A Parametrization-Based Surface Reconstruction System for Triangular Mesh Simplification with Application to Large Scale Scenes

Adriana Wise

Graduate Center, City University of New York

How does access to this work benefit you? Let us know!

Follow this and additional works at: http://academicworks.cuny.edu/gc_etds

Part of the Computer Sciences Commons

Recommended Citation

Wise, Adriana, "A Parametrization-Based Surface Reconstruction System for Triangular Mesh Simplification with Application to Large Scale Scenes" (2014). CUNY Academic Works.
http://academicworks.cuny.edu/gc_etds/128
A Parametrization-Based Surface Reconstruction System for Triangular Mesh Simplification

with Application to Large Scale Scenes

by

Adriana Wise
This manuscript has been read and accepted by the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Date

Professor Ioannis Stamos, Ph.D.
Chair of the Examining Committee

Date

Distinguished Professor Robert Haralick, Ph.D.
Executive Officer

Distinguished Professor Gabor Herman, Ph.D.

Professor George Wolberg, Ph.D.

Professor Michael Reed, Ph.D., Senior Research Associate at Blue Sky Studios

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK
Abstract

A Parametrization-Based Surface Reconstruction System for Triangular Mesh Simplification

by

Adriana Wise

Advisor: Professor Ioannis Stamos

The laser scanner is nowadays widely used to capture the geometry of art, animation maquettes, or large architectural, industrial, and land form models. It thus poses specific problems depending on the model scale. This thesis provides a solution for simplification of triangulated data and for surface reconstruction of large data sets, where feature edges provide an obvious segmentation structure. It also explores a new method for model segmentation, with the goal of applying multiresolution techniques to data sets characterized by curvy areas and the lack of clear demarcation features. The preliminary stage of surface segmentation, which takes as input single or multiple scan data files, generates surface patches which are processed independently. The surface components are mapped onto a two-dimensional domain with boundary constraints, using a novel parametrization weight coefficient. This stage generates valid parameter domain points, which can be fed as arguments to parametric modeling functions or surface approximation schemes. On this domain, our approach explores two types of remeshing. First, we generate points in a regular grid pattern, achieving multiresolution through a flexible grid step, which nevertheless is designed to produce a globally uniform resampling aspect. In this case, for reconstruction, we attempt to solve the open problem of border reconciliation across adjacent domains by retriangulating the border gap between the grid and the fixed irregular border. Alternatively, we straighten the domain borders in the parameter domain and coarsely triangulate the resulting simplified polygons, resampling the base domain triangles in a 1-4 subdivision pattern, achieving multiresolution from the number of subdivision steps. For mesh reconstruction, we use a linear interpolation method based on the original mesh triangles as control points on local planes, using a saved triangle correspondence between the original mesh and the parametric domain. We also use a region-wide approximation method, applied to the parameter grid points, which first generates data-trained control points, and then uses them to obtain the reconstruc-
tion values at the resamples. In the grid resampling scheme, due to the border constraints, the reassembly of the segmented, sequentially processed data sets is seamless. In the subdivision scheme, we align adjacent border fragments in the parameter space, and use a region-to-fragment map to achieve the same border reconstruction across two neighboring components. We successfully process data sets up to 1,000,000 points in one pass of our program, and are capable of assembling larger scenes from sequential runs. Our program consists of a single run, without intermediate storage. Where we process large input data files, we fragment the input using a nested application of our segmentation algorithm to reduce the size of the input scenes, and our pipeline reassembles the reconstruction output from multiple data files into a unique view.
CONTENTS

Chapter 1. INTRODUCTION .............................................. 1
  1.1. Problem Statement ............................................ 1
  1.2. The Laser Scanner ............................................ 2
  1.3. The 3D Data Processing Pipeline ......................... 3
  1.4. Data Simplification and Surface Design .............. 5
  1.5. Our Solution ............................................... 11
  1.6. Original Contribution .................................. 12

Chapter 2. RELATED WORK ............................................ 16
  2.1. Decimation .................................................. 16
  2.2. Simplification through Mesh Parametrization ........ 22
  2.3. Discrete Parametrization Methods .................... 29
  2.4. Pipeline Systems using B-Spline Surface Fitting ... 35

Chapter 3. SEGMENTATION ............................................ 44
  3.1. Free-Form Boundary Segmentation ..................... 44
  3.2. Our Contribution ......................................... 45
  3.3. The Segmentation Algorithm .......................... 45
    3.3.1. Proxy Calculation .................................. 45
    3.3.2. Discussion of the Cohen-Steiner vs. Wise Error Metrics 46
    3.3.3. Global Priority Queue ................................ 51
    3.3.4. Iterative Region Reconfiguration ................. 51

Chapter 4. PARAMETRIZATION ....................................... 58
  4.1. Fix-Boundary Parametrization .......................... 58
  4.2. Our Contribution ......................................... 60
    4.2.1. Parametric Mesh Simplification .................... 60
    4.2.2. Fixpoint Boundaries ................................ 61
    4.2.3. Multiresolution Remeshing ........................ 61
  4.3. The Parametrization Scheme ............................ 62

Chapter 5. RESAMPLING ............................................... 68
  5.1. Parameter Space Resampling ............................ 68
  5.2. Our Contribution ......................................... 68
  5.3. Grid Remeshing ............................................ 69
  5.4. Subdivision ................................................. 71
Contents

5.5. Least Squares Approximation ........................................ 72

Chapter 6. RETRIANGULATION ........................................... 75
6.1. Our Contribution ..................................................... 75
6.2. Quartering the Border Gap ........................................... 76
6.3. Trapezoidalization .................................................... 78
6.4. Monotone Polygon Triangulation .................................... 85

Chapter 7. RECONSTRUCTION .............................................. 89

Chapter 8. ERROR CALCULATION ......................................... 90
8.1. Parametrization Error ................................................ 90
8.2. Reconstruction Error ................................................ 92
8.3. Pipeline Error ......................................................... 94

Chapter 9. RESULTS .......................................................... 97
9.1. Grid Triangles ........................................................ 97
9.2. Decimation in Parameter Space and Delaunay Triangulation .... 102
9.3. Least Squares Approximation from Data-Trained Control Points 104
9.4. Uniform Subdivision ................................................ 105
9.5. Large Scenes ......................................................... 111

Chapter 10. CONCLUSIONS AND FUTURE WORK ......................... 119
10.1. Limitations of Our System .......................................... 119
10.2. Possible Improvements to Our System ............................. 120
10.3. Future Work ......................................................... 121
10.3.1. Point-Based Reconstruction ...................................... 121
10.3.2. Subdivision for Scanned Models ................................. 121

APPENDICES

Appendix A. SURFACE DESIGN FUNDAMENTALS ....................... 122
A.1. Construction ........................................................ 122
A.1.1. Curves .......................................................... 122
A.1.2. Bipolynomial Surfaces .......................................... 127
A.1.3. Total Degree Surfaces ........................................... 136
A.2. Approximation ........................................................ 143
A.2.1. Data-Trained Control Points .................................... 143
A.2.2. Least-Squares Reconstruction .................................... 144
A.2.3. Approximation through Local Linear Interpolation .......... 145
A.3. Subdivision .......................................................... 147
A.4. Interpolation ........................................................ 147

vii
Contents

Appendix B. **PARAMETRIZATION FUNDAMENTALS** .............................................. 149

  B.0.1. The Euclidean Metric Tensor .......................................................... 149
  B.0.2. Equiareal (Authalic) Maps ............................................................ 152
  B.0.3. Conformal Maps ............................................................................. 153
  B.0.4. Discrete Harmonic Maps ................................................................. 156

Appendix C. **SLOPE MONOTONICITY ANALYSIS FOR THE ERROR METRICS** .... 167

**BIBLIOGRAPHY** ......................................................................................... 169
1. Mesh simplification methods ........................................ 41
2. Key for Table 1 ......................................................... 43
3. Performance of segmentation ....................................... 55
4. Performance of parametrization .................................... 66
5. Errors: reconstruction vs. reference surface ..................... 96
6. Performance of grid triangles reconstruction .................... 97
7. Performance of decimation with Delaunay retriangulation ... 102
8. Performance of Bernstein approximation reconstruction .... 104
9. Performance of subdivision with linear interpolation reconstruc-
   tion ................................................................. 105
10. Bernstein coefficient/function values for various $\lambda$ values, planar curve . 126
11. Parametric plane values for reference rectangular grid (bilinear interpolation) ........................................ 130
12. Parametric hyperboloid values for reference rectangular grid (bilinear interpolation) ........................................ 135
13. Parametric plane values for reference triangular grid (barycentric interpolation) ........................................ 138
14. Control point values for the hyperbolic paraboloid based on a triangular grid ........................................ 141
15. Parametric hyperbolic paraboloid values for reference triangular grid (barycentric interpolation) .......................... 142
## List of Algorithms

1. One Partitioning Step ................................................. 56
2. Compute Umbrella .................................................... 65
3. Compute Four Sections ............................................... 77
4. Line Sweep .............................................................. 80
5. Make Partition Lines .................................................. 81
6. Make Monotone Polygons ............................................. 82
7. Recursive Make Monotone Polygons ............................... 84
8. Monotone Polygon Triangulation ................................... 88
LIST OF FIGURES

1. Point neighborhood and sliding triangle. .................................................. 31
2. Atomic map $g$ from 3D triangle $T$ to 2D triangle $t$. ........................... 33
3. Monotonicity of error metrics: $E_{\text{Cohen-Steiner}}$ vs. $E_{\text{Wise}}$. .......... 49
4. Segmentation of the bunny model with 50 regions. ................................. 50
5. Spurious points created in an interior scene by reflection of data in the mirror. 53
6. Segmentation on different models. ............................................................. 54
7. Partial scan of the Great Hall (CCNY). .................................................... 55
8. Different levels of segmentation on the bunny model. ................................ 55
9. Views of the segmentation of the fandisk model. ...................................... 56
10. Segmentation of scan 3 of the Great Hall with 144 regions. ....................... 57
11. Parametrization for the hemisphere. .......................................................... 59
12. Hemisphere grid point reconstruction with different parametrization weights,
    15 regions and an oversampling factor of 8. ............................................. 64
13. Parametrization triangulation for 50K detail. .......................................... 66
14. Parametrization on various models. ......................................................... 67
15. Simplified border line-up for subdivision (hemisphere). ........................... 72
16. Polygon partitioning and triangulation. .................................................... 76
17. Multiple cusps: our intersection parity check solution. ............................ 78
18. Phases of triangulation for double-cusp polygon. ...................................... 83
19. Monotone polygon triangulation (theoretical example). ........................... 85
20. Monotone polygon triangulation (actual implementation). ........................ 85
21. Sample polygon with double cusps. .......................................................... 86
22. Border gap sectioning, monotone polygon partitioning and triangulation
    (hemisphere regions). ............................................................................. 87
23. Our parametrization of simple polynomial surfaces. .................................. 91
24. Analytic surface computation from reference grid vs. our parameter points
    (parametrization error). ........................................................................... 92
25. Analytic surface computation vs. our reconstruction from our grid resamples
    (reconstruction error). ........................................................................... 93
26. Monkey saddle parametrization, grid resampling, and reconstruction.......... 93
27. Analytical surface computation from reference triangular grid vs. our recon-
    struction (pipeline error). ........................................................................ 95
List of Figures

28. Grid triangles reconstruction (fandisk). ........................................ 98
29. Grid triangles reconstruction (hemisphere). ................................. 98
30. Grid triangles reconstruction (bunny). ......................................... 99
31. Grid triangles reconstruction (Great Hall detail). ......................... 100
32. Grid triangles reconstruction (armadillo). ................................. 101
33. Decimation with Delaunay retriangulation (horse). ....................... 102
34. Decimation with Delaunay retriangulation (hemisphere). ............... 103
35. Decimation with Delaunay retriangulation (bunny). ....................... 103
36. Bernstein reconstruction (various models). ................................. 104
37. Region border straightening for subdivision (bunny). .................... 106
38. Uniform subdivision (hemisphere). .......................................... 107
40. Uniform subdivision (bunny). ................................................. 109
41. Uniform subdivision (armadillo). .............................................. 110
42. Original triangle sequencing in Great Hall scan data file. .............. 111
43. Great Hall adjacent scans: original triangulation ...................... 112
44. Scan 3 of the Great Hall, basis for sequential processing. ............. 113
45. Scan 3 of the Great Hall, simplified mesh. ................................. 113
46. Great Hall, detail. ..................................................................... 113
47. Great Hall: original vs. our triangulation. .................................. 114
48. Scan 4 of the Great Hall reconstruction with simplification. .......... 115
49. Scans 21 through 26 of the Great Hall: grid triangles reconstruction. 116
50. Scans 21 through 26 of the Great Hall: points reconstruction .......... 117
51. Scan 26 of the Great Hall, simplified mesh, vs. photo. ................. 118
52. Sideways parabola through bilinear interpolation, showing interpolation values and control points. ......................................................... 125
53. Parametric plane: rectangular parameter grid and polynomial surface. 129
54. Elliptic paraboloid: rectangular parameter domain and quadratic surface. 131
55. Hyperbolic paraboloid: rectangular parameter grid and quadratic surface from coordinate functions. ......................................................... 136
56. Plane: triangular parameter grid and total degree surface. ............. 138
57. Hyperbolic paraboloid as a total degree quadratic surface. .......... 141
58. Hyperbolic paraboloid: control points. ..................................... 143
59. Three-dimensional surface patch as a mapping of a planar patch. .... 152
60. Minimal surfaces transformation f. ............................................. 158
61. Vertex neighborhood, a.k.a. umbrella, a.k.a. stencil (for parametrization weights). ................................................................. 163
INTRODUCTION

1.1 PROBLEM STATEMENT

The three-dimensional reconstruction of objects of varying scales is an area of computer vision and computer graphics that covers the combined process of capturing their shape and appearance, of processing the resulting surface data, and of rendering the final computerized model. The need for data processing is tightly bound to the type of data collection method, as each method has its strengths and limitations. Capturing data for large scenes is performed with time-of-flight scanners, which output point clouds, i.e. sets of three coordinates plus additional information for each point. Points are collected in a rasterized manner, following the motion of the scanner, which covers all reachable surface points in the field of view in a sweeping pattern. Processing these points requires re-creating adjacency information, aligning data from the various fields of view, as well as providing an approximation of the surface of workable size.

This thesis provides a solution for the stages of simplification of the point cloud data and of surface reconstruction. Our method uses a preliminary stage of surface segmentation, which generates surface patches that are mapped onto a two-dimensional domain with boundary constraints. On this domain — which provides, for each original three-dimensional region, an affine mapping in the plane — our method re-samples points in a regular grid pattern, at a flexible resolution. To resolve the previously open problem of border reconciliation across adjacent domains after reconstruction, our method discards re-sampled grid points that do not fall within the fixed boundary of the two-dimensional domains and re-triangulates the border gap between the grid and the fixed irregular border. It then proceeds with the reconstruction as follows: for a point cloud reconstruction, restores previous border points as part of the re-sampled set, and computes for the rest of the points a linear interpolation of their neighbors; while for a triangulated reconstruction, our method reconstructs the whole re-triangulation, using the same interpolation scheme, but adding connectivity information from 2D to the reconstructed 3D points. All stages are our original implementation, and are part of the same program, which runs sequentially the stages of segmentation, parametrization, resampling, retriangulation, and reconstruction. The output consists of a triangular mesh.
We will begin by briefly categorizing and describing the data capturing methods, in order to place our own work in the appropriate context. The methods of capturing information about three-dimensional surfaces may be active or passive. The passive methods do not interfere with the reconstructed object, but only use a sensor to measure the radiance reflected or emitted by the object’s surface to infer its three-dimensional shape. The sensor can be an image sensor in a camera, sensitive to visible light. Reconstruction based on processing and rendering of image data is called image-based reconstruction.

The active methods interfere with the object being studied either mechanically or radiometrically. Mechanical active methods function on the principle of direct contact with the object, from a known location, to measure distance: a depth gauge is used to find the distance to the object placed on a rotating turntable. Contact 3D scanners (e.g. CMMs, or coordinate measuring machines) are an implementation of this method. Radiometric active methods refer to using radiation emitted toward the object, then measuring the reflection time to infer the distance to its surface. Among this latter category, examples range from moving light sources, time-of-flight scanners, to microwave and ultrasound.

The choice of data capturing method depends on the susceptibility to microscopic surface destruction during measurement, as well as on the accessibility and the size of the object. In the field of 3D reconstruction, when working with large objects, to which access is possible — for various reasons: traffic, historic preservation, natural obstacles, size — only from a distance, the method of choice is time-of-flight scanning. The position of 3D surface points using time-of-flight scanners is calculated by measuring the time it takes for a laser beam emitted by the scanner to travel to the surface and back, given that the speed of the laser beam (the speed of light $c = 299,792,458 \text{ m/s}$) is known:

$$c = \frac{2d}{t} \implies d = \frac{c \cdot t}{2}$$

(1)

In order for the distance calculation to be accurate, given the large value of $c$ relative to the distance from the scanner to the object, the time measurement has to be very accurate itself. For instance, the time of travel for the laser beam over the distance of one meter would be $t = \frac{d}{c} = \frac{1\text{ m}}{299,792,458 \text{ m/s}} \approx 3.3564 \cdot 10^{-9} \text{s} \approx 3.34 \text{ns}$! Due to the difficulty of implementing time measurement devices with this accuracy, the precision of scanner data is only of the order of millimeters. High density (every $\frac{1}{4}\text{ in.}$) and high speed data collection (from 10,000 to 1,000,000 readings per second) both become
necessary in order for enough surface information to be available to perform noise elimination or correction for inaccurate measurements.

On the other hand, for the same reason (distance determination using time measurement for travel of laser beams), time-of-flight scanners are very suitable for data collection from large distances (up to 200m), which is why they are used for scanning large urban scenes, buildings (exteriors and interiors, industrial or architectural), large constructions (highway systems, industrial sites), or land forms. But with this type of usage, extraneous objects may be present or even move in the field of view, and their interference introduces false readings, and, even after their elimination, introduces gaps in the data for the real surface where such occlusions occurred.

Another example of error introduced by the laser scanner are the readings at the object’s edges, due to the fact that, as a compensation for the low accuracy, scanners usually collect information in more than one pass over slightly different locations for each point and then average them. This results in edge points being averaged with points from third objects or from the background, which generates point location inaccuracies, or introduces the need for the prior elimination of these points from the measurement set altogether.

But perhaps the most important limitation occurring with scanner data is that, given its unpredictable accuracy (very good otherwise, of the order of millimeters, but presenting the potential of local variations of the same order as features), as well as its high sampling rate (requiring minutes or more for a complete scan), each measurement is very susceptible to motion; which is why scanners are placed on tripods in fixed positions, and compensations for the slightest vibrations and even for changes in temperature have to be made internally. Data can be collected only in batches, one per field of view, which means, one, that partial data sets collected from several fields of view have to be reconciled afterwards within a same coordinate system, a stage of scanner data processing known as registration; and two, that to acquire enough surface information to compensate for the level of accuracy, the data is collected in large amounts, which requires intelligent simplification.

1.3 THE 3D DATA PROCESSING PIPELINE

These inherent shortcomings — good but variable precision, high sampling rate, partial data sets from each field of view in different coordinate systems, occlusions — can be, to some extent, corrected through further conditioning of the acquired data: alignment, deduplication, simplification, smoothing, fitting to analytical functions where applicable. Our interest is, however, limited to only one of them: simplification. To
give perspective over the whole process, and again context to our area of preoccupa-
tion, we summarize below the 3D processing pipeline, in its most general sequence

REGISTRATION Data resulted from all the fixed scanner positions, one from each
deck of view, must be aligned in a unified set. The recomposition of data from the
partial views requires the reconciliation of all the different reference systems into a
unique view, during the phase of registration. This step also compensates for the fact
that the different subsets of data will have been collected across partially overlapping
sections of the surface, with duplicate data referring to the same points, but taken in
reference to different systems of coordinates. It is sometimes necessary to perform an
initial phase of segmentation prior to the registration, in order to identify markers to
use with various registration methods.

SEGMENTATION After all the data subsets have been regrouped into a unique co-
ordinate system, forming what is called a point cloud, the millions of points may
be divided into subsets during a process called segmentation, in which, as the name
suggests, suitable data subsets are created within the natural boundaries represented
by surface features or even by small local variations of curvature. This enables self-
contained regions of the surface to be processed independently, a necessity in com-
puter vision for a number of reasons — surface segments will most often present
different levels of detail, the number of points for the entire object may be too large
for processing all at once, parallel computation may be envisioned for each segment
etc.

SIMPLIFICATION Prior to, simultaneously with, or after segmentation, the data in
each segment may still present redundancy or size issues, which makes it necessary
that large data sets undergo the process of simplification, a reduction in the number of
points in the data set. This has to be done in such a way that the shape of the object be
able to retain its structure and features, and only truly redundant points from areas of
the surface with little or no variation of curvature should disappear during this phase.
Simplification is necessary in order for multi-million point data sets to be reduced to
sizes at which they can be manipulated or rendered in close to real time.

As shown in Chapter 2, this sequence is not obligatory: segmentation may be done prior to registration, as in [Yu 2010].
The range scanner provides a set of point locations and other properties (i.e. reflectance) of the object’s surface, integral or partial. As mentioned, the sampling rate is very high, of up to the order of millions of points per scan. The usual scenes being described via three dimensional modelling will often contain large areas with redundancy — building façades, open roads, interior walls, man-made construction features — or their scanning will result in point clouds of a much larger density than the need for processing, or than the real time rendering capability. This introduces the need for intelligent data point reduction, which should neither eliminate feature points, nor indiscriminately lower density of representation where the level of detail demands it. The most popular solutions for point reduction currently in existence operate directly on the three dimensional point cloud.

In the field of computer-aided graphic design, however, many successful data processing methods are formulated the context of discrete surface parametrization, which assumes that an equivalent planar representation of the data points along with their connectivity — triangular, quadrangular, polygonal — is available or can be computed from the data points, to mimic the behavior of a parametric generating domain. This has been historically the case because a surface formulated in the implicit model, in which the 3D surface is the zero set of a function of three variables \( F(x, y, z) = 0 \), \( \mathcal{Z}(F) = \{(x, y, z) \in \mathbb{R}^3 | F(x, y, z) = 0\} \), is not necessarily defined: there is no guarantee that \( \mathcal{Z} \neq \emptyset \) (i.o.w. there is no guarantee that \( F(x, y, z) : \mathbb{R}^3 \to \mathbb{R} \) is injective, i.e. \( \forall a \in \mathbb{R} \exists (x, y, z) \in \mathbb{R}^3 \text{ s.t. } F(x, y, z) = a \)), therefore it is much more convenient to assume the existence of a two-dimensional generating domain. In addition, the point of the surface can be easily derived from the corresponding parameters, rather than calculated from algebraic equations of high-degree. This also corresponds to the fact that the interest in surfaces has sprouted from the problem of surface construction (in the auto industry).

A parametric function \( F : \mathcal{P} \subset \mathbb{R}^2 \to \mathbb{R}^3 \), where \( \mathcal{P} \) is the parameter space, describing a surface, is usually of the form:

\[
F = \begin{cases} 
    x = F_1(U, V) \\
    y = F_2(U, V) \\
    z = F_3(U, V),
\end{cases}
\]

where the bivariate polynomial functions \( F_i(U, V) : \mathbb{R}^2 \to \mathbb{R} \) are of various degrees \( p, q \) in \( U \) and \( V \) respectively. The parametric multi-polynomial function \( F : \mathbb{R}^2 \to \mathbb{R}^3 \)
can be treated as of either bipolynomial degree $\langle p, q \rangle$ or as of total degree $m = \max\{p, q\}$ and has $n = \frac{(m+1)(m+2)}{2}$ coefficients:

$$
F(U, V) = \begin{cases} 
F_1(U, V) = a_1 U^p + a_2 U^{p-1} V + \ldots + a_{n-2} V^{q-1} U + a_{n-1} V^q + a_n \\
F_2(U, V) = b_1 U^p + b_2 U^{p-1} V + \ldots + b_{n-2} V^{q-1} U + b_{n-1} V^q + b_n \\
F_3(U, V) = c_1 U^p + c_2 U^{p-1} V + \ldots + c_{n-2} V^{q-1} U + c_{n-1} V^q + c_n 
\end{cases}
$$

(3)

The main idea behind surface design using parametric polynomials is that, by introducing, for each variable, a set of new variables of multiplicity equal to the value of its highest exponent, any polynomial function can be rendered linear and symmetric in each new variable. This procedure is called polarization and transforms the 2-variable function $F_i(U, V), i = 1, 3$ of bipolynomial degree $\langle p, q \rangle$ or of total degree $m$ into a polarized form of $F_i$, denoted from here on, as a notation convention, in small cap $f_i$, which is linear and symmetric in each of its $p + q$ (the bipolynomial case) or $m$ (the total degree case) variables: $f_i(u, \ldots, u, v, \ldots, v) : \mathbb{R}^{p+q} \rightarrow \mathbb{R}$, or $f_i((u, v), \ldots, (u, v)) : \mathbb{R}^m \rightarrow \mathbb{R}$. Then, the surface generated by the three coordinate polynomial functions can be determined at every point $a = (u, v)$ through linear interpolations in each variable, with ratios of interpolation $\lambda$ for $u$ and $\gamma$ for $v$ within a given interpolation interval $[r; s]$.

Any problem of three-dimensional data processing, and, in particular, our simplification problem, can be reformulated — using parametric bivariate polynomial representations of functions — in terms of one of the following fundamental problems:

**CONSTRUCTION**

Given:

- the parametric polynomials $F_i$ representing the surface;
- an initial reference patch, a parametric range, rectangular $[r; s] \times [r; s]$, where $r, s \in \mathbb{R}$ or triangular $\triangle rst$, where $r, s, t \in \mathbb{R}^2$;
- a ratio of interpolation $\lambda$;

Find:

- control points (the complete control net) corresponding to the polynomials $F_i$;
- points on the surface as the range of $F$ at various $\lambda$s.

---

2 Representations of the form $f(x, y, z) = f(x, y, z(x, y))$ are implicit, not parametric.
### Interpolation

**Given:**
- real data points;
- a surface patch, for instance triangular, $\triangle RST = F(\triangle rst) = \triangle F(r)F(s)F(t)$, with $F$ unknown anywhere except at its endpoints;

**Find:**
- points on the surface as the range of $F$ at various $\lambda$s, with $F$ passing through the original data points.

### Approximation (Fitting)

**Parametric**

**Given:**
- real data points;

**Find:**
- pseudo-control nets (i.e. derived from the data in the absence of functions);
- an approximating surface (mimicking a parametric function or set of functions), not necessarily passing through the data points;
- samples of the resulting approximating surface at new points.

**Non-Parametric**

**Given:**
- real data points;

**Find:**
- implicit functions (height functions) that approximate the data points, by using error minimization to find the best fit function for parts of, or for the whole set of points;
- samples of the resulting functions at new points.

The main difference between interpolation and approximation, in the context of multilinear interpolation of functions for surface design, is the following. In interpolation, through the existence of control points, subsets of which are actually function values at the endpoints of a given range, we aim to find a function which, by definition, passes through these control points. Since the control points are part of the
surface, sampling new points in the parameter domain and calculating their image through multilinear interpolation will interpolate this function, thus producing new valid points in between original points, on the same surface, even in the absence of its descriptive function.

In approximation, it is deemed more important to find a polynomial description of a surface that does not pass through the data points, but retains their general shape. Thus, the control points are not known, since we don’t even require the data points to be on the surface, but data-trained to yield samples of the approximation function near the data points. Some implementations use iterative minimizations of the error between the approximation function and real data. In reality, in most cases, a combination of both is used, with some real data providing an initial control net (thus making the surface pass through a subset of the real data points), and the remainder of computed points used to evaluate the quality of the approximation.

We would like to emphasize that approximation surfaces are also computed through multilinear interpolations, not to be confused with what is called in the field “interpolation surfaces”. Our own linear interpolation in each original mesh triangle interpolates planes — which is good enough, given that our goal is simplification, and that the original mesh triangles are dense and small. But, our interpolation method yields an approximation surface. As we will see later, higher order surfaces (quadratics, cubics etc.) could have been used. Although they require more control points (6 for quadratics, 10 for cubics etc.), these can be determined as combinations of the polar variables derived from the reference triangle parameter endpoints, and given as arguments to the polar forms of the coordinate functions, provided these are known. Thus, all that is needed in computing a surface are the reference triangle, and the value of the function at the parametric control points. Because the plane is the only surface that is completely determined by the three vertices of the reference triangle, it is frequently used as interpolation domain for reconstruction applications where the density of points is relatively high, and thus locally approximating the surface with a plane does not introduce significant errors.

**Subdivision** We refer here only to triangular subdivision. Given:

- a known parametric polynomial function;

- an initial reference triangle \( \triangle RST \subset \mathbb{R}^3 \) (from a parameter reference triangle \( \triangle rst \)), defining an initial planar control net;
Find:

- a limit surface defined by \( n \) steps of linear barycentric interpolation of the initial triangles and all subsequent resulting triangles, which themselves become at every step more refined control nets, converging to the surface.

This method makes best use of the fact that the surface can be interpolated without the knowledge of its polynomial description, but only with the knowledge of its control points. Given a triangulated surface, subdivision applied once produces a new control net that includes only points obtained from the old triangulation through barycentric interpolation and additional averaging in the third coordinate, depending on the scheme. Applied repeatedly, subdivision produces in the limit the same smooth surface as defined by the original triangulation as control points, if the original control points were used in a resampling scheme. It is visually indistinguishable from the surface obtained through direct parametric calculation from parameter triangles, or from interpolation using as coefficients the values obtained by converting parameter point values into interpolation coefficients \( \lambda, \gamma, \mu \) within the reference triangle. However, it is calculated with nested linear interpolations using the same \( \lambda, \gamma, \mu \), performed recursively in each subdivided triangle taken as reference triangle at each step.

Hence, the difference between interpolation and subdivision is more subtle. In subdivision, the control points become part of the surface at each step, while in interpolation only reference triangle endpoints (i.e. the initial control net) are part of the surface. Subdivision also uses an additional averaging step, which relocates the new points from their purely interpolated locations. This is important for the following reason. Supposing the surface is not defined by a function, but contains sparse triangulation points, defining its general shape. Applying one step of barycentric interpolation produces the next layer of control points, while averaging can be viewed as a modification to the purely linear interpolation. In the limit, the control net obtained through repeated interpolation followed by averaging identifies with the surface, and thus, an unknown surface can be adequately represented only by linear interpolations, a vertex mask, and an initial triangular mesh control net. This can be used in surface approximation, however with the caveat that, the smoother the surface, the more it shrinks away from the original points.

The problem of data simplification can be cast as either one of these fundamental surface design problems.

---

3 Barycentric interpolation applies only in the first two coordinates.
Construction To design a surface, the constructive approach requires the knowledge of a function or a set of functions. If the functions are unknown, which is the case for modelling surfaces using scanned data, then the problem of surface reconstruction can be treated as a problem of either approximation, interpolation, or subdivision.

Interpolation If, for instance, the data is known to be planar in certain regions, a convenient linear bivariate parametric polynomial (or a non-parametric linear trivariate polynomial) can be chosen to describe the plane passing through the data, and most likely fitting everywhere in between, with coefficients derived through error minimization from an appropriate subset of (or all of) the data points. Then, new points can be sampled on this surface, now entirely defined through its function, in any configuration. This would be an application of surface interpolation, since the surface is known to pass through the points used for training the coefficients through error minimization.

Approximation If, instead of fitting data points exactly, the interest is in finding control points which would generate the surface near, but not passing through, the original points, an approximation approach would consist of iteratively improving the control points by minimizing the error between the reconstructions and the real data. This is the case when interpolating would result in an overfit surface, for example a bumpy higher order surface passing through data points when an approximating plane would be the right answer.

Subdivision Subdivision is a newer approach which uses a recursive refinement of control points to derive an extended, more refined control net which converges, at limit, with the surface. However, since the only type of surface passing through all its control points is the plane, subdivision uses linear interpolation combined with an averaging step in 3D. The shape of the surface is derived from the interpolation weights applied to a stencil around a vertex. Used with arbitrary data points, subdivision could approximate an unknown surface by first computing its own control nets as if the surface were linear, averaging the results with a vertex mask, and then updating the mask weights through a feedback process from original data points.
1.5 OUR SOLUTION

Our method is also based on discrete surface parametrization, inspired by the idea that, in the interpolation formulation of the problem, a surface point could be derived from an initial mesh triangle, considered as a complete 3-point control net for a plane (for which all three control points are on the surface). A simplified version of the surface should be possible at any resolution, through resamples easily designed in the parameter domain in regular configurations. Furthermore, since we are going from dense to sparse, the support net for our interpolation need not be extended, since the best possible surface approximation at the resample is on the dense surface, and not beneath. Each initial surface triangle could be its own control net in a one-step linear interpolation putting the resample directly on the surface.

Resampling and retriangulation (with or without introduction of synthetic points) are easier to implement in the plane. The second building block of our solution is, therefore, mapping regions of the 3D surface onto generating planar domains, which are our resampling domains, and the quality of which plays a big part in the quality of the reconstruction. The 2D maps are not orthogonal projections onto planar (or other) domains, because this would lose significant information about the shape of the triangulation, as well as produce non-surjective images and therefore ambiguities in reconstruction. Good quality 2D mappings produce flattened versions of the 3D surface which preserve, locally, important properties or the original surface, such as triangle shape and area. The preservation of these properties ensures the existence of a quasi-affine mapping between the 3D domain and the 2D domain, and vice-versa, and thus makes possible the generation of any point \((x, y, z)\) of the final surface from a resample \((u, v)\) from the 2D domain without distortion.

We are given a set of data points, but no function to construct a surface from parameter points, nor a function to construct a parameter domain. Our task is to construct a parameter space reflecting the data well, to resample at different resolutions in the parameter space, and to find a reconstruction scheme. We segment the surface into regions. We then implement a conformal parametrization combined with a uniform resampling in the parameter space. For reconstruction, two approximation solutions were interesting. First, we computed a least-squares approximation for the surface, using the resampled points as arguments for Bernstein functions in a matrix of Bernstein coefficients, since they are arranged in a grid. The second solution was to consider the resampled 2D points as arguments of bilinear polynomials defined locally on each parameter triangle and use one step of linear interpolation to reconstruct the resamples in the local planes of each triangle. The barycentric location of the resampled points
with respect to their reference triangles can be viewed as one step of bilinear interpolation of the original triangle generating a first order surface (a plane), by using a one-to-one triangle correspondence preserved through a hash table from the original data.

We create multiresolution versions of the surface using the planar mapping corresponding to each surface region as the staging ground for point resampling at various grid sizes. The new grid point distribution in the two-dimensional space reproduces well in the three-dimensional space due to the quality of the parametrization. The regular point distribution at a density attuned to the local need for features in different regions of the surface is implemented through a new method of calculation of the grid step for each parameter region, which depends on the desired point reduction ratio, applied only to the grid points falling within the irregularly shaped surface regions, as they result from our segmentation algorithm.

We also implement a simplified version of subdivision, combined with an averaging step that considers only a triangular stencil for each resampled point, and barycentric coordinates as weights.

We computed a 2D retriangulation of the resamples in parameter space. The reconstruction then reproduces the point connectivity available in 2D from this new triangulation, since the computation of the 3D triangular meshing reduces to a simple correspondence of point indices between 2D resamples and their reconstructed image, as previously mentioned.

In conclusion, in our solution, the 2D placement of the resampled points obeys a regular grid, while the reconstruction can be recovered by a barycentric linear combination of the 3D points (corresponding to the 2D reference points) surrounding the resample. The same exact triangulation computed in 2D can be transported in 3D through a one-to-one point correspondence.

1.6 Original Contribution

Parametric Mesh Simplification The novelty of our approach consists in using discrete parametrization for point simplification, which is traditionally performed in the 3D original domain. The consecrated method, implemented in CGAL (the Computational Geometry Algorithms Library), is a 3D method based on mesh edge collapses for edges whose cost of disappearance (expressing redundancy of the edge scheduled to be deleted) meets a user-specified threshold. The reason 2D mapping is not more widely used for mesh simplification comes from the difficulty of segmenting the original mesh in domains matching at the borders reconstruction: The 2D para-
metric boundaries that fall out of sync with one another after reconstruction, and the
reconciliation of the reconstructed 3D boundaries from adjacent domains remain an
open problem, which requires extensive preprocessing steps to smooth and align the
boundaries of adjacent surface patches in many approaches we studied. Some solve
the border reconciliation problem through repeated steps of parametrization and sub-
division, others solve it through minimizing edge paths along the narrow band of
triangulations adjacent to the border — both of which we see as a complexity over-
head and a major impediment in scaling these methods to large models.

**Fixpoint Border Parametrization** This thesis circumvents this problem by
using parametrizations of mesh surface regions with border constraints, keeping the
region borders in their original configuration, and reusing this configuration for bor-
der reconstruction. This way, the different regions will join perfectly in 3D along these
borders, which becomes especially important when joining output triangles from sep-
arate program runs. If the higher density of points along these borders becomes
undesirable after one iteration of mesh simplification, they can be simplified independ-
ently and averaged across adjacent regions in a single step going through all regions
two by two (through a map, for instance, as we explain in Section A.3).

**Multiresolution Through Gridding With Region Size-Dependent Reduction Factor** Another novel part this thesis brings is the fact that any resolution
can be specified as a reduction factor, which will determine, according to our novel
grid step calculation, the resulting number of points and the size of the grid step for
each region, departing from the original number of points. In contrast with the state-
of-the-art in multi-resolution meshing, which stores base mesh point positions and
offsets to obtain successive levels of refinement/simplification, our method generates
the level of coarseness directly in the parameter domain, by computing the region-
individualized square grid step that obeys the globally specified reduction factor. We
have not experimented with such methods, and cannot attest to the gain in speed from
rendering time LOD retrieval methods. Our method does not store base resolution in-
formation and detail coefficients, as at each coarseness level, in our method there is
only one step of direct interpolation of the original surface.

