adding additional distortion due to the constraints). Aside from the generality of this approach, we also proposed a different (additive) balance between area and angle distortion. This energy provides the standard angle versus area conservation tradeoff. Using symbolic algebra methods to deal with the energies and their derivatives of first and second order, coupled with the use of canned, highly tuned solvers gives us robust methods for a variety of parameterizations on the sphere with very little implementation effort. We expect that these methods can be further improved through the use of hierarchical preconditioning techniques.

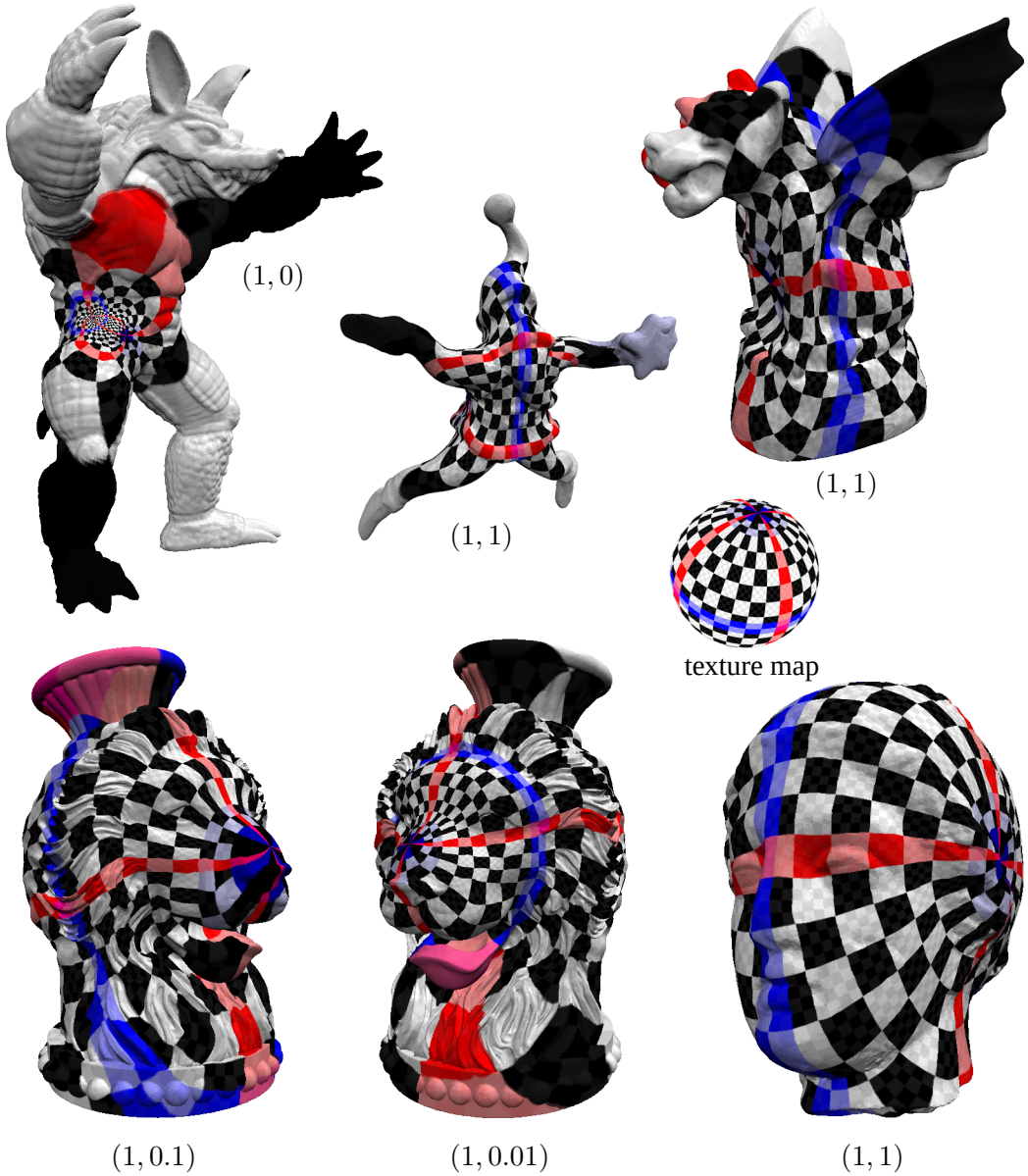


Figure 4.5: *We computed parameterizations for several large (*$200k \ldots 400k$ *triangle) meshes using the combined energy* $\hat{E}_{\mathbf{C}}$ *with weightings* $(w_{\mathbf{D}}, w_{\mathbf{A}})$ *as denoted in the image. No conditions were enforced during the solve, nevertheless the parameterizations are fold-free.*

Maple code for spherical parameterization energy

Energy:=proc(θ_i, ϕ_i, θ_j, ϕ_j, θ_k, ϕ_k)
 # convert angles into 3d coordinates
x_i:=cos(θ_i) sin(ϕ_i);
y_i:=sin(θ_i) sin(ϕ_i);
z_i:=cos(ϕ_i);
 # same for x_j, y_j, z_j *and* x_k, y_k, z_k

 # triangle edges
Ax:=$x_i - x_j$; Ay:=$y_i - y_j$; Az:=$z_i - z_j$;
Bx:=$x_i - x_k$; By:=$y_i - y_k$; Bz:=$z_i - z_k$;
Cx:=$x_j - x_k$; Cy:=$y_j - y_k$; Cz:=$z_j - z_k$;

 # compute dot products the old fashioned way
AA:=Ax*Ax+Ay*Ay+Az*Az;
BB:=Bx*Bx+By*By+Bz*Bz;
CC:=Cx*Cx+Cy*Cy+Cz*Cz;
AB:= Ax*Bx+Ay*By+Az*Bz;
AC:=-(Ax*Cx+Ay*Cy+Az*Cz);
BC:= Bx*Cx+By*Cy+Bz*Cz;

 # Area squared (use symmetric formula)
Area2:=$1/16 * (2 * AA * BB + 2 * AA * CC + 2 * CC * BB - AA^2 - BB^2 - CC^2)$;
 # radius of circumcircle on secant
RR:=$AA * BB * CC/(16 * Area2)$;

 # if triangle is acute d_{min} *is distance to circumcircle center*
if $AB \geq 0$ and $AC \geq 0$ and $BC \geq 0$ then
d_{min}^{-2}:=$1/(1 - RR)$;
tmp:=$d_{min}^{-2} * (w_{\mathbf{A}} * \frac{Area2}{area_{ijk}} + w_{\mathbf{D}} * (\alpha_{ij} * AA + \alpha_{ik} * BB + \alpha_{jk} * CC))$;
else
 # otherwise d_{min} *is distance to midpoint of longest edge*
if $AA \geq BB$ and $AA \geq CC$ then d_{min}^{-2}:=$4/((x_i + x_j)^2 + (y_i + y_j)^2 + (z_i + z_j)^2)$;
elif $BB \geq AA$ and $BB \geq CC$ then d_{min}^{-2}:=$4/((x_i + x_k)^2 + (y_i + y_k)^2 + (z_i + z_k)^2)$;
else d_{min}^{-2}:=$4/((x_k + x_j)^2 + (y_k + y_j)^2 + (z_k + z_j)^2)$;
end if;
tmp:=$d_{min}^{-2} * (w_{\mathbf{A}} * \frac{Area2}{area_{ijk}} + w_{\mathbf{D}} * (\alpha_{ij} * AA + \alpha_{ik} * BB + \alpha_{jk} * CC))$;
end if;
tmp;
end proc:

Figure 4.6: *Maple code for weighted area and spring energies.*

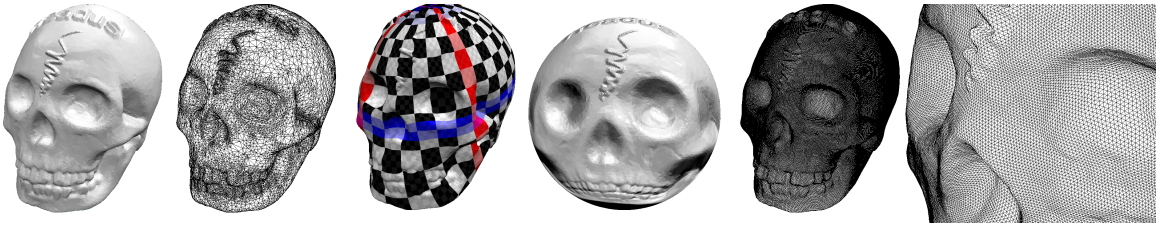


Figure 4.7: *Parameterization and remeshing of skull model. From left to right: original model, hidden-line irregular mesh, texture-mapped parameterization* $(w_D, w_A) = (1, 1)$*, normal shaded view of the spherical domain, the remesh and a close-up of the remesh.*

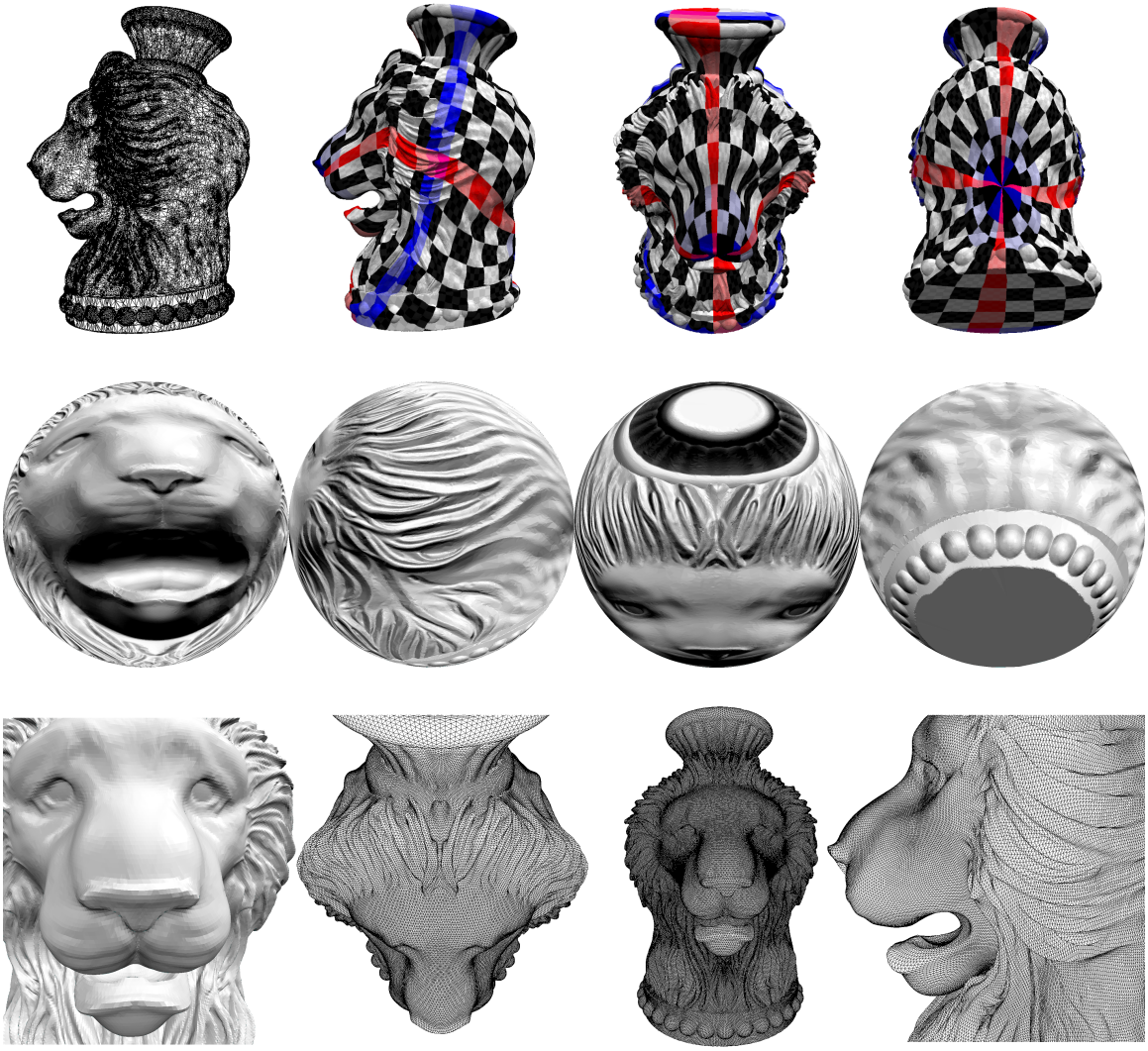


Figure 4.8: *Parameterization and remeshing of vase-lion model. The first row shows texture maps of the computed parameterization* $(w_D, w_A) = (1, 1)$*. The second row displays normal shaded views of the domain. The third row shows the remesh from different views.*