**New Conformal Parametrization Formula** The parametrization method
itself, which uses geometric 3D information from around each point in order to deter-
mine the placement of that point in the 2D map, is aligned with the general theory of
shape and area preservation, but uses a new coefficient in the calculation of the 2D
point positions. This new coefficient will be discussed in more detail in the chapter on parametrization, and visual examples will be given for comparison among several coefficients widely used.

**Multi-hole border gap triangulation** The retriangulation of the 2D domains, after regular grid sampling, is another new approach for mesh reconstruction. Our method was inspired by what is known in computational geometry as monotone polygon partitioning, applied to what we call the border gap — the space between the border of the inner regular grid and the irregular outer region border. We triangulate this gap by sectioning it into four polygons, by recursively partitioning these four sections into monotone sub-polygons, and then by triangulating the resulting monotone sub-polygons using consecutive edge angle values as triggers to drive the triangulation. The extension of monotone polygon triangulation to polygons with holes is ours. The recursive partitioning of the four border gap sections is a new take on the serial approach proposed in the literature for turning a random polygon into a monotone one. Compared to the sequential version, it simplifies the shape, and reduces the size, of each polygon subject to partitioning in the next step, which results in a dramatic speed-up of the partitioning step, and in the elimination of the need to store more than one cusp vertex information at a time (the site of polygon partitioning, a cusp is a disruption in polygon monotonicity).

**Multiresolution through subdivision and parameter border straightening** Region borders, retained as such for the reconstruction, tend to be much denser than the interior resampled points. In a simplification for large buildings setting, this is not a problem, since our algorithm was devised to be applied a number of times and absorb the dense boundaries as interior points in a wider region in each next iteration. This happens because the regions, at first, do not have feature-dictated borders, due the the large initial number of points and to the requirement that regions do not exceed 500 points, to limit the parametrization matrix to the capabilities of our matrix library (Lapack++). However, in models of smaller scale, or even for buildings after several rounds of point reduction, the region boundaries will eventually align with features. This calls for region border simplification, as well. We implement this through border straightening in the parameter domain, and retaining only *corner points*, i.e. points at the concurrence of three or more regions. We take the resulting simplified polygon for each region and triangulate it using the same monotone polygon triangulation designed for the border gap triangulation. We then subdivide the triangles in 2D using a simple 1-4 subdivision, and reconstruct as before.
All the code is our original implementation, except where, only for comparison to our triangulation method, we used Richard Sciewcuck’s Delaunay triangulation. In conclusion, here are our original contributions at a glance:

- Mesh segmentation using a modified version of the variational shape approximation method;
- Mesh simplification through parametrization combined with multiresolution resampling;
- Fix-point border parametrization;
- New affine mapping coefficient for equidistant and equiareal 2D parametrization of a 3D surface irregular patch;
- New reduction factor to provide uniform point reduction;
- New recursive triangulation method through the fixed points of inner grid border and outer region border. Unlike Delaunay, this method does not introduce artificial points to the re-sampled set;
- New subdivision implementation through region border simplification, followed by triangulation, subdivision and reconstruction. The novelty consists in the method of building the so-called *base domains*, the triangular surface patches that host the subdivision.
2 RELATED WORK

2.1 DECIMATION

Since we are using a planar mapping method for mesh simplification, an overview of the best known methods for point reduction directly in 3D is necessary to establish the challenges for using such methods for large meshes, as well as to give credit to the competing approach — decimation in 3D —, whose popularity is mainly due to the QEM algorithm presented in Garland and Heckbert \cite{Garland1997} and implemented in CGAL, a public repository of implementations for many computer graphics algorithms.

Retiling Polygonal Surfaces (RTPS) — Turk, 1991

\textbf{Turk} \cite{Turk1992} decimates the mesh by inserting new points on the mesh and selectively computing their 3D positions, according to a scheme of vertex repulsion. He superimposes the new points over the old triangulation, and, as more new vertices are added to the mesh, isolates and removes the old points in the new context. A brief step of parametrization is used to remove and retriangulate the gaps left behind by the old points.

The process of vertex repulsion, which determines the location of a new vertex, at first places random points on the 3D surface, producing what the author calls a \textit{mutual tessellation}. A repulsion function decreasing with distance (and zero beyond a fixed radius) is defined for each inserted point as a composite of neighborhood point projections onto the plane tangent at this point. Since the function decreases with distance, only a small neighborhood needs to be referenced for this calculation. The point is moved on the surface in the direction of the resultant force, by an amount proportional with the value of the force (the neighborhood point projection sum). The new locations of these points on the surface represent the vertices of a new triangulation.

The similarity with our method is that it performs the decimation through a complete resampling of the mesh, but it does so in 3D, based on the repulsion forces driving points to new locations. Another aspect that we found similar to our method is the removal of edges in a two-dimensional domain, but while in \textbf{Turk} \cite{Turk1992} this is achieved by projecting points orthogonally onto local tangent planes, our method employs a full-fledged geometry-preserving parametrization. The method presented
2.1 Decimation

by Turk [1992] reconstructs the 3D local triangulation by adding a neighborhood-averaged z coordinate to the new points, while our method can be viewed as performing a weighted average of all the new 3D coordinates. Finally, this paper also introduces nested spaces for smooth interpolation between levels of detail (LODs).

Decimation of Triangle Meshes (DTM) — Schröder and Lorensen, 1992

Schröder et al. [1992] revisit an older mesh reconstruction idea presented originally in Lorensen and Cline [1987], in an attempt to reduce the number of polygons generated by their earlier marching cubes algorithm, which produced counts of 500,000 to 2,000,000 triangles for MRIs or industrial tomography applications. Besides the range scanner, they extend the use of their method to data resulted from other sampling devices producing similarly large polygonal models: range cameras, digital elevation data, satellite data. Their algorithm performs vertex removals using a heap-ordered selection of candidate vertices based on an vertex removal cost function. For each removal, the hole left behind the disappearing vertex is retriangulated through a recursive loop-splitting procedure (where by loop they mean hole boundary). The split line is added to connect two non-neighboring vertices in the loop. This part of the retriangulation resembles vaguely our recursive polygon partitioning, in the sense that they use (non-monotone) polygon partitioning on very simple polygons to add triangulations over small holes.


The theory of mesh reconstruction based on minimization energies characterizing the resample versus the original point set is laid out in Hugues Hoppe’s Ph.D. thesis, Hoppe [1994]. He provides a complete reconstruction pipeline consisting of three phases: triangulation, optimization (simplification), subdivision. In short, he simplifies the input point cloud and connects it into a bare bones mesh, which serves as base domain. The triangulation is obtained, similar to the marching cubes method of Lorensen and Cline [1987], through the evaluation of the zero set of, in this case, a distance function between the data points and a previously computed set of estimated tangent planes (the original marching cubes was evaluating the zero set of a density function obtained through tomography). Once the triangulation is obtained, the energy minimization is used in a second stage to validate the edge collapse, split, and swap operations leading to the simplified representation. The base mesh is then
subdivided, in a third stage, according to the Loop subdivision rules for smooth vertices, while features (creases, corners, and darts) are treated with specific sharpness-preserving weight mask rules. Once again, during this stage, as in the optimization stage, the energy minimization is used to validate the subdivision points at each subdivision level.

This thesis has found its way in the literature in the form of three communications, Hoppe et al. [1992] (detailing phase 1), Hoppe et al. [1993] (detailing phase 2), and DeRose et al. [1994] (detailing phase 3), but the exposé is much more coherent in the completed thesis of Hoppe [1994], which is yet another example of seminal work done under the supervision of professor Tony DeRose, then at the Department of Computer Science and Engineering from the University of Washington. We detail only the second one below.

Mesh Optimization (MO) — Hoppe, DeRose, Duchamp et al., 1993

In Hoppe et al. [1993], detailing the second and more relevant to the problem of simplification phase of the Hoppe pipeline, the authors cast mesh simplification as an energy minimization problem over the connectivity \( K \) and over the 3D vertex positions \( V \), in a double nested loop \( \min_K \{ \min_V \{ E(K,V) \} \} \). They define the mesh energy as:

\[
E(K,V) = E_{\text{dist}}(K,V) + E_{\text{rep}}(K) + E_{\text{spring}}(K,V)
\]

In Equation (4) the minimization of the energy terms defines “optimal mesh”: distance against the original mesh, number of vertices, a spring energy that ensures that a minimum of the first two terms will exist. In detail:

\[
E_{\text{dist}}(K,V) = \sum_{i=1}^{n} d^2(x_i, \Phi_V(|K|))
\]

is the distance energy, where \( x_i \) are the input points, and \( \Phi_V : \mathbb{R}^2 \rightarrow \mathbb{R}^3, \Phi_V(|K|) = V \) is the parametric function translating 2D barycentric coordinates into 3D vertex positions. The connectivity set \( K \subset \mathbb{R}^2 \) provides a frame of reference for the new points, and the distance energy simply measures the deviation of the reconstruction of the resamples from the original point positions \( x_i \):

\[
d^2(x_i, \Phi_V(|K|)) = \min_{b_i \in |K|} \| x_i - \Phi_V(b_i) \|
\]
2.1 Decimation

The *representation energy*, proportional to the number of vertices $m$ of $K$, penalizes a large number of vertices in the optimized mesh:

$$E_{\text{rep}} = c_{\text{rep}} m,$$

where $c_{\text{rep}}$ is an empirical proportionality constant, user-specified.

Interestingly, the third term, the *spring energy*:

$$E_{\text{spring}}(K, V) = \sum_{(j,k) \in K} \kappa \|v_j - v_k\|^2$$

is introduced to eliminate the presence of spikes in the reconstruction, which the authors attribute to employing only the first two terms of the energy sum, $E_{\text{dist}} + E_{\text{rep}}$. The presence of $E_{\text{spring}}$ is necessary to eliminate these spikes, which are a manifestation of an imperfect minimum finding for the first two terms alone. We encounter these spikes in our reconstruction also, but since we do not build the remesh based on a minimization process, we looked elsewhere for an explanation. In our case, the spikes seem to be an artifact of an improper barycentric placement of the resamples. We concluded that they are, in fact, an artifact of reconstruction, which is based on solving a $3 \times 3$ matrix system, whose condition is unknown. Returning to Hoppe et al. [1993], in the formula for $E_{\text{spring}}$, $\kappa$ is a spring constant of small subunit values ($10^{-2}, 10^{-3}, 10^{-4}, 10^{-8}$).

The mesh is first simplified to the bare bones by a procedure that assigns each face to a graph node, and builds graph edges only where the endpoint nodes correspond to adjacent faces whose dihedral angle is below a threshold. Similar to Turk [1992], in the next step random samples are placed on the mesh. The energy minimization method is applied to the oversampled mesh with varying representation coefficients $c_{\text{rep}}$ leading to varying simplification levels. The actual simplification technique is based on edge collapse, split, and swap operations, under specified conditions that select the appropriate case.

*Quadric Error Metrics (QEM) — Garland and Heckbert, 1997*

Garland and Heckbert [1997] bring, as stated before, a very popular mesh decimation scheme, in which iterative contractions of vertex pairs are performed in 3D. If we denote $M_n$ the pre-triangulated input model, and $M_0$ the target approximation, Garland and Heckbert [1997] introduce the generalized contraction sequence $\mathbf{v} = (v_1, v_2, \ldots, v_k)$, based on simultaneous contraction pairs $\langle v_i, v_{i+1} \rangle$. A single run of
2.1 Decimation

the algorithm produces a series of decreasing resolution meshes \( M_n, M_{n-1}, \ldots, M_g \), each mesh in the series being the product of multiple locally performed edge contractions aggregating into a global generalized edge contraction per resolution level.

Contraction takes into account edges \( \langle v_1, v_2 \rangle \) as candidates, if the distance \( \| v_1 - v_2 \| < t \), where \( t \) is a threshold parameter, and according to a contraction cost, defined as the error produced by the contraction vertex \( v \) against the original surface, locally approximated as a collection of tangent planes at points in the neighborhoods of \( v_1, v_2 \). A 4x4 matrix \( Q \) is associated with each vertex \( v = (v_x, v_y, v_z, 1)^T \) to approximate the error at \( v \) after each edge contraction \( \langle v_1, v_2 \rangle \rightarrow v \). The matrix \( Q \) is initialized as the sum of matrices \( K_p \), which arise from the sum over neighborhood \( N_v \) of individual calculations of the squared distance from a vertex to each incident plane:

\[
(p^T v)^2 = (p^T v)(p^T v) = v^T (pp^T) v = v^T K_p v, \quad p \in N_v,
\]

where \( p = (abcd)^T \) is the plane defined by the equation \( ax + by + cz + d = 0 \). Calculating the sum of squared distances from \( v \) to all incident planes \( \bigcap_{p \in N_v} p \) results in the vertex error:

\[
\Delta(v) = \sum_{p \in N_v} (p^T v)^2 = \sum_{p \in N_v} v^T K_p v
\]

where the squared distance matrix \( K_p \), or fundamental error quadric, is:

\[
K_p = pp^T = \begin{pmatrix}
  a^2 & ab & ac & ad \\
  ab & b^2 & bc & bd \\
  ac & bc & c^2 & cd \\
  ad & bd & cd & d^2
\end{pmatrix}
\]

Thus \( Q \) is defined as the sum:

\[
Q = \sum_{p \in N_v} K_p,
\]

and, throughout each contraction \( \langle v_1, v_2 \rangle \rightarrow v \), has the recursive definition \( Q = Q_1 + Q_2 \). With \( Q \) thus defined, the total vertex error over \( N_v \) at each edge contraction step is defined as \( \Delta v = v^T Q v \). This vertex function is the edge removal cost, with \( v \) emplaced on the surface to minimize \( \Delta v \), which is a quadratic form, whose minimization is linear.

All the contraction costs of candidate edges are arranged in a priority queue, which pops the least global cost. This triggers the update of the remaining costs, and the process begins again. However, the simplicity of this algorithm comes with a disclaimer:
it closes topological holes and joins unconnected regions, because the authors believe (rightly so for some applications) that sometimes “topology may be less important than overall shape”.

*Multiresolution Modeling of Point-Sampled Geometry (MRMPSG) — Pauly, Kobbelt, and Gross (2002)*

In [Pauly et al. 2002], the authors create a system of multiresolution representation and editing of point-sampled geometry, in the absence of a mesh structure. They redefine LOD in the context of point-cloud data as not necessarily a reduction in point count, but rather a reduction in geometric detail, meaning that low geometric detail still requires the same point density, only without the high frequency features. The authors keep the vertex density at all resolution levels, and achieve low-frequency versions of the model through smoothing, not downsampling, and they obtain their multiresolution levels through normal displacements of the same sample. Point-cloud simplification is only necessary to speed up the convergence of the iterative Gaussian smoothing method, which is implemented here as a combination of smoothing, down-, and upsampling steps:

\[
S' = (\Phi \cdot \Psi^{-1} \cdot \Phi \cdot \ldots \cdot \Phi \cdot \Psi \cdot \Phi)(S),
\]

where \(S'\) is the smooth approximation, \(S\) is the original point set, \(\Phi\) is the smoothing operator, and \(\Psi\) is the downsampling operator. Simplification is a direct adaptation of the QEM technique from [Garland and Heckbert 1997] to meshless point sets, in which normal estimation is done through eigenanalysis of the covariance matrix describing a point and its neighborhood, and neighborhood information is collected from an \(r\)-radius sphere centered at the point:

\[
C \cdot v = \lambda \cdot vC = \begin{pmatrix}
(p_1 - \overline{p})^T \\
\vdots \\
(p_k - \overline{p})
\end{pmatrix} \cdot \begin{pmatrix}
(p_1 - \overline{p}) \\
\vdots \\
(p_k - \overline{p})
\end{pmatrix}
\]

with \(p_k \in N_p\) and \(\overline{p} = \frac{\sum p_k \in N_p \cdot \Phi_k p_k}{\sum_{k \in N_p} \Phi_k p_k}\), and \(\Phi_k(\|p_k - p\|) = \Phi_k(x) = e^{-\frac{x^2}{\sigma^2}}\), where \(\sigma\) is a parameter controlling the locality of the approximation. Then, the normal vector \(\overline{n}_p\) at \(p\) is the eigenvector with the smallest eigenvalue. The normal displacement offsets \(d_0, \ldots, d_{n-1}\) relate a point to its correspondent in the lowest resolution level \(0\).
through a number of terms reflecting the displacement of the higher resolution level $j$ against the preceding levels in a hierarchy:

$$p_j = p_0 + d_0 \vec{n}_0 + d_1 \vec{n}_1 + \ldots + d_{j-1} \vec{n}_{j-1}$$

The normal estimates at $p_j$ for resolution level $j$ are computed with neighborhood information from the same level.

The third component of this resampling scheme is a decomposition operator which separates the low frequency (global) shape information from the high frequency (local) information. Since the base domain points have to relate to the higher resolution through normal displacements, their computation is in fact a resampling, which the authors achieve through iterative projections $r$ of original points $p$ onto weighted least-square plane neighborhood approximations.

This paper’s focus is the application of deformations to high point count models through editing subsampled versions of the surface, after which all the parameters of the deformation can be transferred to the high resolution model and the deformation computed offline. The deformation technique creates a smooth transition between the total deformation area and the zero deformation areas around. Deformation is not our focus, but this explains why the authors place importance on the speed with which the normal displacement hierarchy (and thus the base domain) is created, and not on the offline computation of the detail coefficients which create the resolution levels.

2.2 SIMPLIFICATION THROUGH MESH PARAMETRIZATION

Methods in this category construct discrete parametrizations in 2D of large surface patches serving as domain initializations, obtained through prior simplifications in 3D. These two-dimensional domains are resampled in various configurations, while aiming to cover the parameter domain without boundary discontinuities between adjacent domains. The resolution of the resulting point set is controlled through a choice in the number of subdivision steps. The subdivision schemes address both the rule of resampling, as well as the problem of finding an averaging vertex mask which will approximate or interpolate the original data points.

Multiresolution Analysis of Surfaces of Arbitrary Topology (MRASAT) — DeRose, Lounsbery and Warren, 1993

In [DeRose et al. 1993], the authors describe surface subdivision as a two-step procedure: a splitting step, generating the regular parameter 1-4 subtriangulation, and an averaging step, which is a 3D local weighted averaging of new points obtained
in the splitting step, using a neighborhood weight mask. This mask carries the description of the function being rendered, while the 2D subdivision rule carries the description of the interpolation ratio. Vertex functions are written as orthogonal decompositions into scaling functions and wavelets, with level $j$ representation being a linear combination of one level 0 scaling function (a “hat” function with support in the one-ring neighborhood of a vertex) and $j$ wavelet functions of levels $j-1, \ldots, 0$. Then, the vertex functions representing different multiresolution levels can be rendered at viewpoint-appropriate resolutions by truncating the wavelet terms corresponding to the finer levels. This yields the sequence of mesh instances from the coarsest, $M^0$, to more refined subdivided levels $M^{s-1}, M^s$ etc. For uniform and stationary schemes, the subdivision and detail information are stored as matrices of coefficients of the scaling and wavelet functions, recursively related. Using this framework, for down-sampling, pass/block information can be captured in a pair of filter matrices called analysis filters. However, this treatment can only be applied to surfaces with subdivision connectivity, i.e. that were obtained through subdivision, since the calculation of the analysis filters is possible only with the knowledge of the synthesis (construction) filters.

The averaging step (performed with 3D point information from prior steps and from the current splitting step) moves every 3D point in the current (refined, but not averaged) mesh instance into a new 3D position determined as a weighted combination of its 3D neighbors (identified in 2D). Depending on the weights, this results in various types of surface. Loop’s subdivision is reviewed and Loop’s weights are given as:

$$w(n) = n\beta(n) / (1 - \beta(n)),$$

where $\beta(n) = 2\left(\frac{3}{8} + \cos \frac{2n\pi}{4}\right)^2 - 1/4$, $n$ is the size of $u_i$’s one-ring neighborhood $N_i$, and $w(n)$ equal weights which, applied to points $u_j \in N_i$, determine where $u_i$ moves:

$$u_i = \sum_{u_j \in N_i} w(n)u_j$$

Loop’s weights rearrange subdivision vertices into a locally curved surface, as a particularization of which unit-valued weights produce piecewise linear surfaces.

DeRose et al. [1993] also discuss the preservation of barycentric coordinates through a series of nested spaces: these are a sequence of mesh subdivisions, starting from the coarsest, $M^0$, to more refined subdivided levels $M^{s-1}, M^s$ etc. If a point $\hat{x}$ from a coarser, subdivided, but not averaged level $\hat{M}^{s-1}$, is located in the $\triangle \hat{A}\hat{B}\hat{C} \in \hat{M}^s$ from the next refined level, also subdivided but not averaged, with barycentric coordinates
α, β, γ: \( \hat{x} = \alpha \hat{A} + \beta \hat{B} + \gamma \hat{C} \), then after the corresponding level of splitting of averaged \( M^{s-1} \) into \( M^s \), \( x \)'s new position on the averaged surface \( M^s \) will be related to the positions of averaged \( A, B, C \) by the same barycentric coordinates: \( x = \alpha A + \beta B + \gamma C \).

The averaging process, which essentially relaxes a point’s position in its neighborhood to achieve a regularly sampled parametric surface of degree \( m \), where \( m \) is the number of subdivision steps, will preserve barycentric coordinates by virtue of the affinity it creates between the successive spaces of subdivided vertices, as well as between the corresponding spaces before and after the averaging step. As long as all vertex insertions and relocations are linear at each step, the correspondence through barycentric coordinates can be preserved. Hoppe, DeRose et al. also address the over-smoothing effect of Loop surfaces by designing a different set of subdivision matrices for feature vertices, which is shown to better preserve sharpness.

However, this technique applies to providing detail to surfaces in a constructive approach, with the ability to withhold and store resolution-appropriate detail for fast rendering (in animation). It is not immediately applicable to arbitrary surfaces, which are represented by meshes obtained with data acquisition devices, and therefore do not present subdivision structure. In order to adapt this technique to arbitrary surfaces, they have to be first simplified and rebuilt with subdivision connectivity. This introduces the problems of finding subdivision domains, of finding a vertex averaging mask that fits the data, and, finally, of smoothness across domains.

Multiresolution Analysis of Arbitrary Meshes (MAAM) — Eck, DeRose and Duchamp, 1995

Eck et al. [1995] generalize multiresolution analysis to arbitrary surfaces. To convert an arbitrary surface (e.g. one obtained as range data) into a multiresolution surface, they create a base mesh consisting of very few triangles on which they perform subdivision, combined with a pairwise parametrization of the domains to ensure smoothness across domains. To obtain the triangular regions, the authors develop Voronoi regions, whose barycenters, connected through surface edge paths, produce a dual triangulation. For parametrization, they construct a harmonic map (angle, distance preserving) \( h : \bigcup F_i \subset \mathbb{R}^3 \rightarrow \bigcup T_i \subset \mathbb{R}^2 \), where \( F_i \) are the sparse curved triangular surface patches and \( T_i \) are the planar triangles they map onto. The interior of each triangular surface patch \( F_i \) is mapped onto the interior of the flat triangle \( T_i \) using a local harmonic map \( \rho_i : F_i \rightarrow T_i \). Each planar triangulation \( T_i \) is then resampled through 1-4 recursive subdivision (parametrically uniform resampling) combined with a triangle area-equalizing technique (geometrically uniform resampling). The geometrically uniform resampling computes a split coefficient for each edge, which aims at creating
equiareal triangles adjacent to that edge. Finally, for reconstruction, the resamples are located within $\rho = \bigcup \rho_i$.

Of particular interest to our application is the straightening of the edge paths obtained as shortest paths over the surface between barycenters of adjacent Voronoi regions, that create the initial surface partitioning. These edge paths are straightened through a harmonic parametrization of adjoining triangular regions, in which the new border of the two patches is resampled along the diagonal of the hinge map quadrilateral. The authors simply state that they “replace the edge by the image of the corresponding diagonal of the quadrilateral under the inverse harmonic map”. It is not clear where are the new diagonals resampled, nor how exactly a harmonic map that was discretized using vertex neighborhood information can be inverted and applied to resamples. We assume that the diagonals will be resampled at subdivision points during the subdivision step, and that the resulting points will be linearly interpolated in their (now complete and available) parameter triangles. This means that the patches must be processed in pairs at all times, so that information on triangles spanning two adjacent patches can be available for the reconstruction of each border subdivision point. Another possibility is that, knowing in advance the number of subdivisions, the straight border edges could be resampled at the appropriate regular intervals in advance. This way, in a single pass over the patch pairs, new 3D borders can be computed and can represent their own reconstructed image in a way similar to ours.

Another application of the idea of edge straightening in parameter space would be in feature detection. First using a relatively good segmentation algorithm, one could address the jagged shape of surface driven region boundaries by mapping, as discussed above, regions two by two, retracing the boundaries between corner points, and interpolating resamples along the straight lines in 3D. However, since on large data sets segmentation algorithms may result in segmentation of coplanar surfaces only by virtue of the number of regions being chosen to not exceed 500 points, several passes over the non-feature region boundaries could be be made, to ensure that boundaries dividing adjacent regions with mutual normal deviation below a certain threshold are eliminated.

**Multiresolution Adaptive Parameterization of Surfaces (MAPS) — Lee and Sweldens, 1998**

In [Lee et al., 1998], known in the literature as MAPS, the authors construct a sequence of multi-resolution meshes ranging from the original dense mesh to a very sparse base mesh — again, as in [DeRose et al., 1993] and [Eck et al., 1995], a 3D simplified mesh
which contains few large triangular faces connecting original points. The resamples are obtained through a modified version of the Loop subdivision algorithm applied to the base patches, which is shown to improve smoothness at the base domain boundaries. The base domain is not constructed using growing Voronoi tiles and the dual Delaunay triangulation, like in Eck et al. [1995], but by selective vertex removal. Vertices are scheduled for removal based on a metric which is inversely proportional with a linear combination of normalized area and curvature, thus penalizing small, flat one-ring neighborhoods. The metric is inversely proportional with a linear combination of normalized area and curvature:

\[ w(i) = \lambda \frac{\text{Area}(N_i)}{\max_{j \in M} \text{Area}(N_j)} + (1 - \lambda) \frac{\kappa(N_i)}{\max_{j \in M} \kappa(N_j)}, \]

where \( \lambda \) is a coefficient penalizing one or the other of the two components, as necessary. The gap left by each just-removed vertex is retriangulated using a constrained (fixed boundary) Delaunay triangulation of this neighborhood. The local one-ring neighborhoods or the vertices marked for removal are flattened using an invertible discrete conformal (angle preserving) mapping \( x_i \in \mathbb{R}^3 \rightarrow u_i \in \mathbb{R}^2 \):

\[
\begin{align*}
\mu_i(x_i) &= 0 \\
\mu_i(x_j \neq i) &= \|x_i x_j\| e^{i \theta_k} a,
\end{align*}
\]

where: \( \theta_k = \sum_{j=1}^{\left|N_i\right|} \angle(x_j x_i x_{j+1}) \) and \( a = \frac{2\pi}{\theta_k} \). After \( u_i \)'s computation, the gap left behind by the just removed vertex is retriangulated using a constrained (fix boundary) Delaunay triangulation. The remaining 3D vertices after all removals and retriangulations form the fixpoint base domain.

The next step parametrizes the vertices from each wave of removal by setting their image as the flat barycentric combination of the enclosing triangle resulted after the retriangulation. This means that the triangles after each stage of removal are a frame of reference for the vertices that were removed. Thus the vertices from each wave of removal (decomposition) are memorized through their flat barycentric position within the enclosing triangle resulting from the retriangulation. The fact that the (fewer) triangles after each stage of removal provide a frame of reference for the vertices that were removed corresponds to the setup of control nets for linear interpolations in reverse (the synthesis layers), having at step one the base domain vertices as control points, at step two the second-coarsest level vertices as control points, and so on. The detail coefficients of the resamples are related to the irregular removed points. Each original vertex will appear in the refining control net for some level of subdivision. The vertices for the next refined control net are computed through barycentric location with respect to the control points in the previous iteration. Thus MAPS is a remeshing
method in which unrolling the recursion of barycentric interpolation down to the base domain corresponds to finding an initial, sparse, control net followed by subdivision with custom control points. This generalizes, in effect, the computation for control points to arbitrary positions at each step, while still achieving a regular approximation of the surface through regular subdivision, thanks to a beautiful and ingenious algorithm. The multiresolution steps being essential for achieving a high compression rate with quick LOD retrieval, location of the subdivision point can be done with respect to a coarser triangular structure for lower LODs. Of course, if fast rendering is not an issue, only the base domain, subdivision scheme, and the original triangular structure are enough.

To achieve smoothness, if a 2D triangle (or, more generally, the local support stencil) straddles several adjacent 2D triangular domains, the reconstruction is computed relative to the enclosing triangle of the union of these domains. The construction of the base domains, whose shape is inherently triangular through the method of simplification in 2D, will result in straight edges through original points only after hinge map computation of linear borders and of their 3D image. This simple shape of the generating domains is key for the smooth reconstruction across domain borders, similar to Eck et al. [1995], while our approach generates jagged borders consisting of original points. In contemplating this way of surface approximation with subdivision over recursively parametrized domains, we believe that the size of our data sets would not support this implementation, which requires global operations: the triangular partitioning, the hinge map computations.

By comparison, our implementation is sequential. It is based on partitions which cannot be obtained through naïve sectioning of the data files, especially when they come from several scans whose registration resulted in a non-rasterized reorganization of vertex listing. Because these partitions need to match after processing, we adopted a method which sequentializes all phases of processing (parametrization, simplification, retriangulation, reconstruction). This requires fixpoint borders across partitions. The preservation of these borders introduces our jagged edge parametric domains. In experimenting with subdivision, we worked around the need for global knowledge of the regions, by introducing a border fragment map, which stores a border edge sequence and its straightened 2D version, updating with the same border value the border entry for both regions in which it appears.

Subdivision is also used with iterative geometric algorithms (minimizing approximation error), for constructing interpolating smooth meshes from arbitrary points, by assuming the input mesh or a subset of features as a control net for a known subdivision scheme (Loop, Catmull-Clark), and by iteratively updating the control vertices.
by a closest point computation, which retrofits points of the actual mesh in the control structure at every subdivision step. These methods are described in [Litke et al. 2001, Ivekovic and Trucco 2007, Cheng et al. 2004].

**Consistent Mesh Parameterizations (CMP) — Praun, Sweldens, Schröder, 2001**

In [Praun et al. 2001] (2001), the authors assume the preexistence of the base mesh domain. We interpret this as simply allowing the user to specify feature points from the original set, which are then connected by straight edges to form large triangular domains. Although the motivation of this paper is morphing (texture transfer), where a “meta-base-domain-mesh” is necessary to exist across different shapes, the idea of smoothing the base mesh boundaries to minimize discontinuities and distortions at the edges is relevant to our interests. The authors identify and address the problem of “tracing fair boundary curves” at the incidence of two reconstruction patches. They contend that the base domain construction as presented in [Lee et al. 1998] is not a desirable implementation in the context of texture transfer, since the coarsification of the original mesh through vertex removal in the parameter plane, departing from different models, would obviously result in different base domains. Hence the choice of user-specified feature points, which can be made to correspond in a visual way across the different models.

The parametrization function, again, maps the triangular boundary vertices onto themselves in a global parametrization, and the inner points with a 2D conformal image using Floater weights as in [Floater 1996]. Feature points are then connected through edge paths on the original mesh, following the locus of the minima of a vertex function that encodes only the “pull” exerted onto path vertices by their one-ring neighbors in the parameter domain. The goal is to minimize the influence of other feature vertices, in order to enforce a shortest path on the original mesh, since the function only considers, for each vertex, the local effect of vertex neighbors in the parameter space. This function is based on the $\lambda$ coefficients from the matrix row for the boundary vertex from the same matrix that computed the parametrization. The path will follow vertices popped from a local priority queue storing values of the vertex function in decreasing order, choosing, in effect, vertices from the one-ring neighborhood of minimum $\lambda$, i.e of maximum local curvature. As we now know, the conformal $\lambda$-s are proportional with tangents of angles formed by the vertex with its star vertices, and since tan is an increasing function on $(-\frac{\pi}{2}, \frac{\pi}{2})$, the smaller the $\lambda$, the smaller the face angles, the higher the elevation, and thus the greater the local curvature at that point.
Once the paths are traced on the original mesh, remeshing can be done in the parameter space using a 1-4 subdivision algorithm, with the resamples inheriting an interpolated version of the surrounding vertices from the original mesh.

We took away from this paper the fact that the best parameter domain borders do have to pass through original points. On the other hand, the bookkeeping involved in making sure that the mesh shortest paths do not intersect and are smooth seems complex, and so the simplicity of the boundary smoothing as a method, through minimization of a $\lambda$-based vertex function, comes at the cost of testing for self-intersections and for smoothness of the traced boundaries. In addition, feature points are user-specified, which would be impossible for large meshes, which have to deal with feature detection on a different scale, possibly iteratively, as well. The algorithm in [Praun et al. 2001] does not include the phase of vertex selection, and the results tables only time separately the phases of base mesh triangulation, parametrization, and remeshing.

2.3 DISCRETE PARAMETRIZATION METHODS

A separate presentation of discrete parametrization methods is useful, since many interesting developments in parametrization are not necessarily associated with mesh simplification. The problem arises mostly in texture mapping (texture-to-model) and in texture transfer (model-to-model). The importance of a good correspondence between a 2D domain and the 3D reality of the data set is evident.

Piecewise Surface Flattening for Non-Distorted Texture Mapping (PSF) — Bennis, 1991

In [Bennis et al. 1991], we find a review of the previous general approach in texturing: the surface is originally sampled at a fine grid (for instance, scanned). This grid provides isoparametric curves (curves obtained by holding one of the parameters fixed, and varying the other). Quad regions comprised between parallel isoparametrics are chosen, for example four "squares" around a vertex. The quad regions are flattened to represent the isoparametrics through the vertex as straight lines. The size of the quad is chosen so distortion at edges due to flattening is below a threshold. The flat grid is triangulated. Texture mapping is done by sampling texture points inside this grid and reconstructing them using barycentric interpolation.

Against this background on texturing, Bennis maps isoparametrics onto plane curves that will preserve geodesic curvature, as opposed to sampling the surface at grid points without regard to geometry. He reminds the Frenet frame and the expression of its components (normal, tangent, binormal) in terms of each other and of the main
2.3 Discrete Parametrization Methods

curvature $\kappa$ and torsion $\tau$ at each point $s$ on the surface, viewing these surface attributes as the angular velocities of the normal and the osculating planes as $s$ moves along the curve $c_S$ on the surface $S$.

Observing the fact that the geodesic curvature $\kappa_S(x)$ of a three-dimensional curve $c_S$ on a surface $S$ at a point $x$ is equal to the curvature $\kappa_P(X)$ at $X$ of the planar curve $c_P$ obtained by projecting $c_S$ onto the tangent plane through $x$, Bennis exploits this property to construct a parametrization. For a triangle of the surface $\triangle M_1 M_2 M_3$, he projects segment $|M_1 M_2|$ onto the tangent plane at $M_1$. The projection is $P_1 P_2$. Onto this plane, he constructs $P_3$ which preserves segment length $\|P_2 P_3\| = \|M_2 M_3\|$ and angle $\theta_1 \equiv \angle(M_1 M_2 M_3) = \theta_2 \equiv \angle(P_1 P_2 P_3)$. This defines the angle preserver operator $P(M_3) = P_3$. This procedure unfolds (develops) isocurve points from $S$ onto $P$.

Since this is a local operation affecting one angle, and the application of the angle preserver may not result in angles summing up to $360^\circ$ around a vertex, the issue next becomes how to combine this with the other isocurves passing through $x$. For this, four edges incident on $x$ produce four projections onto the tangent plane to $S$ at $x$. Three of them can be projected with the operator above, and the fourth one inherits the difference. The aggregate of the unfolding process along all isocurves produces a quadrangulated flat surface. Bennis illustrates this on developable surfaces (cone, cylinder), as well as on the sphere. To account for the unfairly represented fourth angle, a relaxation procedure repositions each point as the average of the 12 points consisting of its 4 cross neighbors and each of their cross neighbors (3 more for each minus 4 repeats making 8): $4 + 8 = 12$. The relaxation is iterated until no significant position changes occur globally. On this feature net, new samples are then placed in a grid structure and reconstructed with neighborhood information.

To conclude the review of this important contribution, we note that [Bennis et al. 1991] introduced the idea of a conformal mapping operator per point applied to a developable portion of, or to the entire mesh. It then applied relaxation as a local neighborhood averaging process[1]. The idea of the locally computed conformal mapping operator was later refined in many other works, among which, notably [Pinkall and Polthier 1993], [Eck et al. 1995], [Floater 1996], [Lee et al. 1998], [Sheffer and de Sturler 2000], [Sheffer and de Sturler 2002], [Floater 2003]. We have already presented in Chapter B a more in depth study of some of these methods.

---

1 Since in all online copies of [Bennis et al. 1991] the pictures are missing, we couldn’t evaluate the problem of the feature boundaries even visually, where the method was applied to portions, as opposed to the entire mesh.
Figure 1.: Neighborhood around point \( p \) and sliding triangle \( \triangle p, p_k, p_{k+1} \).

*Parametrization and Smooth Approximation of Surface Triangulations (PSA) — Floater, 1997*

In Floater [1996], the author introduces the idea of combining the quality of the mapping (isometric, conformal, equiareal etc.) with the one-to-one weighted mapping. The latter is presented as a generalization of barycentric mapping to a mapping based on any convex combination. Floater discusses various types of parametrization, starting from the purely barycentric combination of points interior to a fixed boundary — originally presented by Tutte for drawing a straight line graph through a given configuration of curve-connected points. He proves this transformation to be equivalent to a uniform parametrization, which minimizes the sum of the squares of edge lengths.