Chapter 5

Conclusion

We have presented methods for normal remeshing. Our new method simplifies existing methods and extends them to approximating construction. Because these methods rely on existing parameterizations we developed a method for spherical parameterizations which does not require the specification of artificial constraints. As a consequence distortion was reduced and less user input required. This chapter summarizes our work. We will also discuss links to other work and outline possible future directions.

5.1 Summary

We have presented a novel geometry pipeline based on unconstrained spherical parameterization and normal remeshing. There were three contributions:

First we showed how to increase the stability of Normal Mesh construction while speeding it up by decomposing the process into two stages: parameterization and remeshing. We showed that the remeshing step can be seen as resampling under a small perturbation of the given parameterization. Based on this observation we described a novel algorithm for efficient and stable (interpolating) normal mesh construction via parameterization perturbation.

Our second contribution was the introduction of Variational Normal Meshes. We described a novel algorithm for encoding these meshes and used our implementation to argue, that variational normal meshes have a higher approximation quality than interpolating normal meshes as expected. In particular we demonstrated that interpolating normal meshes have about 60 percent higher Hausdorff approximation error for the same number of vertices than our novel variational normal meshes. We also showed that variational normal meshes have less aliasing artifacts than interpolating normal meshes.

Our third contribution was the on parameterizations for unstructured genus zero meshes. Previous approaches could only avoid collapses by introducing artificial constraints or continuous reprojections, which are avoided by our method. The key idea was to define *upper bound* energies that are still good approximations. We achieve this by dividing classical planar triangle energies by the minimum distance to the sphere center. We proved that these simple modification provides the desired upper bounds and are good approximations in the finite element sense.

5.2 Future Work

Critically sampled normal basis Laplacian pyramids are not orthogonal and are only the first step toward the development of critically sampled basis transformation using approximating functions. The basis functions in our variational normal mesh construction were still interpolating (though not the approximation itself). Perhaps even better approximations can be built when using, *e.g.*, cubic B-splines. As the naïve lifting construction from coarse to fine fails due to missing orthogonality (see section 3.7.1) one would have to solve for the coefficients on all levels simultaneously in a *non-linear* minimization problem. The situation is complicated further, as the number of coefficients in this construction is not constant: depending on the insertion of non-normal vertices

the dimension of the problem can range between n in the normal and $3n$ in the fully vectorial case. Variable dimensions are not trivially covered by the classic optimization theory, posing a major challenge toward just *formulating* the problem.

Numerical methods It should be possible to improve the optimization times of the spherical parameterization problem by using more sophisticated numerical methods. The trust-region solver used [BMMS04] could not be used reliably in combination with a preconditioner. Trust-region methods need positive definite preconditioning matrices, while planar (linear) parameterization problems respond well to hierarchical preconditioning [AKS]. Another promising direction is to extend the trust-region method and permit temporary increase of the objective function during minimization. In cases with narrow and curved valleys in the objective function landscape this relaxation is sometimes more efficient than imposing strict decline [CGT00].

Displaced volumes Botsch and Kobbelt showed in [BK03] that the displacement of volumes provides good metaphor for editing subdivision surfaces. They presented a scheme reconstructing a refined surface from displaced volumes over a single level. Using volumes to specify displacements leads to an underconstrained optimization problem. The authors augmented this by a smoothing term to obtain a solution. The connection to variational normal meshes can be seen by the good volume preservation of our method as observed in section 3.6. The application in [BK03] did not require the measurement of reconstruction errors. Apparently no attempts were made to construct a multi-level hierarchy. Their regularization during reconstruction requires the knowledge of many original surface properties. Nevertheless the definition of surface detail as displacement of volumes is a very attractive idea for the future development of variational normal meshes.

Appendix A

Notes on Optimization

This chapter provides a high level review on solving certain types of nonlinear optimization problems on continuous domains. The key ideas, algorithms and references for a range of problems are given. Several theoretical limitations are stated and tips on the selection of algorithms are provided. This chapter is intended to be an introduction to the main ideas and and cannot be exhaustive.

Numerical optimization is a concept for dealing with particular linear and nonlinear systems. We have used optimization for important algorithmic steps throughout this thesis. To illuminate these steps, this chapter is intended to present background information on a range of ideas for solving optimization problems. We hope this will give the reader a better insight into our motivation when designing these algorithms. To establish context and notation Section A.1 introduces the continuous optimization problem. In Section A.2 we attempt to classify objective functions by the structure of available information. Section A.3 surveys a number of numerical techniques for solving these problems and recalls important convergence results.

The more experienced reader may skip ahead to Section A.4 where we will discuss some of the experiences that were gained during our experiments.

A.1 The Optimization Problem

A continuous optimization problem is given by an objective function

$$f : \Omega \to \mathbb{R}$$

with associated domain Ω. We typically assume that $\Omega \subseteq \mathbb{R}^n$, where n is the number of variables. Our goal is to find a *local* minimizer x_{min} of the objective, such that

$$f(x_{min}) \leq f(x), \quad \forall x \in U(x_{min}) \tag{A.1}$$

for all values x in a small enough neighborhood $U \subseteq \Omega$ of x_{min}. The domain Ω is allowed to be all or a subset of $\mathbb{R}^n$. In the latter case, we speak of a *constrained* optimization problem.

This is a *much* simpler task than the solving the *global* minimization problem, where the minimum over all $x \in \Omega$ is sought. We focus on this subproblem for several reasons:

- ◇ Some problems have only one minimum. Finding the local solution is equivalent to obtaining the global answer.
- ◇ Many physically motivated processes achieve only local minima: *Newton's apple fell straight to the ground - it did not tunnel to the lowest point on earth.* Hence, finding a nearby local solution is sometimes more useful than obtaining the global minimizer.
- ◇ Finding local minimizer is a subproblem of finding the global optimizer.
- ◇ Efficient methods have been developed for finding local minima of smooth objective functions.

What makes the continuous problem more tractable than combinatorial optimization is the ability

to resort to neighborhood arguments and the use of derivatives to find downhill directions. Many ideas for solving the continuous problem are based on the Taylor series expansion.

Theorem 1 (Taylor Series Expansion) *Let $f : \mathbb{R}^n \to \mathbb{R}$ be analytic and three times continuously differentiable. Then*

$$f(x+d) = f(x) + \mathbf{grad}\, f(x)^T \cdot d + \frac{1}{2} d^T \cdot \mathbf{Hess}\, f(x) \cdot d + O(\|d\|^3). \tag{A.2}$$

We will repeatedly use the Taylor series expansion to obtain first and second order models of the objective function f. There are different approaches that can be used for minimization, such as stochastic processes [Spa03, Pol87] and convexity [BV04]. Unfortunately, none of these other approaches is enjoying the same success and popularity in the literature as method based on the Taylor polynomial.

Most optimization methods are iterative and produce a series of better and better approximations $x_0, x_1, x_2, \ldots, x_i$ of the minimizer. Depending on availability, these methods are permitted to query some of the following properties:

- $\diamond$ $f(x_i)$ - the scalar value of the objective function. This data is used to check for progress, e.g., to make sure $f(x_{i-1}) > f(x_i)$ at a sufficiently fast rate (Wolfe condition [NW99, Kel95, CGT00]).
- $\diamond$ $\mathbf{grad}\, f(x_i)$ - the gradient of the objective function. The gradient at x_i can be represented as an $n-$dimensional vector. A vanishing gradient is a strong indicator for a nearby stationary point of the objective function. (Often called the first order necessary condition.) The gradient can be used to define descent directions.
- $\diamond$ $\mathbf{Hess}\, f(x_i)$ - the Hessian of the objective function. The Hessian can be represented as an (hopefully sparse!) $n \times n$ matrix and is often used for scaling or preconditioning the gradient [BV04, NW99]. Using the Hessian in Newton-like algorithms leads to q-superlinear convergence near the solution. The Hessian is in general not positive definite (only near the solution). Positive definiteness in addition to a vanishing gradient is a sufficient condition for convergence to a minimum. (Also called the *second order sufficient condition*.)

For constrained problems it is usually assumed that Ω can be described in a simple way, for example by equality and inequalities. There are also developments for solving constraints defined by general nonlinear equations [CGT00].

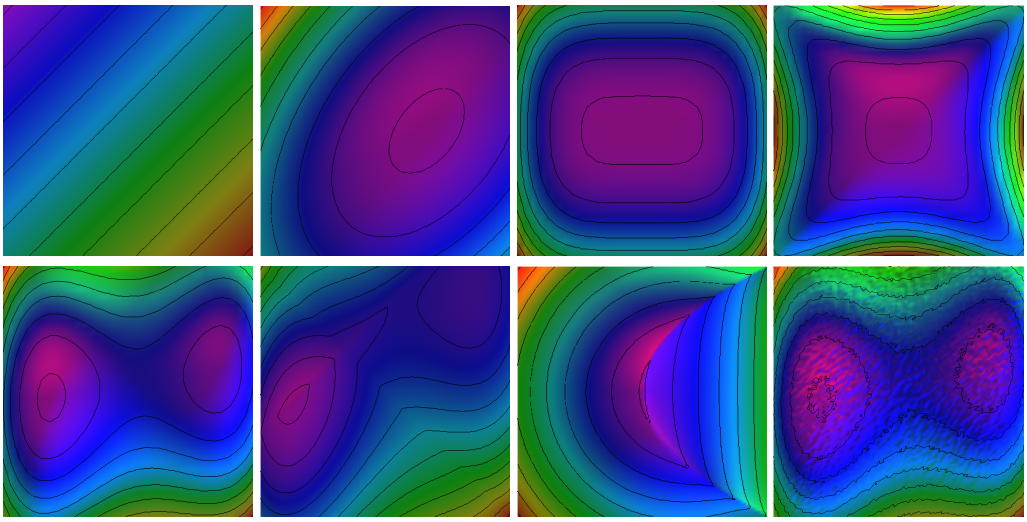


Figure A.1: *Contour plots of objective functions using extra light sources for shading. From left to right top row: linear function, quadratic function, convex function and non-convex function with unique minimum. From left to right bottom row: smooth function with two local minima, function with discontinuous Hessian, function with discontinuous gradient at minimum and a noisy function.*

A.2 An Attempt to classify Optimization Problems

In the previous section, we have described uses of the objective function, gradient and Hessian. The Taylor series expansion also gives us also a way to roughly classify optimization problems by difficulty. All things being equal, we can sort objective functions by increased unpredictability of the error term. For now we assume an unconstrained objective function, where all x_i can be chosen arbitrarily from $\Omega = \mathbb{R}^n$ (examples are given in Figure A.1). Ideally the objective function is very smooth (at least two continuous derivatives).

1. Without constraints, affine objective functions (defined by zero second and higher order derivatives) have no interesting minimizers. Solving constrained affine problems on the other hand has many important applications for instance in Operations Research and is called *linear programming* [Van01].

2. Quadratic functions are the simplest unconstrained objectives. These functions are characterized by constant, positive definite Hessian matrices. (Otherwise the function shape would be a saddle and unbounded from below.) Quadratic objectives are usually minimized with linear equation systems. This means these problems are simple, well understood and very popular in comparison to problems with variable or no Hessian.

3. Objective functions with varying, but positive definite Hessian (everywhere). Such functions

are convex (or "bowl"-shaped). (But convex functions in general are only C^0.) Convex functions have a unique minimum. Convexity is not easy to identify, but once known a broad theory with efficient solution methods exist [BV04]. Nice convex functions can be solved at the equivalent cost of a few linear solves.

4. Non-convex functions with unique minima. The Hessian is indefinite in some regions away from the minimum. As a consequence minimization approaches will have to deal with zero or negative curvature directions (multiple valleys in Figure A.1, upper right) Because of these negative curvature directions, methods specialized on convex functions will often fail, because for efficiency reasons they often rely on methods like the Cholesky decomposition of the Hessian. In regions away from the minimizer *multiple* narrow valleys might exist, with the potential of increasing the ill-conditioning of the problem. While there might be a single minimizer, computationally this problem has to be solved with the same methods as the next.

5. The last fairly nice function class is that of objectives with arbitrary (but hopefully smooth and bounded) Hessians. These objective functions are assumed to have multiple minima. Given a reasonable starting guess many algorithms using descent directions will produce a solution that is in some sense near the starting point. But in general there is no control over which of the local minimizers is picked. For an efficiently solved problem in this class see [FMS03].

6. Objectives with piecewise discontinuous or unbounded Hessians lead in the experience of the author to less well conditioned problems (as similar objective functions with nicer Hessians). In practice using this Hessian information to precondition the gradient is often still desirable.

7. Objective functions that are only C^1 continuous with no access to Hessian information for preconditioning. Simple problems of this class can still be solved with high accuracy using first order (gradient descent) methods.

8. A very hard case are C^0 objectives with piecewise continuous or no gradient at all. Here access to the main stopping criterion (a vanishing gradient) is lacking. With no additional information (like subgradients [Pol87]) one can only expect to obtain a relaxation of the objective value.

9. It is even harder to deal with stochastic objective functions. These are assumed to be composed of a deterministic function that is perturbed by an independently, identically distributed zero mean noise term [Spa03].

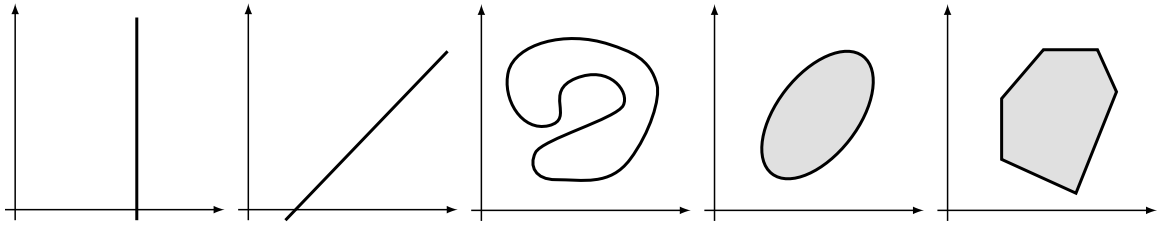


Figure A.2: *Examples of simple constraints (from left to right): identity constraints, linear equality constraints, nonlinear equality constraints, ball (or distance) constraints and linear inequality constraints.*

Functions that resist above classification scheme are for instance C^0 convex or discontinuous quasi-convex. The global knowledge implied by the convexity arguments leads to more efficient modeling than using the Taylor series expansion alone (see Chapter 5 in [Pol87] and Chapter 11 in [CGT00]). Above classification is mostly done by structure into. Additional difficulty is arises if a particular problem is ill-conditioned. With this we mean the appearance of long and thin (and even worse: curved) valleys in the objective function landscape. These valleys are tracked by many methods with high precision along the bottom, which makes the search for the minimum quite expensive.

Optimization problems can be further classified by their type of constraints. Constraints are used to describe the domain Ω which is usually called the *feasible set*. (Examples are given in Figure A.2.) Constraints significantly complicate finding the solution of optimization problems. For this reason we expect that a feasible set Ω is described in a simple way. In particular one wants to have efficient tests for feasibility $x \in^? \Omega$ and often it helps to know how to navigate on the boundary $\partial\Omega$.

1. Identity constraints: some variables are assumed to be constant $x^j = c^j$. This is the simplest form of constraints and often implemented using Lagrange multipliers by modifying gradient and Hessian of the objective. Such modifications can be numerically rigid and visibly disturb the gradient and error residual in a neighborhood of the affected variables.

2. Linear equality constraints: A linear combination of variable is assumed to be constant $Ax = c$. This constraint is often numerically nicer than the identity constraint, because the disturbance is distributed over multiple variables. Eliminating equality constraints can make a sparse Hessian denser, hence some care needs to be taken in their implementation [BV04] (Chapter 10).

3. Nonlinear equality constraints: The solution vector x has to satisfy a set of nonlinear equations $c(x) = 0$. Solving nonlinear equations is general *at least as difficult* as minimizing an

objective, so this is a hard problem. This is complicated by the fact that there are now two residuals (one for the optimization and one for the nonlinear equations) leading in general to (slightly) suboptimal and (slightly) infeasible solutions. Sometimes it is possible to combine both nonlinear problems into either a single objective function or nonlinear system [CGT00].

4. Inequality constraints: Variables are assumed to be bounded: $0 \leq Ax+b$. Multiple constraints of this type will form the faces of a simplex [BV04] (Chapter 11).

5. Ball constraints: Given a norm $\|.\|$, center c and radius r a ball is defined by $B = \{\|x - c\| \leq r\}$. Solutions of this problem for simple (often quadratic) objective functions form the basis of trust-region methods [CGT00].

Finding an initial point $x_0 \in \Omega$ can be a difficult task. Combinations of many constraints can shrink the domain $\Omega \subseteq \mathbb{R}^n$ into the empty set and make the optimization problem *infeasible*.

A.3 Methods for Solving Optimization Problems

For any non-stationary[1] argument x_i and vector d, one of the directions d or $-d$ points downhill. Roughly speaking this means half of the directions will lead to a reduction of the objective. Why don't we just follow one of these directions and see where it leads us?

Not surprisingly some directions are better than others. A popular choice is to minimize the objective by varying only one of the coordinates of x_i. This is motivated by the Gauss-Seidel iteration for solving linear systems and in this context called *coordinate descent* methods. Surprisingly in the nonlinear setting it was shown that these methods are not guaranteed to converge. Even if they converge, they might do so only very slowly (see [NW99] p.53ff). Powell constructed in 1973 several examples of three-dimensional objectives for which coordinate descent methods cycle infinitely between 6 different attractors and fail to converge to a point with zero gradient [Pow73]. One problem of this method is that the descent directions can get arbitrarily close to orthogonal to the gradient. This makes progress *very slow*.

Still, this idea is useful because it introduces the idea of line-searches. The one-dimensional problem - minimizing a function along a line $x + \lambda d$ - is fairly easy to solve (for instance brute force using binary search). The exact solution is often not required. In practice one searches for values of $\lambda \in (0, 1]$ are for which $f(x + \lambda d)$ is in some sense *sufficiently smaller* than $f(x)$. This can be

[1] A point x is called *stationary* or *first order critical* if $\mathbf{grad}\, f(x) = 0$, *e.g.*, the tangent plane of the objective function is horizontal at x.

achieved via *backtracking methods*, which start from $\lambda_0 = 1$ and iteratively decrease λ_i until the sufficient decrease of the objective in direction d is achieved. This final λ is often referred to as *step length*.[2]

We are going to state the convergence rates of different algorithms in terms of their asymptotic behavior. We will distinguish three different types, namely q-linear, q-superlinear and q-quadratic rates [Kel95].[3] Q-Linear convergence denotes a growth of significant digits that is linear in the iteration number, e.g., $O(i)$ (in other words the error declines with geometric rate). q-superlinear convergence refers to more than geometric error decline, e.g., an escalating increase of significant digits in each step, e.g., $\Omega(i)$. Q-Quadratic convergence stands for a doubling of significant digits in each iteration, or exponential growth of significant digits $O(2^i)$.

A.3.1 First Order Methods

A better idea than using simple coordinate descent is based on building a first order model of the objective, namely

$$f(x_i + d_i) \approx f(x_i) + \mathbf{grad}\, f(x_i) \cdot d_i \tag{A.3}$$

and to follow a descent direction d that is obtained from the gradient in each iteration step. If the descent direction $d_i = -\mathbf{grad}\, f(x_i)$ is the negative gradient and a step length λ_i is chosen to find the minimum along this direction, then the very greedy *steepest descent* method (in the Euclidean norm) is obtained [NW99, BV04, Pol87]. It is surprising, but has been demonstrated again and again [NW99, BV04, Pol87] that the performance of unpreconditioned descent methods can be quite poor. One reason for this poor behavior lies in the abrupt direction change in each minimization step, e.g., for the steepest descent method $d_{i-1} \cdot d_i = 0$.

Theorem 2 (Convergence rate steepest descent) *Let f be twice continuously differentiable. If the steepest descent method is started sufficiently close to a minimum x^* with positive definite* $\mathbf{Hess}\, f(x^*)$*, then the sequence $f(x_i)$ converges with rate*

$$\frac{f(x_{i+1}) - f(x^*)}{f(x_i) - f(x^*)} \leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 \tag{A.4}$$

where λ_1 is the smallest and λ_n is the largest eigenvalue of $\mathbf{Hess}\, f(x^*)$ *(Theorem 3.4 in [NW99]*

[2] This is slightly misleading as in general $\|d\| \neq 1$.

[3] The "q" refers to "quotient" and is chosen to disambiguate the introduced terms.

p.49). We also obtain convergence for the point sequence x_i

$$\frac{\|x_i - x^*\|}{\|x_0 - x^*\|} \leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1} + \epsilon\right)^i \tag{A.5}$$

as Theorem 4 in [Pol87] p.27 shows.

The convergence rate is q-linear and easily observed in practice. The main problem is a factor that is close to 1 for many problems, even of very modest size. Because the factor depends on the condition number of the Hessian, it can be improved by carefully changing to a different set of variables defining the objective. In general this is not easy and hence methods have been developed that are more robust to this problem.