To achieve shape-preserving parametrization, he explores the combination of the length-preserving quality with the barycentric combination. Orthogonal projections onto either the least-squares plane or the plane whose normal is an average of the triangle normals in \( 3D \) are considered unstable. Instead, he uses a method based on Welch and Witkin [1994], preserving arc length in each radial direction, in which parameter points \( p \) are chosen in relationship with their one-ring neighbors \( p_k \in N_p \) such that:

\[
\| p_k - p \| = \| x_k - x \| \quad \text{and} \quad m(\angle p_k, p, p_{k+1}) = 2\pi \frac{m(\angle x_k, x, x_{k+1})}{\theta_i},
\]

where \( x \) is the image of \( p \), and \( x_k \) are in the neighborhood \( N_i \). To simplify notations, we refer to the kernel of a 2D neighborhood as \( i \), and to any of its \( k \) neighbors as \( j \). Floater expresses a vertex in its one-ring neighborhood as a barycentric combination of a sliding triangle through the neighborhood points, defined, for every \( j \in N_i \), with vertices \( j \) and the endpoints of edge \( e \) intersected by ray \( |j,i| \). There are \( j \) such sliding triangles. A barycentric relationship with the all neighborhood points fol-
2.3 Discrete Parametrization Methods

Floater further shows that such a parametrization is an affine mapping from $\mathbb{R}^3(x,y,z)$ to $\mathbb{R}^3(u,v,0)$, thus preserving linear combinations. He recommends, for the mapping of the surface boundary:

- unit square, if the surface is a tensor-product spline approximation;
- unit circle, if the approximation is in the form of triangular patches;
- chord length, if the approximation is shape preserving.

Finally, as numerical considerations, Floater warns that LU decomposition only works for $n \times n$ matrices with $n \leq 500$. We encountered the same instability with our matrix library, Lapack++. Beyond that size, he suggests iterative solutions, such as Bi-CGSTAB, a variant of the conjugate gradient method for non-symmetric matrices. To evaluate the parametrization, Floater uses three $C^1$ bi-cubic coordinate functions $x(u,v), y(u,v), z(u,v)$, defining a cubic surface (the salt dome). This surface $S(u,v)$ is sampled on a square grid and, alternatively, interpolated by a $C^2$ cubic tensor-product spline using the parameter values (corresponding to the 3D grid samples) produced by the neighborhood barycentric method, generating $S'(u',v')$. This resembles our own error calculation method.

In a different retriangulation of the parameter space, using a Delaunay triangulation, a good, smooth reconstruction was achieved by using a tensor-product surface approximation.

MIPS: An Efficient Global Parametrization Method (MIPS) — Hormann and Greiner, 1999

Hormann and Greiner [2000] construct a composite of atomic linear maps $\Psi \circ g$, each of which map a 3D triangle $T \subset \mathbb{R}^3$ onto a 2D triangle $t \subset \mathbb{R}^2$. The $g$ component of these maps corresponds to only the 2D to 2D linear transformations $A$, describing the stretch of triangle $T$ to its parameter image $t$ (thus $\Psi$ describes the parametric coordinate functions for $T$):

$$g(x) = Ax + b = \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} x + \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

The deformation induced by this transformation is characterized by the diagonal matrix resulting from the singular value decomposition:

$$\Lambda = U^T \Sigma V = U^T \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} V$$
2.3 DISCRETE PARAMETRIZATION METHODS

![Diagram of atomic map](image)

Figure 2.: Atomic map $g$ from 3D triangle $T$ to 2D triangle $t$ as in [Hormann and Greiner](2000).

First as the 2-norm of matrix $A$, $\kappa_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2 = \sigma_1 \sigma_2$, then as the Frobenius norm:

$$
\kappa_F(A) = \|A\|_F \cdot \|A^{-1}\|_F = \sqrt{\sigma_1^2 + \sigma_2^2} \sqrt{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} = \kappa_2(A) + \frac{1}{\kappa_2(A)} = \frac{\text{trace}(A^T A)}{\det A}
$$

The Frobenius norm associated with each atomic linear map is computed as:

$$
\kappa_F(g) = \frac{\cot \alpha |a|^2 + \cot \beta |b|^2 + \cot \gamma |c|^2}{\det \nabla \Psi},
$$

where $\alpha, \beta, \gamma$ are angles in the 3D triangle $T$, $a, b, c$ are sides in the image triangle $t$, and $\Psi$ is the direct coordinate function $\Psi : \Omega \subset \mathbb{R}^2 \rightarrow S \subset \mathbb{R}^3$, such that $g$ is the composition $g = (\phi^{-1} \circ \Psi)(x, y, z) = \Psi(\phi^{-1}(x, y, z)) = \Psi(u(x, y, z), v(x, y, z))$. Thus $\Psi$ is the 2D to 2D coordinate function expressing $t(u, v)$. All quantities but $a, b, c$ and $\Psi$ are known. The parameter points $A, B, C$ forming triangles $t$ are found by minimizing the functional $\kappa_F = \sum_j \kappa_F(g_j)$. The main advantage of this method is that “there is no more need to fix the parameter values of the boundary points in advance. Instead, the boundary of the parametrization will develop most naturally in such a way that the deformation energy $\kappa$ is minimized” ([Hormann and Greiner](2000), p.158).
In Sheffer and de Sturler [2000] the authors implement a new parametrization metric. Since they seek a conformal (angle preserving) parametrization, the new 2D point positions are chosen to minimize, over the entire mesh, the sum — over a triangular face — of the three square differences between 2D angles and their 3D counterparts. This minimization is applied with some constraints: for instance, once a traversal order of the original mesh triangles is established, and thus the angles of each triangle have positive values, the 2D angles are valid only if they are also positive, in order to maintain the same (outward or inward) face orientation. Other constraints are enforced as well: the sum of angles in a 2D triangle is $\pi$, the sum of angles around a 2D vertex is $2\pi$, the length of the first incident edge coincides with the length of the last incident edge to finish the neighborhood traversal cleanly. All these validity constraints are aggregated into an objective function, whose minimization is solved over the number of mesh vertices (or to the restriction of interior nodes where necessary), by Newton’s method.

This method generates a few boundary self-intersections. Their cause being unknown, they are handled post-processing. Like our method, the authors compute the reconstruction by barycentric coordinates. However, they use a third party remeshing triangulation in the plane. In comparison, our method, by using fix-point border projections in the least squares plane, will generate no self-intersections. We also use, just for illustration, a third party Delaunay triangulation implementation, but we found confidence in using our own triangulation method designed for the border gap, in our implementation of subdivision, which needs a coarse triangulation to create the base domains.

In Sheffer and de Sturler [2002], the same authors create a conformal map of $T_1 : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ through ABF (angle-based flattening), an earlier paper of theirs (Sheffer and de Sturler [2000]). In $\mathbb{R}^2$, they overlay a grid resample point set, just like our method. This resample is then smoothed by another map $T_2 : \mathbb{R}^3 \rightarrow \mathbb{R}^2$, which is meant to reduce length distortions in reconstruction. In parameter space, they compute a bounding box grid and its triangulation, $G_1$. They then smooth this triangulation through the sizing function $T_2$, which replaces $G_1$ with $G_2$, to minimize linear distortion in reconstruction.

The sizing function is defined for each vertex $v_i$ from the 2D grid, computes all the distortion ratios $\langle v_i, v_j \rangle \subset \mathbb{R}^2$ to $\langle v_i, v_j \rangle \subset \mathbb{R}^3$ and averages them, moving $v_i$ in its new 2D location $\tilde{S}(v_i)$. The smoothed 2D points are located within parent triangles.
through barycentric coordinates $\alpha, \beta, \gamma$. The reconstruction is then $\hat{S}(v_i) = \alpha \tilde{S}(v_i) + \beta \tilde{S}(v_i) + \gamma \tilde{S}(v_i)$.

2.4 PIPELINE SYSTEMS USING B-SPLINE SURFACE FITTING

Fitting Smooth Surfaces to Dense Polygon Meshes (FSSDPM) — Krishnamurthy and Levoy, 2000

Krishnamurthy and Levoy [1996] present a pipeline reconstruction system, also based on segmentation and gridding, just like ours. The pipeline steps are:

- an interactive boundary curve painting phase, similar in functionality, but not implementation, to our segmentation phase. Theirs allows the user to manually specify boundary points, which are then connected through a parametric straight line; 3D point resamples, a.k.a. “face points”, are then computed at uniform spacing on the line intersections of the surface with the plane determined by the parametric straight line, and perpendicular onto the parameter plane. This piecewise linear face-point curve is then modelled using a B-spline curve. Points sampled at regular intervals on the B-spline curve are then projected onto the surface to obtain a smoother face-point curve;

- a gridded resampling of each quad mesh patch defined by two boundary pairs of face-point curves, again similar in functionality, but not implementation, to our resampling phase. Theirs is performed in 3D over the B-spline approximation curves computed from the iso-curves from the previous phase;

- a surface fitting method which finds a B-spline approximation surface to the grid points, similar to the Bernstein least squares approximation method we investigated briefly in this thesis, which computes “control points” from the data, then outputs grid resamples generated from these control points. However, in their implementation, as mentioned before, a square grid subdivision is first applied to the initial quad polygons, generating grid iso-curves and grid points on the 3D surface. The B-spline fitting is applied to these 3D resamples (as opposed to real data), with the result that the corresponding 3D B-spline surface points can be associated one-to-one with the grid points to generate displacement maps.

Interestingly, the initial iso-curves are not computed as 3D images of a 2D gridding, but as 3D curves connecting feature points, using Dijkstra’s shortest path algorithm. This $O(n \log n)$ vertex search phase is restricted only to patch vertices, and thus
deemed to not significantly add to the complexity of the overall algorithm. The boundary iso-curves obtained as shortest paths, connecting user selected feature points, define quad patches. Curve approximations for each patch are obtained through B-spline curve fitting, with regularly sampled points along the boundaries, the “face points”, defining the grid spacing. The resulting iso-curves form a “spring mesh” subject to a later step of relaxation, and thus are at this point only a first approximation to the final quad mesh.

A second approximation is obtained by relaxing the position of each mesh point along the direction of, and proportional to, the resultant of the four elastic force vectors from the point subject to relocation, to its four neighboring quad mesh points. The similarity to the relaxation of springs is suggested by the expression of the elastic force for a spring, which is $F = \frac{k \cdot x^2}{2}$, where $k$ is the elastic constant (and can be ignored in these calculations, since all the “springs” are supposed to have the same $k$), and $x$ is the elongation of the spring (its position relative to the equilibrium state). The relaxed iso-curves are resampled to achieve a regular point distribution along both directions, $u$ and $v$.

The final approximation is obtained through a B-spline surface fitting to the grid points obtained in the previous phase. This is in contrast to most approaches, which use surface fitting directly to original data points.

Subsequent refinement operations attempt to reduce the “wiggling” of the relaxed elastic iso-curves by minimizing a displacement function, which measures, for each vertex, the displacement between the spline surface and the resampled parametrized spring mesh surface, i.e. the displacement between the input and the output of the preceding phase. The continuity across patches is ensured by requiring the same resampling resolution and the same number of control points at the shared boundaries.


Greiner and Hormann [1997] also offer a whole pipeline reconstruction system, similarly based on tensor product B-spline surface fitting to data points, with a parametrization based on the minimization of the elastic energy of edges from the original surface. The surface edges, similar to the approach of Krishnamurthy and Levoy [1996], constitute networks of springs whose endpoints, subject to the elastic energy minimization, will relocate in positions of a global energy minimum for the whole network, thus determining parameter positions $(u, v) \in \mathbb{R}^2$. In addition to the surface approximation problem, for which a minimization between the B-spline approximating surface and
the original data points is imposed, the authors also address the fairing problem, for which they specify a fairing functional which adds a fairing term to the minimization:

$$J_\omega = J_0 + \omega J_{\text{fair}},$$

where:

- $J_\omega$ — the term whose minimization reflects a good fit of the B-spline approximating surface:
  $$J_0(F) = \sum_i \|P_i - F(Q_i)\|,$$

  with $P_i$ the data points, $Q_i$ the parameter points, and $F(Q_i)$ the B-spline approximation from parameter points;

- $J_{\text{fair}}$ — the fairing functional, defined as:
  $$J_{\text{fair}} \equiv J_S \overset{\text{def.}}{=} \int_\Omega \text{trace}(\text{Hess}_S^2(F)) \, d\omega_S,$$

  with $S \subset \mathbb{R}^3$ being the 3D image of the grid points lying within the triangulation of the parameter values $Q_i$ through linear interpolation.

**Piecewise Surface Reconstruction from Range Data — Gene Yu, 2010**

We wanted to give special attention to a recent thesis from the Graduate Center, whose focus on large scene reconstruction makes this contribution especially relevant.

**Surface Smoothing and Discrete Curvature Computation**

In his thesis [Yu 2010](#), he classifies the surface locally into 8 types, depending on mean curvature $H$ and Gaussian curvature $K$, which define an $HK$-sign map. The input surface is classified using this criterion with local curvature values per vertex computed discretely from the one-ring neighborhood. Because the stencil is small (of the order of magnitude of the range data density), it is sensitive to noise: curvature computed this way is of the same order of magnitude as the noise itself. The author uses an anisotropic smoothing operator, which fits a polynomial to the points in the neighborhood of each point $x_i$, smoothed with a weight coefficient, derived with neighborhood information:

$$f(x_i) = \min \left\{ \sum_{j \in N_i} w_j (\|x_i - x_j\|) \cdot \|g^T(x_j) c(x_i) - f(x_j)\| \right\},$$

37
where \( x_i \) is the point whose neighborhood \( N_i \) is fit against the polynomial \( g^T(x_j)c(x_i) \), with \( g(x) = (x^n \ x^{n-1} \ldots \ x \ 1)^T \) the polynomial basis vector and \( c(x) \) the vector of coefficients to be determined for each \( x_i \). The range of the fit functions across the mesh is the new surface, now smoothed and better equipped for providing discrete curvature information. The local surface approximations will also serve later, during the phase of surface fitting over larger surface components. For this purpose local neighborhoods are stored in a fast-search \( kd-tree \) data structure.

**Mesh Segmentation**

**Region Growing** Gene Yu performs segmentation on one scan at a time, prior to registration. This reduces the data set subject to segmentation, which is very similar to our approach. We perform segmentation scan by scan, then, due to the fixborder reconstruction, we can reassemble the reconstructed scans seamlessly by having triangle indices from multiple triangle files pointing to a unique point file. In his presentation of a segmentation algorithm, Gene Yu first introduces a number of popular algorithms, including our choice of region growing. He observes that the local distance measures employed by these algorithms do not scale globally. We disagree with the application of this characterization to our method, since our distance measures are global — for the triangles in the “zone”, errors are computed against a number of region proxies, each situated at the center-of-mass of its region, possibly far away across the surface from the triangle itself and from one another — and are compared globally. In contrast, the implementation shown on p. 51 is, indeed, a threshold-based growing process, rather than a global error minimization process.

**Normalized Cut** Another method, characterized this time as global, is the normalized cut, but is described as having another drawback, as it involves matrix computations scaled to data sets too large for the available solvers. Digression: we encountered the same problem in our parametrization algorithm, which forced us to impose size limits to our regions, at the expense of interfering with the efficiency of the algorithm. In the end, we used regions more for the benefit of arbitrary data segmentation in sizable partitions, then for the benefit of separating the surface into components of similar curvature. This being said, even though the more numerous regions result in smaller region sizes, the distance criterion still applies accurately. Returning to the methods examined by Gene Yu, he describes the normalized cut as a global method using, essentially, the same principle as traditional region growing, but in reverse: a global criterion of dissimilarity is applied to cut the scene, organized as a graph, into
dissimilar components, until the the cuts no longer satisfy the separation criterion. The same criticism about the criterion not being global enough is reiterated on p. 61. Thus, Gene Yu ends his exposition of segmentation methods with the 3D Hough transform, which “transfers points relations to a global histogram space in which the peaks correspond to the optimal parameter values” (Yu [2010], p.50) describing the candidate fit parametric functions for each component.

3D Hough Transform  We see a certain degree of similarity between clustering surface components as fit functions generating peaks in a Hough transform space, versus as collections of triangles of minimum errors in a global error space, our GPQ. But there is a main difference — we do not assign function descriptions to our regions, but simply state that they are similar in virtue of the distance error criterion applied. The function fit is, perhaps, a better approach for large scenes, and, in any case, creating this output in segmentation is one step ahead in the pipeline. On the other hand, the method is focused on planes. Extracting the best fit function, a.k.a. model selection, is an iterative search that the author is improving by creating a fast version of the segmentation method, based on RANSAC sampling.

In the end, for segmentation, Gene Yu settles on a combination of normalized cut (removing edges that satisfy a separation criterion) combined with fitting planes. To speed up the process of plane fitting (model selection), which runs in up to two hours for scans containing $10^6$ points, he uses a sampling algorithm (RANSAC) that structures the scan into an octree (a nested sequence of parallelepipeds), which drastically reduces the number of points selected to drive the primitive fitting process (to the number of coefficients needed to describe that primitive by its implicit equation, for instance three points for a plane). The output of the segmentation process is a collection of connected coplanar components.

Registration  For registration (across different scans, now each separated into components), he builds a scan pairwise graph having a node for each component, and an edge for each component intersection. The graph is initialized as bipartite, with all components from each scan connecting to all components from the other scan. Connections are removed in three iterations: an axis-aligned bounding box intersection test, a tight bounding box test, and a parameter space boolean test. New scans are added one by one. The parameter spaces are chosen as projection spaces consisting of geometric primitives: planes, cylinders and cones. The planes resulted from segmentation are projected onto the geometric primitive onto which the projection does not produce foldovers.
**2.4 PIPELINE SYSTEMS USING B-SPINE SURFACE FITTING**

**RECONSTRUCTION** For reconstruction, he uses the parameter points to fit polynomials to the surface components, now in a registered scan set. The components too complex to fit a polynomial function are approximated as a B-spline surface. The polynomial fitting is done using implicit polynomials: “we seek a function of two variables $f(x, y)$ that takes the value $z_c$ at $(x_c, y_c)$” ([Yu](#), p. 89). The choice of polynomial coefficients is the solution to a matrix system in which the coefficients are, for each row, the linearized entries of the outer product $u \otimes v^T$ (from the projection points), and the free terms are the $z$ coordinates of the corresponding 3D point. The B-spline approximation follows the scheme used for quadrangular patches fitting and joining quadratic surfaces.

The error whose minimization is used to make the choice of polynomial coefficients that result from the matrix system described above — when used with competing polynomial degrees $d$ for each patch — encodes two elements: one, the quality, and two, the complexity of the fit function. The quality component is implemented as the difference in height obtained through the fit function versus the heights of points from a training data set. The efficiency component of the error is evaluated as the bit-rate ratio measured on the compressed versus uncompressed data files consisting of points and their *residual heights*. The residual heights are defined as the difference in height obtained through using a higher degree polynomial (requiring more coefficients/more support around the vertex) versus using a plane. Through the amount of redundancy eliminated during compression, the bit-rate of the compressed data file will actually measure the aggregate deviation of the $z$ coordinates computed through the higher polynomial fit against the $z$ coordinates computed from the fit with the plane of least squares around the data. The idea of this comparison reinforces the concept of favoring approximation versus interpolation, when the approximation describes more faithfully the general shape, even though it does not include original data points.

**TRIANGULATION** For remeshing, the polynomial fit functions, one per scene component, are sampled on the intersections between components (domain boundaries) and on the interior, on the projection spaces for each component. The projections described in Section 2.4 are triangulated using an off-the-shelf Delaunay triangulation, and the triangulation is lifted into 3D by adding the $z$ coordinate to the $x, y$ pairs as explained above, while preserving the computed triangular connectivity during the reconstruction.

In conclusion, the polynomial fitting approach seems to be a good way to treat large scenes, which contain man-made shapes over large mesh areas, and redundancy
can be eliminated through using as few points as possible to generate the surface. Reconstruction is then as elegant as computing function values at very sparse samples.

### Simplification Methods: Comparison Table

<table>
<thead>
<tr>
<th>Method</th>
<th>Shape</th>
<th>IN Triangles</th>
<th>OUT Triangles</th>
<th>Time</th>
<th>Phase</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTPS</td>
<td>Connoly surface</td>
<td>3,675</td>
<td>201</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ADS</td>
<td>tube</td>
<td>75,264</td>
<td>3,617</td>
<td>–</td>
<td>SSR (SDV)</td>
<td>IRIS MIPS R 3000</td>
</tr>
<tr>
<td>DTM</td>
<td>skull</td>
<td>569,000</td>
<td>57,000</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MRASAT</td>
<td>Spock head</td>
<td>32,768</td>
<td>4,054</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MO</td>
<td>pipe fitting</td>
<td>3,832</td>
<td>916</td>
<td>9.9m</td>
<td>10.2m</td>
<td>PPL –</td>
</tr>
<tr>
<td>PSSR</td>
<td>fandisk</td>
<td>16,475</td>
<td>184</td>
<td>2.2h</td>
<td>SSR (SDV)</td>
<td>–</td>
</tr>
<tr>
<td>MAAM</td>
<td>3-hole torus bunny</td>
<td>17,920</td>
<td>366</td>
<td>4.6m</td>
<td>33.5m</td>
<td>SGI Onyx Reality Engine 2</td>
</tr>
<tr>
<td>SE</td>
<td>bunny phone</td>
<td>69,473</td>
<td>10,793</td>
<td>11m</td>
<td>38m</td>
<td>PPL HP 735/125 80MB</td>
</tr>
<tr>
<td>QEM</td>
<td>bunny cow</td>
<td>60,451</td>
<td>10</td>
<td>15.3s</td>
<td>0.91s</td>
<td>ISM (SMP) SGI Indigo2 195MHz R10,000</td>
</tr>
<tr>
<td>MAPS</td>
<td>fandisk horse</td>
<td>12,946</td>
<td>168</td>
<td>23s</td>
<td>163s</td>
<td>HBL (SDV) 200MHz Pentium Pro</td>
</tr>
<tr>
<td>MSSR</td>
<td>car</td>
<td>1,114</td>
<td>378</td>
<td>–</td>
<td>–</td>
<td>PPL –</td>
</tr>
<tr>
<td>ROP</td>
<td>head</td>
<td>21,680</td>
<td>256</td>
<td>13m34s</td>
<td>SSR (SDV)</td>
<td>195MHz SG O2 R 10,000</td>
</tr>
<tr>
<td>NM</td>
<td>skull horse</td>
<td>20,002</td>
<td>112</td>
<td>2.5m</td>
<td>6.8m</td>
<td>SSR (SDV) –</td>
</tr>
<tr>
<td>CMP</td>
<td>horse face</td>
<td>100,000</td>
<td>43,000</td>
<td>7m37s</td>
<td>10m08</td>
<td>SSR 900MHz Pentium III</td>
</tr>
<tr>
<td>ISOR</td>
<td>cat</td>
<td>50,000</td>
<td>20,629</td>
<td>2m</td>
<td>–</td>
<td>REC Pentium IV 2.4GHz, 512MB</td>
</tr>
</tbody>
</table>

Table 1: Performance of various mesh simplification methods.
The results for all subdivision methods include the remeshing and upsampling, since these methods operate on an initial bare bones mesh segmentation which is not in itself the purpose of the mesh processing, but rather the initial triangular base domain on which subdivision is performed. Table 1 doesn’t show, therefore, the final mesh counts for these examples.

Phase Acronyms

To comply with the required printed size within the page, Table 1 uses acronyms for two types of entries: paper identifiers and computational phase accounted for in the computation times shown in the original papers. Regarding the pipeline phase timed in various methods, we included this particular entry because our own system is sequential, and each subsequent phase includes all the previous component phases. It is thus a pipeline system. In contrast, many of the papers present only parts of the pipeline, which are the ones accounted for in their own computation time tables. Below is a legend for the phase acronyms:

SSR — simplification-subdivision-reconstruction
SDV — subdivision only
PPL — pipeline
ISM — initialization-simplification
SMP — simplification only
HBL — hierarchy building (for subdivision)
REC — reconstruction only

Paper Identifier Acronyms

The paper identifier acronyms used are shown in Table 2 in which the full titles are printed. The acronyms are largely constructed using the capital letters of the key words in each paper title.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Title</th>
<th>Citation Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTPS</td>
<td>Re-Tiling Polygonal Surfaces</td>
<td>Turk [1992]</td>
</tr>
<tr>
<td>ADS</td>
<td>Simplification of Objects Rendered by Polygonal Approximations</td>
<td>Dehaemer and Zyda [1991]</td>
</tr>
<tr>
<td>DTM</td>
<td>Decimation of Triangle Meshes</td>
<td>Schröder et al. [1992]</td>
</tr>
<tr>
<td>MRASAT</td>
<td>Multiresolution Analysis for Surfaces of Arbitrary Topological Type</td>
<td>DeRose et al. [1993]</td>
</tr>
<tr>
<td>MO</td>
<td>Mesh Optimization</td>
<td>Hoppe et al. [1993]</td>
</tr>
<tr>
<td>PSSR</td>
<td>Piecewise Smooth Surface Reconstruction</td>
<td>DeRose et al. [1994]</td>
</tr>
<tr>
<td>MAAM</td>
<td>Multiresolution Analysis of Arbitrary Meshes</td>
<td>Eck et al. [1995]</td>
</tr>
<tr>
<td>SE QEM</td>
<td>Surface Simplification Using Quadric Error Metrics</td>
<td>Garland and Heckbert [1997]</td>
</tr>
<tr>
<td>MAPS</td>
<td>MAPS: Multiresolution Adaptive Parameterization of Meshes</td>
<td>Lee et al. [1998]</td>
</tr>
<tr>
<td>MSSR</td>
<td>Mesh Simplification with Smooth Surface Reconstruction</td>
<td>Volpin et al. [1998]</td>
</tr>
<tr>
<td>ROP</td>
<td>Remeshing Triangulated Surfaces with Optimal Parameterizations</td>
<td>Hormann et al. [2000]</td>
</tr>
<tr>
<td>NM</td>
<td>Normal Meshes</td>
<td>Guskov et al. [2000]</td>
</tr>
<tr>
<td>CMP</td>
<td>Consistent Mesh Parameterizations</td>
<td>Praun et al. [2001]</td>
</tr>
<tr>
<td>ISOR</td>
<td>Isotropic Remeshing of Surfaces: A Local Parameterization Approach</td>
<td>Surazhsky et al. [2003]</td>
</tr>
<tr>
<td>FSSDPM</td>
<td>Fitting Smooth Surfaces to Dense Polygon Meshes</td>
<td>Krishnamurthy and Levoy [1996]</td>
</tr>
<tr>
<td>PSRFRD</td>
<td>Piecewise Surface Reconstruction from Range Data</td>
<td>Yu [2010]</td>
</tr>
<tr>
<td>IADHTB</td>
<td>Interpolating [...] 3D-data with Hierarchical Tensor Product B-Splines</td>
<td>Greiner and Hormann [1997]</td>
</tr>
</tbody>
</table>

Table 2. Acronyms used in Table 1
3

SEGMENTATION

3.1 FREE-FORM BOUNDARY SEGMENTATION

The method we implement uses a normal-based clustering algorithm to segment the data into regions, which constitute the domains over which a discrete harmonic parametrization is later computed. The error to the cluster proxy from [Cohen-Steiner et al.] [2004], used as the criterion for our segmentation, was tested in different variations, which use face normals and triangle areas, to penalize large normal deviations to the cluster proxy, as well as to prevent large triangles at the extremities of a region from significantly altering the region’s configuration “at the last moment”. Due to the particularity of scanner data sets, which present regular initial sampling and thus roughly equal triangle areas, triangle area may not be a factor in the shape of the clusters during the first pass, but if the method were applied several times, the triangles resulting from our grid resampling method may differ in size from the border triangles, thus triangle area becomes important. The points of the boundaries of the resulting regions are ordered sequentially, and their positions remain fixed.

We categorize this type of segmentation as surface-driven segmentation, since it depends on its overall shape, and the borders do not pass through preordained points. In contrast, we encountered many methods performing surface partitioning through shortest paths along the surface, connecting centroids of Voronoi regions or otherwise user-selected feature points, which we categorize as curve-drawing segmentation. These methods appear especially in subdivision, where it is important to obtain surface patches of triangular shape (triangular base domains). Inspired by these methods, but based on our own segmentation, we also implemented triangular base-domain generation, through region border simplification and simplified border triangulation in the parameter domain, described in Section A.3.

This algorithm generates free, irregular borders, which correspond to large variations of face normals, such as at sharp edges and discontinuities. In scanned surfaces from buildings, region boundaries are generated at the edges of large features, such as doors, windows, architectural features and scan edges. In scanned surfaces from curvy objects (the Stanford bunny etc.), regions tend to form around low curvature areas, generating borders along lines of relatively higher curvature. The number of regions in this algorithm does not change dynamically with the demand for regions, i.e.
a prespecified number of regions will divide a roughly coplanar area, even when this is undesirable or unnecessary. However, in the context of our need to process regions in a linear solver based on an \( n \times n \) square matrix where \( n \) is the number of region points, it is useful to control the number of regions through this rigid specification of number of regions, in order to limit the size of the matrix to the capabilities of our matrix computation software (LaPACK++).

3.2 OUR CONTRIBUTION

This algorithm is inspired by the work in Cohen-Steiner et al. [2004] and Sander [2003]. We reimplemented the algorithm as described, with the possible added value of the simultaneous GPQ computation and region growing implementation, which we perform in one atomic step, to improve code performance, because it requires only one pass through the scene triangle superset, while keeping the size of the queue low at all times. Another contribution was the introduction of a new error formulation, which we describe in Section 3.3.1.

3.3 THE SEGMENTATION ALGORITHM

The segmentation is based on an iterative algorithm which starts at each step with a set of region triangle seeds around which regions will form through additions of adjacent triangles. The seeds can be also specified as points, in which case a preliminary parent triangle finding is necessary. By default, our algorithm selects random triangle seeds in the first iteration. All subsequent iterations recalculate the seeds, which are now named “proxies”, since they “represent” the region forming around them in the next step, and are the reference against which an error will be calculated for each triangle.

3.3.1 Proxy Calculation

For every region computed in an earlier iteration, a proxy will be chosen to provide the error calculation reference for a new clustering step. The computation of a new proxy consists of three steps: first, a center-of-mass proxy is generated from the normals and areas of all triangles currently in the region, as the vector:

\[
\overrightarrow{N_i} = \frac{\sum_{j \in R_i} \overrightarrow{n_j} \text{Area}(\triangle t_j)}{\sum_{j \in R_i} \text{Area}(\triangle t_j)}
\]
In the second step, all region triangles $\triangle t_j \in R_i$ produce $j$ errors to the center-of-mass proxy, using one of the following errors $E(\vec{n}_j, \vec{N}_i)$, starting with a simple vector difference:

$$E_{\text{normal}} = \vec{N}_i - \vec{n}_j$$

The error-to-proxy calculation can be done in a few more different ways, all of which result in fairly similar region configurations. Below is an error inspired by Sander [2003]:

$$E_{\text{Sander}} = (1 - \vec{N}_i \cdot \vec{n}_j)\|\mathbf{Bb}\|,$$

where $\mathbf{B}$ is the barycenter of the proxy triangle of normal $\vec{N}_i$, and $\mathbf{b}$ is the barycenter of the triangle $\triangle t_j$ whose error is calculated, of normal $\vec{n}_j$. Another error is due to Cohen-Steiner et al. [2004]:

$$E_{\text{Cohen-Steiner}} = \text{Area}(\triangle t_j)\|\vec{N}_i - \vec{n}_j\|^2$$

Finally, our own version produced the results throughout this paper:

$$E_{\text{Wise}} = (1 - \vec{N}_i \cdot \vec{n}_j)^2 \text{Area}(\triangle t_j)$$

In the third step, the new proxy is found through a region search of the region triangle which minimizes error $E$ to the center-of-mass. This makes sure that the proxy is a real triangle, since the center-of-mass is only a calculated normal vector, with no face association, and which may not coincide at all with an existing one.

### 3.3.2 Discussion of the Cohen-Steiner vs. Wise Error Metrics

Our error calculation was based on the idea from Cohen-Steiner et al. [2004], but improved to decrease sensitivity to insignificant normal vector angle differences, which may arise with noise, or with gently curved surfaces. Another aspect of our error formulation was the concern for speed of computation, slightly improved as well, since there are no vector calculations past the original dot product from our formula, which brings the computation to a difference and a square of scalars.

#### Our Segmentation Error Metric

We present a simple comparison between the Cohen-Steiner error and our own, omitting the triangle area factor, which is the same in both cases. The formulæ are:

$$E_{\text{Cohen-Steiner}} = \|\vec{v}_1 - \vec{v}_2\|^2 \quad \text{and} \quad E_{\text{Wise}} = (1 - \vec{v}_1 \cdot \vec{v}_2)^2$$

(8)
To keep the comparison simple, we consider the 2D case:

\[ E_{\text{Cohen-Steiner}} = \| (v_1 \vec{i} + v_{1y} \vec{j}) - (v_2 \vec{i} + v_{2y} \vec{j}) \|^2 = \| (v_1 \vec{i} - v_2 \vec{i}) + (v_{1y} - v_{2y} \vec{j}) \|^2 \]

vs.

\[ E_{\text{Wise}} = \left[ 1 - (v_1 \vec{i} + v_{1y} \vec{j}) (v_2 \vec{i} + v_{2y} \vec{j}) \right]^2 = (1 - v_{1x} v_{2x} - v_{1y} v_{2y})^2 \]

The Two Metrics Are Different

In this section we formally prove that the two errors are different, due to the square in our formulation:

\[ \| \vec{v}_1 - \vec{v}_2 \|^2 = (v_{1x} - v_{2x})^2 + (v_{1y} - v_{2y})^2 \]

vs.

\[ (1 - \vec{v}_1 \vec{v}_2)^2 = 1 + v_{1x}^2 v_{2x}^2 + v_{1y}^2 v_{2y}^2 - 2 v_{1x} v_{2x} - 2 v_{1y} v_{2y} + 2 v_{1x} v_{2x} v_{1y} v_{2y} \]

With reductions on both sides, keeping in mind that \( \vec{v}_1 \) and \( \vec{v}_2 \) are normal unit vectors, the comparison becomes:

\[ v_{1x}^2 + v_{2x}^2 + v_{1y}^2 + v_{2y}^2 = 2 \text{ vs. } 1 + (v_{1x} v_{2x} + v_{1y} v_{2y})^2 \quad (9) \]

It now suffices to consider a simple counterexample to prove that the two quantities in Equation 9 are, indeed, different. Let \( \vec{v}_1 = \sqrt{3} \vec{i} + \frac{1}{2} \vec{j} \) and \( \vec{v}_2 = \frac{\sqrt{2}}{2} \vec{i} + \frac{\sqrt{2}}{2} \vec{j} \) two unit vectors in the plane. After the numeric replacements in Equation 9, we get:

1 vs. \( \left( \sqrt{3} \frac{\sqrt{2}}{2} + 1 \frac{\sqrt{2}}{2} \right)^2 = \left( \frac{\sqrt{6} + \sqrt{2}}{4} \right)^2 \simeq 0.933 \neq 1 \)

(Before reductions, the actual values of the two metrics for the counterexample used are \( E_{\text{Cohen-Steiner}} \simeq 0.068148 \) and \( E_{\text{Wise}} \simeq 0.001161 \), respectively.)

Sensitivity to Angle Difference

We now compare the two metrics from the point of view of how well each reflects angle difference between normal unit vectors. We claim that our error metric, \( E_{\text{Wise}} \), is less reactive to small such differences, which gives the segmentation a better aspect in the presence of noise and gentle local curvature, without compromising the necessary delineation at sharp features. This is because its dependency of \( \alpha - \beta \), the angle...
between the two vectors, always graphs under the same dependency derived from $E_{\text{Cohen-Steiner}}$.

How does $|\vec{v}_1 - \vec{v}_2|^2$ vary with $\alpha - \beta$? Without losing generality, we consider again the two unit vectors in the plane, forming angles $\alpha$ and $\beta$ with the Ox axis, respectively.

In the Cohen-Steiner formulation, we relate $E_{\text{Cohen-Steiner}}$ to $\alpha - \beta$ as follows:

$$
|\vec{v}_1 - \vec{v}_2|^2 = (v_1 - v_2)^2 = (v_1 \cos \alpha - v_2 \cos \beta)^2 + (v_1 \sin \alpha - v_2 \sin \beta)^2 = v_1^2 + v_2^2 - 2v_1v_2(\cos \alpha \cos \beta + \sin \alpha \sin \beta) = 2[1 - \cos(\alpha - \beta)] = 4\left[1 - \cos\left(\frac{2\alpha - \beta}{2}\right)\right] = 4\sin^2\frac{\alpha - \beta}{2} \quad (10)
$$

In Equation 10, we kept the assumption that $|\vec{v}_1| = v_1 = 1$ and $|\vec{v}_2| = v_2 = 1$. If we further consider, in the worst case, that the angle difference can range from $0$ to $\pi$, in other words that $\alpha - \beta \in [0; \pi]$, then $\frac{2\alpha - \beta}{2} \in [0; \frac{\pi}{2}]$, interval over which the sine function is positive and increasing. This means that, indeed, this error formulation provides a meaningful correlation between normal angle difference and region assignment.

How does $(1 - \vec{v}_1 \cdot \vec{v}_2)^2$ vary with $\alpha - \beta$? In our formulation, we can relate $E_{\text{Wise}}$ to $\alpha - \beta$ as follows:

$$
(1 - \vec{v}_1 \cdot \vec{v}_2)^2 = \left[1 - \cos(\alpha - \beta)\right]^2 = 4\left[1 - \cos\left(\frac{2\alpha - \beta}{2}\right)\right]^2 = 4\sin^4\frac{\alpha - \beta}{2} \quad (11)
$$

The same considerations of function domain apply. Comparing Equations 10 and 11, we see that $E_{\text{Wise}} = \frac{1}{4}E^2_{\text{Cohen-Steiner}}$, but also that, on the definition domain of $\sin \frac{\alpha - \beta}{2}$, the curve of $\sin^4\frac{\alpha - \beta}{2}$ lies consistently under the curve of $\sin^2\frac{\alpha - \beta}{2}$, thus making our error more conservative for small increases in angle differences than the one presented in Cohen-Steiner et al. [2004]. Because our error increases more slowly for (a wider range of) small angle differences, it presents a higher tolerance for mildly curved neighborhoods, which can be assimilated more easily into the same region:

$$
\begin{align*}
  f'_1\left(\frac{\alpha - \beta}{2}\right) &= f'_1(x) = (\sin^2 x)' = 2 \sin x \cos x = \sin(\alpha - \beta) \\
  f'_2\left(\frac{\alpha - \beta}{2}\right) &= f'_2(x) = (\sin^4 x)' = 4 \sin^3 x \cos x = 2 \sin(2x) \sin^2 x = 2 \sin(\alpha - \beta) \sin^2\left(\frac{\alpha - \beta}{2}\right)
\end{align*}
$$

48
If we compare, for instance, the slopes for \( x = \frac{\pi}{6} \), we get:

\[
\begin{align*}
    f'_1\left(\frac{\pi}{6}\right) &= \sin\frac{\pi}{3} = \frac{\sqrt{3}}{2} \\
    f'_2\left(\frac{\pi}{6}\right) &= 2\sin\frac{\pi}{3}\sin\frac{2\pi}{6} = \frac{\sqrt{3}}{4} < f'_1\left(\frac{\pi}{6}\right),
\end{align*}
\]

we see that \( f_2 \)'s slope is lower for the interval \( x \in \left[0; \frac{\pi}{4}\right] \), or \( \alpha - \beta \in \left[0; \frac{\pi}{2}\right] \). Rigorously, \( \sin^4 x \) has the inflexion point at a higher \( x \) than \( \sin^2 x \) (\( \frac{\pi}{3} \) vs. \( \frac{\pi}{4} \)), and the slope of \( \sin^4 x \) is below the slope of \( \sin^2 x \) for \( x \in \left[0; \frac{\pi}{4}\right] \) (see Appendix C). This behavior of our error metric is better for smoothly curved surfaces (such as the Stanford bunny), as well as for surfaces acquired with noise (such as our scanned data), because it tends to produce smoother boundaries. Segmentation must not occur at local normal differences that are insignificant, or that might produce concavities in the boundary shape, unless strictly necessary. It must occur only at sharp features or at the point-count limit for a region, with boundaries as smooth as possible. This is a necessity in our implementation because we use the trace of these boundaries on the surface to produce our subdivision model reconstructions, and concavities will interfere with the per region method of reconstruction. In the fixpoint border reconstruction, irregular borders are also a negative, as they force unsightly border gap retriangulations. The monotonicity characteristics of these two functions can be visualized in Figure 3.

![Figure 3: Monotonicity of \( E_{\text{Cohen-Steiner}} \left( \frac{\alpha-\beta}{2} \right) \) (red) vs. \( E_{\text{Wise}} \left( \frac{\alpha-\beta}{2} \right) \) (blue)](image)

However, segmentation of continuous surfaces, especially in the presence of the constraint of a fixed number of regions, is a very subjective matter. In effect, the number of regions imposes an angle limit on normal differences between adjacent regions, therefore the shape of the segmentation is the result of a user choice. While it is possible
3.3 The Segmentation Algorithm

and desirable to segment at features that can be agreed on visually, segmentation with higher granularity remains a preference (the number of applications that generate segmentation based on user input stands proof to this) or (in the presence of automated algorithms) an accepted result depending on the usability of the end product. For medical applications, for instance, where one must distinguish between tissue A and tissue B, the number of output segments is an objective and a priori known quantity. In contrast, for objects and buildings, the number of surface regions — not an end in itself — is more or less appropriate only depending on how the regions can be used in later stages of processing without loss of information for the whole; and also, to a smaller extent, depending on how quickly the segmentation can be applied to larger data sets, or to a large number of data sets in a limited amount of time.

Figure 4.: Segmentation of the bunny model with R=50, in various implementations of the error metrics: (a) Cohen-Steiner from Cohen-Steiner et al. [2004], (b) Sander from Sander [2003], and (c) our own.
3.3 THE SEGMENTATION ALGORITHM

3.3.3 Global Priority Queue

Besides a new proxy, a new region configuration also fills in a global priority queue, which holds triplets consisting of triangles, their error to their own region proxy, and that proxy. This GPQ provides the pool from which triangles are popped, in increasing order of error to their respective proxies, during the region assignment step. Unlike the proxy calculation, which is based on the previous region configuration, the GPQ filling and emptying (for region assignment) both happen during the same iteration.

Each tuple pushed onto the GPQ consists of a triangle, its error to proxy, and its proxy:

\[
\text{tuple} = (\Delta t_j, E(\Delta t_j, \Delta T_i), \Delta T_j)
\]

The arrangement of all such tuples into the GPQ is keyed by triangle, and sorted by error. A triangle’s addition to a region and a push of its neighbors into the GPQ are one atomic step. However, pushing triangles onto the GPQ (building the pool of unassigned triangles) and adding triangles to a proxy (region growing) are two different operations. Pushing a triangle tuple onto the GPQ also stores the region adjacency for that triangle. A triangle can end up in the GPQ by being pushed from several different regions.

3.3.4 Iterative Region Reconfiguration

To reconfigure the regions, each iteration uses two previously computed items: region triangle proxy (computed in the previous iteration) and global priority queue (computed during the current iteration, one adjacency layer ahead of the look-up for popping). The reconfiguration happens through additions of adjacent triangles to the updated proxy, after which old region assignments have outlived their usefulness. As mentioned, the algorithm computes errors of triangles \(t_j\) only to proxies to which they are connected through adjacency, but arranges all such tuples into a unique GPQ, keyed by triangle, and globally sorted by error. Thus, errors computed against different proxies will be pushed onto the same queue. Multiple pushes of the same triangle onto the queue happen when the wave of adjacency reaches it from neighboring regions simultaneously. This creates entries of the same triangle, but with different errors. This is the situation where the algorithm makes an assignment choice among competing regions, claiming membership of the same triangle.
3.3 The Segmentation Algorithm

**Multiple Entries in the GPQ for the Same Triangle** The region growing process goes through the existing GPQ, pops the triangle with the least error (among choices of triangles with errors to other seeds), and assigns it to its own seed. At this time, the neighbors of each added triangle are themselves pushed into the GPQ, thus preserving the identity of the seed they came from (through the neighboring relationship), stored in the proxy field. Each triangle pushed into the GPQ will therefore have associated to it the error to the seed that it was connected to through an adjacency relationship. Thus, even though triangle additions to a region are only possible for triangles adjacent to it already, only those assignments will be made for which the error to their adjacency seed is smallest. This means that a region with errors of its own outer adjacent triangles smaller than another region’s outer band error will favor more additions to itself.

**Adjacency Property** Growing a region around a proxy follows a path of strict adjacency. This is necessary to ensure region contiguity in a data space riddled with gaps. In fact, getting rid of unwanted scene artifacts or extraneous objects is as simple as skipping a seed in that area. We applied this principle to the segmentation of the room interior data set, which included a mirror. The mirror reflected data points, which appeared as spurious data in a virtual space behind the mirror. To eliminate that duplicated part of the scene, we manually selected initial seeds, skipping the mirror area. In another scene, we eliminated a chandelier in the same manner. Triangle addition to a region is final for an iteration — there are no back and forth re-assignments. Adding adjacent triangles around a proxy only stops when that region’s growth conflicts at the edges with another region’s growth.

**Minimum Error Choice** This conflict happens as the same triangle has several entries in the GPQ, from different seeds, with different errors to each one, but only the entry with the minimum error (being at the top of the queue) will get assigned to its own seed. The other instances of the same triangle are tagged with the new region assignment, to avoid their future selection from the GPQ (they are not removed per se). Presence in the GPQ means availability for addition only if also the tag indicates “unassigned status”. This is faster than searching, for each pop operation, all other same triangle instances from the GPQ. If the pop operation is successful, and the triangle was previously unassigned, our region claims it. If it had been assigned already, our region stops growing in that direction.
3.3 THE SEGMENTATION ALGORITHM

Figure 5: Spurious points created in an interior scene by reflection of data in the mirror.

**Global Error Principle**  The principle of this algorithm is not based on all-to-all triangle to updated proxies comparison. Nor is it based on a fully precomputed GPQ with errors to regions from the past iteration. It is based on a one step ahead adjacency layer pushing. This achieves two purposes: it produces only one global pass through the scene triangles, and it reduces the search space for region additions to the one layer of former adjacencies. The addition will happen based on each region’s having smaller errors of its outer band, and thus producing more numerous additions to itself, while other regions will lose the same triangles. The only decision between additions to different regions happens during the assignment of the last band, in which the triangles will appear as adjacent to more than one triangle (and thus region), and there will be several errors (and thus proxies) to choose from. The least error to its own proxy will be the one at the top of the GPQ, and will benefit that region with one more addition.

**Algorithm Termination**  All remaining instances in the GPQ of a triangle which was assigned to a region will be tagged as assigned and popped without assignment when they resurface at the top of the queue. This prevents a triangle to be assigned to more than one region. The emptying of the GPQ signals the termination of the region partitioning step. After a partitioning step has been completed, the newly created regions will form new proxies. The proxies computed on the new regions will replace the old ones and a new partitioning stage starts again.
Thus, adding adjacent triangles to a region is conditional: namely, the candidate triangle (selected only through adjacency) is located in the GPQ. If it is not there, it means it hasn’t been previously assigned to any other region. Therefore, it will be assigned to the region to whose proxy it is connected through a chain of adjacency. But if it is already in the GPQ, it means that adjacency has reached it faster from a neighboring region, to which it is now already assigned. At this point, a second condition has to be met: this time, with an assignment already made, if the error of the candidate triangle against this region is smaller than the error against the region to which it had been previously assigned, the assignment will be changed, and, for the time being, this region will attempt to add the triangle to itself. If no other region claims this triangle, the assignment remains final. On the other hand, if the error already found in the queue is smaller than the current one, this region stops growing in that direction, and only third region claims may produce smaller errors and possible reassignments.

In our experiments, four iterations are usually enough for convergence. This step is virtually instantaneous for every model up to 100,000 points (the hemisphere, the bunny), and very fast for sets of range scanner data from 1,500 points (building corner) to 1,000,000 points (building slice, room, larger academic models).
3.3. The Segmentation Algorithm

<table>
<thead>
<tr>
<th>Model</th>
<th>Triangles</th>
<th>Regions</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>fandisk</td>
<td>23,965</td>
<td>15</td>
<td>0m30.791s</td>
</tr>
<tr>
<td>dragon</td>
<td>47,795</td>
<td>20</td>
<td>1m42.331s</td>
</tr>
<tr>
<td>50k</td>
<td>50,000</td>
<td>15</td>
<td>2m26.186s</td>
</tr>
<tr>
<td>buddha</td>
<td>67,241</td>
<td>20</td>
<td>2m36.440s</td>
</tr>
<tr>
<td>buddha</td>
<td>1,087,717</td>
<td>100</td>
<td>8m47.256s</td>
</tr>
<tr>
<td>bunny</td>
<td>69,451</td>
<td>20</td>
<td>2m15.512s</td>
</tr>
<tr>
<td>bunny</td>
<td>69,541</td>
<td>50</td>
<td>0m55.849s</td>
</tr>
<tr>
<td>horse</td>
<td>96,967</td>
<td>20</td>
<td>12m18.853s</td>
</tr>
<tr>
<td>armadillo</td>
<td>345,945</td>
<td>1000</td>
<td>2m29.826s</td>
</tr>
<tr>
<td>scan3</td>
<td>927,284</td>
<td>1000</td>
<td>3m45.545s</td>
</tr>
</tbody>
</table>

Table 3.: Segmentation times for various models

Figure 7.: Scan 3 of the Great Hall of the Shepherd Hall at the CCNY campus, with 1000 regions

Figure 8.: Different levels of segmentation on the bunny model.
3.3 The Segmentation Algorithm

**Algorithm 1** One Partitioning Step

// Initialize the region assignment status array
for triangle ∈ triangleSuperSet do
    status[triangle] ← −1;
end for

// Initialize regions with their seeds
for k = 1 to K ∈ seedSet do
    $\mathcal{R}_k$.ADDTRIANGLE(k);
    GPQ.PUSHNEIGHBORS(k);
end for

// Assign and push atomic step
while not EMPTY(GPQ) do
    topTriangle ← Top(GPQ);
    Pop(GPQ);
    if status[topTriangle] == −1 then
        status[topTriangle] ← k;
        $\mathcal{R}_k$.ADDTRIANGLE(topTriangle);
        GPQ.PUSHNEIGHBORS(topTriangle);
    else
        continue;
    end if
end while

return regionSet;

Figure 9.: Different views of the segmentation of the fandisk model, R=20.
Figure 10.: Scan 3 of the Great Hall of the Shepherd Hall at the CCNY campus, with 144 user-specified regions
4.1 Fix-Boundary Parametrization

As stated in Section B.0.4, the main problem in the field of parametrization remains finding a good parametrization metric, which would result in the minimization of a particular distortion of the parameter map versus the original 3D surface. We mentioned that the three main types of ideal transformations (isometric, equiareal, and conformal) result in a series of constraints on the singular values $\sigma_1$ and $\sigma_2$ of the Jacobian of the transformation defining the 2D map, in Section B.0.1. Conformal (angle preserving) maps being best for the quality of the reconstruction, we also mentioned how harmonic maps came to approximate conformal maps in Sections B.0.3 and B.0.4. Since no exact solution to the problem of finding a discrete conformal or harmonic transformation, or mapping, from 3D to 2D exists, numerous approximations have been proposed, all resulting from the translation of the various mathematical requirements defining conformality, into a discrete function that maps 3D points onto 2D parameter points, as hinted at in the Section B.0.4 and amply presented in Sections B.0.4 and B.0.4, which describe the most important contributions in conformal mapping, with numerous papers presenting derivative formulæ that reference these findings, such as [Pinkall and Polthier 1993], [Floater 1996], [Hormann and Greiner 2000], [Haker et al. 2000], [Sheffer and de Sturler 2001], [Sander et al. 2001], [Floater 2003], [Hormann and Floater 2006], and many others. All these approaches attempt to minimize angular distortion, or to preserve distances, or area, or a combination thereof.

One of our contributions is such a discrete function, which minimizes the angular distortion as well as the stretch in the parameter domain. Our transformation is defined for each point in 2D as a displacement of a point relative to its 3D neighborhood, as most other methods. We chose a displacement formula which aims to preserve angles within all triangles incident on a point, as well as the length of the edge connecting the point to each of its neighbors. This formula was inspired by the triangle congruence case ASA (angle-side-angle), and requires the weight for each vertex (contributing to the relocation of the point) to be a normalized average of weights computed with angles and edge lengths from one incident triangle at a time, from the
Figure 11.: Parametrization for the hemisphere.

vertex neighborhood. The formulæ, shown in Equations \ref{equation15} and \ref{equation16} are presented in Section 4.3.

This segmentation algorithm generates free-form boundaries. We use this result in a fix-boundary parametrization setting, in which the theory recommends — when using regular shapes (unit square, unit circle etc.) is not possible — the projection of real borders onto a least squares plane, as domain boundaries. Most real world surface reconstruction work traditionally defines the domain boundaries by specifically locating points of interest on the mesh and connecting them through geodesic paths to generate mesh cuts. In contrast, our parameter domains are surface-driven, and, in our fix-boundary implementation, we don’t impose any constraints to their shape even in reconstruction. Parameter values obtained with this particular constraint are very reliable in obtaining accurate reconstructions, as demonstrated in Chapter 8. When combined with our resampling scheme, allowing the boundaries to vary along with the interior points would generate planar domains of wider areas, from which recon-
structured points close to the margin of a region would overstep the fix boundaries in reconstruction.

The parametrization method basically unfolds the 3D surface onto the plane, in a way that preserves the shape of the original triangles. This is important in creating a domain where resampling and retriangulation are easy to perform. But it is essential in reconstruction, where the parameter values must generate valid surface points through interpolation. To generate the 2D triangles, each point on the original surface is averaged with neighboring points, using neighborhood information to weigh each neighboring point’s contribution. The neighborhood mask is the one-ring region around each vertex, which means the method has local support. As such, the matrix solving for the new 2D point locations is sparse, and the linear system can be solved efficiently — with the only caveat of size restriction.

4.2 OUR CONTRIBUTION

4.2.1 Parametric Mesh Simplification

We use the fix, free form boundary 2D parametrization to simplify the data set, relying on the quality of the segmentation and parametrization to avoid topological errors after simplification. Even with aggressive simplification in resampling, we never reconstruct away from the original mesh. Current methods use 3D locally iterative approaches to operate vertex reductions with no guarantee of preserving topology. For instance, Garland and Heckbert [1997], in one of the widely circulated mesh simplification methods, acknowledge that this method does not preserve topology. With ambitious simplification ratios, even though the reconstructed points are in provably good locations relative to the original mesh, their connection may be forced over areas that curve below the connections. This is true about any method. Our method does not produce points not on the surface of the original mesh, and, due to the regions being, in general, bordered by features, connection of previously discontiguous mesh areas is not possible.

Using just one pass, due to the point resampling in parameter space, we can achieve virtually any ratio of simplification. A limit exists only in the case of the Delaunay method — which we use for comparison only, since we proudly provide a retriangulation method ourselves — because it requires a minimum number of triangles to be kept in the triangulation, in order to preserve the Delaunay property. Another limit exists in the case of the fixpoint border reconstruction, where our triangulation re-
mains denser along the borders, due to the higher density of points along the borders; however, the alternative (the Delaunay triangulation) exhibits the same characteristic.

A newer approach we explored was building the surface up at various resolutions using subdivision. This is not original, and we have a long way to go to master that technique. However, we found a new way to generate the base domains, through parametrization of the borders, simplification and reconciliation of the borders in 2D, and then triangulation of the resulting simplified polygons. This yielded spectacular results in the case of academic shapes such as the hemisphere, the Stanford bunny, and the Armadillo, both visually and speed-wise.

4.2.2 Fixpoint Boundaries

Another new idea of our mesh simplification method is the rejoining of regions during the remeshing step, along original shared boundaries. The reconciliation of the region borders is possible through retaining the original the borders as the fixpoint of the parametrization and of the reconstruction process. We constrain the parametrization to an orthogonal projection of the original borders onto the least squares plane of each region, as recommended in Floater [1996], Floater [2003], and Hormann and Floater [2006]. The points along borders also remain unchanged through the reconstruction phase, where, due to the border constraint in parametrization, the lifting in 3D of the interior points will fall within the region hull. With this choice, however, we introduce the problem of a different sampling density along the borders. But each retriangulation method we use addresses the border density in its own specific way. The Delaunay triangulation is flexible and discards points as necessary, even though, as stated in Section 4.2.1, it famously results in denser, smaller triangles along the borders. The border gap triangulation can be assimilated in a further round of segmentation, if the whole algorithm is applied multiple times. The straight border triangulation does not preserve original borders, and we mention it here for completion, and because it is based on a subset of points on the borders; but it reconstructs identically across regions through a map we create during resampling.

4.2.3 Multiresolution Remeshing

The resulting 2D regions can be retriangulated or resampled at different densities, which are specified as a region point reduction factor (in the case of the grid resampling method), or as a number of subdivisions (in the case of the triangular subdivision method). The desired resolution can be achieved directly from the original mesh.
4.3 THE PARAMETRIZATION SCHEME

As a future interest, we would like to explore level-of-detail rendering methods, such as wavelets.

The grid step for regions widely varying in size, such as result from our segmentation, is calculated to scale to each region, based on the reduction ratio mentioned above. Furthermore, to prevent the difference in grid step sizes that may result even after scaling — due to the fact that the grid is calculated from one of the dimensions of the region bounding box, therefore step sizes not correlated with the actual region size, which, through shape irregularity, could be much smaller than its bounding box, or than one of its two dimensions — we developed an original step formula which keeps proportionality with the original region point count after resampling, regardless of region size. Through this proportionality, given that the reduction applies to a number of points originally regularly scanned, we actually achieve regular resampling throughout.

4.3 THE PARAMETRIZATION SCHEME

The regions formed as a result of the algorithm described in Section 3.3 are natural domains for parametrization. They unfold onto planar representations under the constraint of the preservation of their borders, as mentioned in Section 4.2.2, by having these borders projected onto the least squares plane, while the inner points are calculated to preserve angles. The transformation from 3D to 2D is implemented through a sparse linear system, which creates parameter $u = (u, v)$ values for the region points according to the following cases:

$$u_i = \begin{cases} 
    w_{ij}u_i, & \text{if } u_i \not\in \Gamma \text{ and } u_j \in N_i, \\
    u_i, & \text{if } u_i \in \Gamma.
\end{cases} \quad (12)$$

Throughout this section, $\Gamma$ represents the boundary of $\Omega$ (the domain of the transformation), $u_i$ represents the 2D parameter point, the subscript $j$ indicates its neighbors, and the neighborhood of $u_i$ is denoted by $N_i$. The weights $w_{ij}$ applied to each point are dependent on the geometry of the one-ring neighborhood, according to the mathematical justification presented in Section B.0.3.

The simultaneous requirements of the form $12$ produce a linear system $Au = b$, where the matrix $A$ of the coefficients includes weight entries for each interior point, the vector unknown $u$ represents the calculated parameter 2D points, and the free term vector $b$ is formed with the fixpoint border coordinates, scaled by their weight factor, if they occur in an interior point’s neighborhood, or zeroes, if not.
If we denote by $n$ the number of interior points and by $b$ the number of border points in a region, then the size of the matrix will still be only $n \times n$ (since we compute only interior points). Thus, each row $i$, representing a point $u_i$, will be:

$$u_{ii} = \lambda_{i1} u_1 + \ldots + \lambda_{ij} u_j + \ldots + \lambda_{in} u_n, \ j \neq i$$ \hspace{1cm} (13)

or, grouping the terms based on relationship to $N_i$,

$$\sum_{u_k \not\in N_i} \lambda_{ik} u_k + \sum_{u_j \in N_i} \lambda_{ij} u_j - u_{ii} = 0.$$ 

The influence of non-neighborhood points over $u_i$ is zero: $\sum_{u_k \not\in N_i} \lambda_{ik} u_k = 0$. For clarity, and because these zero entries are specified as such during the construction of the matrix of coefficients, we will not remove from the row equation the corresponding term. After moving explicitly the known values of the border points (which don’t change) in the free term column, the row will look like this:

$$\sum_{u_k \not\in N_i} \lambda_{ik} u_k + \sum_{u_j \in N_i} \lambda_{ij} u_j - u_{ii} = -\sum_{u_b \in N_i} \lambda_{ib} u_b$$

Accordingly, the matrix $A$ will include three types of coefficients:

$$\begin{cases} 
\lambda_{ij}, & \text{if } i \neq j \text{ and } u_j \in N_i, \\
-1, & \text{if } i = j, \\
0, & \text{if } i \neq j \text{ and } u_j \not\in N_i.
\end{cases}$$ \hspace{1cm} (14)

The free term vector is filled as follows:

$$b_i = \begin{cases} 
-\lambda_{ib} u_b, & \text{if } u_b \in N_i, \\
0, & \text{ow.}
\end{cases}$$

The coefficients $\lambda_{ij}$ are the normalized weights

$$\lambda_{ij} = \frac{w_{ij}}{\sum_{u_j \in N_i} w_{ij}},$$ \hspace{1cm} (15)

where weight $w_{ij}$ corresponding to the triangle $\triangle i, j, j+1$, incident on point $u_i$, is:

$$w_{Wise} = \frac{\tan \frac{\alpha_{ij}}{2} + \tan \frac{\beta_{ij}}{2}}{r_{ij}},$$ \hspace{1cm} (16)
4.3 The Parametrization Scheme

where $\alpha_{ij} = \angle(j, i, j + 1)$, $\beta_{ij} = \angle(i, j, j + 1)$, and $r_{ij} = \|u_i - u_j\|$. This formula preserves angles comparably with Pinkall and Polthier [1993], and arguably better than Floater [2003], as illustrated in the hemisphere reconstructions of Figures 12. Compared to Floater [2003], it optimizes computation by using only one incident triangle at a time, which is a big gain in speed, if no incident triangle ordering is necessary, and these triangles can be accessed randomly. The ordering of incident triangles uses edge adjacency, and thus it tests each incident triangle against the remainder of the set, on each of its three sides. Compounded for the scene size, this is one of the computational bottlenecks of the entire program.

We used several other weight formulæ for comparison, first the one derived by Pinkall and Polthier [1993]:

$$w_{\text{Pinkall-Polthier}} = \cot \beta_{ij} + \cot \gamma_{ij},$$

(17)

and the other by Floater [1996]:

$$w_{\text{Floater}} = \frac{\tan \alpha_{i,j} + \tan \beta_{i,j+1}}{r_{ij}}$$

(18)

Figure 12.: Hemisphere grid point reconstruction with different parametrization weights, 15 regions and an oversampling factor of 8.
Algorithm 2 Compute Umbrella

```plaintext
struct Umbrella U; // Initialize the umbrella structure
U.i ← currentPoint; U.j ← new POINTS();
incidentTriangleSet ← ORDERINCIDENTTRIANGLES(i);
U.degree ← −1;
U.weightsArray[1...j]; U.averageWeight ← 0;
U.α[1...j]; U.β[1...j]; U.γ[1...j];
// Fill in incident triangle △j = △ABC data for each j ∈ Ni
for j = 1 to Size(umbrellaTriangleSet) do
    △j ← umbrellaTriangleSet.at(j);
    Line |ij| ← △j.getSIDE_C();
    j ← △j.getVERTEX_B();
    U.degree++;
    U.j.AddPoint(j);
    U.α[U.degree] ← ANGLE(△j, A);
    U.β[U.degree] ← ANGLE(△j, B);
    tan α/2 ← U.α[U.degree].HALFTANGENT();
    tan β/2 ← U.β[U.degree].HALFTANGENT();
    // Compute cumulative neighboring point weights
    U.weightsArray[j] ← (tan α/2 + tan β/2)/|ij|.CALCSEGMENTLENGTH();
    U.averageWeight += U.weightsArray[j];
end for
return U;
```

Algorithm 2 collects information from the umbrella stencil around a point, such as edge lengths, angles of incident triangles, and calculates average weights. As mentioned earlier, ordering of the incident triangles around a point is one of the expensive parts of the whole program, but is necessary only for those formulations of the weights $w_{ij}$ that need information from two adjacent triangles simultaneously. In our redesigned weight formula, we need angles from only one incident triangle at a time, which would not need the call to ORDERINCIDENTTRIANGLES(). The call to FINDINCIDENTTRIANGLESET(), implemented at the beginning of the program to cache this information for the region growing phase, performs a search through all scene triangles; at the time of umbrella calculation, this information is already available.

We use versions of the industry-standard BLAS and LAPACK optimized for modern hardware by Apple and distributed with MacOS X as our linear solver, with the open source LAPACK++ library as a wrapper to provide a C++ interface to these libraries. Once the 2D points are computed, they can be displayed using a coordinate system transformation that shows them in the least squares plane computed for each region. For visualization, we use the open source Coin3D implementation of the OpenInventor scene graph rendering system, implemented on top of OpenGL.

1 LAPACK++ is hosted on sourceforge.net.
Figure 13: Parametrization triangulation, using $w_{Wise}$ of the 50K detail of the Great Hall. The triangulation is shown in the least square plane of each region.

Table 4: Parametrization times for various models. These are **cumulative**, including the previous stage of segmentation.
Figure 14.: Parametrization on various models. Each region is flat, and represented in its least squares plane.
RESAMPLING

5.1 PARAMETER SPACE RESAMPLING

We sample the parameter space at a uniform square grid, with the step designed to remain proportional with the original density across regions of varying sizes, and to account for the point loss resulted from the region stamping — the carving of the region shape out of the rectangular grid. We then reconstruct the vertices of this grid, along with the original points on each region border, through barycentric interpolation. For a mesh output, we reconstruct the obvious triangulation of the grid, and we also provide a triangulation of the space between the inner border and the outer border, the border gap triangulation. This is a very difficult step, because the outer border is irregular and non-convex. We resorted to the monotone polygon triangulation described in O’Rourke [2005], which we extended to polygons with holes and improved with a recursive partitioning algorithm.

5.2 OUR CONTRIBUTION

One of our motivations was for the whole algorithm (segmentation, parametrization, resampling, reconstruction) to iterate multiple times through the process and thus integrate non-feature border points from the previous iterations into a future region interior. This situation arises in spite of the good threshold metric of the segmentation algorithm, because for large data sets there is a hard resource limit on the size of the matrix system computing the parametrization. Thus, during a first global iteration of our algorithm, separate regions can be created even for smoothly varying surface components, because the region count has to be correlated with the size of the data set in order to limit the point count per region and keep the size of the matrix low. This consideration makes a triangulated output a necessity. An alternative would be to work on non-triangulated data sets and to derive the geometric characteristics (normals at points, point neighborhoods, and their elements) in the absence of a triangular structure, which is a direction we did not pursue, against better advice.

Another motivation was to enable sequential processing of large scenes without border gaps between scene components, even for pre-segmented components run in separate instances of the program. These can be easily reassembled into a complete
scene using a shell script looping through the triangle output files, which index against an all-points file. The resulting triangle files will not generate seam gaps if the borders appear connected identically (with the same triplets of indices) in the triangle files for adjacent regions. The border edge indices are the same in both triangle files only if the reconstructed borders coincide. The side effect of this region joining is a higher density of reconstructed points along the borders, relative to the simplified interior. However, at least for building reconstructions, the separation between different scene components will (hopefully) correspond naturally to feature borders in the original scene, and thus our fixpoint method will not introduce more densely sampled outlines in undesirable places after reconstruction.

We contribute, therefore, a border gap triangulation, based on the classic monotone polygon triangulation presented in O’Rourke [2005], with the two improvements (holes and recursive partitioning) discussed in Section 5.1 and detailed in Chapter 6.

Once our triangulation was functional, one of our realizations was that it could be used to triangulate whole region outlines, and thus generate triangular base domains similar to those employed by many subdivision schemes. We returned to one of our earlier abandoned efforts, in which we had separated and ordered in a consistent edge traversal order all the complex region borders, as well as their fragments. We call fragments subsets of the border edge sets comprised between corner points, which, in turn, are points at the intersection of three or more regions. We replaced the border fragments with straight lines in the parameter domain, we triangulated the resulting simplified polygons, and subdivided the triangles for a much faster resampling scheme. Our second contribution in resampling is the generation of the base domain triangles through segmentation and border straightening, from where the subdivision can proceed.

5.3 Grid Remeshing

The 2D regions can be resampled at any density. The grid for the planar regions is region-sensitive, but uniform. Region-sensitive, because it depends on the region point count; and uniform, because it achieves roughly the same aspect throughout the mesh even as the regions are of widely varying sizes; all by specifying a unique scene-wide reduction factor. The grid step is derived from a combination of: region original point count; size of the axes-aligned region bounding box; and the user-specified reduction factor. We developed a step calculation formula which keeps the new number of points in proportionality with the original region point count regardless of region size.
We created an axes-aligned bounding box around each planar region, and we gridded this box based on a division factor to be applied to the smaller of its dimensions, to obtain the unit-length (the step) of the grid:

$$\text{step} = \frac{\text{width}}{\text{factor}}$$

The larger dimension was then ruled using the same step, so that the grid elements become approximately square. The irregular region shape was then stamped out of this rectangular grid, to obtain the region interior points. The border points were passed unchanged. The sampling step formula must scale the new density — after the excess of the bounding box that does not fall within the flat region borders is discarded during the region stamping — to the original point count of each region. Therefore, we needed a new point count

$$n = \frac{N}{f}, \quad (19)$$

where

$N$ — pre-sampling region point count

$f$ — reduction factor.

We want to stamp $n$ points out of $n_{BB}$ points in the bounding box around the region, aligned in a rectangular pattern

$$n \leq n_{BB} = n_w \cdot n_1,$$

where

$n_w$ — number of resampled points along width

$n_1$ — number of resampled points along length.

We denote $\text{Area}(B) = \text{width} \cdot \text{length}$ the area of the bounding box around the region, where width, length, are its dimensions (and the directions along which to rule with the new step). The relationship between $n$ and $n_{BB}$ can be established through the ratio of areas of the region and its bounding box, given that the sampling density is the same:

$$\rho = \frac{n}{\text{Area}(R)} = \frac{n_{BB}}{\text{Area}(B)}$$
If we define $\lambda \overset{\text{def.}}{=} \frac{\text{Area}(\mathcal{R})}{\text{Area}(\mathcal{B})} \leq 1$, then we can relate $n$ and $n_{BB}$ through the proportion of the two areas:

$$n = \lambda n_{BB} \tag{20}$$

To obtain square grid units, we equalize the grid step size in each direction. The ratio between $\text{Area}(\mathcal{B})$ and new number of points $n$, representing the area of the unit square in the grid, can be expressed as the product of the ratios of $\frac{\text{width}}{n_w}$ and $\frac{\text{length}}{n_l}$, representing equal steps in each direction:

$$\frac{\text{Area}(\mathcal{B})}{n_{BB}} = \frac{\text{width} \cdot \text{length}}{n_w \cdot n_l} = k = \sqrt{k} \sqrt{k}.$$

We get:

$$\begin{cases}
\frac{\text{width}}{n_w} = \sqrt{k} \\
\frac{\text{length}}{n_l} = \sqrt{k}
\end{cases} \implies \begin{cases}
\frac{\text{width}}{n_w} = \frac{\text{width}}{\sqrt{k}} \\
\frac{\text{length}}{n_l} = \frac{\text{length}}{\sqrt{k}}
\end{cases}$$

We can now calculate the width-based step:

$$\text{step} = \frac{\text{width}}{n_w} = \frac{\text{width}}{\sqrt{\text{Area}(\mathcal{B})/n_{BB}}} = \frac{\text{Area}(\mathcal{B})}{\frac{n}{\lambda}} = \sqrt{\frac{\lambda \cdot \text{Area}(\mathcal{B})}{N}}.$$

and we thus introduce the step formula:

$$\text{step} = \sqrt{\frac{f \cdot \text{Area}(\mathcal{R})}{N}} \tag{21}$$

5.4 SUBDIVISION

We studied recent work on surface subdivision, and were tempted to apply its basic principle to our reconstruction method after we realized that we can easily obtain a coarse triangular partitioning of the mesh using our triangulation method, applied to the simplified borders. We obtain simplified border from our original region shapes by retaining, from the original region boundaries, only points at the intersection of three or more regions (the corner points). We then connect the corner points in 2D
5.5 LEAST SQUARES APPROXIMATION

In contrast to retriangulation through selected existing points, which is an approximation based on the barycentric interpolation of the resamples, we also implement a least-squares surface approximation method, by solving a linear system to compute...
data-trained control points, as described in Section A.2. In this method, inspired by 
[Farin 2002], pp.278–281, the original data points are the known quantity, as well as
their associated Bernstein polynomials, which describe the locations of the parameter
images \( u_i = (u_i, v_i) \) of the data points within the reference patch, while the control
points \( b_j \) are unknown:

\[
\begin{pmatrix}
    p_0 \\
p_1 \\
p_K \\
\end{pmatrix}_{(K+1) \times 1} = \begin{pmatrix}
    B^0_0(u_0)B^0_0(v_0) & \ldots & B^0_m(u_0)B^n_0(v_0) \\
    \vdots & \ddots & \vdots \\
    B^0_m(u_K)B^0_0(v_K) & \ldots & B^0_m(u_K)B^n_0(v_K) \\
\end{pmatrix}_{K \times (m+1)(n+1)} \begin{pmatrix}
    b_{0,0} \\
    \vdots \\
    b_{m,n} \\
\end{pmatrix}_{(m+1)(n+1) \times 1}. \tag{22}
\]

The system size is

\[(K + 1, 1) = (K + 1, (m + 1)(n + 1)) \star ((m + 1)(n + 1), 1),\]

where \( K + 1 \) is the number of original scanned points \( p_K \), \((m + 1)(n + 1)\) is the
(arbitrary) dimension of the resulting control grid \( b_{m,n} \), expressing the polar degree of
the polynomial that this approximation attempts to fit, and \((K + 1) \times (m + 1)(n + 1)\)
is the size of the matrix of the Bernstein coefficients, calculated with parameter values
of the original points. The control points vector is a linearized version of the geometry,
or design matrix:

\[ B = \begin{pmatrix}
    b_{0,0} & \ldots & b_{0,n} \\
    \vdots & \ddots & \vdots \\
    b_{m,0} & \ldots & b_{m,n} \\
\end{pmatrix} \]

The Bernstein polynomials capture information about the interpolation coefficients for
the parameter points \( u_k \), one per row, in each of the two dimensions \( u_k \) and \( v_k \), with
respect to a rectangular patch \([r_1; s_1] \times [r_2; s_2] \):

\[ B^m_i(u_k)B^n_j(v_k) = C^i_m u_k^m(1 - u_k)^{m-i} C^j_n v_k^n(1 - v_k)^{n-j}, \quad C^i_m = \frac{n!}{i!(n-i)!}. \]

Indices \( i \) refer to \( u_i \) coordinates within \([r_1; s_1]\), while indices \( j \) — to \( v_j \in [r_2; s_2] \).
We computed the reference patch end values by arbitrarily factorizing the number of
input points.

The control points are computed by solving for \( B \) what we call the forward system
from Equation 22

\[ P = MB, \]
where we labeled $M$ the Bernstein coefficients matrix, and $B$, the unknown control points vector. $P$ is, of course, the vector of input data points. Since matrix $M$ is non-square, hence not invertible, points $B$ are calculated by actually solving instead:

$$
M^T P = M^T M B.
$$

(23)

The new 3D points are obtained from the return system, in which, this time, the unknown vector is the vector of reconstructed points $P''_K$, and the Bernstein matrix is computed with resampled (grid) points as arguments to the Bernstein polynomials:

$$
P'' = M' B,
$$

where $P''$ are the reconstructed approximation points, $C'$ is the Bernstein coefficients matrix, recalculated with the resampled values $u'_i, v'_i$, and $B$ is the control points vector calculated above. Rewritten in matrix form, this system is:

$$
\begin{pmatrix}
  p_1 \\
  \vdots \\
  p_L
\end{pmatrix} = 
\begin{pmatrix}
  B_0(u'_0)B_0(v'_0) & \cdots & B_m(u'_0)B_n(v'_0) \\
  \vdots & \ddots & \vdots \\
  B_L(u'_L)B_L(v'_L) & \cdots & B_m(u'_L)B_n(v'_L)
\end{pmatrix}
\begin{pmatrix}
  b_{00} \\
  \vdots \\
  b_{mn}
\end{pmatrix}
$$

(24)

The output is a set of $L \neq K$ regular 3D points, reproducing in 3D the grid distribution of the resamples in 2D. Just like in the interpolation reconstruction case, here, too, by specifying the grid size we implicitly control the size of the reconstruction, i.e. the point reduction ratio.