It is quite surprising that following randomly chosen descent directions as for instance described in the SPSA method is *on average as good as following the exact gradient* [Spa03, Pol87]. From this one should realize that following the gradient downhill is not a particularly original or efficient. This argument should also show on an intuitive level, that better downhill directions than the gradient exist (because the average includes many directions that perform worse than the gradient).

One idea that performs better than the steepest descent method is physically motivated and simulates a "heavy ball" rolling down the objective landscape. This dampens the direction changes and achieves under ideal parameter selection for the damping the same convergence rate as the popular nonlinear conjugate gradient method (refer to [Pol87] p.74).

Observation 1 (Convergence rate linear conjugate gradient) *The linear conjugate gradient method with constant Hessian* A *converges with geometric (q-linear) rate. The error is bound by*

$$\frac{\|x_i - x^*\|_A}{\|x_0 - x^*\|_A} \leq 2\left(\frac{\sqrt{\frac{\lambda_1}{\lambda_n}} - 1}{\sqrt{\frac{\lambda_1}{\lambda_n}} + 1}\right)^i. \tag{A.6}$$

(See equation 2.15 in [Kel95] and for a sharper bound Theorem 5.5 in [NW99].))

Compared to the steepest descent method the main difference is the square root on the spectral condition number $\frac{\lambda_1}{\lambda_n}$. This means that, *at least in the linear setting for quadratic energies*, the conjugate gradient method converges much faster than the steepest descent method. It also does not require any tuning of a damping parameter (as for the heavy ball method) to achieve this rate. Different *nonlinear* extensions to the conjugate gradient methods exist and behave similarly well in practice. But because they have a "memory" of previously encountered nonlinear data, their

theoretical analysis is complex. The Fletcher-Reeves variant is often less efficient than the Polak-Ribiére+ algorithm and recommended by different authors [NW99, PTVF92].[4]

One reason causing the gradient to be a poor downhill direction is its *dependence* even on simple affine scaling of the variables.

$$\mathbf{grad}\,(f(Ax)) = A \cdot \mathbf{grad}\, f(Ax) \tag{A.7}$$

This means a steepest descent algorithm's performance will depend on the chosen "parameterization" - or on the scaling of the variables [NW99, BV04]. While this appears to be a minor problem, this sensitivity has great practical implications: for instance in the parameterization problems discussed in Chapter 4 *huge* disparities on the edge lengths appear during the minimization. Such situations need to be handled in a robust way across all scale! Newton-steps are independent of affine transformations [NW99].[5]

Poor scaling of the gradient has also the potential to ruin a main termination criterion, which is based on the vanishing norm of the gradient. For this reason users are required to carefully pick an *application dependent* threshold ϵ for the gradient norm $\|\mathbf{grad}\, f(x_i)\| \leq \epsilon$ for termination [BMMS04].

A range of methods has been developed to automatically define good preconditioning matrices to obtain better search directions. Some (often positive definite) matrices B_i are updated in each iteration step from the available gradient information. The gradient direction is scaled with the inverse $H_i = B_i^{-1}$ of this matrix to obtain the new step direction $d_i = -H_i \cdot \mathbf{grad}\, f(x_i)$. These methods often try to converge to the Newton method and are commonly referred to as *quasi-Newton* iterations. Different construction rules for the B_i are, for instance, Broyden's method [Kel95, Pol87, NW99], the Davidson-Fletcher-Powell method (DFP) [Pol87, NW99] or the SR1 method [NW99]. What makes these methods particularly attractive, is that direct update-rules for the H_i exist. This means there is no need to invert a matrix in each iteration step![6] All the mentioned methods converge in a neighborhood of the minimum with q-linear or even q-superlinear rate [Kel95, Pol87, NW99]. But q-superlinear rate is only achieved if the B_i converge to the Hessian - which for large dimension n is rarely practical to await.

Observation 2 (First order methods) *The cost of first order methods is very low if counted on a*

[4] The simpler Polak-Ribiére (no "plus") method performs well in practice but can fail to converge without periodic restarts.

[5] The underlying problem does not magically disappear but is handed over to the linear equation systems solver.

[6] But H_i might not be sparse.

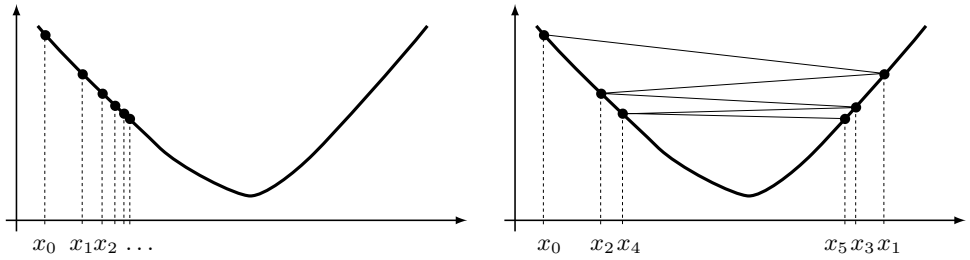


Figure A.3: *Without step length control the decrease of the objective function can be too small either because of too short (left graph) or too long (right graph) step lengths.*

per iteration basis. Their strength lies in the following situations

a) the sequence x_i is still far away from the solution.

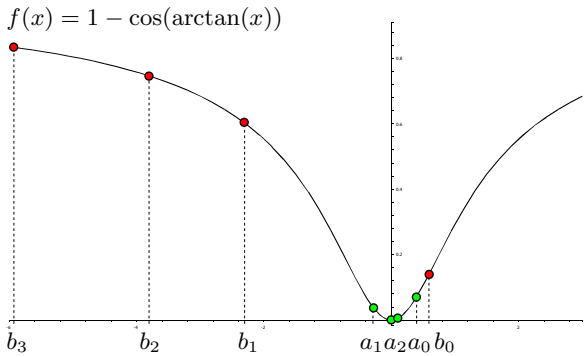
b) the dimension n of the problem is reasonably small or well-posed.

c) or if Hessian information can not be obtained. In practice the most efficient first order methods are the nonlinear conjugate gradient for well-conditioned and quasi-Newton methods for ill-conditioned objectives.

A.3.2 Ensuring Convergence

Before we go into more detail and discuss second order algorithms for solving the optimization problem, we will motivate some results for ensuring convergence for arbitrary starting guess. Just having a decreasing sequence of objectives $f(x_0) > f(x_1) > \cdots > f(x_i) > \ldots$ does not guarantee convergence *to the sought for minimum* $f(x_{min})$. Indeed the sequence this sequence might converge to some value larger than $f(x_{min})$ as shown by the two examples in Figure A.3. To simplify convergence arguments a number of criteria have been developed that provide simple to check conditions on the step length guaranteeing sufficient progress toward a solution. For line search methods these checks are based either on the *Wolfe conditions* (leading in combination with backtracking methods to the *Armijo rule*) or the less often used *Goldstein conditions* [NW99, Kel95]. For trust-region methods the most often used progress criterion is the *Cauchy point* [NW99, CGT00].

The nature of these conditions is technical, and for this reason we will not repeat the exact formulas here. But we can't stress their practical importance enough. Not only do these conditions virtually guarantee convergence, but they are also designed to be simple and inexpensive to check. Furthermore they are designed to not interfere with rapid convergence. In particular none of the three conditions will interfere with the rapid convergence rates achievable by Newton-like methods.



i	a_i	$f(a_i)$
0	$0.40e-0$	$0.72e-1$
1	$-0.28e-0$	$0.37e-1$
2	$0.80e-1$	$0.32e-2$
3	$-0.15e-2$	$0.12e-5$
4	$-0.11e-7$	$0.69e-16$

i	b_i	$f(b_i)$
0	$0.60e-0$	0.14
1	$-2.31e-0$	0.60
2	$-3.83e-0$	0.75
3	$-5.94e-0$	0.83

Figure A.4: *The function $f(x) = 1 - \cos(\arccos(x))$ has a unique minimum at $x = 0$. The Newton method without step length control converges for the starting guess $a_0 = 0.4$ and diverges for the slightly larger initialization $b_0 = 0.6$. Using step length control the Newton method can be made globally convergent. But even then q-superlinear convergence is only observable near the solution!*

A.3.3 Second Order Methods

Second order models can be motivated by several observations. We will assume that we can approximate the objective in each iteration step by a model constructed using the Taylor series expansion:

$$f(x_{i+1}) = f(x_i + d_i) = f(x_i) + \mathbf{grad}\, f(x_i)^T \cdot d_i + \frac{1}{2} d_i^T \cdot \mathbf{Hess}\, f(x_i) \cdot d_i$$

Assuming a positive definite Hessian, we can solve the constant quadratic model by taking the gradient with respect to d_i

$$d_i = -\mathbf{Hess}\, f(x_i)^{-1} \cdot \mathbf{grad}\, f(x_i) \tag{A.8}$$

to obtain the formula for the *Newton-iteration*. Instead of inverting the Hessian, the new search direction d_i is best obtained by solving a linear equation system. The new search direction d_i can also be interpreted as the linearization of the first order optimality condition[7] or as the direction of steepest descent *in the Hessian norm* $\|d\| = (d^T \mathbf{Hess}\, f(x) d)^{0.5}$. This iterative procedure does not converge when x_0 is set to be far away from the minimizer, as the example in Figure A.4 shows. But if convergence is achieved, it happens at an astonishing rate:

Theorem 3 (Convergence rate Newton method) *Let f be twice differentiable and* $\mathbf{Hess}\, f$ *be Lipschitz continuous and in the neighborhood of the minimum x_{min}. If the Newton iteration A.8 is started with x_0 sufficiently close to x_{min} the generated point sequence converges $x_i \to x_{min}$. The convergence rate of $\|x_i - x_{min}\|$ and $\|\mathbf{grad}\, f(x_i)\|$ is q-quadratic.*

[7] This idea is used for indefinite matrices occurring in non-linear equation systems.

The Newton method is designed to minimize an objective that happens to be purely quadratic *in a single step*. But without step length control as discussed in Section A.3.2 all Newton-like methods will fail *even for simple convex functions* when started sufficiently far from the solution (as happened to sequence b_i in Figure A.4)!

To benefit from the good final convergence rate, essentially all second order models will try to perform the Newton step first. If the Newton step fails to produce a sufficient decrease of the objective, like that specified by the Wolfe condition, alternative steps have to be considered. These alternatives are often obtained by backtracking via step length reduction. Computing the Newton direction requires the solution of a linear problem. This is an expensive step that pays off either when q-quadratic convergence is observed, or when the problem is too ill-posed for first order methods. A clever implementation will not spend too much time on the linear solve and terminate early with an approximate solution, if it can decide that q-quadratic convergence is unlikely. This can be done, for instance, when negative curvature directions are encountered in the Hessian matrix. Defining good early truncation criteria is still actively researched and are discussed further in Section A.4.3.

Observation 3 (Newton methods) *Depending on the implementation of the line search, Newton methods can be fragile when operating far away from the solution. Their ultimate strength is the end game, where superior convergence rates are achieved. Newton methods can use a variety of linear solvers (for non-convex problems they must be able to handle indefinite Hessian matrices) and have no particular restrictions on preconditioners.*

Trust-region methods provide a more flexible framework compared to line-search methods. Instead of finding (approximate) minimizers along line segments, the search is extended to cover finite volumes. To keep dealing with these regions simple, they are usually defined as balls measured in some norm $\|.\|$. Figure A.5 shows the basic framework of a trust-region minimizer as described in [CGT00]. One motivation behind introducing trust-region methods is the desire to increase the granularity of the solution process: not only do balls cover more search space than line segments, but one has more flexibility in designing algorithms for obtaining a solution of the current step. Interestingly, this is achieved by creating a sequence of simple but non-trivially constrained subproblems.

A popular second order trust-region method was proposed independently by Steihaug and by Toint [NW99, CGT00] and was implemented by the author in [FMS03] and also available in TAO [BMMS04]. At the heart of this method is a modified linear/nonlinear conjugate gradient solver. This method is not difficult to implement and works very well in practice. Its main strength is

Basic trust-region algorithm

Step 0: Initialization.
An initial point x_0 and an initial trust-region radius Δ_0 are given.
Compute $f(x_0)$ and set $i = 0$.

Step 1: Model Definition.
Chose a norm $\|.\|_i$ that defines the shape of a ball B_i centered at x_i with radius Δ_i.
Define a model m_i for the objective f on the trust-region B_i.

Step 2: Step calculation.
Compute a step d_i that "sufficiently reduces" the model m_i and
stays inside of the trust-region, e.g., $x_i + d_i \in B_i$.

Step 3: Acceptance.
Compute $f(x_i + d_i)$ and decide if the model predicted the decrease of
the objective function well.
If the prediction was good the step is accepted and $x_{i+1} := x_i + d_i$.
Otherwise the step is rejected and $x_{i+1} := x_i$.

Step 4: Trust-region radius update.
If the prediction of the model turned out to be
very accurate: Allow for a larger trust-region and increase the radius.
reasonably good: Keep the trust region radius $\Delta_{i+1} := \Delta_i$.
poor: Decrease the trust region radius.

Increment i by 1 and continue with step 1.

Figure A.5: *The basic trust-region algorithm as described in [CGT00].*

the relatively efficient handling of points x_i that are still far away from the region of Newton-convergence. But there is a serious drawback of this method compared to line-search Newton solvers in the quadratically convergent region: currently there is no theory on the preconditioning of trust-region solver with *arbitrary* preconditioners. According to [CGT00] the preconditioning matrix M has to be positive definite so it can be used to define a norm $\|.\|_M$ for measuring the trust-region radius. Maybe this problem is minor and easily fixed, but a lack of preconditioning has serious implications on the practically achievable convergence rate, as we will discuss later.

A.3.4 Nonlinear Equations

Observation 4 *If we could only solve nonlinear equation systems efficiently, we would have no need for developing methods for the more specialized optimization problem.*

The first order optimality condition $\mathbf{grad}\, f(x_{min}) = 0$ connects the continuous optimization problem defined by equation A.1 with the solution of a system of nonlinear equations. Nonlinear equa-

tions are defined by

$$g(x) = 0 \tag{A.9}$$
$$g : \mathbb{R}^n \to \mathbb{R}^n$$
$$\mathbf{Jac}\, g : \mathbb{R}^n \to \mathbb{R}^{n \times n}$$

Two differences are obvious between both problems: nonlinear equations lack an objective function to check for progress. The Jacobian of g is also usually neither symmetric or positive definite — even at the solution! The indefiniteness of the Jacobian is not a serious problem for solving smooth nonlinear equation systems using the Newton method (see [Kel95, CGT00] and also the detailed discussion in [BMN01]). But the lack of an objective function has consequences. To be able to check for progress toward a solution one usually introduces the norm of the residual

$$\frac{1}{2}\|g(x)\|_2^2 \tag{A.10}$$

as a *merit function* or pseudo energy [NW99, Kel95, CGT00]. Using the square of the L_2 norm, transforms solving nonlinear equations into a smooth *global* optimization problem (Chapter 16 in [CGT00]). But not all nonlinear equations have a solution: in this case even the global minimum is meaningless. Similarly, many local minima of A.10 will not satisfy $g(x) = 0$. Local methods might get stuck when trapped between local maxima (see Figure A.6).

In this context, it is worth pointing out the simplicity of the linear conjugated gradient method (a Krylov iteration solver for quadratic systems defining an objective function) with the algorithmic complexity for instance of the generalized minimum residual (GMRES) method (which can deal with indefinite linear equation systems) [Kel95]. In some sense this increased complexity reflects the consequences that the loss of structure — of not having a proper objective function — has.

We must be cautious of treating formula A.10 as a proper energy from which gradient and Hessian energy are derived. In the simple case of a linear, but indefinite equation system $g(x) = Ax$, we could obtain

$$f(x) = \tfrac{1}{2}x^T A^T A x$$
$$g(x) = Ax \qquad \mathbf{grad}\, f(x) = A^T A x$$
$$\mathbf{Jac}\, g(x) = A \qquad \mathbf{Hess}\, f(x) = A^T A.$$

The new objective $f(x)$ still needs to be globally minimized to obtain a solution to $g(x) = 0$. What makes this transformation really impractical is a condition number of the Hessian $A^T A$ that is the square of the condition number of the Jacobian A. (For a deeper treatment compare the remarks on the CGNR and CGNE modifications on page 25 in [Kel95].)

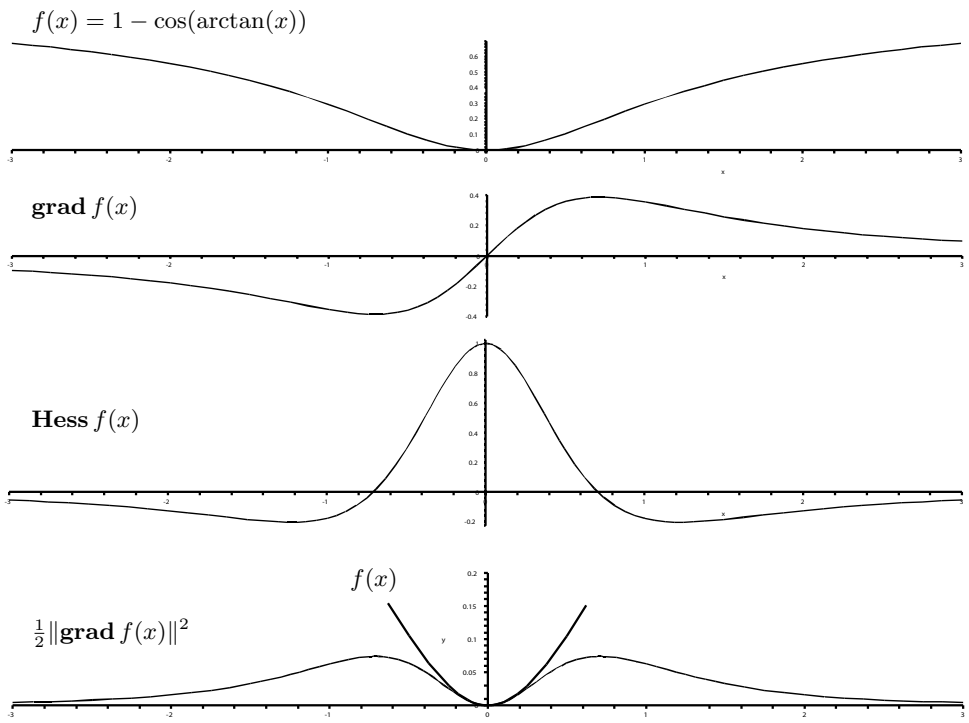


Figure A.6: *The objective $f(x) = 1 - \cos(\arctan(x))$ (top) has a unique minimum and poses no problem to minimization using most local schemes and arbitrary starting guess. If $f(x)$ (top graph) is replaced by $\frac{1}{2}\|\mathbf{grad}\, f(x)\|^2$ (bottom) convergence to x_{min} requires global minimization if a starting guess x_0 with $\mathbf{Hess}\, f(x_0) < 0$ is used.*

Observation 5 *We conclude that solving nonlinear equations is a similar, but structurally more difficult problem than minimizing an objective function. In particular nonlinear equations* require *a starting guess near the solution, while for optimization problems this is only* desirable.

Fortunately there are many problems, in particular time-dependent systems, that allow to track solutions while varying some parameters. Such ideas are also formally explored in continuation methods ([NW99] Chapter 11.3).

A.4 Practical Considerations

In this section we discuss a range of topics that arise during the practical design of objective functions and numerical minimization. In reality, computations are not performed with infinite precision, might not last long enough to see asymptotic behavior, might fail due to ill-posedness or indefiniteness of the objective and so on.

A.4.1 The Presence of Numerical Noise

So far, we assumed that all numerical computations were performed with infinite precision. Obviously this assumption does not hold for machine numbers. But there are many more sources of inaccuracy. For example, the objective function is noisy, because it comes from a physical process or is computed using numerical quadrature, a gradient is noisy because it is computed via finite differences, or the Newton-step is noisy because an iterative method for is used for solving the linear equation system. In general, there are preventable errors, that are reduced by diligent work and errors that escape control. How do these errors affect the minimization process and which consequences does their propagation have for the convergence results? Higher order algorithms are more sensitive to noise than simpler methods. This is fairly intuitive, but can be shown rigorously [Pol87]. In general, stochastic noise poses less of an obstacle than deterministic noise, mainly because it has the tendency to cancel out by the law of large numbers. This sometimes happens naturally for the algorithms discussed so far, but can be strictly enforced as discussed in [Pol87] (p.98ff) and [Spa03].

First order methods The behavior of the gradient descent method using noisy gradients $\mathbf{grad}\, f(x_i) + r_i$ yields no surprise. As long as the noise level is smaller than the magnitude of the exact gradient $\|r_i\| < \|\mathbf{grad}\, f(x_i)\|$, the usual rate of progress is made toward the minimum. Once the noise level gets larger than the gradient, the descent breaks down, in general somewhere near the solution [Pol87].

Second order methods The analysis of the Newton-steps shows more insights. Because of ill-conditioning the Newton-direction can be noisy, even if gradient and Hessian are exact to machine precision. This situation can still be analyzed using perturbed gradients.

Theorem 4 (Convergence of inexact Newton-iteration) *Let us assume a setting (positive Hessian, starting guess near minimum) where the exact Newton iteration* $d_i = -\mathbf{Hess}\, f(x_i)^{-1}\mathbf{grad}\, f(x_i)$ *converges q-quadratically to the minimum* x_**. Let us instead use a noisy gradient (or solve the linear system inexactly) such that the residual* $r_i = \mathbf{Hess}\, f(x_i)d_i + \mathbf{grad}\, f(x_i)$ *is bound by* $\|r_i\| \leq \eta_i \|\mathbf{grad}\, f(x_i)\|$*. Then the sequences* $\|x_i - x_*\|$ *and* $\|\mathbf{grad}\, f(x_i)\|$ *converge*

- ◇ *q-linearly, if* $\eta_i \leq \eta$ *for some* $\eta \in [0, 1)$*.*
- ◇ *q-superlinearly, if* $\eta_i \to 0$*.*
- ◇ *q-quadratically, if* $\eta_i = O(\|\mathbf{grad}\, f(x_i)\|)$*.*

(Compare with [NW99] p.136, [Kel95] p.96 or [Pol87] p.103)

This result can be directly used for tuning the accuracy of the linear step using particular forcing sequences η_i as is illustrated in Figure A.8. It assures us that convergence of the inexact Newton-iteration is at least linear with such a simple choice as $\eta_i = 0.5$ and can be tweaked in the limit by using, for example, $\eta_i = \sqrt{\|\mathbf{grad}\, f(x_i)\|}$ for q-superlinear or $\eta_i = \|\mathbf{grad}\, f(x_i)\|$ for q-quadratic rate. The q-linear rate can be achieved even in the presence of ill-conditioned Hessians as discussed by Theorem 6.1.3 in [Kel95].