This method presents, however, numerical instabilities, which the authors of Farin [2002] suggest are fixable through additional boundary conditions for each domain, such as imposing that the surface twist is zero at each corner point. We did not pursue this additional constraint, for fear it would call for further discrete approximations of the twist calculation that would add to the overall program complexity.
RETRIANGULATION

We achieve the retriangulation in several different ways. We experiment with decimation in the parameter plane (in the real meaning of the term, actually supressing every \( n^{th} \) point from the flattened region), followed by a Delaunay triangulation of the remaining original points, since this triangulation will restructure itself adding Steiner points as needed to achieve good triangle shapes. This happens in order for the triangulation to preserve the Delaunay property (that triangle vertices be barycenters of Voronoi regions, each of which contains points that are closer to itself than they are to any other region). This is the only code we borrowed, with due credit at the end of this chapter. This triangulation gives good results, but limits the simplification exactly due to the necessity to introduce new points. We observed a clear lower threshold on the number of triangles resulted from the Delaunay triangulation, no matter how aggressive a reduction factor we specified.

As an alternative to this off-the-shelf retriangulation, we triangulate the rectangular grid, and provide a triangulation of the border gap. Our algorithm was inspired by the theory on polygon triangulations presented in O’Rourke [2005], which we extend by implementing a triangulation of polygons with holes and by reinterpreting the previously sequential monotone partitioning algorithm into a simpler, more reliable recursive version.

6.1 OUR CONTRIBUTION

The border gap forms a topological “doughnut” shape around the center grid. In order to obtain a triangulation, we separate this gap into polygons. We create four sections. To obtain a consistent, non-intersecting triangulation, each section needs to be further partitioned into monotone polygons. Monotone polygons are polygons in which the vertices can be divided into two monotone chains, one on either side of a monotonicity axis. A monotone chain preserves an increasing ordering of vertex coordinates along this axis, as vertices are traversed in connectivity (edge) order in its direction. Our four-section approach results in two pairs of polygons: two polygons (top and bottom) along an Ox axis, and two (left and right) along an Oy axis. These polygons are then

---

1 As presented, the algorithm in O’Rourke [2005] did not deal with multi-depth cusps (“cusps within cusps”). Our algorithm does.
partitioned to achieve the monotonic property for each of the resulting partitions, which can be triangulated without edge intersections.

Our contribution is the polygon accounting setup and the doughnut polygonal shape partitioning, in order to apply partitioning to a shape that doesn’t support it natively. Another contribution is the treatment of multicusps, which arise abundantly in our border shape, and which cannot be separated adequately by the sequential version of the partitioning algorithm as outlined in O’Rourke [2005]. We resolve this problem by implementing a recursive version of this algorithm.

6.2 QUARTERING THE BORDER GAP

A reliable triangulation of the irregular, non-convex border gap starts with obtaining four sections. We compute four pairs of corresponding inner and outer border points, according to a heuristic that will result in points approximately placed at the NW, NE, SE, and SW cardinal points of a compass:

\[
\begin{align*}
    x_{NW} &= (\min X, \max Y) \\
    x_{NE} &= (\max X, \max Y) \\
    x_{SE} &= (\max X, \min Y) \\
    x_{SW} &= (\min X, \min Y)
\end{align*}
\]  

This heuristic is applied to the inner border. The corresponding point on the outer border is chosen as the geometrically closest point. The reason the compass points were chosen on the inner border, even though the closest point search space would have been much reduced in the opposite case (i.e. outer border compass points, inner border closest point to each), is that the inner polygon being composed of squares, finding these points is much more accurate. By comparison, the outer border being extremely irregular, the heuristic would not work, for example, in cases with regions

Figure 16.: Sample polygon partitioning and triangulation: cusp detection and trapezoidalization (left), partitioning (middle), and triangulation (right)
diagonally shaped, where the point \((\text{maxX}, \text{maxY})\) is too far to the left of where the corresponding inner border point would be (much like the tilt of the island of Manhattan, where the Upper West Side is actually to the east of the Lower East Side).

### Algorithm 3 Compute Four Sections

1. \textbf{vector(Polygon)} fourSections; // Declare return variable
2. \textbf{Polygon} outerPolygon ← \textbf{POLYGON}(outerBorder);
3. \textbf{Polygon} innerPolygon ← \textbf{POLYGON}(innerBorder);
   // Compute inner border compass points
4. \textbf{Point} innerNW = \text{MIN}(innerPolygon.points, XminusYorder);
5. \textbf{Point} innerNE = \text{MAX}(innerPolygon.points, XplusYorder);
6. \textbf{Point} innerSE = \text{MAX}(innerPolygon.points, XminusYorder);
7. \textbf{Point} innerSW = \text{MIN}(innerPolygon.points, XplusYorder);
   // Make chains NW \rightarrow NE, NE \rightarrow SE, SE \rightarrow SW, SW \rightarrow NW
8. \textbf{Points} innerChainN ← innerPolygon.MAKECHAIN(ascending, innerNW, innerNE);
9. \textbf{Points} innerChainE ← innerPolygon.MAKECHAIN(ascending, innerNE, innerSE);
10. \textbf{Points} innerChainS ← innerPolygon.MAKECHAIN(ascending, innerSE, innerSW);
11. \textbf{Points} innerChainW ← innerPolygon.MAKECHAIN(ascending, innerSW, innerNW);
   // Calculate the outer closest point to each of these
12. \textbf{Point} outerNW ← outerPolygon.points.CLOSESTPOINT(innerNW);
13. \textbf{Point} outerNE ← outerPolygon.points.CLOSESTPOINT(innerNE);
14. \textbf{Point} outerSE ← outerPolygon.points.CLOSESTPOINT(innerSE);
15. \textbf{Point} outerSW ← outerPolygon.points.CLOSESTPOINT(innerSW);
   // Find the quarter chains between these outer points
16. \textbf{Points} outerChainN1 ← outerPolygon.MAKECHAIN(ascending, outerNW, outerNE);
17. \textbf{Points} outerChainN2 ← outerPolygon.MAKECHAIN(descending, outerNW, outerNE);
18. \textbf{if} outerChainN1.CONTAINSPOINT(outerSE) \textbf{then}
19. \quad \textbf{Points} outerChainN ← outerChainN2;
20. \textbf{else}
21. \quad \textbf{Points} outerChainN ← outerChainN1.REVERSE();
22. \textbf{end if}
   // Omitting the other three chains...
23. \textbf{Points} chainN.AddPoints(outerChainN); chainN.AddPoints(innerChain);
24. \textbf{Polygon} polygonN ← \textbf{POLYGON}(chainN);
   // Omitting the other three polygons...
25. \textbf{Add} the polygons to the return polygon vector
26. fourSections.Push_Back(polygonN);
   // Omitting the other three polygons...
27. \textbf{return} fourSections;
Algorithm 3 describes the partitioning of the border gap into four closed polygons. Statement 18 is necessary to make sure the correct chain is built between the specified endpoints. There will be two possible loops between the endpoints, the correct one, and the one all the way around the other three sections. Applying a shorter versus longer loop heuristic to determine the correct one does not work, due to the very irregular borders: the longer loop may very well be the one needed to complete the quarter polygon in question. Instead, we ask that the loop we choose does not include the diagonally opposed compass point. This works in most cases, but we have found exceptions to that, too.

6.3 Trapezoidalization

The polygons resulted from the quartering of the border gap are non-monotone. This means that vertex connectivity does not induce vertex coordinate increase with respect to some monotonicity axis. Intuitively, we call the vertices with out-of-sequence coordinates return vertices; the literature calls them cusps. The idea is to partition each non-monotone polygon into monotone sub-polygons, by connecting return (cusp) vertices to the next available original polygon vertex “across” the monotonicity axis, in the direction of the return, as shown in Figure 16 in red lines. To detect cusps, as well as to connect with the appropriate original vertex (called support vertex), a trapezoidalization of the polygon must be computed, also as shown in Figure 16 in blue lines.

Figure 17.: The problem of multicusps: without the parity check (left), with parity check (right). This is region 80 from the bunny model.

The trapezoidalization constructs sweep lines through each original polygon point, finds all intersections with polygon edges, and determines, based on the number of edge intersections on either side of the point, whether it is a cusp or not. The ideal

\footnote{This sample polygon was constructed to test the method. It did not arise in any of the real cases, which are more complex.}
case, of single cusps, will generate single edge intersections on both sides. Regular
vertices will generate single intersections on only one side. Multiple cusps will gener-
ate multiple intersections possibly on both sides, but this time, their parity matters: if
there are odd intersections on both sides, the cusp is a “regular” cusp (a return from
the natural progression of vertices on that chain); but if there are even intersections
on either side, the cusp is an “inverted” cusp (a return from the return!!!). In that
case, it is skipped. Algorithm 4 shows the list of events for all the polygon vertices,
previously axis-ordered: the sweep line generation, the edge intersections generation,
the counting of the edge intersections, and the selective addition to the cusp set of the
valid cusps with their sweep lines. The return value is the set of all triplets (vertex,
sweepLine, cuspStatus).

In a second pass over the cusps alone, the algorithm determines the cusp direction
based on the relevant coordinate values of the prior and subsequent vertices in con-
nectivity order, drawing the partition lines at the same time, that is, drawing the con-
nection of each cusp with the opposing support vertex in the trapezoid determined by
the corresponding sweep lines. This stage produces a set of partition lines that should
be used to generate monotone sub-polygons in a later sequential step. The generation
of partition lines is illustrated in Algorithm 5 which goes over the cusps, extracts each
cusp’s previous and next vertex from the polygon in connectivity order, and, based on
coordinate differences, determines the direction of the cusp and which support vertex
it should be connected to. This algorithm returns the polygon-wide set of partition
lines and the final cusp set. The cusp set is not final until this stage because of cusps
determined by an edge that happens to be parallel with the monotonicity axis. Such a
connection should be disallowed. After the partition lines are created, they are fed to
a polygon construction algorithm, which takes the original, non-monotonic polygon
together with its set of partition lines as arguments, and returns the multiple mono-
tonic polygons determined by the partition lines. Closed polygons are necessary as
closed edge sequences in the triangulation algorithm.

We initially saw the stage of forming monotone polygons out of a non-monotone
polygon and its partition lines as a naturally recursive algorithm. Later, we realized
that by including in this recursion the step of cataloguing the cusps and forming the
partition lines would reduce substantially the complexity of the shape of the polygon
undergoing the monotone partitioning: instead of computing partition lines sequen-
tially, then building the corresponding polygon, we build a recursion which examines
the cusps, draws the partition lines and builds monotone polygons, all in one func-
tion. This improvement brought speed and reliability to the border gap triangulation,
which is fully functional even in the craziest of the border gap configurations.
Algorithm 4 Line Sweep

sweepSet ← vector<struct vertex(sweepLine, isCusp, isSupport)>;
axisOrderedPolygonPoints ← ORDERPOLYGONPOINTS();
for vertex ∈ axisOrderedPolygonPoints do
    vertex.SweepLine ← LINE(direction, vertex);
sweepPoints = EDGEINTERSECTIONS(sweepLine);
    for sweepPoint ∈ sweepPoints do
        distance ← ∥vertex.Y() − sweepPoint.Y∥;
        if distance < 0.0 then
            aboveSet.ADD(sweepPoint);
        else
            belowSet.ADD(sweepPoint);
        end if
    end for
if aboveSet.EMPTY() && belowSet.EMPTY() then
    vertex.isCusp ← false;
    vertex.isSupport ← false;
else if aboveSet.EMPTY() then
    trapezoidLines.ADD(Line(vertex, aboveSet.RIGHTBELOW()));
    vertex.isCusp ← false;
    vertex.isSupport ← true;
else if belowSet.EMPTY() then
    trapezoidLines.ADD(Line(vertex, aboveSet.RIGHTABOVE()));
    vertex.isCusp ← false;
    vertex.isSupport ← true;
else
    if aboveSet.SIZE()%2 == 1 && belowSet.SIZE()%2 == 1 then
        vertex.isCusp ← true;
        vertex.isSupport ← true;
        cuspSet.ADD(vertex);
    else
        vertex.isCusp ← false;
        vertex.isSupport ← false;
    end if
    end if
sweepSet.ADD(vertex);
end for
return sweepSet;

Algorithm 6 shows the computation of monotone polygons from a random polygon and its set of partition lines in its original version, which uses recursion on a precomputed partition line set. The cusp finding and the sweep line computation were done in a separate function, shown in Algorithm 5, based on the number of intersec-
Algorithm 5 Make Partition Lines

for vertex ∈ sweepSet do
    if vertex.isCusp == true then
        preVertex ← ComputeConnectedPrevious(vertex);
        while preVertex.X() == vertex.X() ∨ preVertex.isSupport() == false do
            skip preVertex;
            preVertex ← ComputeConnectedPrevious(vertex);
            if preVertex == sweepSet.begin() then
                preVertex.isSupport() ← true;
                break;
            end if
        end while
        postVertex ← ComputeConnectedNext(vertex);
        while postVertex.X() == vertex.callX() ∨ postVertex.isSupport() == false do
            skip postVertex;
            postVertex ← ComputeConnectedNext(vertex);
            if postVertex == sweepSet.end() then
                postVertex.isSupport() ← true;
                break;
            end if
        end while
        if preVertex.X() > vertex.X() ∧ postVertex.callX() ≥ vertex.X() then
            partitionLine = Line(vertex, preVertex);
        else if preVertex.X() < vertex.X() ∧ postVertex.callX() ≤ vertex.X() then
            partitionLine = Line(vertex, postVertex);
        else if preVertex.X() == vertex.X() ∧ postVertex.callX() > vertex.X() then
            partitionLine = Line(vertex, postVertex);
        else
            partitionLine = Line(vertex, preVertex);
        end if
        cuspSet.ADD(vertex);
        partitionLines.ADD(partitionLine);
    end if
end for
return cuspSet, partitionLines;

6.3 TRAPEZOIDALIZATION

The vertices generating intersections on both sides of themselves, along the sweep line (perpendicular onto the monotonicity axis), are cusps. The vertices generating intersections on only one side are regular vertices. Double cusps are detected through the odd-even counting of their intersections. In Figure 18, the cusp to the right end of the purple partition line has 1 intersection above, and would have 3 intersections below, in a serial imple-
Algorithm 6 Make Monotone Polygons

if polygon.Size() < 3 then
    return
end if
if partitionLines.Empty() || polygon.Size() == 3 then
    // Base case: push polygon into the subPolygons set
    subPolygons.Add(polygon);
end if
if polygon.Size() > 2 then
    MakeMonotonePolygons(subPolygon1, partitionLines1);
end if
if polygon.Size() > 2 then
    MakeMonotonePolygons(subPolygon2, partitionLines2);
end if
return
6.3 Trapezoidalization

— therefore it would be a double cusp. But the vertex at the opposing tip in the triangle formed with the purple base and two blue edges would have 2 upper intersections and 2 lower intersections — therefore it would not be declared as a cusp, even though it produces multiple intersections on both sides. This consideration is not presented in [O'Rourke 2005].

Figure 18.: Sample double-cusp polygon with trapezoidalization (a), monotone polygon partitioning (b), monotonicity chains (c), and final triangulation (d)

Algorithm 7 shows the improvement to make the recursion include the cusp examination, the partition line drawing, and the polygon forming all in one function. This version was inspired by the fact that each partition line triggers the split of the current polygon into two subpolygons, and since the two events are associated, they could be merged into one recursion. None of these implementations are in the original description from [O'Rourke 2005], where, nevertheless, the essential elements to understand the process of triangulation through trapezoidalization are laid out as a general framework, from where any particular vision about the data structures, accounting, and implementation can proceed. The sequential version illustrated in [O'Rourke 2005]
is presented here because it was our original implementation, and to acquaint the reader with the idea of partitioning across the polygon starting from the cusp vertices, to eliminate non-monotonic chains.

The challenges of either implementation were daunting, given the many corner cases actually encountered in a complex polygon shape (edges overlapping the sweep line; the double cusps; cases where the eligibility of regular vertices as support vertices generates incorrect partition lines), as well as given the need for edge orientation consistency, which was especially hard to implement at the stage of the initial partitioning of the border gap in four sections.

Algorithm 7 Recursive Make Monotone Polygons

```plaintext
axisOrderedPolygonPoints ← ORDERPOLYGONPOINTS();
for vertex ∈ axisOrderedPolygonPoints do
    sweepLine = LINE(axis, vertex);
    sweepPoints = EDGEINTERSECTIONS(sweepLine);
    for sweepPoint ∈ sweepPoints do
        distance = ∥sweepPoint.X() - vertex.X()∥;
        if distance < 0.0 then
            negativeSet.Add(sweepPoint);
            negativeNext ← sweepPoint;
        else
            positiveSet.Add(sweepPoint);
            positiveNext ← sweepPoint;
        end if
        if negativeSet.Size() % 2 == 1 && positiveSet.Size() % 2 == 1 then
            cuspSet.Add(vertex);
            cuspType ← vertex.CUSP_TYPE(); // Left or right
            support ← vertex.FIND_SUPPORT(cuspType);
            partitionLine ← LINE(vertex, support);
            partitionSet.Add(partitionLine);
            pair (subPolygon1, subPolygon2) ← polygon.MAKE_SUBPOLYGONS(partitionLine);
            RecursiveMakeMonotonePolygons(subPolygon1);
            RecursiveMakeMonotonePolygons(subPolygon2);
        end if
    end for
    subPolygons.Add(polygon); // This adds an indivisible polygon
end for
return
```

6.4 MONOTONE POLYGON TRIANGULATION

Triangulating each monotone polygon presumes the existence of two distinct monotone chains (joined at the ends). The triangulation algorithm starts from the first point on one of the two monotone chains and makes connections with same chain points as long as the angles at these points are not reflex, to ensure visibility to the connecting vertex. If a reflex angle is encountered, the connection is made with points from the opposite chain.

Figure 19. Monotone polygon triangulation. Note the path the triangulation takes around reflex vertex D.

Figure 20. The polygon of Figure 19 run through Algorithm 8. The chain endpoints are shown in orange.

Algorithm 8 describes the process for the case where the monotonicity axis is horizontal. The key to the algorithm is the fact that, when a reflex angle is encountered, it is not removed from the advancing end of the list L. As the next vertex in axis order
Figure 21.: Example polygon exhibiting double cusps: (a) phase 1 determines the cusps (yellow squares); (b) phase 2 traces sweep lines through all polygon vertices, determining the connections the cusps should make to avoid polygon edge intersections; (c) phase 3 isolates the monotone sub-polygons constructed with the connections at (b), and collects them as individual polygon objects; (d) phase 4 constructs a triangulation for each monotone sub-polygon.

is added to L, and as this new vertex becomes the current point to attempt to add to the triangulation, the triangle formed will include L’s first, second elements, then, skipping the reflex vertex, the current vertex P. This is illustrated in Figure 19 where the red arrows show the direction of the triangulation, as it finds next axis-ordered vertices on the opposite chain, the blue arrow shows the triangulation skipping point E and moving on to the next available vertex along Ox, point I (which happens to be on the opposite chain), then the yellow arrow shows how E, previously skipped, is now reconsidered in a further triangle, △DIE.
Figure 22: Border gap sectioning, monotone sub-polygonalization, and triangulation for the hemisphere, with $R=10$ and $F=5$ ($F$ is the reduction factor, which governs the density along the inner regular border).
Algorithm 8 Monotone Polygon Triangulation

```
Triangles polygonTriangles;
if polygon.points.Size() < 3 then
  return
else if polygon.points.Size() == 3 then
  polygonTriangles.Push_Back(Triangle(polygon));
  return
else
  Points V ← polygon.points.OrderByAxis(Ox);
  // L will hold the advancing end of the wave of points that will triangulate
  Points L; // Initialize L with first two points of V
  L.Push_Back(V.Front()); V.Pop_Front(); L.Push_Back(V.Front()); V.Pop_Front();
  while V.Empty() == false do
    Point P ← V.Front();
    if polygon.InSameChain(P, L) == false then
      while L.Size() > 1 do
        if polygon.UpperChain(P) == true then
          Angle A;
          A ← Angle(L.Back(), L.at(L.Size()-2), P, CLOCKWISE);
          if A > π then
            break;
          end if
          polygonTriangles.Push_Back(Triangle(L.Back(), L.at(L.Size()-2), P));
        else
          A ← Angle(L.Back(), L.at(L.Size()-2), P, COUNTERCLOCKWISE);
          if A > π then
            break;
          end if
          polygonTriangles.Push_Back(Triangle(L.Back(), P, L.at(L.Size()-2)));
        end if
        L.Pop_Front();
      end while
      L.Push_Back(P);
    else
      while L.Size() > 1 do
        Angle A;
        if polygon.UpperChain(P) == true then
          A ← Angle(L.Back(), L.at(L.Size()-2), P, CLOCKWISE);
          if A > π then
            break;
          end if
          polygonTriangles.Push_Back(Triangle(L.Back(), L.at(L.Size()-2), P));
        else
          A ← Angle(L.Back(), L.at(L.Size()-2), P, COUNTERCLOCKWISE);
          if A > π then
            break;
          end if
          polygonTriangles.Push_Back(Triangle(L.Back(), P, L.at(L.Size()-2)));
        end if
        L.Pop_Back();
      end while
      L.Push_Back(P);
    end if
  end if
  return polygonTriangles;
```

Lifting the new points into 3D space is achieved through linear interpolation within the faces of the original mesh. The interpolation coefficients are the barycentric coefficients with respect to the 3D triangle whose 2D parameter image hosts each new point. This requires a method of first finding the 2D triangle enclosing the new point in parameter space. We implemented this search through an $O(n)$ test, within each region, of which triangle $\triangle ABC$ satisfies

$$\text{Area}(\triangle A'B'P) + \text{Area}(\triangle B'C'P) + \text{Area}(\triangle C'A'P) = \text{Area}(\triangle A'B'C')$$

with respect to the new point $P$. The primed letters are a notation convention for parameter points. When the area test fails to produce a parent triangle, we default to a closest triangle check. (Unfortunately, this sometimes creates reconstruction artifacts.)

The 2D triangle $\triangle A'B'C'$ is related one-to-one to its 3D parent triangle through a hash table computed during parametrization. With the three 3D points $A, B, C \in \mathbb{R}^3$ corresponding to $A', B', C' \in \mathbb{R}^2$, we can compute the interpolation

$$P'' = A\alpha + B\beta + C\gamma,$$

where $\alpha, \beta, \gamma$ are the same as $P$’s barycentric coordinates within triangle $\triangle A'B'C' \subset \mathbb{R}^2$, and were previously computed as solutions to the system:

$$\begin{cases}
\alpha u_{A'} + \beta u_{B'} + \gamma u_{C'} = u_P \\
\alpha v_{A'} + \beta v_{B'} + \gamma v_{C'} = v_P \\
\alpha + \beta + \gamma = 1
\end{cases} \tag{26}$$

These coordinates place the new points on the faces of the old triangulation. Since the old triangulation was dense, this approximation is sufficient and gives very good results.
We calculate errors for the parametrization alone, reconstruction alone, and finally for the whole pipeline.

We adopted an error calculation based on three types of known surfaces: the plane, a couple of quadratics (the hyperbolic paraboloid, the elliptic paraboloid), and a cubic (the monkey saddle). These were implemented on a reference triangle \( \Delta r, s, t \in \mathbb{R}^2 \), where we make the distinction between \( r, s \) as real scalar endpoint intervals (encountered during the analysis of the quadrangular, or tensor product, patches) and \( r, s, t \) as points in \( \mathbb{R}^2 \), for the triangular patches (or Bézier triangles).

The synthetic 3D points, along with their triangulation connectivity, are stored in two files, of the same format as the input files for the large scenes we used in this project. These files, \texttt{vertices_<model>.ascii} and \texttt{triangles_<model>.ascii} (where the \texttt{<model>} pattern represents the different surfaces) are input files for our VSA program, which computes, in a single run (no stages or intermediate storage), two output files, one for the resulting point set, the other for the corresponding triangulation, as well as a visual representation in Inventor, the surface \( S' \). We feed our quadratic surface synthetic points, which are the real, reference surface, into our pipeline, and calculate errors according to the schemes presented in the following sections.

8.1 \hspace{1em} \textbf{PARAMETRIZATION ERROR}

Our parametrization method is designed to work well with a linear interpolation type of reconstruction (placing reconstructed points on the faces of the initial mesh, as if interpolating small local planes). To verify the quality of the parametrization, we parametrize three surfaces obtained from coordinate functions applied to an equilateral reference triangle centered in the origin. We first compare this parametrization directly with triangular grid parameter values, computed from the reference triangle. We then feed our parameter values into three types of parametric surface functions (plane, quadratic, cubic), and compare the resulting points with the initial 3D surface points they correspond to, for the same three surfaces, respectively.

\textbf{DIRECT PARAMETRIZATION COMPARISON} \hspace{1em} Since our parametrization is constrained at the boundaries by a projection of the surface border on the parameter plane, we
expect it not to produce an equilateral triangle, but rather a curved triangle, inside which the parameter points are not arranged in a grid, but in a tessellation with quasi-equilateral triangles. This is illustrated in Figure 23.

The error calculation scheme is:

\[
\begin{align*}
\text{synthetic surface} & \quad \rightarrow \quad \text{subdivision grid} \\
& \quad \downarrow \text{compare} \\
\text{synthetic surface} & \quad \rightarrow \quad \text{our parametrization}
\end{align*}
\]

Figure 23: Our parametrization: hyperbolic paraboloid (green), elliptic paraboloid (purple), monkey saddle (orange) vs. the subdivided triangular grid (red).

**Analytic Surface Comparison** Next, we feed our computed parameter points into the analytic expressions of three different types of functions, describing a plane, a quadratic, and a cubic surface, respectively. The output points will thus correspond directly, one for one, to the original 3D input points, which were obtained on a regular grid, by subdividing an equilateral triangle centered in the origin. A comparison of the original 3D points with those computed, using the same functions, from our parameter points, will attest to the quality of the parametrization method.

This comparison holds the reconstruction step fixed, while varying the parameter points it is based on:
8.2 RECONSTRUCTION ERROR

For the purpose of reconstruction error calculation, we hold the parameter points fixed, while varying the method of reconstruction:

resampled parameter grid $\rightarrow$ computed surface $\uparrow$ compare

resampled parameter grid $\rightarrow$ our reconstruction

Using the pre-calculated surface $S$, sampled at triangular grids, as an input mesh, we run this data through our whole pipeline, obtaining a reconstruction. Saving the resampled parameter grid, we run this data through the surface parametric computation, obtaining a valid analytic surface sampled at our multiresolution grid (as opposed to at the subdivision triangles). The parameter points being the same in both cases, the comparison of the 3D output points in each case will attest to the quality of the reconstruction.
Since the reconstruction is linear, we also compare the mean error across surfaces of varying degree, to see how well the linear reconstruction holds for different surfaces. We noticed that, the higher degree, the closer the area fit was. We also noticed the spike artifacts that happen when a resampled point falls over a parameter point, which results in instabilities in the computation of the new 3D location through barycentric coordinates. These results are shown in Figure 25.

Figure 25.: Analytic surface computation from our grid resampled points (white) and our reconstruction from our grid resampled points (colors: red — plane, green — hyperbolic paraboloid, purple — elliptic paraboloid, orange — monkey saddle. Notice the spike appearing in one of the corners of the Monkey saddle.

Figure 26.: Monkey saddle parametrization and grid resampling points (a) and the error in reconstruction using the barycentric coordinate placement method (b).
8.3 Pipeline Error

In this section, we use only quadratics. To measure the error for the whole pipeline, since we are losing the one-to-one point correspondence that would make the comparison meaningful, we have to resort to surface distance between the analytical surface and the parametrized, resampled, and reconstructed one. We would like to know how far are the reconstructed points from the correct, reference surface. For this purpose, we collect our reconstructed points, and feed only their $x$, $y$ coordinates into the implicit form of the hyperbolic paraboloid:

$$z' = x^2 - y^2$$  \hspace{1cm} (27)

and of the elliptic paraboloid:

$$z' = \frac{x^2}{a^2} + \frac{y^2}{b^2}$$  \hspace{1cm} (28)

Saving the output $z$ in each case, we compare it to the third coordinate $z$ from the output file. This shows the distance between the correct surface and the reconstructed one in the direction of the $Oz$ axis.

Figure 27 shows the pipeline surface (computed from the reference surface using our pipeline: parametrization, resampling, reconstruction) overlaid with the reference surface (computed from pipeline output $(x, y)$ pairs through the implicit function describing the same surface — again, the hyperbolic paraboloid, which exhibits the most curvature). From the viewing angle, what shows are the points from each surface that are slightly above the other. Seeing the white (pipeline) mixing with the red (implicit), we notice that both surfaces really blend together, meaning the error should be really small. Visually, there is no noticeable difference. We tabulated the errors, point by point, in Table 5.

The error calculation scheme is:

\[
\begin{align*}
\text{reconstructed surface } (x, y, z) &\quad \rightarrow \quad z \text{ coordinate} \\
\downarrow \text{compare} \\
\text{computed surface (implicit form) } z' = f(x, y) &\quad \rightarrow \quad z' \text{ coordinate}
\end{align*}
\]

The output point file and triangle file are compared against the known surface samples, a subset of which we give in Table 5 below, which shows the $z$-error per point:

$$
\epsilon_z = \frac{|z_{\text{implicit}} - z_{\text{pipeline}}|}{\text{boundingBoxSize}} \times 100\%
$$  \hspace{1cm} (29)

94
8.3 Pipeline Error

Figure 27.: Surface $S$ (white) was computed from pairs $(x, y)$ of our pipeline output points $(x, y, z) \in S'$ (red), using the implicit equation $z = \frac{x^2 - y^2}{10}$. The difference $|z - z'|$ expresses the pipeline error $z_{\text{error}}$, which becomes $\epsilon_z$ expressed as a percent relative to the size of the bounding box along $Oz$.

For completeness, we also compute the vector distance $t$-error and its norm $\|PP'\|$, with $P \in S$ and $P' \in S'$, where $S$ is the reference surface and $S'$ is the pipeline surface, leading to the length error $\epsilon_t$ (per point):

$$\epsilon_t = \frac{t_{\text{error}}}{\|\hat{P}\|} \times 100\%,$$

where

$$\|PP'\| = \sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2}$$

and $\hat{P} = \vec{OP}$,

and the angular error $\epsilon_\theta$ (per point):

$$\epsilon_\theta = \angle(\overrightarrow{PP'}, \hat{P})$$

The length errors $\epsilon_z, \epsilon_t$ are expressed in units used by Inventor. To make them meaningful, we related them to the scale of the model, which we inferred by computing a bounding box of the model. We scaled the $z$-errors to the length of the bounding box along the $Oz$ axis; we scaled the length of the vector difference to the length of the position vector $\hat{P}$ of the point on the reference surface $P \in S$; we calculated the angle of the vector difference with the position vector $\hat{P}$.

These additional errors are not as meaningful in the case of the comparison of the implicit surface versus the pipeline surface, which only differ in $z$, as they are in the cases where the point-to-point correspondence between the reference surface and the
Table 5: Sample errors: z-errors and t-errors for our pipeline against the reference (implicit) surface.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>z′</th>
<th>εz</th>
<th>εt</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5.17839</td>
<td>-1.18965</td>
<td>2.52898</td>
<td>2.54005</td>
<td>0.055328%</td>
<td>0.187896%</td>
</tr>
<tr>
<td>-5.07885</td>
<td>0.46773</td>
<td>2.54509</td>
<td>2.55759</td>
<td>0.0625222%</td>
<td>0.219158%</td>
</tr>
<tr>
<td>-4.02797</td>
<td>-2.21158</td>
<td>1.12919</td>
<td>1.13335</td>
<td>0.0207776%</td>
<td>0.0878009%</td>
</tr>
<tr>
<td>-3.91705</td>
<td>0.267244</td>
<td>1.55083</td>
<td>1.52719</td>
<td>0.118215%</td>
<td>0.561227%</td>
</tr>
<tr>
<td>-3.83635</td>
<td>1.20512</td>
<td>1.31426</td>
<td>1.32653</td>
<td>0.0613335%</td>
<td>0.289696%</td>
</tr>
<tr>
<td>-3.78813</td>
<td>2.44695</td>
<td>0.823696</td>
<td>0.836237</td>
<td>0.0627026%</td>
<td>0.273417%</td>
</tr>
<tr>
<td>-3.76468</td>
<td>3.51201</td>
<td>0.177812</td>
<td>0.18386</td>
<td>0.03024%</td>
<td>0.117396%</td>
</tr>
<tr>
<td>-2.77791</td>
<td>-3.10212</td>
<td>-0.209259</td>
<td>-0.190636</td>
<td>0.0931131%</td>
<td>0.446747%</td>
</tr>
<tr>
<td>-2.71495</td>
<td>-0.927925</td>
<td>0.635743</td>
<td>0.650991</td>
<td>0.0762396%</td>
<td>0.518271%</td>
</tr>
<tr>
<td>-2.61469</td>
<td>0.738216</td>
<td>0.63983</td>
<td>0.629164</td>
<td>0.0533294%</td>
<td>0.382453%</td>
</tr>
</tbody>
</table>

computed one leads to points of different z coordinates, as well, and therefore the vector difference $\vec{PP'}$ has a more significant length and deviation from $\hat{P}$.

In Table 5 we show the first ten non-zero error entries. We omitted the points $P'$ that are identical to the reference (implicit) points $P$, due to the fact that our fixpoint border algorithm will replace in 3D the reconstructed border with the original one. We thus eliminate even the matches that may be due to correct reconstructions. The purpose of this table is to give an idea of the error size for a surface containing high curvature variations, such as our quadratic.
9.1 Grid Triangles

This method is our original idea of gridding and retriangulating in parameter space. In our experiments, it worked far better in the large scene setting, where there are sizable regions of almost coplanar data, which can be easily gridded and lifted in 3D without distortion using the barycentric location method. Another reason it worked well in this setting was the fact that the region boundaries are themselves almost coplanar with the interior, thus constraining the parametrization to the LSP projection did not add to the distortion. Finally, the data is very irregular, and most of the efficiency of our method translates into a regular gridding of the interior points, even though the boundary points remain as jagged as originally in the scan input files. The gain is that they are properly detected by our segmentation algorithm, and the presence of regions structures the scan files into self-contained units within which even random decimation cannot remove major features.

Regarding the use of the different error metrics for the segmentation algorithm, our own error \( E_{\text{Wise}} \) works best for buildings. The competing error metric, \( E_{\text{Sander}} \), favors more evenly distributed regions and is thus more suitable for round models without holes, such as the hemisphere. We used both error metrics, each for the models it worked best with, however we mostly used our metric.

<table>
<thead>
<tr>
<th>Model</th>
<th>Regions</th>
<th>Factor</th>
<th>IN Triangles–Points</th>
<th>OUT Triangles–Points</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hemisphere</td>
<td>10</td>
<td>5</td>
<td>1,563–803</td>
<td>749–398</td>
<td>om2.279s</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>5</td>
<td></td>
<td>998–520</td>
<td>om2.754s</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td></td>
<td>589–318</td>
<td>om2.726s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td></td>
<td>436–239</td>
<td>om2.939s</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2</td>
<td></td>
<td>1,347–697</td>
<td>om2.994s</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
<td></td>
<td>2,403–1,225</td>
<td>om3.494s</td>
</tr>
<tr>
<td>bunny</td>
<td>50</td>
<td>5</td>
<td>69,451–35,947</td>
<td>23,520–11,967</td>
<td>8m7.012s</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>5</td>
<td></td>
<td>16,138–8,398</td>
<td>7m14.193s</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>10</td>
<td></td>
<td>27,304–13,918</td>
<td>4m16.321s</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>5</td>
<td></td>
<td>19,289–9,962</td>
<td>3m39.304s</td>
</tr>
<tr>
<td>fandisk</td>
<td>16</td>
<td>10</td>
<td>23,964–11,984</td>
<td>6,979–3,557</td>
<td>3m17.432s</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>5</td>
<td></td>
<td>4,490–2,375</td>
<td>2m33.806s</td>
</tr>
<tr>
<td>building</td>
<td>50</td>
<td>5</td>
<td>50,000— — —</td>
<td>14,835–8,876</td>
<td>15m27.392s</td>
</tr>
</tbody>
</table>

Table 6.: Grid triangles performance includes all phases, plus image capturing time.
Figure 28.: Fandisk, grid triangles reconstruction: 15 regions with 20 (above) and 5 (below) reduction factors

(a) $R=5$, $F=10$
(b) $R=5$, $F=4$
(c) $R=5$, $F=3$
(d) $R=5$, $F=2$

Figure 29.: Hemisphere, grid triangles reconstruction.
9.1 GRID TRIANGLES

Figure 30.: Bunny, grid triangles reconstruction with various reduction factors.
Figure 31.: Building 50K detail, grid triangles reconstruction with various reduction factors.
(a) Front

(b) Back.

Figure 32.: Armadillo, grid triangles reconstruction: $R=2000$, $F=20$. 
In this method we rely on the fact that our segmentation will create region borders at features. Therefore, a random removal of points in parameter space, followed by a Delaunay triangulation, will not remove features, even though the triangulation is only optimized for triangle shape and for connecting with all border points, and thus will tend to create large inner triangles and smaller border triangles, missing potential inner features (even with our barycentric reconstruction, which guarantees that the reconstructed points will be placed on the surface, there may be missed interior features). However, either most features will have generated region borders, or they may be small enough to be safely ignored.