Achievable convergence rates At this point we would like to highlight the confusing nature of Theorem 4. How can it be that a Newton-solver — which for the sake of argument is based on a linear conjugate gradient method and uses *a constant (!) second order objective function model* — converge q-quadratically, while a nonlinear solver *having access to continuously updated, exact objective information* shows only q-linear convergence?[8] We offer two answers to this question. First the simple answer: in our example, the q-quadratic convergence is due to bad accounting. Even under ideal early termination conditions, CG-Newton-steps should rarely beat the the non-linear conjugate gradient method! But this answer is slightly naïve. In real implementations, the Newton-method has two advantages. First, it saves the cost of *many* exact objective function and gradient evaluations (used in line-searches) at the cost of a *single* Hessian evaluation, which is used to formulate the quadratic model. If objective function and gradient evaluations are expensive, this provides time savings by a *constant* factor. The second, more hidden advantage is that the Newton-step defines a large, modular "chunk" of *well-understood* work. By Theorem 4 near the solution, the Newton step reduces the problem of efficiently solving nonlinear systems to efficiently solving linear equations! Particular linear systems may have solution methods and preconditioners that outperform the simple conjugate gradient method! This allows to leverage from known techniques and potentially achieve a q-superlinear speedup as measured in real CPU cycles.

A.4.2 A Simple but More Realistic Cost Model

In Section A.3 we stated the the algorithmic progress in the limit as a function of iteration steps, where each step was either a line-search or a minimization in a trust-region. We noticed in Section A.4.1 that the cost of each iterative step can be highly variable and consequentially influence how we think about the observed convergence rate.

There are two components that contribute to this cost. The first is *problem specific* and accounts

[8] This question is rarely asked in the optimization literature. Theorem 3 on page 72 in Polyak discusses a similar problem.

for how expensive the evaluation of the objective function, its gradient and Hessian are. The second component is *method specific* and has to account for how frequently objective information is updated and how it is processed. The processing is often dominated by linear algebra routines, in particular operations on high dimensional vectors and matrices.

A.4.2.1 The Cost of Objective Function Information

If not stated otherwise we will assume that the objective function and its gradient can be evaluated with high precision.

The objective function Clearly there is no theoretical limit on the complexity of objective functions. Nevertheless if we are interested in solving problems with a large number of variables $n = 10^3 \dots 10^6$, we have to restrict our attention to functions that can be evaluated efficiently. This means in particular linear (or at most quadratic) time. One objective type that fits this description is additive[9]

$$f(x^1, \dots, x^n) = \sum_{i=1}^{n} \sum_{j \in \mathcal{N}(i)} f_{ij}(x^i, x^j). \tag{A.11}$$

Such objective functions appear frequently enough as the solution of integrals over finitely supported basis functions. For now we assume that the $f_{ij}(x^i, x^j)$ are fairly simple formulas and the sets of interacting variables $|\mathcal{N}(i)| < k$ are not too large (say $k = 3 \dots 100$). This particular structure permits us to evaluate the objective function with very little loss in machine precision $\epsilon \approx 10^{-16}$ in linear time $O(n)$.

The gradient Differentiating equation A.11 gives us a formula

$$\mathbf{grad}\, f(x^1, \dots, x^n) = \sum_{i=1}^{n} \sum_{j \in \mathcal{N}(i)} \mathbf{grad}\, f_{ij}(x^i, x^j) \tag{A.12}$$

that allows for easy assembling of the gradient from $\mathbf{grad}\, f_{ij}(x^i, x^j)$. If the chain rule is used to obtain the formulas for $\mathbf{grad}\, f_{ij}(x^i, x^j)$ one typically observes increased algebraic complexity of evaluating $\mathbf{grad}\, f_{ij}(x^i, x^j)$ over $f_{ij}(x^i, x^j)$. For this reason computing $\mathbf{grad}\, f_{ij}(x^i, x^j)$ via finite differences might appear competitive. In some situations this could indeed lead to more efficient evaluation than symbolic differentiation. But finite differences are hard to tune. Even under perfect conditions the finite difference gradient will have at most half the number of significant digits

[9] Multiplicative formulas can be converted into a summation by taking the logarithm, keeping monotonicity.

than the objective function it is computed with ($\epsilon \approx 10^{-8}$). Most iterative algorithms will evaluate the gradient at most once per iteration, while calling the objective function multiple times (particularly for step-length control reasons). This means, in general, that the computation of the gradient unlikely to be a performance bottleneck.

The Hessian Differentiating our model objective function one more time yields a sparse Hessian

$$\mathbf{Hess}\, f(x^1, \ldots, x^n) = \sum_{i=1}^{n} \sum_{j \in \mathcal{N}(i)} \mathbf{Hess}\, f_{ij}(x^i, x^j). \tag{A.13}$$

It is often very desirable to store the Hessian matrix in main memory for efficient access by the linear solver. Memory availability restricts us to store *full* matrices to problems with less than $10^3 \ldots 10^4$ variables. It can be beneficial to define matrices that have a sparser structure and entries with less accuracy than the exact Hessian. Such matrices might be used as discussed in Section A.3.3 for preconditioning inexact or quasi-Newton steps.

Enforcing some simple constraints like the average of *all* variables leads to full Hessians. One way to handle these without storing huge matrices is to define "matrix-free" matrices. These could be hybrid, where a sparse part of the Hessian is stored and the simpler full part is evaluated dynamically. Particularly iterative linear solvers only need to have access to the results of the matrix-vector multiplication. The drawback of this approach is a limitation to available matrix-free or even custom preconditioners. (Many available preconditioners don't work without explicit matrix access!)

Evaluating Hessians $\mathbf{Hess}\, f_{ij}(x^i, x^j)$ via symbolic differentiation often leads to computations that nicely fit into the cache and are limited only by FPU throughput, and not by main memory bandwidth. Matrix entries can be computed independently. For this reason, using a compiler supporting loop parallelization can have dramatic impact on the assembly times of the Hessian.[10] The relative accuracy of the Hessian appears to be less critical than the accuracy of the gradient for some applications like parameterization (Chapter 4). Hypothetically it might be beneficial to evaluate and store the entries of the Hessian only with 32-bit accuracy. In our experiments we did not observe a significant advantage from this approach. We do not have a satisfactory explanation for this.

A.4.2.2 The Cost of Linear Algebra Subroutines

Solving large linear equation systems can be an expensive task. This is particularly true, if ill-conditioned matrices are encountered. But for systems with large numbers of variables a mundane

[10] The Intel Pentium IV CPU for instance supports SIMD SSE2 instructions and hyper-threading.

problem moves into the center stage: insufficient cache size and low main memory bandwidth.

Observation 6 *A 3 GHz Pentium IV processor has a theoretical peak performance of 6 GFlop/s. The bus bandwidth of 2 GBytes/s limits just accessing large vectors like $x = (x^1, \ldots, x^n)$ or $\mathbf{grad}\, f(x)$ to at most 250 million numbers per second! Using vector operations the FPU can't be utilized with more than 3 percent peak performance.*

There is nothing we can do about the cost of vector-vector operations that are $O(n)$ and can be precisely accounted with the memory bandwidth. The cost of matrix inversion often leads a lot of room for creativity.

It would be nice if one could partition the optimization problem into cache friendly chunks. One step in this direction are direct linear solvers like SuperLU [DEG+99], but this only addresses one part of the problem. At a certain problem size, the vector of variables $x \in \mathbb{R}^n$ will not fit into the cache and just accessing this (or any other) vector will make the cache useless. The obvious solution is to only work on a moderately sized subset of variables that can reside in the cache and is independent of the problem size. (On the order of $100 \ldots 1000$ variables for current processors and reasonably sparse problems.) This subset might be periodically selected by analyzing the descent direction generated by the classic method and picking the variables x^j that contributed the most to this direction. (Some form of principal component analysis?) This idea in some sense is a generalization of coordinates descent or hierarchical methods. (But with a search direction selection based on the gradient.) If such a method would converge, it will most likely have a higher iteration count than the original descent method. But with the memory gap continuing to grow (and already having only 3 to 10 percent utilization in the linear algebra kernel), such an approach might well pay off when factoring memory access in.

Finally, one could implement the linear algebra kernel on hardware that is not as bandwidth limited, as are GPUs on graphics cards or FPGAs. Currently these chips don't have high floating point precision. But they would still be useful in obtaining an approximate solution. The solution could be improved to full precision with a few Newton steps on the CPU.

A.4.3 Early Truncation and Inexact Solutions

Now that we have cost models both for obtaining objective information and the linear algebra routines, we need to discuss how to distribute the effort between the two computations. The minimization algorithms discussed in this chapter are modular and alternate between obtaining objective information and using it in linear algebra subroutines.

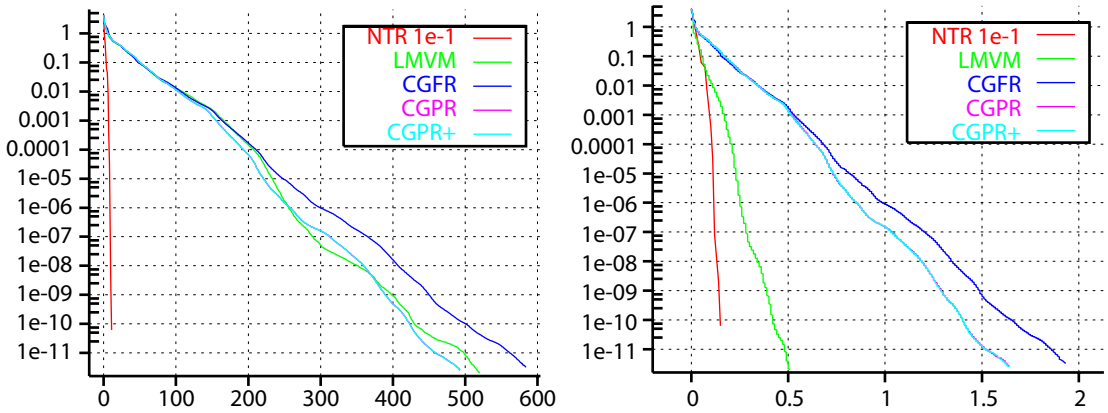


Figure A.7: *Plotting the distance $f(x_i) - f(x)$ over the iteration number i (left graph) shows very fast convergence for the Newton trust-region method compared to the q-linear convergence of all other methods. Plotting progress over CPU-time (right) the picture is more differentiated. The Newton method still wins, but only by a moderate margin compared to LMVM, a limited memory quasi-Newton method. (The results for CGPR and CGPR+ are identical in this case.) The test problem was a small spherical parameterization problem with about 1000 variables (a very coarse version of the igea model in Figure 4.5) and the library used was TAO [BMMS04].*

During the time spent in the linear solver, the gradient and Hessian information is assumed to be constant. This is appropriate, if the objective truly is a linear or quadratic function. But in general functions have nonzero second and third derivatives as discussed earlier in Section A.2. For this reason, we need to get a new view of gradient and Hessian information periodically. The question is, how often should the objective information be updated?

Iterative solvers rarely compute exact solutions in a finite number of steps. Instead, they will terminate with some residual error. If the objective information used in the solver was relatively expensive to obtain, we have a motivation to squeeze the last bit of information out of it and solve the linear algebra computations with high precision. If, on the other hand, the objective information is cheap to update, then it should be beneficial to do so often to keep the current model as accurate as possible.

One interesting result is that being inexact often pays off. It can be shown that in many situations we can afford lower order inaccuracies as long as sufficient progress is made toward the solution. Examples of ideas where inexact solutions are efficient include: choosing backtracking over exact line search, solving for the Newton step with only limited accuracy, or approximately enforcing constraints while being being far from a solution.

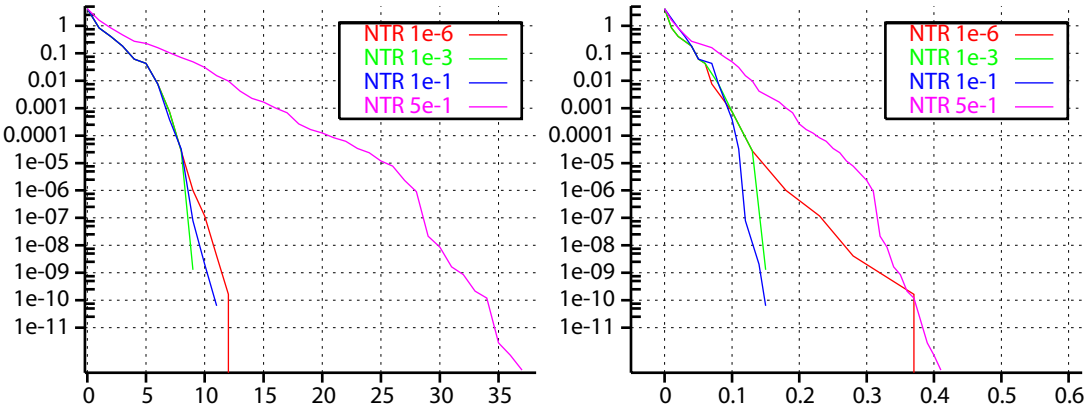


Figure A.8: *Comparing the convergence rates of the Newton trust-region methods [BMMS04] using forcing sequences $\eta_i = 10^{-6}, 10^{-3}, 0.1$ and 0.5 for the accuracy in the linear step. Using iteration count as cost model the left graph shows faster than linear convergence for steps with high accuracy, but mostly linear convergence the low accuracy step $\eta_i = 0.5$. Plotting the convergence rate over CPU time shows a slightly different picture: steps with very high and very low accuracy take more time to converge than steps with intermediate accuracy. The test problem is the same as in Figure A.7.*

A.4.4 Termination Criteria

At some point, any computation has to stop and return a result. How can we monitor convergence and make the decision that it is not worth continuing? Only in test cases do we know the solution x_{min}. In most other cases, even the objective value $f(x_{min})$ is unknown at the minimum! All we have access to are the objective reduction $\delta f_i = f(x_{i+1}) - f(x_i)$, the step $d_i = x_{i+1} - x_i$ and the gradient of the objective $\mathbf{grad}\, f(x_i)$. How are these related? For smooth functions we can refer once more to the Taylor series expansion. We see that the objective function reduction $\delta f_i = O(\|x_i - x_{min}\|^2)$ is quadratic in the distance from the minimum . (Compare also the numerical results using test functions stated in [Pol87] p.384ff.) This only restates that smooth functions appear quadratic at the minimum. This also means that they are very shallow. For convergent algorithms, the angle between search direction and gradient is bound from below, and hence their length is of the same order $\|d_i\| = O(\|\mathbf{grad}\, f(x_i)\|)$ (assuming no scaling by step length control). For ill-conditioned Newton-iterations [Pol87] argues for using d_i as a termination criterion that automatically scales with the problem. We were interested in solutions with high accuracy and reduced $\|\mathbf{grad}\, f(x_i)\|$ until the method broke down.

Observation 7 (Influence of machine precision) *Assume a high machine precision of $\epsilon = 10^{-16}$ for the computation of the objective function f. Then the objective function is virtually constant in an area $\|x - x_{min}\| \leq \sqrt{\epsilon} \approx 10^{-8}$ around the exact minimum x_{min}. (These errors can be much larger, due to cancellation of large numbers in the objective function.)*

Even in the best case, how does this fairly large residual relate to the individual variables of the computed vector $x = (x^1, \ldots, x^n)$? Remembering the equivalence of norms we have

$$\frac{1}{\sqrt{n}} \cdot \|x\|_2. \leq \|x\|_\infty \leq \|x\|_2 \leq \sqrt{n} \cdot \|x\|_\infty. \tag{A.14}$$

For a large problem with 10^6, variables the individual error is reasonably bound by $0.001 \cdot \|x\|_2 \leq \|x\|_\infty \leq \|x\|_2$.

A.4.5 Miscellaneous Remarks

Scaling and choice of variables The Newton method is invariant to *affine* transformations of variables, but it is not invariant to *arbitrary* variable changes. The Newton method will deal very well with long and *straight* valleys in the objective function landscape, as long as good preconditioners are available that compress these valleys virtually into nice round bowls. For the lack of such preconditioners, first order methods have great difficulty with any kind of elongated valleys. But the Newton step has its limits, as it is only linear, it can't do much in the presence of long and *curved* valleys. For this reason, it is of great practical importance to select variables in which the valleys in the objective function landscape appear as "straight" as possible.

Singular Hessian and the Newton Step One often faces objective functions that are invariant to certain variable transformations. Spherical parameterization energies for instance remain constant under rotations (Chapter 4). A consequence of this invariance is that Hessians are singular everywhere. Most convergence theory breaks down in the presence of non-positive definite Hessians ([Pol87] Chapter 6.1). In particular, a convergence of $\|x_i - x_*\|$ might not happen or be very slow. Indeed, in our spherical parameterization example the data might rotate on the sphere during minimization without having influence on the energy. One might be tempted to add constraints — for instance, in the form of Lagrange multipliers — and force the Hessian to be positive definite. But is this necessary or even desirable? The answer is not clear. Setting a-priori constraints complicates the computation of gradient and Hessian somewhat. It might also not make much of a difference to the unconstrained case if one is only interested in the convergence of $\|\mathbf{grad}\, f(x_i)\|$ and doesn't care about which particular final x is chosen by the minimizer.

Singular Hessians encountered away from the minimum can be a serious problem. The same is true for Hessians with negative curvature directions. In the first case, the Newton-step can lead to a very long (or even infinite length) step into the zero-curvature direction. In the second case,

the step will point locally upward when projected to the negative curvature direction (compare with the example in Figure A.4). This means non-positive curvature directions often cause a reduction of the step-length and may completely ruin the Newton-step. For this reason, methods have been developed that analyze Hessians and compute similar, but positive definite matrices for use by the Newton-step [NW99]. But this is often fairly expensive and the problem often better dealt with conjugate gradient based (trust-region) methods [NW99, CGT00].

Discontinuities at a distance from the minimum In our experience discontinuities are unlikely to cause any of the first or even second order methods to fail, at least if they don't pass through the minimum. But discontinuities that are attempted to be "stepped over" may trigger a reduction of step length or trust-region size by the control algorithm. This may force very small step lengths and significantly increase the number of iterations.

Poles Infeasible regions are often delimited by poles. Because one can assign the function value $+\infty$ to these regions one can think of poles as discontinuities of infinite height — with similar consequences for the solvers. Other poles, like $f(x) = x^{-2}$ at $x = 0$, do not form boundaries of infeasible regions. They might partition the domain Ω into multiple regions. But in our opinion their semantics can be troublesome, especially if one is interested in a local minimum that is *near* the starting point: the possibility of crossing a pole is real.

The Taylor series expansion has a convergence radius that does not extend over poles. For this reason one might be forced to use very short step lengths, especially near poles! (This was very noticeable in some of our experiments.) When designing objective functions it seems for solution efficiency reasons very desirable to move poles as far away as possible from solution and initialization.

Self-concordance Very little theory exists on the minimization progress for non-quadratic functions far away from the solution. But for some special convex functions $f : \mathbb{R} \to \mathbb{R}$ with the *self-concordance* property

$$\|f'''(x)\| \leq 2f''(x)^{3/2} \tag{A.15}$$

results on fast convergence are known. The negative logarithm function $f(x) = -\log(x)$ is self-concordant and for this reason very popular for defining pole barriers. This comes handy when dealing with inequality constraints and used with *interior point* methods [BV04] (Sections 9.6 and 11.5).

Methods with memory and restarts The conjugate gradient, heavy ball, quasi-Newton methods and some trust-region minimizers keep a memory of objective function data encountered several iterations ago. This historic data can be misleading, particularly when collected far away from the minimum. For this reason, implementations often provide a procedure to clear the data and restart the minimization with the current iterate x_i. Restarts can be done periodically, but they are expensive. A restarted minimizer often begins with the steepest descent and slowly learns more about the objective function. The improved convergence rate will often be achieved in the limit, but only if no further restarts happen. Sometimes users have the desire to interfere with the minimization, for instance to perform periodic projections to enforce certain additional conditions. Such interference by the user invalidates the historic data and might require a restart of the solver.

A.4.5.1 Libraries and Further Reading

Nonlinear problems cannot be solved efficiently without a basic understanding of the available numerical methods. Some first order methods and even the Newton-step are fairly simple to implement. Fine tuning these methods requires a lot of experience. It also often happens, that particular problems respond in somewhat unpredictable ways to different solution methods. For this reason being able to experiment with competing algorithms is very valuable. For this reason we argue, that one should at least try some of the existing optimization libraries before attempting to implement competing methods.

Software When searching for libraries, a good starting point is the "Decision Tree for Optimization Software" [MS05]. The freely available Toolkit for Advanced Optimization (TAO) offers implementations of conjugate gradient, quasi-Newton, Newton and trust-region solvers [BMMS04]. TAO makes heavy use of PETSc [BEG$^+$97], which provides parallel implementations of linear and nonlinear solvers. A commercially available solver for convex problems is MOSEK.[11] Linear solvers and preconditioners are for instance provided by the free libraries SuperLU (an efficient direct solver [DEG$^+$99]) and hypre (a collection of high performance preconditioners [FBC$^+$]). Many numerical libraries depend on linear algebra kernels, such as the self-tuning ATLAS [Atl] or the proprietary Math Kernel Library (MKL) for Intel processors [Int], for the efficient computation of basic vector or matrix operations.

[11] http://www.mosek.com

Literature The following literature has been used for compiling this survey.

- ◇ First order methods — [Pol87, NW99]
- ◇ Newton methods — [Pol87, Kel95, NW99, CGT00]
- ◇ Trust-region methods — [NW99, CGT00]
- ◇ Constrained optimization — [Pol87, CGT00, BV04]
- ◇ Convex problems — [BV04, Pol87]
- ◇ Stochastic methods — [Pol87, Spa03]

[Pol87] provides unique theoretical insights, especially into first order and simple Newton methods. This book is slightly outdated (it was written in 1987) as, for instance, trust-region methods are not mentioned. The treatment is unique in its clarity and depth. [NW99] provides a good modern introduction to optimization. It has a brief discussion of first order methods but focuses on trust-region and quasi-Newton methods. [CGT00] is an exhaustive and recent survey of trust-region methods. It discusses many adaptions of the basic trust-region algorithm A.5 to problems of practical interest, such as the treatment of constrained optimization and heuristics for difficult problems, like allowing non-monotonous decline of the objective function. [BV04] is specialized on convex problems. The treatment is somewhat theoretical, as its focus is on the formulation of dual problems, feasibility and other specialized topics are presented. Many examples are given to illustrate new concepts. Some computational methods are discussed, but not in particular detail. [Kel95] motivates in great detail linear solvers like conjugate gradient and GMRES. Based on these iterative methods extensions for nonlinear equations like the Newton and quasi-Newton (Broyden's) method are developed. The author carefully examines the interplay between linear and nonlinear methods, particular early truncation, and gives many numerical examples.