<table>
<thead>
<tr>
<th>Model</th>
<th>Regions</th>
<th>Factor</th>
<th>IN</th>
<th>OUT</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>hemisphere</td>
<td>5</td>
<td>100</td>
<td>1,563–803</td>
<td>494–268</td>
<td>om1.666s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td></td>
<td>670–356</td>
<td>om1.792s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td></td>
<td>744–392</td>
<td>om1.964s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td></td>
<td>848–445</td>
<td>om1.896s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2</td>
<td></td>
<td>1,048–545</td>
<td>om1.859s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td></td>
<td>1,714–878</td>
<td>om1.950s</td>
</tr>
<tr>
<td>bunny</td>
<td>50</td>
<td>100</td>
<td>69,451–35,947</td>
<td>16,024–7,985</td>
<td>3m55.433s</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>5</td>
<td></td>
<td>26,183–13,094</td>
<td>3m55.835s</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>4</td>
<td></td>
<td>29,129–14,537</td>
<td>3m58.539s</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>5</td>
<td></td>
<td>27,998–14,109</td>
<td>2m7.049s</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>4</td>
<td></td>
<td>30,619–15,419</td>
<td>2m10.031s</td>
</tr>
<tr>
<td>horse</td>
<td>50</td>
<td>5</td>
<td>96,966–48,485</td>
<td>21,478–11,378</td>
<td>12m28.207s</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>5</td>
<td></td>
<td>39,813–20,083</td>
<td>4m9.113s</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>10</td>
<td></td>
<td>32,305–16,329</td>
<td>4m8.191s</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>50</td>
<td></td>
<td>27,817–14,085</td>
<td>4m9.297s</td>
</tr>
</tbody>
</table>

Table 7.: Decimation Delaunay performance includes all phases, plus capturing time.

Figure 33.: Horse: Decimation Delaunay method. Both reconstructions have a small spike.
9.2 Decimation in Parameter Space and Delaunay Triangulation

Figure 34: Hemisphere: Decimation Delaunay method with various decimation factors. The Delaunay triangulation rearranges triangles close to the apex (d).

Figure 35: Bunny: Decimation Delaunay method with various decimation factors. At F=100 the Delaunay triangulation has hit a lower bound on number of $\Delta$s.
9.3 Least squares approximation from data-trained control points

This method is based on the theory presented in Section 5.5. The data points and the parameter points in 2D space (calculated with our tangent formula), will train a control net. This control net — characterizing the surface as if it contained polarized function values at reference interval endpoints — is then used with the new point locations in parameter space to calculate the 3D locations corresponding to these points. We implement this method, too, one region at a time. This method generates directly 3D points, which therefore cannot be triangulated, since the correspondence between the reconstructed points and their 2D pre-image is lost, as the matrix computation produces them in an order uncorrelated with their ordering in the Bernstein matrix (even though this follows the rasterized order of the grid and its triangulation).

<table>
<thead>
<tr>
<th>Model</th>
<th>Regions</th>
<th>Control Net</th>
<th>Factor</th>
<th>IN Triangles-Points</th>
<th>OUT Points</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>hemisphere</td>
<td>16</td>
<td>5</td>
<td>$4 \times 4$</td>
<td>1,563–803</td>
<td>216</td>
<td>0m3.063s</td>
</tr>
<tr>
<td>bunny</td>
<td>50</td>
<td>5</td>
<td>$3 \times 3$</td>
<td>69,451–35,947</td>
<td>7,675</td>
<td>4m48.927s</td>
</tr>
<tr>
<td>fandisk</td>
<td>16</td>
<td>5</td>
<td>$4 \times 4$</td>
<td>23,964–11,984</td>
<td>2,475</td>
<td>3m1.439s</td>
</tr>
<tr>
<td>horse</td>
<td>100</td>
<td>5</td>
<td>$4 \times 4$</td>
<td>96,966–48,485</td>
<td>11,721</td>
<td>5m16.408s</td>
</tr>
</tbody>
</table>

Table 8: Bernstein method. All phases included, plus image capturing user time.

Figure 36.: Bernstein reconstruction. All models have 16 control points per region.
This method was implemented with a weight factor of 1 all around, meaning that we simply used a 1-4 triangular subdivision in parameter space and reconstructed the subdivision points using the barycentric method without either introducing weights, or using the original points as feedback for vertex rearrangement during each subdivision stage, as performed in many implementations. This means that the subdivision is complete through its final resolution in 2D, as opposed to weighing each round of points from intermediate subdivision levels to obtain 3D points. This ensures that no significant (and advancing with each subdivision level) shrinkage relative to the original model occurs. For the resampling along the borders, we identified corner points (where 3 or more regions meet) and traced straight lines between them in parameter space. The resulting simplified border fragments were subdivided in parameter space, then reconstructed using the barycentric method. The 3D borders were chosen to coincide across adjacent regions, by picking one or the other of the 3D border reconstructions obtained for the two neighboring regions. Otherwise, the reconstruction of the same points within disjoint enclosing triangle sets (parent triangles) from adjacent regions would have resulted in different reconstructions, as illustrated in Figure

<table>
<thead>
<tr>
<th>Model</th>
<th>Regions</th>
<th>Levels</th>
<th>IN</th>
<th>OUT</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Triangles–Points</td>
<td>Triangles–Points</td>
<td></td>
</tr>
<tr>
<td>hemisphere</td>
<td>15</td>
<td>5</td>
<td>1,563–803</td>
<td>47,104–23,682</td>
<td>om28.974s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>10,240–5,185</td>
<td>10,240–5,185</td>
<td>om5.999s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>2,560–1,313</td>
<td>2,560–1,313</td>
<td>om3.386s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>640–337</td>
<td>640–337</td>
<td>om2.848s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2</td>
<td>160–89</td>
<td>160–89</td>
<td>om2.690s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>40–25</td>
<td>40–25</td>
<td>om2.678s</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>10–8</td>
<td>10–8</td>
<td>om2.832s</td>
</tr>
<tr>
<td>fandisk</td>
<td>16</td>
<td>5</td>
<td>23,965–11,985</td>
<td>23,965–11,985</td>
<td>2m59.449s</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>53,248–26,626</td>
<td>53,248–26,626</td>
<td>2m2.825s</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>13,312–6,658</td>
<td>13,312–6,658</td>
<td>1m48.097s</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3,328–1,666</td>
<td>3,328–1,666</td>
<td>1m47.470s</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>832–418</td>
<td>832–418</td>
<td>1m45.993s</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>208–106</td>
<td>208–106</td>
<td>1m44.338s</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>52–28</td>
<td>52–28</td>
<td></td>
</tr>
<tr>
<td>bunny</td>
<td>100</td>
<td>3</td>
<td>69,452–35,948</td>
<td>23,809–5,952</td>
<td>3m14.656s</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>5,953–1,498</td>
<td>5,953–1,498</td>
<td>3m6.322s</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1,489–381</td>
<td>1,489–381</td>
<td>3m2.615s</td>
</tr>
<tr>
<td>armadillo</td>
<td>1000</td>
<td>2</td>
<td>345,945–172,975</td>
<td>111,087–56,267</td>
<td>3m49.884s</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>111,087–56,267</td>
<td>111,087–56,267</td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Subdivision method.
Figure 37.: Bunny model: original borders (white) and subdivision domains (red).

Figure 37 illustrates the original 3D borders (in white) versus the simplified 3D straight lines drawn between corner points. The top figure shows a 0 subdivision level (no subdivision) with only triangular base domains (in red), superimposed over the regions (same color for the same region). The figure on the bottom shows a 3 subdivision level, as well as the fact that no significant shrinkage occurs: the original borders do not appear “far away” from the reconstructed model.
Figure 38: Hemisphere uniform subdivision, R=5.

Figure 38 illustrates six levels of subdivision for the hemisphere model segmented in 5 regions. The base level divides each region (simplified to a rectangle after border straightening) into two large triangular domains. These domains are subdivided using a uniform, regular 1-4 subdivision, and the new points are reconstructed in 3D through parent triangle location. Due to location on the surface of the original triangulated model, no shrinkage in the resulting surface occurs, except for the original loss through the input approximation of data as triangular facets, as opposed to a curved surface. The errors, very small, were calculated for each phase of our reconstruction system using a synthetic polynomial surface in our Section 8. Figures 39, 40, and 41 show similar results for the fandisk, bunny, and armadillo models, respectively.
9.4 Uniform Subdivision

Figure 39.: Fandisk uniform subdivision, R=16.
Figure 40. Bunny uniform subdivision, $R=50$.
9.4 **UNIFORM SUBDIVISION**

Figure 41.: Armadillo uniform subdivision, $R=1000$. 

110
For the larger scenes, we used a collection of scans from the Great Hall of Shepard Hall of the City College of New York. There are 30 scans of various sizes, starting with one end of the room, where: the back wall forms an entire scan (scan 3), consisting of 927,284 triangles and 491,593 points; each additional scan is a side-to-side slice of the room; and the front wall forms scan 26, counting 1,009,265 triangles and 531,613 points.

We process the scans sequentially. Each scan is segmented in an outer loop segmentation scheme: either a *sequential input triangle file segmentation* (10,000 entries from the triangle data file at a time, in the order in which they appear in the file), or using our *segmentation algorithm* (with a number of regions depending on the size of the input triangles.(SCAN_NUMBER) file, for instance 100 for a 1,000,000-triangle file representing a large scan slice). The sequential scheme does not work well, because the triangles are ordered in concentric circles in the scans, as illustrated in Figure 42, so we would be dealing with circular stripes, thus diminishing the benefit of later using segmentation on each one. The advantage of using a segmentation scheme is evident in this case, where it has a structuring effect on unstructured data, or on data that was restructured in an earlier stage of processing to suit a different purpose (in this case, we suspect the circular structure resulted from the application of a registration algorithm).

Figure 42.: Scan 3 of the Great Hall in its original triangle sequencing from the data files. Each stripe represents a sequential block of 10,000 triangles.
The segments from each scan (slice) are saved as data points in separate, numbered triangle files, each with indices pointing to a unique, master point file (the original point file). For simplification and reconstruction, the triangle files are processed in a batch using a shell script, which goes through all numbered sections, and runs our pipeline for each one.

The number of regions per triangle file is customized to its size, given that our segmentation algorithm will have produced triangle files of widely varying sizes — there is no guarantee that feature-based segmentation will result in an even distribution of triangles; quite the contrary. The number of regions is calculated to produce roughly 500-triangle sized regions from each triangle file. Even with this precaution, because some features are really large, while others are really small, this procedure may still need adjustment. In these cases: (a) we restart just the segment (not the entire scan) with a different preset number of regions; or (b) we restart the segment with a different reduction factor, to deal this time with regions that are too small.

![Image](image_url)

(a) Scans 3 (white) and 4 (red).

(b) Arrangement of slices in the scan sequence. (c) Detail, showing original triangulation.

Figure 43: Scans 3 and 4 of the Great Hall. Placement of original triangulation scan slices in the scene. There are discontiguous areas within each slice.
9.5 LARGE SCENES

Figure 44.: Scan 3, R=100, basis for sequential processing.

Figure 45.: Scan 3, simplified mesh, IN=7,387,052△s, OUT=151,201△s.

Figure 46.: Great Hall, detail.
Figure 47: Original triangulation (a) left, (b) top, vs. our retriangulation with F=1 (a) right, (b) bottom.
9.5 LARGE SCENES

Figure 48: Scan 4: reduction from IN=336,459\(\Delta s\) to OUT=79,984\(\Delta s\)+100,945\(\Delta s\) unprocessed.
9.5 LARGE SCENES

Figure 49.: Scans 21 through 26 of the Great Hall: grid triangles reconstruction.
(a) Floor detail, showing continuity of segmentation across scans.

(b) Side view, showing windows with clear segmentations (a).

(c) Side detail, showing windows with clear segmentations (b).

Figure 50: Scans 21 through 26 of the Great Hall: points reconstruction.
Figure 51: Scan 26, IN=1,009,264 $\Delta s$, OUT=351,026 simplified $\Delta s$+29,636 original $\Delta s$: view of the back arch in (a), detail in (b). Actual photo, (c).
10.1 LIMITATIONS OF OUR SYSTEM

The system described presents one particular difficulty in processing large scenes: the size of the regions is limited to the size that the matrix system computing the parametrization can handle, using the current available matrix libraries. Thus, as mentioned throughout this thesis, the ideal size for a region should be around $500 \times 500$ original data points. This comes in conflict with the idea that the segmentation system used should divide the scene only according to features. In most cases, where the large scenes are buildings with lots of architectural details, increasing the number of regions in order to contain the size of each one to within the specified limit for the matrix system is not an impediment — as there are always features small enough to produce a meaningful segmentation even in the presence of this requirement. At the same time, for rounded objects (such as the Stanford bunny), again, the increase in the number of specified regions does not interfere much with the quality of the segmentation, since region boundaries are going to form along smoothly varying features, anyway — it becomes a matter of the segmentation algorithm decreasing the threshold of variation of normals that stops the region growing process earlier, and thus generating more regions overall.

Two immediate consequences of this minor limitation appear: one is the inability of the system to perform an even more drastic reduction in number of points in only one global iteration. This is because, in cases like the Armadillo model, a large number of regions (2000) being necessary to bring down the region sizes, the overall reduction of the number of points is limited to this large number of regions, and to the point reduction that can be achieved within each. The other limitation, closely related to the first one — but more important from the point of view of the visual aspect of the output — is the fact that the more, smaller regions will leave their boundary footprint on the output model, since the boundaries are part of the reconstruction. If the segmentation produced only large regions, and therefore if boundaries — which translate into discontinuities in the appearance of the final triangulation — appeared only where strictly necessary (and this is a matter of subjectivity when it comes to architectural scenes, even though it may not be for medical imaging models, for example), the result would be more pleasing, and the boundaries between regions would strictly
correspond to features, whose presence should be maintained throughout. But if the
boundaries are many, and they delimit small regions, interfering with the regularity
of the final triangulation, not only the final aspect leaves to be desired, but further
applications such as texture mapping are harder to implement.

10.2 POSSIBLE IMPROVEMENTS TO OUR SYSTEM

As an immediate idea for improving the aspect of the triangulation and the input
to output reduction ratio, we considered reapplying the whole algorithm to the final
result, which, just as the input model, can be stored as a collection of data points
and their triangulation, and therefore should react well to multiple iterations. These
iterations were thought to be able to absorb the undesirable triangulation borders
arising from small regions whose presence was imposed only by the size constraint,
and not by actual model features. In practice, this did not work well, due to the
very same irregular triangulation induced by many small region borders, which some-
times produces gaps in our parent triangle-based reconstruction — which works by
placing a resampled point within the same 3D triangle that generated its 2D enclosing
triangle, a.k.a. its “parent”. The mechanism of gaps in the final triangulation is as
follows: with the regular grid sampling, the retriangulation of the resamples follows
grid points in the order of traversal of the grid. The same ordering is then applied
to the 3D resamples, but when the 3D resample is missing due to the reconstruction
method described (an extremely concave border can cause a resample to not find a
“parent” triangle in the current region, because its actual parent is in an adjacent re-

region, for example), a gap will form in the final triangulation. As opposed to a regular
grid triangulation, two other successful methods we used were Delaunay triangula-
tion and subdivision. The Delaunay triangulation works much better, but (a) it may
also present cases of missed reconstructed points, (b) it favors well-formed triangles
regardless of size, thus not producing either a visually pleasing output, or one useable
with texture mapping, and (c) everybody else is doing it, and we wanted to present a
better triangulation method. Finally, the subdivision method is also subject to missed
reconstructed points, and also, more importantly, to the original shape of the domain
triangles — as these are obtained by the straightening of the border fragments connect-
ing tri-region corner points with no reshaping for aspect, the subdivision triangles will
inherit the same shape through similarity.
10.3 Future Work

10.3.1 Point-Based Reconstruction

With these considerations in mind, two main directions of improvement become interesting to pursue: one is the elimination of the retriangulation of the resampled points as a pre-requisite for rerunning the algorithm multiple consecutive times for the same scene. Without the retriangulation, missed reconstructed points will not result in gaps on the final mesh — in fact, there will not be a final mesh, just a point cloud. In this case, an alternative method of deriving point neighborhood information and region formation criteria (both based on triangle adjacency in our current implementation) is necessary as a replacement for all the information derived from an existing triangulation. If based only on point clouds at every stage, the method could be safely applied iteratively; and if a final triangulation is needed, we believe that it would be much easier to obtain from larger, sparser regions.

The point neighborhood information could be derived considering a given radius around each point, and storing, in a pre-processing stage — in a way much similar to our TMesh class —, all the original 3D neighboring points that fall within that radius, together with the normal to the surface induced by them in the local area around the considered point. The region formation by adjacent additions to a centroid could be implemented based on the same pre-computed point neighborhood information derived as before. The size of the radius becomes extremely important, as there could be points satisfying the closeness criterion, but not adjacent in reality on the surface...

10.3.2 Subdivision for Scanned Models

Another direction would be focusing on generating original triangular domains of similar size and equilateral shape; or, if these requirements are hard to achieve in parallel, the size requirement could be dropped and a compensating method to determine the number of subdivisions on a case by case basis for each region could be developed. Personally, I think that this direction alone is an interesting and fruitful avenue for exploration if the very fast and animation-friendly method of subdivision is to be used with actual (as opposed to designed) models, with possible applications in the gaming, movie, and GPS industries.
A.1 Construction

A.1.1 Curves

In this section we would like to show a simple example of generation of a planar curve defined by a quadratic biquadratic polynomial as a motivation for our parametrization method, as well as to convey the beauty of the idea of approximating a surface through repeated linear interpolations once the control points are known, regardless of the function. These are our original derivations for simple cases of curves and surfaces, based on the general theory of multilinearization of bivariate polynomials presented in Gallier [2002].

We chose for our example the following function $F : \mathbb{R} \rightarrow \mathbb{R}^2$ with $F(X) = (F_1(X), F_2(X))$ defined as:

$$F = \begin{cases} 
F_1(X) = a_1X^2 + b_1X + c_1 \\
F_2(X) = a_2X^2 + b_2X + c_2,
\end{cases}$$

(32)

Our goal is to multilinearize this bi-quadratic parametric function of one parameter, in order to enable the generation of its domain through linear interpolations in each of its variables:

$$F_1(t) = f_1(t_1, t_2) = a_1t_1t_2 + b_1\frac{t_1 + t_2}{2} + c_1$$
$$F_2(t) = f_2(t_1, t_2) = a_2t_1t_2 + b_2\frac{t_1 + t_2}{2} + c_2$$

We want to evaluate this function at any point $u$ within the reference interval $[r; s]$, defined by its ratio of interpolation $\lambda$:

$$u(\lambda) = \lambda r + (1 - \lambda)s \implies \lambda = \frac{u - r}{s - r}$$
We want to express

\[ u = \frac{u - r}{s - r} r + \left(1 - \frac{u - r}{s - r}\right) s = \frac{u - r}{s - r} r + \frac{s - r - u + r}{s - r} s = \frac{u - r}{s - r} r + \frac{s - u}{s - r} s \]

Replacing variable \( u \) with its expression depending on the reference interval and the interpolation ration \( \lambda \), we get:

\[
\begin{align*}
 f_1(t_1, t_2) &= f_1(u, u) = a_1 \left[ \frac{u - r}{s - r} r + \frac{s - u}{s - r} s \right]^2 + b_1 \left( \frac{u - r}{s - r} r + \frac{s - u}{s - r} s \right) + c_1 \\
 f_2(t_1, t_2) &= f_2(u, u) = a_2 \left[ \frac{u - r}{s - r} r + \frac{s - u}{s - r} s \right]^2 + b_2 \left( \frac{u - r}{s - r} r + \frac{s - u}{s - r} s \right) + c_2 
\end{align*}
\]

We want to express \( f_1 \) and \( f_2 \) only in terms of the function values at the reference interval endpoints, \( f_i(r, r), f_i(s, s), \) and \( f_i(r, s), \) where \( i = 1, 2 \):

\[
\begin{align*}
 f_i(r, r) &= a_i r^2 + b_i r + c_i \\
 f_i(s, s) &= a_i s^2 + b_i s + c_i \\
 f_i(r, s) &= a_i rs + b_i \frac{r + s}{2} + c_i 
\end{align*}
\]

We develop the expressions \( f_i(u, u) \), hoping to reduce them to terms only depending on \( f_i(r, r) \) and \( f_i(s, s) \):

\[
\begin{align*}
 f_i(u, u) &= a_i \left[ \left( \frac{u - r}{s - r} \right)^2 + 2 \left( \frac{u - r}{s - r} \right) \left( \frac{s - u}{s - r} \right) + \left( \frac{s - u}{s - r} \right)^2 \right] + \\
 &\quad + b_i \left( \frac{u - r}{s - r} r + \frac{s - u}{s - r} s \right) + c_i = \\
 &= a_i \left( \frac{u - r}{s - r} \right)^2 r^2 + 2 a_i \frac{u - r s - u}{s - r s - r} r s + a_i \left( \frac{s - u}{s - r} \right)^2 s^2 + \\
 &\quad + b_i \frac{u - r}{s - r} r + b_i \frac{s - u}{s - r} s + c_i 
\end{align*}
\]

We want to form the terms \( \left( \frac{u - r}{s - r} \right)^2 f_i(r, r) \), \( 2 \frac{u - r}{s - r} \frac{s - u}{s - r} f_i(r, s) \), and \( \left( \frac{s - u}{s - r} \right)^2 f_i(s, s) \) by adding and subtracting terms. Below, the boldface terms appearing with a “−” sign are the subtractions needed to cancel the added terms, while the positive terms are pre-existing terms:

\[
\begin{align*}
 f_i(u, u) &= \left( \frac{u - r}{s - r} \right)^2 f_i(r, r) - b_i \left( \frac{u - r}{s - r} \right)^2 r - c_i \left( \frac{u - r}{s - r} \right)^2 + \\
 &\quad + 2 \frac{u - r}{s - r} \frac{s - u}{s - r} f_i(r, s) - 2 b_i \left( \frac{u - r}{s - r} \right) \left( \frac{s - u}{s - r} \right) \frac{r + s}{2} - 2 c_i \left( \frac{u - r}{s - r} \right) \left( \frac{s - u}{s - r} \right) + \\
 &\quad + \left( \frac{s - u}{s - r} \right)^2 f_i(s, s) - b_i \left( \frac{s - u}{s - r} \right)^2 s - c_i \left( \frac{s - u}{s - r} \right)^2 + \\
 &\quad + b_i \frac{u - r}{s - r} r + b_i \frac{s - u}{s - r} s + c_i
\end{align*}
\]
A.1 construction

Grouping together all the terms in \( b_i r \), they cancel out:

\[
-b_i \left( \frac{u-r}{s-r} \right)^2 r - b_i \left( \frac{u-r}{s-r} \right) \left( \frac{s-u}{s-r} \right) r + b_i \left( \frac{u-r}{s-r} \right) r = \\
= b_i \left[ \frac{u-r}{s-r} - \frac{u-r s - u}{s-r s - r} - \left( \frac{u-r}{s-r} \right)^2 \right] r = \\
= b_i \frac{u-r}{s-r} \left( 1 - \frac{s-u}{s-r} - \frac{u-r}{s-r} \right) r = 0
\]

The same thing happens to the terms in \( b_i s \):

\[
-b_i \frac{u-r}{s-r} s - b_i \left( \frac{s-u}{s-r} \right)^2 s + b_i \frac{s-u}{s-r} s = \\
= b_i s - u \left[ 1 - \frac{u-r}{s-r} \frac{s-u}{s-r} \right] s = 0
\]

Finally, the terms in \( c_i \) also reduce:

\[
-c_i \left( \frac{u-r}{s-r} \right)^2 - 2c_i \left( \frac{u-r s - u}{s-r} \right) - c_i \left( \frac{s-u}{s-r} \right)^2 + c_i = \\
= -c_i \left[ \left( \frac{u-r}{s-r} \right)^2 + 2 \frac{u-r s - u}{s-r} \frac{s-u}{s-r} + \left( \frac{s-u}{s-r} \right)^2 - 1 \right] = \\
= -c_i \left[ \left( \frac{u-r}{s-r} \right)^2 + \frac{s-u}{s-r} \left( \frac{s-u}{s-r} \right)^2 - 1 \right] = 0
\]

We arrive at the following notable result:

\[
f_i(u, u) = \left( \frac{u-r}{s-r} \right)^2 f_i(r, r) + 2 \frac{u-r s - u}{s-r s - r} f_i(r, s) + \left( \frac{s-u}{s-r} \right)^2 f_i(s, s) \quad (33)
\]

Equation 33 states that the value of the function \( f_i \) at parameter \( u \) only depends on the values at the endpoints of the reference interval and on the interpolation ratio \( \lambda \). One only needs to evaluate the function at the Bézier control points \( f_i(r, r) \), \( f_i(s, s) \), and \( f_i(r, s) \), and leave the interpolation to the Bernstein coefficients, which express the position of the variable \( u \) within the reference interval \([r; s]\). Formulated this way, Equation 33 can be rewritten as:

\[
f_i(u, u) = \alpha f_i(r, r) + \beta f_i(s, s) + \gamma f_i(r, s),
\]

where

\[
\alpha(u) = \left( \frac{u-r}{s-r} \right)^2, \quad \text{or } \alpha(\lambda) = \lambda^2,
\]

\[
\beta(u) = \left( \frac{s-u}{s-r} \right)^2, \quad \text{or } \beta(\lambda) = (1-\lambda)^2,
\]

\[
\gamma(u) = 2 \frac{u-r s - u}{s-r s - r}, \quad \text{or } \gamma(\lambda) = 2\lambda(1-\lambda).
\]

are the Bernstein coefficients of second degree.
As an example we consider the following curve (a sideways parabola):

\[
\begin{align*}
F_1(t) &= t^2 + 1 \\
F_2(t) &= 2t - 1
\end{align*}
\]  

(34)

This univariate second degree pair of polynomials describes a function of polar degree 2. It can be polarized by using two polar variables:

\[
\begin{align*}
F_1(t) &= f_1(t_1, t_2) = t_1 t_2 + 1 \\
F_2(t) &= f_2(t_1, t_2) = t_1 + t_2 - 1
\end{align*}
\]

We evaluate this function at varying interpolation coefficients \( \lambda \in [0; 1] \) placing each polar variable in the interval \( u(\lambda) \in [r; s] \subset \mathbb{R} \). We choose for exemplification \( u \in \{-3.5, -1.75, 0, 1.75, 3.5\} \) and \([r; s] = [-3.5; 3.5]\). Accordingly, \( \lambda \in \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\} \). The evaluation of the function is done through linear interpolation using the control points:

\[
\begin{align*}
  f_1(u, u) &= \alpha f_1(r, r) + \beta f_1(r, s) + \gamma f_1(s, s) \\
  f_2(u, u) &= \alpha f_2(r, r) + \beta f_2(r, s) + \gamma f_2(s, s)
\end{align*}
\]

Figure 52: Sideways parabola, showing interpolation points within \([-3.5; 3.5]\) (red) and control points (orange)

The control points are the pairs \((f_1, f_2)(r, r) = ((-3.5)^2 + 1, 2(-3.5) - 1) = (13.25, -8), (f_1, f_2)(s, s) = (3.5^2 + 1, 2 \cdot 3.5 - 1) = (13.25, 6), (f_1, f_2)(r, s) = ((-3.5) \cdot 3.5 + 1, -3.5 \cdot 3.5 - 1) = (-11.25, -1)\). Table 10 shows a few function values, calculated through interpolation. This parabola could have not been computed through an implicit function.
### Table 10: Bernstein coefficients $\alpha$, $\beta$, $\gamma$, and function values for different interpolation $\lambda$ values, in the case of the sideways parabola of Figure 52

<table>
<thead>
<tr>
<th>$u$</th>
<th>$\lambda$</th>
<th>$\alpha$, $\beta$, $\gamma$</th>
<th>$x = f_1(u, u)$</th>
<th>$y = f_2(u, u)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3.5</td>
<td>1</td>
<td>1, 0, 0</td>
<td>13.25</td>
<td>-8</td>
</tr>
<tr>
<td>-1.75</td>
<td>$\frac{3}{4}$</td>
<td>0, $\frac{1}{4}$, $\frac{1}{4}$</td>
<td>4.0625</td>
<td>-4.5</td>
</tr>
<tr>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$, $\frac{1}{2}$, $\frac{1}{2}$</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1.75</td>
<td>$\frac{3}{2}$</td>
<td>$\frac{1}{2}$, $\frac{1}{2}$, $\frac{1}{2}$</td>
<td>4.0625</td>
<td>2.5</td>
</tr>
<tr>
<td>3.5</td>
<td>3</td>
<td>0, 0, 1</td>
<td>13.25</td>
<td>6</td>
</tr>
</tbody>
</table>

This can be generalized first to degree 3, where, given a triplet of generating polynomials of third degree of one parameter defining the parametric function $F : \mathbb{R} \rightarrow \mathbb{R}$:

$$F = \begin{cases} 
F_1(X) = a_1X^3 + b_1X^2 + c_1X + d_1 \\
F_2(X) = a_2X^3 + b_2X^2 + c_2X + d_2 \\
F_3(X) = a_3X^3 + b_3X^2 + c_3X + d_3 
\end{cases},$$

the polar form for each component $F_i$ is the symmetric triaffine function $f_i : \mathbb{R}^3 \rightarrow \mathbb{R}^3$:

$$f_i(t_1, t_2, t_3) = a_i(t_1t_2t_3 + b_i t_1t_2 + t_2t_3 + t_3t_1 + c_i t_1 + t_2 + t_3) + d_i,$$

where $t_i \in [r; s]$, $i = 1, 3$. Similar to the second degree case, it can be shown that, for $u = t_1 = t_2 = t_3 = \lambda r + (1 - \lambda)s$, we have

$$f_i(u, u, u) = (1 - \lambda)^3 f_i(r, r, r) + 3(1 - \lambda)^2 \lambda f_i(r, r, s) +$$

$$+ 3(1 - \lambda) \lambda^2 f_i(r, s, s) + \lambda^3 f_i(s, s, s),$$

where the polynomials $(1 - \lambda)^3$, $3(1 - \lambda)^2 \lambda$, $3(1 - \lambda) \lambda^2$, $\lambda^3$ are the Bernstein polynomials of degree 3. Again, the importance of Equation 35 lies in the fact that, to evaluate $F$ at parameter value $u$, all that is needed are the values of $f_i(t_1, t_2, t_3)$ at $(r, r, r)$, $(r, r, s)$, $(r, s, s)$, $(s, s, s)$ and the ratio of interpolation $\lambda$. The image values $f_i(r, r, r)$, $f_i(r, r, s)$, $f_i(r, s, s)$, $f_i(s, s, s)$ are the Bézier control points of the function.

The univariate parametric cubic function $F$ traces a curve in $\mathbb{R}^3$.

Higher order curves can be obtained by describing the coordinate functions through higher degree polynomials.
A.1 construction

A.1.2 Bipolynomial Surfaces

For surfaces defined parametrically, we are interested in bivariate functions of the form \( F : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^3 \) with components \( F_i : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \) and polarized forms \( f_i : (\mathbb{R})^p \times (\mathbb{R})^q \to \mathbb{R} \), which are obtained after linearizing \( F_i(U, V) \), where the degree of variable \( U \) is at most \( p \), and the degree of variable \( V \) is at most \( q \). Alternately, we are interested in functions \( F : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^3 \), with polarized components \( f_i : (\mathbb{R})^m \to \mathbb{R} \) where \( m \) indicates the highest combined degree of \( U^j V^k \) in polynomial \( F_i(U, V) \), or the total degree of \( F \).

After multilinearizing in \( p + q \) variables, for bipolynomial surfaces we get a polar degree of \( m = \max_{1 \leq j \leq p, 1 \leq k \leq q} \{j + k\} \) with \( m \leq p + q \). These surfaces are described in Section A.1.2. For total degree surfaces we get a total degree of \( m = \max\{p, q\} \). These surfaces are treated in Section A.1.3.

The Plane as a Bipolynomial Surface

For the moment, to illustrate polarization and surface construction through linear interpolation, we consider the simplest case, in which \( x(U, V), y(U, V), z(U, V) \) are linear:

\[
F = \begin{cases} 
F_1(U, V) = a_1 U + b_1 V + c_1 \\
F_2(U, V) = a_2 U + b_2 V + c_2 \\
F_3(U, V) = a_3 U + b_3 V + c_3 
\end{cases} 
\]  

(36)

These bivariate coordinate functions define a plane in \( \mathbb{R}^3 \). Each \( F_i(U, V), i = 1, 3 \) can be polarized as follows, in order to obtain symmetric linear functions \( f_i \):

\[
f_i(u_1, v_1) = a_i \frac{u_1 + v_1}{2} + b_i \frac{u_1 + v_1}{2} + c_i 
\]

Having denoted the maximum exponent for each of the two parameter variables \( p \) and \( q \), respectively, in this case we have \( p = 1, q = 1 \), and, considering it as a bipolynomial surface, the bipolynomial degree \( m = p + q = 2 \); considering it as a total surface, the total degree is \( m = \max\{p, q\} = 1 \). Informally, the polar degree represents the added multiplicities of each polar variable, whether \( p + q \) scalars or \( \max\{p, q\} \) points (parameter pairs).

The polar variables can be expressed in terms of the endpoints of the reference interval \([r; s] \) and the ratios of interpolation \( \lambda_1, \lambda_2 \):

\[
u = \lambda_1 r + (1 - \lambda_1) s \text{ and } v = \lambda_2 r + (1 - \lambda_2) s
\]
A.1 Construction

The goal is to express the function \( f_i \) only in terms of \( \lambda_1, \lambda_2 \) and of function values

\[
\begin{align*}
  f_i(r, r) &= a_i r + b_i r + c_i \\
  f_i(r, s) &= a_i r + b_i s + c_i \\
  f_i(s, s) &= a_i s + b_i s + c_i.
\end{align*}
\]

Let \( g_i(u) = a_i u + B_i \) and \( h_i(v) = b_i v + c_i \) two linear functions in one variable, for which it is easy to prove the preliminary result that, for \( u = \lambda_1 r + (1 - \lambda_1) s \) and \( v = \lambda_2 r + (1 - \lambda_2) s \), the following affine relationships hold: \( g_i(u) = \lambda_1 g_i(r) + (1 - \lambda_1) g_i(s) \) and \( h_i(v) = \lambda_2 h_i(r) + (1 - \lambda_2) h_i(s) \). This is because the composition of two affine functions remains affine: \( (u \circ g_i)(r, s) \) is affine if \( g_i(u(r, s)) \) and \( u(r, s) \) are affine. Then, we can extend the result from the case of one variable to the case of two variables as follows:

\[
  f_i(u, v) = a_i u + B_i = g_i(u) = \lambda_1 g_i(r) + (1 - \lambda_1) g_i(s) = \lambda_1 (a_i r + B_i) + (1 - \lambda_1)(a_i s + B_i)
\]

Replacing \( B_i \) with \( h_i(v) \)'s expression as an affine combination of \( r, s \), we get:

\[
\begin{align*}
  f_i(u, v) &= \lambda_1 (a_i r + \lambda_2 h_i(r) + (1 - \lambda_2) h_i(s)) + \\
  &
  + (1 - \lambda_1) [a_i s + \lambda_2 h_i(r) + (1 - \lambda_2) h_i(s)] = \\
  &= \lambda_1 [a_i r + \lambda_2 (b_i r + c_i) + (1 - \lambda_2)(b_i s + c_i)] + \\
  &
  + (1 - \lambda_1) [a_i s + \lambda_2 (b_i r + c_i) + (1 - \lambda_2)(b_i s + c_i)].
\end{align*}
\]

In the first parenthesis, by expressing \( a_i r = [\lambda_2 + (1 - \lambda_2)] a_i r \), we get:

\[
\begin{align*}
  [\lambda_2 + (1 - \lambda_2)] a_i r + \lambda_2 (b_i r + c_i) + (1 - \lambda_2)(b_i s + c_i) = \\
  = \lambda_2 (a_i r + b_i r + c_i) + (1 - \lambda_2)(a_i s + b_i s + c_i) = \lambda_2 f_i(r, r) + (1 - \lambda_2)f_i(r, s)
\end{align*}
\]

Similarly, by expressing \( a_i s = [\lambda_2 + (1 - \lambda_2)] a_i s \) in the second parenthesis, we also get:

\[
\begin{align*}
  [\lambda_2 + (1 - \lambda_2)] a_i s + \lambda_2 (b_i r + c_i) + (1 - \lambda_2)(b_i s + c_i) = \\
  = \lambda_2 f_i(s, r) + (1 - \lambda_2)f_i(s, s)
\end{align*}
\]

The whole expression of \( f_i \) becomes

\[
  f_i(u, v) = \lambda_1 [\lambda_2 f_i(r, r) + (1 - \lambda_2)f_i(r, s)] + (1 - \lambda_1) [\lambda_2 f_i(s, r) + (1 - \lambda_2)f_i(s, s)]
\]

128
In other words, we have proved the following result for linear functions in two variables:

\[ f_i(u, v) = \lambda_1 \lambda_2 f_i(r, r) + \lambda_1 (1 - \lambda_2) f_i(r, s) + (1 - \lambda_1) \lambda_2 f_i(s, r) + (1 - \lambda_1)(1 - \lambda_2) f_i(s, s) \]  

Equation (37) reflects the fact that, in order to represent a point on a plane in \( \mathbb{R}^3 \) defined parametrically by the coordinate functions from Equation (36), one needs only the values of the function at endpoints of the reference interval where the parameters \( u, v \) take values, and the interpolation values \( \lambda_1, \lambda_2 \). Later we will show that our own linear interpolation is similar, except that it operates on the assumption of total degree \( m = 1 \), rather than of polar degree \( \langle p, q \rangle = \langle 1, 1 \rangle \), for which the interpolation, instead of being calculated for each variable \( u \) and \( v \), considers pairs \( (u, v) \) as points in the parameter plane, and is computed as a barycentric combination of a reference triangle \( \triangle rst \).