A.5 Conclusion

Solving smooth nonlinear optimization problems or nonlinear equations with a starting guess near the solution can be considered solved from a theoretic point of view. Practically, there is still a lot of room for tweaking small trade-offs that can add up to large constants. We argue that this could be done automatic examination of some test problems with an approach similar to that of the ATLAS library [Atl].

For starting guess of x that is far away from the solution, we know conditions that guarantee (slow) progress toward a solution for optimization problems. We have argued that solving nonlinear equations in this setting is a substantially harder problem.

For a bad starting guess, typically *much more time* is needed for advancing the sequence x_i closer to the region of Newton-convergence than is needed to solve the final Newton-iterations. Trust-region minimization is the most promising method in these situations. In particular if the standard quadratic Taylor-series model can be replaced by a model capturing the qualities of the particular objective function class in a better way.

The formulation of constraints poses special problems and is actively researched. Best understood are linear equality and convex inequality constraints, especially for convex problems.

Problems with discontinuous gradient are difficult, because we cannot rely on the standard Taylor series model. For convex problems, *subgradient* methods appear to be promising [Pol87, CGT00].

The theory of stochastic function optimization is surprisingly advanced [Spa03, Pol87]. Efficient practical algorithms are focus of current research [Spa03].

Bibliography

[AGP+04] Marc Alexa, Markus Gross, Mark Pauly, Hanspeter Pfister, Marc Stamminger, and Matthias Zwicker, editors. *Point-Based Computer Graphics*. Course Notes. ACM SIGGRAPH, 2004.

[AKS] Burak Aksoylu, Andrei Khodakovsky, and Peter Schröder. Multilevel solvers for unstructured surface meshes. accepted, SIAM J. Sci. Comput.

[AMD02] Pierre Alliez, Mark Meyer, and Mathieu Desbrun. Interactive geometry remeshing. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 347–354, New York, NY, USA, 2002. ACM Press.

[Atl] Atlas. Automatically tuned linear algebra software. http://math-atlas.sourceforge.net.

[BA83] Peter J. Burt and Edward H. Adelson. The Laplacian Pyramid as a compact image code. *IEEE Transactions on Communications*, 31:532–540, 1983.

[BBE+04] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.

[BBG+01] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2001. http://www.mcs.anl.gov/petsc.

[BEG+97] Satish Balay, Victor Eijkhout, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[BF01] Samuel R. Buss and Jay P. Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Trans. Graph.*, 20(2):95–126, 2001.

[BK03] Mario Botsch and Leif Kobbelt. Multiresolution surface representation based on displacement volumes. In *SGP '03: Proceedings of the Eurographics/ACM SIGGRAPH*, pages 483–491. Eurographics Association, 2003.

[BMMS04] Steven J. Benson, Lois Curfman McInnes, Jorge Moré, and Jason Sarich. TAO user manual (revision 1.7). Technical Report ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, 2004. http://www.mcs.anl.gov/tao.

[BMN01] Richard Byrd, Marcelo Marazzi, and Jorge Nocedal. On the convergence of newton iterations to non-stationary points. Technical report, 2001.

[BV04] Steven Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CGT00] Andrew R. Conn, Nicholas I. M. Gould, and Phillipe L. Toint. *Trust-Region Methods*. SIAM/MPS, 2000.

[CRS98] Paolo Cignoni, C. Rocchini, and Roberto Scopigno. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[DEG+99] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.

[DFS05] Neil A. Dodgeson, Michael S. Floater, and Malcom A. Sabin, editors. *Advances in Multiresolution for Geometric Modelling*. ACM Siggraph, 2005.

[DLG90] N. Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graph.*, 9(2):160–169, 1990.

[DRS04] Ingrid Daubechies, Olof Runborg, and Wim Sweldens. Normal Multiresolution Approximation of Curves. *Constructive Approximation*, 2004.

[FBC+] Rob Falgout, Allison Baker, Edmond Chow, Van Emden Henson, Ellen Hill, Jim Jones, Tzanio Kolev, Barry Lee, Jeff Painter, Charles Tong, Panayot Vassilevski, and Ulrike Meier Yang. hypre. http://www.llnl.gov/CASC/hypre/software.html.

[FH05] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geomet-*

ric Modelling, Mathematics and Visualization, pages 157–186. Springer, Berlin, Heidelberg, 2005.

[Flo97] M. S. Floater. Parameterization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14:231–250, 1997.

[Flo03] Michael S. Floater. Mean value coordinates. *Comput. Aided Geom. Des.*, 20(1):19–27, 2003.

[FMS03] Ilja Friedel, Patrick Mullen, and Peter Schröder. Data-dependent fairing of subdivision surfaces. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 185–195, New York, NY, USA, 2003. ACM Press.

[FSK04] Ilja Friedel, Peter Schröder, and Andrei Khodakovsky. Variational normal meshes. *ACM Trans. Graph.*, 23(4):1061–1073, 2004.

[GGH02] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry Images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.

[GGS03] Craig Gotsman, Xianfeng Gu, and Alla Sheffer. Fundamentals of spherical parameterization for 3d meshes. *ACM Transactions on Graphics*, 22(3):358–363, July 2003.

[Gri03] Eitan Grinspun. *The basis refinement method*. PhD thesis, Caltech, Pasadena, California, 2003.

[GVSS00] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal Meshes. *Proceedings of SIGGRAPH 2000*, pages 95–102, 2000.

[GWC+04] Xianfeng Gu, Yalin Wang, Tony F. Chan, Paul M. Thompson, and Shing-Tung Yau. Genus zero surface conformal mapping and its application to brain surface mapping. *IEEE Transaction on Medical Imaging*, 23(7), 2004.

[HAT+00] Steven Haker, Sigurd Angenent, Allen Tannenbaum, Ron Kikinis, Guillermo Sapiro, and Michael Halle. Conformal surface parameterization for texture mapping. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):181–189, 2000.

[HBS+99] Monica K. Hurdal, Philip L. Bowers, Ken Stephenson, De Witt L. Sumners, Kelly Rehm, Kirt Schaper, and David A. Rottenberg. Quasi-conformally flat mapping the human

cerebellum. In *MICCAI '99: Proceedings of the Second International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 279–286. Springer-Verlag, 1999.

[HG00] K. Hormann and G. Greiner. MIPS: An efficient global parametrization method. In P.-J. Laurent, P. Sablonnière, and L. L. Schumaker, editors, *Curve and Surface Design: Saint-Malo 1999*, Innovations in Applied Mathematics, pages 153–162. Vanderbilt University Press, Nashville, 2000.

[Hop96] Hugues Hoppe. Progressive Meshes. *Proceedings of SIGGRAPH 96*, pages 99–108, 1996.

[Int] Intel. Math kernel library. http://www.intel.com/software/products/mkl/.

[JBL03] Maarten Jansen, Richard Baraniuk, and Sridhar Lavu. Multiscale Approximation of Piecewise Smooth Two-Dimensional Functions using Normal Triangulated Meshes. *Submitted for publication.*, 2003.

[KB04] Leif Kobbelt and Mario Botsch. A Survey of Point-Based Techniques in Computer Graphics. 2004.

[Kel95] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.

[KG02] Andrei Khodakovsky and Igor Guskov. Normal Mesh Compression. In Guido Brunnett, Bernd Hamann, and Heinrich Müller, editors, *Geometric Modeling for Scientific Visualization*. Springer Verlag, 2002.

[KGH04] D. Kirsanov, S. Gortler, and H. Hoppe. Fast exact and approximate geodesic paths on meshes. Technical Report TR 10-04, Harvard University Computer Science, 2004.

[KLS03] Andrei Khodakovsky, Nathan Litke, and Peter Schröder. Globally Smooth Parameterizations With Low Distortion. *ACM Transactions on Graphics*, 22(3):350–357, 2003.

[LDW97] Michael Lounsbery, Tony D. DeRose, and Joe Warren. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics*, 16(1):34–73, 1997.

[LKK03] Kyu-Yeul Lee, Seong-Chan Kang, and Tae-Wan Kim. Remeshing into normal meshes with boundaries using subdivision. *Computers in Industry*, 50(3):303–317, 2003.

[LMH00] Aaron Lee, Henry Moreton, and Hugues Hoppe. Displaced Subdivision Surfaces. *Proceedings of ACM SIGGRAPH 2000*, pages 85–94, 2000.

[LSS+98] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Proceedings of SIGGRAPH 98*, pages 95–104, 1998.

[MBLD02] Mark Meyer, Alan Barr, Haeyoung Lee, and Mathieu Desbrun. Generalized barycentric coordinates on irregular polygons. *J. Graph. Tools*, 7(1):13–22, 2002.

[McC02] John McCleary. Trigonometries. *American Mathematical Monthly*, 109:623–638, 2002.

[Mey04] Mark Meyer. *Discrete differential operators for computer graphics*. PhD thesis, Caltech, Pasadena, California, 2004.

[MS05] H.D. Mittelmann and P. Spellucci. Decision tree for optimization software. 2005. http://plato.asu.edu/guide.html.

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer series in operations research, 1999.

[PH03] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. *ACM Trans. Graph.*, 22(3):340–349, 2003.

[Pol87] Boris Theodorovich Polyak. *Introduction to Optimization*. Optimization Software, Inc., 1987.

[Pow73] M.J.D. Powell. On search directions for minimization algorithms. *Mathematical Programming*, 4:193–201, 1973.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces. *Experimental Mathematics*, 2,1:15–36, 1993.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[SAPH] J. Schreiner, A. Asirvatham, E. Praun, and H.. Hoppe. Inter-surface mapping. accepted, SIGGRAPH 2004.

[SGD03] A. Sheffer, C. Gotsman, and N. Dyn. Robust spherical parameterization of triangular meshes. In *In Proceedings of 4th Israel-Korea Binational Workshop on Computer Graphics and Geometric Modeling*, pages 94–99, 2003.

[Spa03] James C. Spall. *Introduction to Stochastic Search and Optimization*. Wiley-Interscience, 2003.

[SS95] Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. *Computer Graphics Proceedings (SIGGRAPH 95)*, pages 161–172, 1995.

[SS96] Peter Schröder and Wim Sweldens, editors. *Wavelets in Computer Graphics*. Course Notes. ACM SIGGRAPH, 1996.

[SS01] Peter Schröder and Wim Sweldens, editors. *Digital Geometry Processing*. Course Notes. ACM SIGGRAPH, 2001.

[SSGH01] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM Press, 2001.

[SV89] John P. Snyder and Philip M. Voxland. *An Album of Map Projections*. U.S. Geological Survey, 1989.

[Swe96] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.

[Van01] Robert J. Vanderbei. *Linear Programming*. Springer, 2001.

[ZSS96] Denis Zorin, Peter Schröder, and Wim Sweldens. Interpolating Subdivision for Meshes with Arbitrary Topology. *Proceedings of SIGGRAPH 96*, pages 189–192, 1996.

[ZSS97] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive Multiresolution Mesh Editing. *Proceedings of SIGGRAPH 97*, pages 259–268, 1997.