Figure 53 represents the plane of parametric equations

\[
\begin{align*}
X(U, V) &= U + 2V + 3 \\
Y(U, V) &= 2U + V + 2 \\
Z(U, V) &= U + V + 1
\end{align*}
\]  

Figure 53.: Plane. The yellow grid is the parameter plane, emplaced at \( xOy \) for representation purposes. The green represents the parametric plane of Equation (38).

If, instead of calculating function values \( F(u, v) \) at each pair \( (u, v) \in [r; s] \times [r; s] \), we consider only the \( 2^{p+q} = 2^2 = 4 \) control points at the endpoints of the reference interval for the same patch that produced Figure 53, namely \([−10.0; 10.0] \times [−10.0; 10.0]\).
A.1 construction

<table>
<thead>
<tr>
<th>$\lambda_1$</th>
<th>$\lambda_2$</th>
<th>$x(\lambda_1, \lambda_2)$</th>
<th>$y(\lambda_1, \lambda_2)$</th>
<th>$z(\lambda_1, \lambda_2)$</th>
<th>$u$</th>
<th>$v$</th>
<th>$x(u, v)$</th>
<th>$y(u, v)$</th>
<th>$z(u, v)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-27.0</td>
<td>-28.0</td>
<td>-19.0</td>
<td>-10.0</td>
<td>-10.0</td>
<td>-27.0</td>
<td>-28.0</td>
<td>-19.0</td>
</tr>
<tr>
<td>$\frac{1}{4}$</td>
<td>$\frac{1}{4}$</td>
<td>-12.0</td>
<td>-13.0</td>
<td>-9.0</td>
<td>-5.0</td>
<td>-5.0</td>
<td>-12.0</td>
<td>-13.0</td>
<td>-9.0</td>
</tr>
<tr>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>-3.0</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-3.0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$\frac{3}{4}$</td>
<td>$\frac{3}{4}$</td>
<td>18.0</td>
<td>17.0</td>
<td>1.0</td>
<td>5.0</td>
<td>5.0</td>
<td>18.0</td>
<td>17.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>33.0</td>
<td>32.0</td>
<td>21.0</td>
<td>10.0</td>
<td>10.0</td>
<td>33.0</td>
<td>32.0</td>
<td>21.0</td>
</tr>
</tbody>
</table>

(a) Point values on the plane of Equation 38 computed through interpolation. (b) Point values on the plane of Equation 38 computed from the coordinate functions.

Table 11: Point values on the plane of Equation 38 (a) interpolation; (b) computation.

$[-10.0; 10.0]$, we can compute any point on the surface defined by its interpolation coefficients $\lambda_{1,2}$ and $F(-10.0, -10.0), F(-10.0, 10.0), F(10.0, 10.0), F(10.0, -10.0)$, its four control points. Table 11b shows, for a small subset of the values used for the display of the surface in Figure 53, points defined by pairs of interpolation ratios within each direction $(\lambda_1, \lambda_2) \in \{(0,0), \left(\frac{1}{4}, \frac{1}{4}\right), \left(\frac{1}{2}, \frac{1}{2}\right), \left(\frac{3}{4}, \frac{3}{4}\right), (1,1)\}$. By comparison, the same $(x, y, z)$ are obtained directly through computation of the coordinate functions for the corresponding values of the parameter pairs $(u, v)$, as can be seen in Table 11a.

**Quadratic Surfaces as Bipolynomial Surfaces**

Let us now consider the quadratic case $F: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^3$:

$$F = \begin{cases} F_1(U, V) = a_1 U^2 + b_1 UV + c_1 V^2 + d_1 U + e_1 V + f_1 \\ F_2(U, V) = a_2 U^2 + b_2 UV + c_2 V^2 + d_2 U + e_2 V + f_2 \\ F_3(U, V) = a_3 U^2 + b_3 UV + c_3 V^2 + d_3 U + e_3 V + f_3 \end{cases}$$

The following parametric equations describe a hyperbolic paraboloid:

$$F = \begin{cases} F_1(U, V) = U \\ F_2(U, V) = V \\ F_3(U, V) = \frac{1}{15} U^2 - \frac{1}{15} V^2 \end{cases} \quad (39)$$

The coefficients of $U^2$, $V^2$ were scaled for representation purposes. Figure 55 shows the quadratic surface described by the parametric Equations 39 obtained from direct computation, as well as from interpolation.
A.1 construction

Another example is the elliptic paraboloid, shown in Figure 54 and described by the parametric equations:

\[
F = \begin{cases} 
F_1(U, V) = U \\
F_2(U, V) = V \\
F_3(U, V) = 2U^2 + V^2
\end{cases}
\]

Figure 54.: Elliptic Paraboloid. The green surface represents the parametric quadratic surface, while the yellow plane represents the parameter domain.

Bipolynomial Surfaces of Degree \( \langle p, q \rangle \)

Recalling the informal definitions given for the bivariate linear case, we have studied so far \( F_i(U, V) \) of total degree 2, \( p = 2, q = 2 \), and polar degree \( m = \max(p + q) = 4 \). In general, the polarization \( f_i \) of each \( F_i \) can be achieved in two ways. The first way to polarize is separately in each of the two variables \( U, V \)

\[
F_i(U, V) = f_i(u_1, \ldots, u_p, v_1, \ldots, v_q),
\]

where \( f_i : (\mathbb{R})^p \times (\mathbb{R})^q \rightarrow \mathbb{R} \). We say that \( F \) is a bipolynomial surface of degree \( \langle p, q \rangle \). Since each variable travelling along reference interval \([r; s]\) can define a uniform partition of this interval, when both variables take values along the axes of a Cartesian coordinate system in the plane at uniform spaces within the planar patch \([r; s] \times [r; s]\), the surfaces generated by the coordinate functions \( f_i(U, V) \) are called tensor product surfaces.

Polarizing in variable \( U \) gives:

\[
f_i(u_1, u_2, V) = a_i u_1 u_2 + b_i \frac{u_1 + u_2}{2} V + c_i V^2 + d_i \frac{u_1 + u_2}{2} + e_i V + f_i
\]
After polarizing in variable $V$ we get:

$$f_i(u_1, v_1, v_2) = a_i u_1 u_2 + b_i \frac{u_1 + u_2}{2} v_1 + \frac{v_1}{2} + c_i u_1 v_2 + d_i \frac{u_1 + u_2}{2} + e_i \frac{v_1 + v_2}{2} + f_i =$$

$$= a_i u_1 u_2 + b_i \frac{(u_1 + u_2)(v_1 + v_2)}{4} + c_i u_1 v_2 + d_i \frac{u_1 + u_2}{2} + e_i \frac{v_1 + v_2}{2} + f_i$$

Each of the polar variables $u_1, u_2, v_1, v_2$ can be written in terms of the reference interval endpoints and the ratios of interpolation $\lambda_i$ (for the $u$ variables) and $\lambda_k$ (for the $v$ variables): 

$$u_{1,2} = \lambda_j r + (1 - \lambda_j) s, \text{ where } j = 1, 2, \text{ (or, generally, } j = \Gamma, p) \text{, and}$$

$$v_{1,2} = \lambda_k r + (1 - \lambda_k) s, \text{ where } k = 3, 4, \text{ (or, generally, } k = p + 1, p + q) \text{.}$$

To avoid confusion between index $i$ differentiating the coordinate functions $f_i(U, V)$ and indices $j, k$ referring to the $p, q$ linear variables, we will omit from here on the indexation of the three coordinate functions. We want to express any point in terms of $\lambda_{j,k}$ and of the $2^{4} = 16$ possible control points: $f(r, r, r, r), f(r, r, r, s), \ldots, f(s, s, s, s)$. Using again the fact that 

$$f(u_1, \ldots, \lambda_j r + (1 - \lambda_j) s, \ldots, u_p, v_1, \ldots, v_q) =$$

$$= \lambda_j f(u_1, \ldots, r, \ldots, u_p, v_1, \ldots, v_q) +$$

$$(1 - \lambda_j) f(u_1, \ldots, s, \ldots, u_p, v_1, \ldots, v_q),$$

the function $f(u_1, \ldots, u_p, v_1, \ldots, v_q)$ can be written as

$$f(u_1, \ldots, u_p, v_1, \ldots, v_q) = \sum_{j=1}^{p} \sum_{k=1}^{q} \left[ \prod_{j \neq k} \lambda_j (1 - \lambda_k) \right] f(r_1, \ldots, r_j, s_1, \ldots, s_k),$$

where $\prod_{j \neq k} \lambda_j (1 - \lambda_k) = B_k^p [r; s] (u) \cdot B_k^q [r; s] (v)$ are the Bernstein polynomials encapsulating the ratios $\lambda_j, \lambda_k$ for variables $u, v \in [r; s]$ (assuming a rectangular parameter patch $[r; s] \times [r; s]$). Index $j$ counts the number of interpolation coefficients $\lambda_j$ for the $u_j$ variables, while index $k$ counts interpolation coefficients $\lambda_k$ for the $v_j$ variables. With this notation, a point of parametric coordinates $u, v$ can be calculated only from the Bernstein polynomials and the control points as follows:

$$F(u, v) = \sum_{j=1}^{p} \sum_{k=m-p+1}^{p+q} B_j^p [r; s] (u) \cdot B_k^q [r; s] (v) \cdot f(r_1, \ldots, r_j, s_1, \ldots, s_k) \tag{41}$$
From this generalization, the quadratic case will have $16 = 2^{p+q} = 2^4$ terms of the form

$$
\prod_{j \neq k} (1 - \lambda_j) \lambda_k f(r_{1 \ldots j}, s_{1 \ldots k}),
$$

which can be rewritten in matrix form as:

$$
\begin{pmatrix}
(1 - \lambda_1)(1 - \lambda_2) & (1 - \lambda_1)\lambda_2 & \lambda_1(1 - \lambda_2) & \lambda_1\lambda_2
\end{pmatrix}
\begin{pmatrix}
(1 - \lambda_3)(1 - \lambda_4) \\
(1 - \lambda_3)\lambda_4 \\
\lambda_3(1 - \lambda_4) \\
\lambda_3\lambda_4
\end{pmatrix}
$$

where

$$
B =
\begin{pmatrix}
 f(r, r, r, r) & f(r, r, r, s) & f(r, r, s, r) & f(r, r, s, s) \\
 f(r, s, r, r) & f(r, s, r, s) & f(r, s, s, r) & f(r, s, s, s) \\
 f(s, r, r, r) & f(s, r, r, s) & f(s, r, s, r) & f(s, r, s, s) \\
 f(s, s, r, r) & f(s, s, r, s) & f(s, s, s, r) & f(s, s, s, s)
\end{pmatrix}
$$

Due to symmetry in $u_1, u_2$ and $v_1, v_2$, if we let $\lambda_1 = \lambda_2 = \lambda$ and $\lambda_3 = \lambda_4 = \gamma$:

$$
F(u(\lambda), v(\gamma)) = \begin{pmatrix}
(1 - \lambda)^2 & (1 - \lambda)\lambda & \lambda(1 - \lambda) & \lambda^2
\end{pmatrix}
\begin{pmatrix}
(1 - \gamma)^2 \\
(1 - \gamma)\gamma \\
\gamma(1 - \gamma) \\
\gamma^2
\end{pmatrix}
$$

Also due to the symmetry in $u_1, u_2$ and $v_1, v_2$, we only have the following types of function values:

$$
\begin{align*}
1 \times f(r, r| r, r), & \quad 2 \times f(r, r| r, s), & \quad 1 \times f(r, r| s, s) \\
2 \times f(r, s| r, r), & \quad 4 \times f(r, s| r, s), & \quad 2 \times f(r, s| s, s) \\
1 \times f(s, s| r, r), & \quad 2 \times f(s, s| r, s), & \quad 1 \times f(s, s| s, s)
\end{align*}
$$

With these repeats, we can rewrite Equation 42 more compactly:

$$
F(u(\lambda), v(\gamma)) = \begin{pmatrix}
(1 - \lambda)^2 & 2(1 - \lambda)\lambda & \lambda^2
\end{pmatrix}
\begin{pmatrix}
(1 - \gamma)^2 \\
2(1 - \gamma)\gamma \\
\gamma^2
\end{pmatrix}
$$

where $G$ is now a $3 \times 3$ matrix with the following entries:

$$
G =
\begin{pmatrix}
 f(r, r, r, r) & f(r, r, r, s) & f(r, r, s, r) \\
 f(r, s, r, r) & f(r, s, r, s) & f(r, s, s, r) \\
 f(s, s, r, r) & f(s, s, r, s) & f(s, s, s, r)
\end{pmatrix}
$$

133
A.1 construction

If, in addition to the symmetry in each of the 2 pairs of variables \( f(u_1, u_2|*,*) = f(u_2, u_1|*,*) \), they also move at the same rate within the reference interval \([r; s] \) (\( f(u,*,*,*) \)), meaning that both coordinates \((u, v)\) of each parameter point have the same interpolation coefficients \( \lambda = \gamma \equiv \lambda \) we only have 5 types of function values:

\[
\begin{align*}
1 \times f(r, r, r, r) &= C_4^4 \\
4 \times f(r, r, r, s) &= C_4^3 \\
6 \times f(r, r, s, s) &= C_4^2 \\
4 \times f(s, s, s, r) &= C_4^3 \\
1 \times f(s, s, s, s) &= C_4^4
\end{align*}
\]

With this insight, Equation \ref{eq:41} can be rewritten once more:

\[
F(u(\lambda), v(\lambda)) = \sum_{i=1}^{p+q} C_{p+q}^i (1-\lambda)^i \lambda^{p+q-i} f(r_{i}, \ldots, r_{i}, s_{i}, \ldots, s_{i}), \tag{43}
\]

or, letting \( m = p + q \):

\[
F(u(\lambda), v(\lambda)) = \sum_{j=1}^{m} C_m^j (1-\lambda)^j \lambda^{m-j} f(r_{j}, \ldots, r_{j}, s_{j}, \ldots, s_{j}) \tag{44}
\]

If we now denote \( B_j^p[r; s] = C_p^j (1-\lambda)^j \lambda^{p-j} \) and \( B_q^q[r; s] = C_q^k (1-\gamma)^k \gamma^{q-k} \) the Bernstein polynomials of the second order if \( p = q = 2 \) or of bipolynomial order \((p, q)\) in general, we can separate, similar to Equation \ref{eq:42} Equation \ref{eq:43} in matrix form as follows, considering different speeds of \( u, v \) along \([r; s]\):

\[
F(u(\lambda), v(\gamma)) = \sum_{j \leq p, k \leq q} \left[ C_p^j (1-\lambda)^j \lambda^{p-j} \cdot B_j^p(\lambda) \cdot C_q^k (1-\gamma)^k \gamma^{q-k} \right] \tag{45}
\]

This is an informal derivation of the matrix expression of a Bézier patch as found in \cite{Farin2002}, p.265, rewritten below with our notations:

\[
F(u(\lambda), v(\gamma)) = \begin{pmatrix} B_0^p(\lambda) & \ldots & B_p^p(\lambda) \end{pmatrix} \begin{pmatrix} b_{00} & \ldots & b_{0q} \\
\vdots & \ddots & \vdots \\
b_{0p} & \ldots & b_{pq} \end{pmatrix} \begin{pmatrix} B_0^q(\gamma) \\
\vdots \\
B_q^q(\gamma) \end{pmatrix} \tag{45}
\]

134
A.1 construction

In this expression, the control points acquire only two subscripts, according to which subset of polar variables in \( f(u_1, u_2; v_1, v_2) \) is held fixed, letting the other switch, for instance \( f(r, r; r, r), f(r, r; r, s), f(r, r; s, s) \) (the first column in B) become \( b_{00}, b_{01}, b_{02} \) for \( p = 2 \). The matrix B of non-repeat control points is called the geometry matrix of the reference patch \([r; s] \times [r; s]\), because it stores the essential geometric information about the 3D patch, encapsulated as 3D control points.

**Example: The Hyperbolic Paraboloid**

We will now proceed with the example quadratic surface from Equation 39, a hyperbolic paraboloid of parametric equations:

\[
F = \begin{cases} 
  F_1(U, V) = U \\
  F_2(U, V) = V \\
  F_3(U, V) = \frac{1}{10}U^2 - \frac{1}{10}V^2
\end{cases}
\]

We constructed this surface using interpolation, as opposed to direct computation, based on the reference patch \([-10;10] \times [-10;10]\) and interpolation coefficients \( \lambda, \gamma \in \left\{ 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1 \right\} \) versus the following pairs of coordinate function polar variables: \((u, v) \in \{(-10.0, -10.0), (-5.0, -5.0), (0.0, 0.0), (5.0, 5.0), (10.0, 10.0)\}\), corresponding to these coefficients (just like in the case of the plane).

<table>
<thead>
<tr>
<th>(u)</th>
<th>(v)</th>
<th>(x(u, v))</th>
<th>(y(u, v))</th>
<th>(z(u, v))</th>
<th>(\lambda)</th>
<th>(\gamma)</th>
<th>(x(\lambda, \gamma))</th>
<th>(y(\lambda, \gamma))</th>
<th>(z(\lambda, \gamma))</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.0</td>
<td>-10.0</td>
<td>-10.0</td>
<td>-10.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>-10</td>
<td>-10</td>
<td>0.0</td>
</tr>
<tr>
<td>-5.0</td>
<td>-5.0</td>
<td>-5.0</td>
<td>-5.0</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>-5</td>
<td>-5</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.8</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>0.0</td>
<td>0.8</td>
<td>0.8</td>
<td>5</td>
<td>5</td>
<td>0.0</td>
</tr>
<tr>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>10.0</td>
<td>0.0</td>
<td>0.8</td>
<td>0.8</td>
<td>10</td>
<td>10</td>
<td>0.0</td>
</tr>
</tbody>
</table>

(a) Point values on the surface of Equation 39, computed from the coordinate functions. (b) Point values on the surface of Equation 39, computed by linear interpolation.

Table 12: Point values for the hyperbolic paraboloid from Figure 55, calculated (a) from equations and (b) using interpolation. They coincide, as expected.

Our implementation uses the matrix representation from Equation 45, in which the Bernstein coefficients can be precomputed and stored in two vectors, and the control points \( f_i(u_1, u_2; v_1, v_2), i = 1, 3 \) can be calculated using a bit mask to switch the \( r, s \) values. Alternatively, the program could use the de Casteljau algorithm. The matrix representation is useful, because it can be used in an approximation setting, and we used it for error calculation in Chapter 8. The idea was to apply our parametrization
A.1 construction

Figure 55: Hyperbolic paraboloid. The blue triangulation is obtained from parameter points \((u, v)\) and coordinate functions. The points calculated from the interpolation Equation 41 are exactly at the grid points.

and reconstruction algorithm to a well-known surface (e.g. a quadratic surface of some known shape), and then to compare the results against the same surface, computed from parameter points. In order to make the comparison meaningful, though, we needed the same parameter points to provide the \(\lambda, \gamma\) interpolation values in both cases (our reconstruction vs. the polynomial construction), which is why the matrix form is preferable, because it allows the selection of the interpolation values from a set of parameter values. However, we did not compute error calculations for the bipolynomial case, because it applies to rectangular patches (with linear coefficients in each direction), as opposed to triangular patches (with barycentric coefficients).

A.1.3 Total Degree Surfaces

Another way to polarize bipolynomial surfaces is by considering each parameter pair \((u, v)\) as a point in the parameter plane, which can be expressed as barycentric coordinates relative to a three-point reference system, the reference triangle \(\triangle rst\), with \(r, s, t\) representing, this time, points in the parameter plane:

\[
r(u_r, v_r), s(u_s, v_s), t(u_t, v_t) \in \mathbb{R}^2
\]

By treating the pair of parameter variables \((U, V)\) as a whole, i.e. as the coordinates of a point \(P'(U, V) \in \mathbb{R}^2\), the polynomials \(F_i(U, V)\) can be polarized in both variables simultaneously, as described in [Gallier 2002]. We say that \(F\) is a polynomial surface of total degree \(m\):

\[
F(U, V) = f\left((u_1, v_1), \ldots, (u_m, v_m)\right),
\]

(46)
where \( f_i : (\mathbb{R} \times \mathbb{R})^m \rightarrow \mathbb{R} \) (each coordinate function is a scalar-valued function).

We will begin again with a set of three bivariate, linear polynomials, describing a plane in \( \mathbb{R}^3 \):

\[
F_i = a_i U + b_i V + c_i, \quad \text{where } i = \{1,3\} \tag{47}
\]

Instead of polarizing in each variable separately, we polarize considering that \( u, v \) are points in \( \mathbb{R}^2 \), expressed in function of the reference triangle \( \triangle r st \):

\[
(u, v) = \begin{cases} 
\lambda u_r + \mu u_s + \nu u_t \\
\lambda v_r + \mu v_s + \nu v_t 
\end{cases}
\]

Replacing into Equation 47, we get:

\[
f_i(u, v) = a_i(\lambda u_r + \mu u_s + \nu u_t) + b_i(\lambda v_r + \mu v_s + \nu v_t) + c_i = \\
\frac{f_i(u_r, v_r)}{f_i(u_r, v_r)} + \frac{f_i(u_s, v_s)}{f_i(u_s, v_s)} + \frac{f_i(u_t, v_t)}{f_i(u_t, v_t)} - (\lambda + \mu + \nu) c_i + c_i,
\]

therefore:

\[
f_i(u, v) = \lambda f_i(u_r, v_r) + \mu f_i(u_s, v_s) + \nu f_i(u_t, v_t) \tag{48}
\]

Equation 48 says that, in order to represent surface \( F_i \), all that is necessary are the function values at the endpoints of the reference triangle \( \triangle r st \) and the barycentric interpolation coefficients \( \lambda, \mu, \nu \), whose values add up to 1: \( \lambda + \mu + \nu = 1 \).

The Plane as a Total-Degree Surface

To verify this statement, as well as to acquire the building blocks necessary in our error calculation in Chapter 8, we redefined our reference patch this time as the parameter triangle \( \triangle \left\{ r(0, a \sqrt{3} / 2), s(-\frac{a}{2}, -a \sqrt{3} / 6), t(\frac{a}{2}, -a \sqrt{3} / 6) \right\} \), where by taking \( a = 20 \), we get \( r(0.0, 11.5470), s(-10.0, -5.7735), t(10.0, -5.7735) \) as approximate values for the reference triangle coordinates. We chose these values to resemble the proportions in the square patch case. We built a triangular net based on subdivision until each side is divided into \( 2^n = 2^4 = 16 \) equal segments, where \( n = 4 \). In Table 13, we show only the subset for \( n = 2 \). Since the values computed from parameters and those computed through interpolation are the same, as expected, and as shown in the Tables 11 and 12, we are collapsing the data into a single table, showing the
A.1 construction

<table>
<thead>
<tr>
<th>$u$</th>
<th>$v$</th>
<th>$\lambda$</th>
<th>$\mu$</th>
<th>$\nu$</th>
<th>$x(u,v)$</th>
<th>$y(u,v)$</th>
<th>$z(u,v)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>-18.547</td>
<td>-23.7735</td>
<td>-14.7735</td>
</tr>
<tr>
<td>-7.5</td>
<td>-1.44338</td>
<td>0.25</td>
<td>0.75</td>
<td>0.25</td>
<td>-7.38675</td>
<td>-14.4434</td>
<td>-7.94338</td>
</tr>
<tr>
<td>-5.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>0.75</td>
<td>0.25</td>
<td>-13.547</td>
<td>-13.7735</td>
<td>-9.7735</td>
</tr>
<tr>
<td>-5.0</td>
<td>2.88675</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>3.7735</td>
<td>-5.11325</td>
<td>-1.11325</td>
</tr>
<tr>
<td>-2.5</td>
<td>-1.44338</td>
<td>0.25</td>
<td>0.5</td>
<td>0.25</td>
<td>-2.38675</td>
<td>-4.44338</td>
<td>-2.94338</td>
</tr>
<tr>
<td>-2.5</td>
<td>7.21687</td>
<td>0.75</td>
<td>0.25</td>
<td>0.0</td>
<td>14.9337</td>
<td>4.21687</td>
<td>5.71687</td>
</tr>
<tr>
<td>0.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>-8.547</td>
<td>-3.7735</td>
<td>-4.7735</td>
</tr>
<tr>
<td>0.0</td>
<td>2.88675</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
<td>8.7735</td>
<td>4.88675</td>
<td>3.88675</td>
</tr>
<tr>
<td>0.0</td>
<td>11.547</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>26.094</td>
<td>13.547</td>
<td>12.547</td>
</tr>
<tr>
<td>2.5</td>
<td>-1.44338</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>2.61325</td>
<td>5.55663</td>
<td>2.05662</td>
</tr>
<tr>
<td>2.5</td>
<td>7.21687</td>
<td>0.75</td>
<td>0.0</td>
<td>0.25</td>
<td>19.9337</td>
<td>14.2169</td>
<td>10.7169</td>
</tr>
<tr>
<td>5.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>0.25</td>
<td>0.75</td>
<td>-3.547</td>
<td>6.2265</td>
<td>0.2265</td>
</tr>
<tr>
<td>5.0</td>
<td>2.88675</td>
<td>0.5</td>
<td>0.0</td>
<td>0.5</td>
<td>13.7735</td>
<td>14.8868</td>
<td>8.88675</td>
</tr>
<tr>
<td>7.5</td>
<td>-1.44338</td>
<td>0.25</td>
<td>0.0</td>
<td>0.75</td>
<td>7.61325</td>
<td>15.5566</td>
<td>7.05663</td>
</tr>
<tr>
<td>10.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.453</td>
<td>16.2265</td>
<td>5.2265</td>
</tr>
</tbody>
</table>

Table 13.: Point values for the plane, calculated from equations and using interpolation.

parameter values, the interpolation coefficients $\lambda$, $\mu$, $\nu$, and the corresponding surface point values.

The control points for the triangular surface defined by the equilateral triangle above are the function values at the vertices of the parameter reference triangle, as follows:

$$F(u_r, v_r) = F(0.0, 11.5470) = R(26.094, 13.547, 12.547)$$
$$F(u_s, v_s) = F(-10.0, -5.7735) = S(-18.547, -23.7735, -14.7735)$$
$$F(u_t, v_t) = F(10.0, -5.7735) = T(1.453, 16.2265, 5.2265)$$

Figure 56.: Plane: The red triangular patch represents the reference parameter triangle. The green plane is the parametric surface, computed (a) from parameter values and (b) from control points and interpolation coefficients.
A.1 construction

Quadratics as Total Degree Surfaces

Again, we will consider now a quadratic surface

\[ F_i = a_i U^2 + b_i UV + c_i V^2 + d_i U + e_i V + f_i, \quad \text{where } i = 1, 2, 3 \]  \hspace{1cm} (49)

Instead of polarizing as \( f_i(u_1, u_2; v_1, v_2) \) we will polarize as \( f_i((u_1, v_1), (u_2, v_2)) \), and, in the process, we will derive informally the expression for an \( m \)-degree bivariate polynomial, with \( m = \max(p, q) \), where \( p \) is the degree in \( U \) and \( q \) is the degree in \( V \). We recall that the resulting surface is an \( m \)-total degree surface.

To polarize a quadratic surface of Equation 49, we polarize into two 2-total degree polar variables,

\[
\begin{align*}
(u_1, v_1) &= (\lambda_1 u_r + \mu_1 u_s + \nu_1 u_t, \lambda_1 v_r + \mu_1 v_s + \nu_1 v_t) \\
(u_2, v_2) &= (\lambda_2 u_r + \mu_2 u_s + \nu_2 u_t, \lambda_2 v_r + \mu_2 v_s + \nu_2 v_t),
\end{align*}
\]

or, denoting \((u_{1,2}, v_{1,2}) = \lambda_{1,2}r + \mu_{1,2}s + \nu_{1,2}t\),

\[ f_i((u_1, v_1), (u_2, v_2)) = f_i(\lambda_1 r + \mu_1 s + \nu_1 t, \lambda_2 r + \mu_2 s + \nu_2 t), \]

where, from now on, dropping the point notation \( r, s, t \), it is understood that \( r, s, t \in \mathbb{R}^2 \) are points of coordinates \( r(u_r, v_r), s(u_s, v_s), t(u_t, v_t) \). Using Equation 48, we can expand:

\[
f_i((u_1, v_1), (u_2, v_2)) = \lambda_1 f_i(r, \lambda_2 r + \mu_2 s + \nu_2 t) + \\
+ \mu_1 f_i(s, \lambda_2 r + \mu_2 s + \nu_2 t) + \nu_1 f_i(t, \lambda_2 r + \mu_2 s + \nu_2 t),
\]

or

\[
f_i((u_1, v_1), (u_2, v_2)) = \lambda_1 \lambda_2 f_i(r, r) + \lambda_1 \mu_2 f_i(r, s) + \lambda_1 \nu_2 f_i(r, t) + \\
+ \mu_1 \lambda_2 f_i(s, r) + \mu_1 \mu_2 f_i(s, s) + \mu_1 \nu_2 f_i(s, t) + \\
+ \nu_1 \lambda_2 f_i(t, r) + \nu_1 \mu_2 f_i(t, s) + \nu_1 \nu_2 f_i(t, t) \tag{50}
\]

In tensor form (since \( r, s, t \) are points in \( \mathbb{R}^2 \)), Equation 50 can be rewritten as:

\[
f_i((u_1, v_1), (u_2, v_2)) = \begin{pmatrix} \lambda_1 & \mu_1 & \nu_1 \end{pmatrix} \begin{pmatrix} f_i(r, r) & f_i(r, s) & f_i(r, t) \\
f_i(s, r) & f_i(s, s) & f_i(s, t) \\
f_i(t, r) & f_i(t, s) & f_i(t, t) \end{pmatrix} \begin{pmatrix} \lambda_2 \\
\mu_2 \\
\nu_2 \end{pmatrix} \tag{51}
\]
In Equation 50, we notice that there are $9 = 3^2 = 3^m$ terms with as many control points, each depending on $m = 2$ polar variables. Due to the symmetry in variables $(u_1, v_1)$ and $(u_2, v_2)$, i.e. $f_i((u_1, u_2), (s, s)) = f_i((s, s), (u_1, u_2))$, there are only $6 = \frac{(2+1)(2+2)}{2} = \frac{(m+1)(m+2)}{2}$ distinct function values:

\[
\begin{align*}
1 \times f_i(r, r) \\
1 \times f_i(s, s) \\
1 \times f_i(t, t) \\
2 \times f_i(r, s) \\
2 \times f_i(r, t) \\
2 \times f_i(s, t),
\end{align*}
\]

as well as the same interpolation coefficients in each polar variable $\lambda_1 = \lambda_2 = \lambda$, $\mu_1 = \mu_2 = \mu$ and $\nu_1 = \nu_2 = \nu$, leading to the final expression:

\[
f((u_1, v_1), (u_2, v_2)) = \lambda^2 f_i(r, r) + \mu^2 f_i(s, s) + \nu^2 f_i(t, t) + 2\lambda\mu f_i(r, s) + 2\lambda\nu f_i(r, t) + 2\mu\nu f_i(s, t),
\]

(52)

**Surfaces of Total Degree $m$**

Dropping the index $i$ from the coordinate function $f_i$, in order to avoid confusion with indices $i, j, k$, we will next use Equation 52 to generalize expression 52 from total degree 2 to total degree $m$:

\[
f((u, v), \ldots, (u, v)) = \sum_{i+j+k=m}^m \frac{m!}{i!j!k!} \lambda^i \mu^j \nu^k f(r, \ldots, r, s, \ldots, s, t, \ldots, t)
\]

(53)

If we let $m = 2$ and $\sigma(i, j, k) = \sigma(1, 1, 0)$ with $i + j + k = m = 2$ in Equation 53, we get the particular case for quadratic surfaces from Equation 52.

**Example: The Hyperbolic Paraboloid as a Total-Degree Surface**

In what follows, we will present an example of quadratic surface obtained from coordinate functions for different parameter values, as well as obtained from linear interpolation for different barycentric coordinates (corresponding to these same parameter
Figure 57: Hyperbolic paraboloid: The red triangular patch represents the reference parameter triangle, with 4 steps of midpoint subdivision. The purple triangulation represents the parametric surface, computed (a) from parameter values and (b) from control points and interpolation coefficients.

values) and control points (polarized function values at the various combinations of polar variables \( r, s, t \)). We detail again the example of the hyperbolic paraboloid:

\[
F = \begin{cases} 
F_1(U, V) = U \\
F_2(U, V) = V \\
F_3(U, V) = \frac{1}{10} U^2 - \frac{1}{10} V^2 
\end{cases}
\]

<table>
<thead>
<tr>
<th>( f(r, s, t) )</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(r, r) )</td>
<td>0.0</td>
<td>11.547</td>
<td>-13.3333</td>
</tr>
<tr>
<td>( f(r, s) )</td>
<td>-5.0</td>
<td>2.88765</td>
<td>6.66666</td>
</tr>
<tr>
<td>( f(r, t) )</td>
<td>5.0</td>
<td>2.88765</td>
<td>6.66666</td>
</tr>
<tr>
<td>( f(s, r) )</td>
<td>-5.0</td>
<td>2.88765</td>
<td>6.66666</td>
</tr>
<tr>
<td>( f(s, s) )</td>
<td>-10.0</td>
<td>-5.7735</td>
<td>6.66666</td>
</tr>
<tr>
<td>( f(s, t) )</td>
<td>0.0</td>
<td>-5.7735</td>
<td>-13.3333</td>
</tr>
<tr>
<td>( f(t, r) )</td>
<td>5.0</td>
<td>2.88675</td>
<td>6.66666</td>
</tr>
<tr>
<td>( f(t, s) )</td>
<td>0.0</td>
<td>-5.7735</td>
<td>-13.3333</td>
</tr>
<tr>
<td>( f(t, t) )</td>
<td>10.0</td>
<td>-5.7735</td>
<td>6.66666</td>
</tr>
</tbody>
</table>

Table 14.: Control point values for the hyperbolic paraboloid, with all the repeats. These values are based on the reference triangle \( \triangle r, s, t \) of parameter coordinates \( r(0.0, 11.5470), s(-10.0, -5.7735), t(10.0, -5.7735) \).
A.1 construction

This surface, considered as a total degree surface of degree 2, will polarize as follows:

\[
\begin{align*}
 f((u_1, v_1), (u_2, v_2)) &= \begin{cases} 
 f_1((u_1, v_1), (u_2, v_2)) = \frac{u_1 + u_2}{2} \\
 f_2((u_1, v_1), (u_2, v_2)) = \frac{v_1 + v_2}{2} \\
 f_3((u_1, v_1), (u_2, v_2)) = \frac{1}{10}(u_1 u_2 - v_1 v_2)
\end{cases}
\]

The 9 control points are represented in Table 14. We chose to keep the repeated control points as such, in order to retain the ability to use the matrix system from Equation 51, which is only 3 x 3 for surfaces of total degree \( m = 2 \), obeying the general size of \( 3^m \) control points (thus \( 3^2 \) control points for \( m = 2 \), arranged in a 3 x 3 matrix), but exhibiting only \( C^2_{m+2} = \frac{(m+1)(m+2)}{2} \) (in this case \( \frac{(2+1)(2+2)}{2} = 6 \)) non-repeat terms, as can be verified informally for \( m = 2 \) from Table 14.

Of these control points, only some are on the surface: those that correspond to actual function values: \( f(r, r) \equiv F(r), f(s, s) \equiv F(s) \), and \( f(t, t) \equiv F(t) \) (remembering that \( f \) is the total degree polar form of \( F \)). In Table 15 we show the points generated through direct computation from parameters, as well as through interpolation, for the same reference triangle, with 2 subdivision steps, and the control points highlighted.

<table>
<thead>
<tr>
<th>u</th>
<th>v</th>
<th>λ</th>
<th>μ</th>
<th>v</th>
<th>x(u,v)</th>
<th>y(u,v)</th>
<th>z(u,v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>-10.0</td>
<td>-5.7735</td>
<td>6.66667</td>
</tr>
<tr>
<td>-7.5</td>
<td>-1.44338</td>
<td>0.25</td>
<td>0.75</td>
<td>0.0</td>
<td>-7.5</td>
<td>-1.44337</td>
<td>5.41667</td>
</tr>
<tr>
<td>-5.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>0.75</td>
<td>0.25</td>
<td>-5.0</td>
<td>-5.7735</td>
<td>-0.83333</td>
</tr>
<tr>
<td>-5.0</td>
<td>2.88675</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>-5.0</td>
<td>2.88675</td>
<td>1.66667</td>
</tr>
<tr>
<td>-2.5</td>
<td>-1.44338</td>
<td>0.25</td>
<td>0.5</td>
<td>0.25</td>
<td>-2.5</td>
<td>-1.44338</td>
<td>0.41667</td>
</tr>
<tr>
<td>-2.5</td>
<td>7.21687</td>
<td>0.75</td>
<td>0.25</td>
<td>0.0</td>
<td>-2.5</td>
<td>7.21687</td>
<td>-4.58333</td>
</tr>
<tr>
<td>0.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
<td>-5.7735</td>
<td>-3.33333</td>
</tr>
<tr>
<td>0.0</td>
<td>2.88675</td>
<td>0.5</td>
<td>0.25</td>
<td>0.25</td>
<td>0.0</td>
<td>2.88675</td>
<td>-0.83333</td>
</tr>
<tr>
<td>0.0</td>
<td>11.547</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>11.547</td>
<td>-13.3333</td>
</tr>
<tr>
<td>2.5</td>
<td>-1.44338</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>2.5</td>
<td>-1.44337</td>
<td>0.41667</td>
</tr>
<tr>
<td>2.5</td>
<td>7.21687</td>
<td>0.75</td>
<td>0.0</td>
<td>0.25</td>
<td>2.5</td>
<td>7.21687</td>
<td>-4.58333</td>
</tr>
<tr>
<td>5.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>0.25</td>
<td>0.75</td>
<td>5.0</td>
<td>-5.7735</td>
<td>-0.83333</td>
</tr>
<tr>
<td>5.0</td>
<td>2.88675</td>
<td>0.5</td>
<td>0.0</td>
<td>0.5</td>
<td>5.0</td>
<td>2.88675</td>
<td>1.66667</td>
</tr>
<tr>
<td>7.5</td>
<td>-1.44338</td>
<td>0.25</td>
<td>0.0</td>
<td>0.75</td>
<td>7.5</td>
<td>-1.44337</td>
<td>5.41667</td>
</tr>
<tr>
<td>10.0</td>
<td>-5.7735</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>10.0</td>
<td>-5.7735</td>
<td>6.66667</td>
</tr>
</tbody>
</table>

Table 15.: Hyperbolic paraboloid point values, from equations vs. interpolation. The boldface values are also control points, as cross-referenced with Table 14.
A.2 approximation

Given real data points, the goal is to generate an approximating surface not necessarily passing through them, when the surface is unknown except at the data points, and the control points are unknown. The least squares approximation method generates control points from Bernstein coefficients calculated at the parameter points corresponding to the data points, and from the real data points. Using these data-trained control points, it computes new points on the surface from 2D new resamples and from the Bernstein coefficients calculated at the resamples.

A.2.1 Data-Trained Control Points

Calculating the control points through a computation inverse to the one yielding a surface point from a support net of degree \((p, q)\) and Bernstein coefficients \(B_{i,j}^p(u)B_{i,j}^q(v)\), with \(i \in \{1, \ldots, p\}, j \in \{1, \ldots, q\}\) is equivalent to finding the pseudo-inverse of the non-square Bernstein matrix:

\[
\begin{pmatrix}
\mathbf{b}_{0,0} \\
\vdots \\
\mathbf{b}_{p,q}
\end{pmatrix} = \left( \begin{pmatrix} B_0^p(u_0)B_0^q(v_0) & \cdots & B_p^p(u_0)B_q^q(v_0) \\ \vdots & \ddots & \vdots \\ B_0^p(u_k)B_0^q(v_k) & \cdots & B_p^p(u_k)B_q^q(v_k) \end{pmatrix} \right)^{-1} \cdot \begin{pmatrix} \mathbf{p}_0 \\
\vdots \\
\mathbf{p}_k
\end{pmatrix}, \quad (54)
\]

Figure 58: Hyperbolic paraboloid in two resolutions, showing the control points.
where \( (p, q) \) is the polar degree of \( F \) (unknown), \( \lambda \) is the interpolation ratio
\[
\langle p, q \rangle \ = \ C_{n}^{i} \lambda^{i}(1 - \lambda)^{p-i}, \text{ with } C_{n}^{i} = \frac{n!}{i!(n-i)!} \text{ and } i \in \{0, \ldots, p\}, j \in \{0, \ldots, q\}. \]
The vector of control points \( (b_{0,0} \ldots b_{p,q})^{T} \) is a linearized version of the control points sequence, the Bernstein coefficients matrix is calculated from parameter values corresponding to original data points, and the vector \( (p_{0} \ldots p_{k})^{T} \) are the data points themselves.

### A.2.2 Least-Squares Reconstruction

Calculating the surface points from the data-trained control points \( (b_{0,0} \ldots b_{p,q}) \) at the new 2D resamples \((u'_{0}, v'_{0}), \ldots, (u'_{j}, v'_{j})\) requires the recalculation of the Bernstein matrix of coefficients to suit resampled parameter points and the control net calculated in the previous step.

\[
\begin{pmatrix}
 p'_{0} \\
 \vdots \\
 p'_{k}
\end{pmatrix} =
\begin{pmatrix}
 B_{0}^{p}(u'_{0})B_{0}^{q}(v'_{0}) & \cdots & B_{p}^{p}(u'_{0})B_{0}^{q}(v'_{0}) \\
 \vdots & \ddots & \vdots \\
 B_{0}^{p}(u'_{j})B_{0}^{q}(v'_{j}) & \cdots & B_{p}^{p}(u'_{j})B_{0}^{q}(v'_{j})
\end{pmatrix}
\begin{pmatrix}
 b_{0,0} \\
 \vdots \\
 b_{p,q}
\end{pmatrix}
\]

The calculation of the points \( (p'_{0} \ldots p'_{k})^{T} \) from only the control points and the Bernstein polynomial coefficients is possible in the absence of the surface polynomial(s) due to the fact that multilinear interpolation of parametric polynomials only depends on the control points and the ratio of interpolation of the resamples. The shape of the function is entirely encapsulated in the control points, while the Bernstein polynomials only contain information on the ratio of interpolation for the new points. The degree of the polynomial is “transferred” onto the part expressing the ratio of interpolation, through the size of the matrix of Bernstein coefficients, while the actual point computation is being done over fixed points (the control points). Examples of particular cases of second-degree parametric curves and third-degree parametric surfaces will be given in the section on reconstruction.

There are more types of surface construction, using rational fractions (rational polynomials), as well as using multiple functions on adjoining domains (such as cubic B-splines, described in Bartels et al. [1987]). In the latter case, at the points of juncture between curve/surface segments, called knots, various degrees of continuity and knot multiplicity conditions define the level of interpolation vs. approximation of the data points, i.e. how precisely the curve/surface conforms to the control polygon (smooth

---

1 It can be different in each direction for rectangular patches, while for triangular patches it is a composite of barycentric coordinates. This explanation is enough to convey the idea that the Bernstein polynomials depend entirely on the ratio of interpolation and the degree of the function being interpolated/approximated.
A.2 approximation

approximation vs. less smooth interpolation). Since, in this case, the shape of the cubic polynomials is a priori determined from the continuity conditions for the four segments defining each basis function, and therefore known and fixed, and since the shape of the curve/surface is a linear combination of the basis functions weighted by the control points, the extension of this method to data fitting is more immediate in applications where a fixed number of points needs to be represented in the original data granularity and (possibly) resampled at different locations.

In our case, the original data is densely sampled, which means that a simple linear interpolation within each original mesh triangle is sufficient to locate 2D resampled parameter points. In the future, it would be interesting to relate a cubic B-spline reconstruction (based on the patches from our segmentation) to the local linear interpolation scheme about to be described next, in Section A.2.3.

A.2.3 Approximation through Local Linear Interpolation

Finding positions on the surface at points in-between data points (resampling) in the parameter domain by assuming a certain shape of the surface (i.e. committing to a certain number of interpolation steps) In this formulation of the problem, the phase of surface reconstruction uses the method from Paragraph 1.4, by taking the resample \( \mathbf{a} = \lambda \mathbf{r} + \mu \mathbf{s} + \nu \mathbf{t} \) within the parameter reference triangle \( \mathbf{t} = \triangle \mathbf{rst} \in P \) and computing a linear or polynomial image \( F(\mathbf{a}) \). For example, in the linear case, if \( T = \triangle ABC \xrightarrow{\phi} \mathbf{t} = \triangle \mathbf{rst} \), where \( \phi : \mathbb{R}^3 \rightarrow \mathbb{R}^2 \) is a geometry-preserving parametrization function that can generate the parameter domain from the data points, the surface can be bilinearly interpolated as follows:

\[
F(\mathbf{a}) = \begin{cases} 
F_1(u, v) &= x(u, v) = \lambda x_A(u, v) + \mu x_B(u, v) + \nu x_C(u, v) \\
F_2(u, v) &= y(u, v) = \lambda y_A(u, v) + \mu y_B(u, v) + \nu y_C(u, v) \\
F_3(u, v) &= z(u, v) = \lambda z_A(u, v) + \mu z_B(u, v) + \nu z_C(u, v),
\end{cases}
\]

where the barycentric coefficients of \( \mathbf{a} \) are equivalent to a linear interpolation in the direction of each vector \( \overrightarrow{rs} = \mathbf{s} - \mathbf{r}, \overrightarrow{rt} = \mathbf{t} - \mathbf{r} \in \mathbf{t} \), but dependent on parametric coordinates \( (u, v) \) with respect to a local coordinate system, thus defining a bilinear interpolation of \( F \) in \( u \) and \( v \). If we denote, for simplification,

\[
\lambda(u, v) \text{ notation } = \lambda_a, \mu(u, v) \text{ notation } = \mu_a, \nu(u, v) \text{ notation } = \nu_a,
\]

we use piecewise linear in our implementation, meaning linear within each reference triangle, with one interpolation step.
then \( a(u, v) \in t = \triangle rst \) will be located with barycentric coordinates as:

\[
\begin{align*}
\lambda &= \lambda_a u_r + \mu_a u_s + \nu_a u_t \\
\mu &= \lambda_a v_r + \mu_a v_s + \nu_a v_t \\
\nu &= \lambda_a + \mu_a + \nu_a = 1,
\end{align*}
\]

or, in matrix form:

\[
\begin{pmatrix}
u_r & u_r & u_t \\
v_s & v_r & v_t \\
1 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
\lambda_a \\
\mu_a \\
\nu_a
\end{pmatrix} =
\begin{pmatrix}
u \\
u
\end{pmatrix}
\]

Thus, in the linear case, the support net for the interpolation consists of only one triangle (the reference triangle \( T = F(t) = F(\triangle rst) \)) and one step of two-dimensional, linear interpolation with barycentric coefficients \((\lambda_a \mu_a \nu_a)\)\(^T\), corresponding to a discrete bivariate polynomial of degree one

\[
F(U, V) \approx \phi^{-1}(U, V),
\]

where \( \phi : \mathbb{R}^3 \to \mathbb{R}^2, F \approx \phi^{-1} : \mathbb{R}^2 \to \mathbb{R}^3 \), with control points retrieved through the inverse of the discrete parametrization \( \phi \):

\[
B_{F(\triangle r,s,t)} = F(b_{\triangle r,s,t}) = \phi^{-1}(b_{\triangle r,s,t}) = \triangle A, B, C
\]

exactly as the original data points. We consider the original triangular structure as a control net over which only one step of linear interpolation is performed.

The reconstructed point \( F(a) \in F(\triangle rst) \)\(^3\) can be therefore retrieved as:

\[
\begin{pmatrix}
\lambda_a \\
\mu_a \\
\nu_a
\end{pmatrix}^{-1}
\begin{pmatrix}
u \\
u
\end{pmatrix}
\]

For higher order polynomials, interpolation is done by deriving the control points (or approximations thereof) from a data set and using these data-trained control points to calculate new 3D points at 2D resamples in the parameter space.

\footnote{For simplicity of notation, the literature replaces \( F(\triangle rst) \) with just triplets \((r, s, t)\), which may confusingly imply that the interpolation over control points is still in parameter space.}
Since all that is needed for generating the surface at any point, once the control points are calculated, is an interpolation ratio that will generate the Bernstein polynomial coefficients, subdivision can also be used for surface reconstruction. Given a control net and a subdivision scheme in the parameter space, new control nets can be created to generate the unknown polynomial surface at finer detail. The difference between subdivision and interpolation consists of the fact that subdivision requires an initial control net and generates new control nets at each step, while interpolation departs from a single initial control net. The shifting of focus towards subdivision stems from the complexity of control nets necessary to design surfaces of higher degrees, and of joining differently defined surfaces, which subdivision avoids through simplifying the interpolation rules by using only barycentric (linear) interpolation, combined with stencil weight masks and, in an approximation setting, an update of the stencil vertices themselves. The Loop scheme is an example of a subdivision approximating scheme, while the Butterfly subdivision is interpolating.

**A.4 INTERPOLATION**

*Data-Trained Control Points*

Through a system similar to the one derived from Equation 45, the \( p \cdot q \) (for the bipolynomial case) and the \( \frac{(m+1)(m+2)}{2} \) (for the total degree case) control points can be derived from a forward system, in which the data points and their Bernstein coefficients are known, provided that a good parametrization of the data points is available. The control points are the unknown. For a single data point:

\[
x(\lambda, \gamma) = \begin{pmatrix} B_0^{p}(\lambda) & \cdots & B_{p}^{p}(\lambda) \\
\vdots & \ddots & \vdots \\
B_0^{q}(\gamma) & \cdots & B_{q}^{q}(\gamma) \end{pmatrix} \begin{pmatrix} b_{00} & \cdots & b_{0q} \\
\vdots & \ddots & \vdots \\
b_{p0} & \cdots & b_{pq} \end{pmatrix} \begin{pmatrix} B_0^{q}(\gamma) \end{pmatrix}
\]

For each resampled point, the elements of the Bernstein coefficient vectors, consisting of \( 2^p \) and \( 2^q \) entries respectively, can be computed for a complete matrix of control points of size \( 2^p2^q = 2^{p+q} \), or, based on repeat entries, can be collapsed into \( p+1, q+1 \) vectors, for a reduced matrix of control points of size \( (p+1)(q+1) \).
For example, in the quadratic case:

\[ x(\lambda, \gamma) = \begin{pmatrix} (1 - \lambda)^2 & (1 - \lambda)\lambda & \lambda(1 - \lambda) & \lambda^2 \end{pmatrix} B \begin{pmatrix} (1 - \gamma)^2 \\ (1 - \gamma)\gamma \\ \gamma(1 - \gamma) \\ \gamma^2 \end{pmatrix} \]

can be collapsed into

\[
\begin{pmatrix} C_2(1 - \lambda)^2 & C_1(1 - \lambda)\lambda & C_0\lambda^2 \end{pmatrix} B \begin{pmatrix} C_2(1 - \gamma)^2 \\ C_1(1 - \gamma)\gamma \\ C_0\gamma^2 \end{pmatrix}
\]

For \( K = (m + 1)(n + 1) \) data points whose parameter values can be associated with the Bernstein coefficients per point as above, we can build the system:

\[
\begin{pmatrix} \mathbf{x}_{00} & \ldots & \mathbf{x}_{0n} \\
\vdots & \ddots & \vdots \\
\mathbf{x}_{m0} & \ldots & \mathbf{x}_{mn} \end{pmatrix} \begin{pmatrix} B_0^0(u_0) & \ldots & B_p^0(u_0) \\
\vdots & \ddots & \vdots \\
B_0^p(u_m) & \ldots & B_p^p(u_m) \end{pmatrix} \mathbf{B} \begin{pmatrix} B_0^0(v_0) & \ldots & B_q^0(v_0) \\
\vdots & \ddots & \vdots \\
B_0^q(v_n) & \ldots & B_q^q(v_n) \end{pmatrix}^T,
\]

where

\[ B_p^j(u_i) = C_p^j(1 - \lambda_i)^j\lambda_i^{p-j} \]

and, similarly,

\[ B_q^k(v_i) = C_q^k(1 - \gamma_i)^k\gamma_i^{q-k}. \]

Solving first for the unknown control points matrix \( \mathbf{B} \), this can be used to generate new 3D points \( \mathbf{X} \) at new 2D resamples in parameter space.

**Reconstructed Points**

Using the now computed control points from Section A.4, calculating the image points \( F(\mathbf{a}) \in \mathbb{R}^3 \), where \( \mathbf{a} = (u, v) \in \mathcal{P} \subset \mathbb{R}^2 \), is equivalent to two matrix-vector multiplications, leading to the calculation of \( \mathbf{X} = F(\mathbf{a}) \). This applies to surfaces whose points can be organized into rectangular patches \( (m + 1)(n + 1) \), i.e. tensor product surfaces. This section described tensor product interpolation.
PARAMETRIZATION FUNDAMENTALS

B.0.1 The Euclidean Metric Tensor

We consider the mapping $F : \Omega \subset \mathbb{R}^2 \rightarrow S' \subset \mathbb{R}^3$. Then the vector $d\mathbf{u}(u, v) \subset \Omega$ maps to the vector $d\mathbf{r}(x(u, v), y(u, v), z(u, v)) \subset \mathbb{R}^3$. We want to approximate the variation $dF(d\mathbf{u})$ of vector $d\mathbf{u}$ under $F$:

$$
\frac{dF}{d\mathbf{u}} = \frac{\partial F}{\partial u} du + \frac{\partial F}{\partial v} dv,
$$
where $F = \begin{cases}
    x(u, v) = F_1(u, v) \\
y(u, v) = F_2(u, v) \\
z(u, v) = F_3(u, v)
\end{cases}$

For the purpose of this approximation, we need the images $F(u, v)$ and $F(u + du, v + dv)$, forming the vector $d\mathbf{r}$, so we can calculate $d\mathbf{r}$ as the vector difference $F(u + du, v + dv) - F(u, v)$. First, we note that:

$$
F(u + du, v + dv) = F(u, v) + dF(u, v) + o(\sqrt{du^2 + dv^2}),
$$
with $o(\sqrt{du^2 + dv^2}) \approx 0$. Then:

$$
d\mathbf{r} \equiv F(u + du, v + dv) - F(u, v) \simeq dF(u, v),
$$
and thus $dF$ approximates the image $d\mathbf{r}$ of vector $d\mathbf{u}$ under $F$:

$$
d\mathbf{r} \simeq dF = \frac{\partial F}{\partial u} du + \frac{\partial F}{\partial v} dv = r_u du + r_v dv
$$

The dot product $d\mathbf{r} \cdot d\mathbf{r}$ encapsulates this difference as a positive scalar. When $du, dv$ are small, vector $d\mathbf{r}$ approximates a spatial elementary arc of curve based at $F(u, v) \in S'$, for which the following quantity is introduced:

$$
I = d\mathbf{r} \cdot d\mathbf{r} = (r_u du + r_v dv) \cdot (r_u du + r_v dv) = (r_u \cdot r_u) du^2 + 2(r_u \cdot r_v) du dv + (r_v \cdot r_v) dv^2,
$$

(63)

149
where

\[ r_u = \frac{\partial F}{\partial u} = \frac{\partial x}{\partial u} i + \frac{\partial y}{\partial u} j + \frac{\partial z}{\partial u} k, \quad \text{and} \quad r_v = \frac{\partial F}{\partial v} = \frac{\partial x}{\partial v} i + \frac{\partial y}{\partial v} j + \frac{\partial z}{\partial v} k. \]

With the notations:

\[ E = r_u \cdot r_u, \quad F = r_u \cdot r_v, \quad \text{and} \quad G = r_v \cdot r_v, \]

the quantity \( dr \cdot dr \) from Equation 63, called the first fundamental form of surface \( r(u,v) \), becomes

\[ I = E \ du^2 + 2F \ du \ dv + G \ dv^2. \]

Because it characterizes the change of \( du \) under \( F \), the first fundamental form can then be interpreted as the stretch of an elementary vector \( du \) under the parametric mapping \( F \). When the surface \( S \) is given, and the parametric domain must be computed discretely, the mapping \( F^{-1} \) must ideally satisfy the condition of minimizing this stretch.

With the alternate notations:

\[ g_{11} = E = \frac{\partial r}{\partial x^1} \cdot \frac{\partial r}{\partial x^1}, \quad g_{12} = g_{21} = F = \frac{\partial r}{\partial x^1} \cdot \frac{\partial r}{\partial x^2}, \quad \text{and} \quad g_{22} = G = \frac{\partial r}{\partial x^2} \cdot \frac{\partial r}{\partial x^2}, \]

we get

\[ I = g_{11} (dx^1)^2 + g_{12} dx^1 \ dx^2 + g_{21} \ dx^1 \ dx^2 + g_{22} (dx^2)^2 = \sum_{i,j=1}^{2} g_{ij} \ dx^i \ dx^j, \]

(or with the summation convention \( I = g_{ij} \ dx^i \ dx^j \)). The fundamental metric tensor is defined as the matrix

\[ g = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \quad (64) \]

Its determinant is the quantity:

\[ g = \det g = \begin{vmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{vmatrix} = g_{11} g_{22} - g_{12} g_{21} = EG - F^2. \]

The following quantities are further introduced:

\[ g^{11} = \frac{g_{22}}{g}, \quad g^{12} = g^{21} = -\frac{g_{12}}{g}, \quad \text{and} \quad g^{22} = \frac{g_{11}}{g}. \]
They are the components of the conjugate metric tensor, which is the inverse of the fundamental metric tensor:

\[
(g^{ij})_{nn} = \frac{1}{\det(g)} \begin{pmatrix} g^{22} & -g^{21} \\ -g^{12} & g^{11} \end{pmatrix} = (g_{ij})^{-1}_{nn}
\]

In tensor analysis, \((g_{ij})\) is a covariant tensor of the second order\(^1\) because it transforms as follows under a coordinate system transformation from \(x^i\) to \(\bar{x}^i\):

\[
\bar{g}_{ij} = g_{rs} \frac{\partial x^r}{\partial x^i} \frac{\partial x^s}{\partial x^j}.
\]

An equally useful result is the following relationship between the metric tensor matrix \(G\) and the Jacobian matrix \(J = \left(\frac{d\bar{x}^i}{dx^j}\right)\) associated with the change of coordinates transformation from a system \((x^i)\) to a rectangular system \((\bar{x}^i)\):

\[
G = J^T J. \quad (65)
\]

If calculating the trace of the matrix \(G\) using \(65\), the following interesting result appears, as carried out in Hormann and Greiner [2000]. First, note that

\[
\text{trace}(G) = \text{trace}(JJ^T), \quad (66)
\]

where

\[
(U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma(V^T V)\Sigma^T U^T = U\Sigma\Sigma^T U^T = U\Sigma^2 U^T = (u_1|u_2) \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}
\]

As \(\text{trace}(A) = \sum \lambda_i\), where \(\lambda_i\) are the eigenvalues of \(A\), and the square roots of the singular values of \(AA^T\), then the trace of the metric tensor matrix and that of the conjugate metric tensor matrix \(G^{-1}\) can be calculated immediately as:

\[
\text{trace}(G) = \sigma_1^2 + \sigma_2^2 \quad \text{and} \quad \text{trace}(G^{-1}) = \frac{\text{trace}(G)}{\det G} = \frac{\sigma_1^2 + \sigma_2^2}{\sigma_1^2 \sigma_2^2}, \quad (67)
\]

based on the well-known identity \(\det A = \prod \lambda_i\), which applied to \(G\) means

\[
\det G = \det JJ^T = (\det J)^2 = (\sigma_1 \sigma_2)^2.
\]

\(^1\) It thus has lower coefficients. Upper coefficients indicate a contravariant tensor.
B.0.2 Equiareal (Authalic) Maps

The transformation of a two-dimensional rectangular patch into a surface curvilinear patch can be analyzed similar to the transformation of a two-dimensional vector into a three-dimensional curve under a mapping \( \mathbf{F} \). The area of a small curvilinear quasi-rectangular patch bounded by points \( \mathbf{r}(u, v) \), \( \mathbf{r}(u + du, v) \), \( \mathbf{r}(u, v + dv) \), and \( \mathbf{r}(u + du, v + dv) \) can be approximated as the area of the parallelogram formed by the vectors \( \mathbf{r}_u(u, v) \) and \( \mathbf{r}_v(u, v) \):

\[
\text{Area}(A) = |\mathbf{r}_u \times \mathbf{r}_v| \ du \ dv = \left| \begin{array}{ccc}
1 & j & k \\
\frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\
\frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v}
\end{array} \right| du \ dv =
\]

\[
= \left| \left( \frac{\partial y}{\partial u} \frac{\partial z}{\partial v} - \frac{\partial z}{\partial u} \frac{\partial y}{\partial v} \right) \mathbf{i} - \left( \frac{\partial x}{\partial u} \frac{\partial z}{\partial v} - \frac{\partial z}{\partial u} \frac{\partial x}{\partial v} \right) \mathbf{j} + \left( \frac{\partial x}{\partial u} \frac{\partial y}{\partial v} - \frac{\partial y}{\partial u} \frac{\partial x}{\partial v} \right) \mathbf{k} \right|
\]

\[
= \left| \frac{\partial(y, z)}{\partial(u, v)} \mathbf{i} + \frac{\partial(z, x)}{\partial(u, v)} \mathbf{j} + \frac{\partial(x, y)}{\partial(u, v)} \mathbf{k} \right| = \sqrt{\left( \frac{\partial(x, y)}{\partial(u, v)} \right)^2 + \left( \frac{\partial(y, z)}{\partial(u, v)} \right)^2 + \left( \frac{\partial(x, z)}{\partial(u, v)} \right)^2}
\]

Using the identity

\[
|\mathbf{a} \times \mathbf{b}|^2 = |\mathbf{a}|^2|\mathbf{b}|^2 - |\mathbf{a} \cdot \mathbf{b}|^2, \quad \forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^n \iff |\mathbf{a} \times \mathbf{b}| = \sqrt{|\mathbf{a}|^2|\mathbf{b}|^2 - |\mathbf{a} \cdot \mathbf{b}|^2},
\]

we find that

\[
\text{Area}(A) = |\mathbf{r}_u \times \mathbf{r}_v| \ du \ dv = \sqrt{|\mathbf{r}_u|^2|\mathbf{r}_v|^2 - |\mathbf{r}_u \cdot \mathbf{r}_v|^2} \ du \ dv = \sqrt{EG - F^2} \ du \ dv.
\]
In order for the areas over the whole respective domains to be equal, \( \text{Area}(S') = \text{Area}(\Omega) \), using the Theorem of Change of Variables in a Double Integral, we get:

\[
\int\int_{S'} f(x,y,z) \, dx \, dy \, dz = \int\int_{\Omega} f(r(u,v)) |r_u \times r_v| \, du \, dv,
\]

or

\[
\int\int_{S'} f(x,y,z) \, dx \, dy \, dz = \int\int_{\Omega} f(r(u,v)) \sqrt{EG - F^2} \, du \, dv,
\]

which requires that

\[
\sqrt{EG - F^2} = 1.
\]

b.0.3 Conformal Maps

We assume the change of coordinates is taking place between orthogonal systems, i.e. \((x_i) = (i, j)\) and \((\bar{x}_i) = (i, j, k)\). In order for a transformation to be conformal, the angle formed by the tangent vectors to two intersecting curves on the surface \(S'\), given by

\[
\cos \alpha_{S'} = \frac{dr \cdot \delta r}{|dr||\delta r|} = \frac{(r_u \, du + r_v \, dv) \cdot (r_u \, \delta u + r_v \, \delta v)}{|r_u \, du + r_v \, dv||r_u \, \delta u + r_v \, \delta v|} = \frac{E \, du \, \delta u + F(du \, \delta v + dv \, \delta u) + G \, dv \, \delta v}{\sqrt{E \, du^2 + 2F \, du \, dv + G \, dv^2} \sqrt{\delta u^2 + 2F \, \delta u \, \delta v + G \, \delta v^2}}
\]

must be equal to the angle formed by the corresponding vectors in parameter space:

\[
\cos \alpha_{\Omega} = \frac{(du + dv) \cdot (\delta u + \delta v)}{|du + dv||\delta u + \delta v|} = \frac{du \, \delta u + (du \cdot \delta v + dv \cdot \delta u) + dv \, \delta v}{\sqrt{du^2 + dv^2} \sqrt{\delta u^2 + \delta v^2}}.
\]

By imposing \( \cos \alpha_{S'} = \cos \alpha_{\Omega} \), after reduction of the orthogonal components \( du \cdot \delta v = dv \cdot \delta u = 0 \), we get:

\[
E = G = \eta, \quad \text{and} \quad F = 0. \quad (68)
\]

This result is consistent with the differential geometry theory, which asserts the general principle of equivalence between the condition of proportionality of the coefficients of the first fundamental form and the conformal property of the mapping. As a particular case, the condition for a 3D to 2D mapping to be conformal, given that the systems of coordinates are orthogonal, is then derived as follows.

If the change of coordinates is taking place between two surfaces, \( S \) and \( S^* \), represented as \( r = r(u^1, u^2) \) and \( r^* = r^*(u^1, u^2) \), we remind that:

\[
\frac{dr}{du^1} \, du^1 + \frac{dr}{du^2} \, du^2 = r_\alpha du^\alpha, \quad \text{where} \quad r_\alpha = \frac{\partial r}{\partial u^\alpha}, \quad \alpha = 1, 2.
\]
The angle between two curves on \( S \), \( u^\alpha = h^\alpha(t) \) and \( u^\alpha = h^\alpha(t) \), is given by

\[
\cos \gamma = \frac{g_{\alpha\beta}h^\alpha h^\beta'}{\sqrt{g_{\mu\nu}h^\mu h^\nu} \sqrt{g_{\sigma\tau}h^\sigma h^\tau}},
\]  

(69)

where \( g_{\alpha\beta} = r_\alpha r_\beta \) characterizes the change of coordinates and \( h^\alpha = \frac{dh^\alpha}{dt} \) are the coordinate-independent components of the tangent vectors to the two curves. Then, according to Kreyszig [1991], pp.193-194, we have the following

**Theorem B.0.1** An allowable mapping of a portion \( S \) of a surface onto a portion \( S^* \) of a surface is conformal if and only if, when on \( S \) and \( S^* \) the same coordinate systems have been introduced, the coefficients \( g_{\alpha\beta} \) and \( g^*_{\alpha\beta} \) of the first fundamental forms of \( S \) and \( S^* \), respectively, are proportional,

\[
g^*_{\alpha\beta} = \eta(u^1, u^2) g_{\alpha\beta}, \quad \eta > 0, \quad \alpha, \beta = 1, 2.
\]  

(70)

As an immediate corollary to Theorem B.0.1 under an allowable mapping of a portion \( S \) of a surface into a plane, with \( (u^1, u^2) \) as Cartesian coordinates in the plane, if \( (u^1, u^2) \) are introduced on \( S \), and the mapping is conformal, then

\[
(dr)^2 = \eta(u^1, u^2)[(du^1)^2 + (du^2)^2],
\]  

(71)

which, compared to

\[
(dr)^2 = I = E \, du^2 + 2F \, du \, dv + G \, dv^2,
\]

becomes equivalent to

\[
E = G = \eta(u^1, u^2), \quad \text{and} \quad F = 0
\]  

(72)

(The difference from the previous proof lies in the proof of proportionality being done prior to the introduction of the condition of the orthogonality of the coordinate systems).

---

2 The Jacobian of the transformation is not null.
3 The proof to Theorem B.0.1 is presented in [Kreyszig, 1991] on pp.193-194.
We now develop the coefficients of $I$ adding the conformality condition from \cite{68}:

\[ E = r_u \cdot r_u = \left( \frac{\partial x}{\partial u} i + \frac{\partial y}{\partial u} j + \frac{\partial z}{\partial u} k \right) \cdot \left( \frac{\partial x}{\partial u} i + \frac{\partial y}{\partial u} j + \frac{\partial z}{\partial u} k \right) = \eta \]

\[ G = r_v \cdot r_v = \left( \frac{\partial x}{\partial v} i + \frac{\partial y}{\partial v} j + \frac{\partial z}{\partial v} k \right) \cdot \left( \frac{\partial x}{\partial v} i + \frac{\partial y}{\partial v} j + \frac{\partial z}{\partial v} k \right) = \eta \]

\[ F = r_u \cdot r_v = \left( \frac{\partial x}{\partial u} i + \frac{\partial y}{\partial u} j + \frac{\partial z}{\partial u} k \right) \cdot \left( \frac{\partial x}{\partial v} i + \frac{\partial y}{\partial v} j + \frac{\partial z}{\partial v} k \right) = 0 \]

By developing the dot products, after eliminating the null products containing orthogonal unit vectors, we get:

\[ \left( \frac{\partial x}{\partial u} \right)^2 + \left( \frac{\partial y}{\partial u} \right)^2 + \left( \frac{\partial z}{\partial u} \right)^2 = \eta \]  \hspace{1cm} (73)

\[ \left( \frac{\partial x}{\partial v} \right)^2 + \left( \frac{\partial y}{\partial v} \right)^2 + \left( \frac{\partial z}{\partial v} \right)^2 = \eta \]  \hspace{1cm} (74)

\[ \frac{\partial x}{\partial u} \frac{\partial x}{\partial v} + \frac{\partial y}{\partial u} \frac{\partial y}{\partial v} + \frac{\partial z}{\partial u} \frac{\partial z}{\partial v} = 0 \]  \hspace{1cm} (75)

By differentiation with respect to $u$ and $v$ in any of the equations \cite{73,75}, we further obtain:

\[ \frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 y}{\partial u^2} + \frac{\partial^2 z}{\partial u^2} = 0, \quad \text{and} \quad \frac{\partial^2 x}{\partial v^2} + \frac{\partial^2 y}{\partial v^2} + \frac{\partial^2 z}{\partial v^2} = 0 \]

After adding and rearranging:

\[ \left( \frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2} \right) + \left( \frac{\partial^2 y}{\partial u^2} + \frac{\partial^2 y}{\partial v^2} \right) + \left( \frac{\partial^2 z}{\partial u^2} + \frac{\partial^2 z}{\partial v^2} \right) = 0, \]

we obtain again

\[ \nabla^2 x(u, v) + \nabla^2 y(u, v) + \nabla^2 z(u, v) = 0 \iff \Delta F(u, v) = 0. \]

Thus we find the conformal mapping be approximated by the solution $F(u, v)$ to the Laplace equation, therefore by a harmonic function relating the surface to its parametrization. The loss of generality from strictly conformal to harmonic occurs as the solution to the Laplace equation becomes more restrictive than the first order PDEs it was derived from. In what follows, the focus will be on how to approximate the Laplace condition on discrete domains, such as triangulated surfaces.
B.0.4 Discrete Harmonic Maps

To find a discrete solution to the Laplace equation describing the harmonic function we are looking for, \( F(du) = dr(u, v) \) of minimal \( I \), and its inverse, \( F^{-1} \), we establish first that we are looking at an elliptic PDE, because its characteristic polynomial exhibits a negative discriminant. Furthermore, because there are no lower order differential terms, and we have a boundary condition, we find ourselves in the case of the Dirichlet boundary problem, where

\[
\nabla^2 F = f \text{ in } \Omega \\
F = g \text{ on } \Gamma,
\]

where \( \Gamma = \partial \Omega, \Gamma \cup \Omega = \overline{\Omega} \), i.e. \( \Gamma \) is the boundary of the open planar region \( \Omega \), \( F \) is the transformation function, \( f \) is the value of \( \nabla^2 F \) inside \( \Omega \), and \( g \) is its value on the boundary \( \Gamma \). The inspirational paper by Pinkall and Polthier [1993] states the goal of the approximation sought for a discrete solution to this PDE, which is to minimize \( E_C \), the conformal energy of the map, defined as the difference:

\[
E_C(F) = E_D(F) - \text{Area}(F(\Omega)), \tag{76}
\]

where \( E_D(F) \) is the so-called Dirichlet energy associated with \( F \). We will derive the expression of, and the necessity for the minimization of this energy in what follows.

The Dirichlet energy and the meaning of its minimization

Looking at our previous results imposing constraints for the singular values of the Jacobian of the transformation in the equiareal and the conformal cases, if a minimization of both the area distortion and of the shape (angle) distortion is sought, the following conflicting conditions have to be both fulfilled as closely as possible at the same time:

\[
\frac{\sigma_1}{\sigma_2} \approx 1 \text{ and } \sigma_1 \cdot \sigma_2 \approx 1.
\]

In order to meet the goals of minimization of both area and shape distortion, the following imposes itself as the governing minimizing condition:

\[
\min_u \left( \frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1} \right). \tag{77}
\]
In [Hormann and Greiner 2000] the following derivation is given putting the Frobenius norm — defined as in 77 — in relationship with the tensor metric matrix \( \mathcal{G} \) and the Jacobian \( J \) of the transformation denoted \( u \) in our Section B.0.4:

\[
\|J\|_F \|J^{-1}\|_F = \sqrt{\text{trace}(\mathcal{G}) \cdot \text{trace}(\mathcal{G}^{-1})} = \sqrt{(\sigma_1^2 + \sigma_2^2) \frac{(\sigma_1^2 + \sigma_2^2)}{\sigma_1^2 \sigma_2^2}},
\]

which leads to the useful expression for the Frobenius norm for matrix \( J \):

\[
\kappa_F(J) = \|J\|_F \|J^{-1}\|_F = \frac{\sigma_1^2 + \sigma_2^2}{\sigma_1 \sigma_2} = \frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1},
\]

This is exactly our minimization argument from 77 or re-written in terms of \( \mathcal{G} \) and \( J \),

\[
\kappa_F(J) = \frac{\text{trace}(\mathcal{G})}{\det J} = \frac{\text{trace}(JJ^T)}{\det J} \quad (78)
\]

With these prerequisites, the Dirichlet energy, taken as just the numerator of \( \kappa_F \) and scaled by a constant factor of \( \frac{1}{2} \), will only satisfy the minimization of \( \frac{\sigma_1}{\sigma_2} \), and therefore will only ensure conformality:

\[
\frac{1}{2} \int_{t_i} \text{trace}(J) = \frac{1}{2} \int_{t_i} \left( \left( \frac{\partial \bar{x}^1}{\partial x^1} \right)^2 + \left( \frac{\partial \bar{x}^2}{\partial x^2} \right)^2 \right) = \frac{1}{2} \int_{t_i} \|\nabla F\|^2, \quad (79)
\]

where \( t_i \) are the 2D triangles whose individual mappings compose the image of the domain \( \Omega \) under the mapping \( F \).

Note: A large number of papers take the minimization of \( E_D \) for granted and omit to justify why. I came across [Cohen-Steiner and Desbrun 2002], a short erratum issued by David Cohen-Steiner and Mathieu Desbrun, authors of numerous papers on parametrization, in which they indirectly confirm that the minimization of the Dirichlet energy is perpetuated in the literature without checks. The case in point was the equivalence between the Least-Squares Conformal Maps (LSCMs), minimizing \( E_C \), and the Discrete Natural Conformal Parametrization (DNCP), minimizing \( E_D \), presented in two different papers as two different methods, but proved in the erratum to characterize the same thing, namely the conformal property of the mapping. (We arrived at the same conclusion independently, above).
In addition, as professor Dennis Zorin of NYU points out in his lecture notes \cite{Zorin2002}, due to the inequality:

$$\det J \leq ||u_1|| \cdot ||u_2|| \leq \frac{1}{2} (||u_1||^2 + ||u_2||^2),$$

the conformality condition (third quantity) appears as less restrictive than the area condition (first quantity), as it should be, and therefore would not guarantee size (area) preservation unless a combination of both criteria was used. Hormann and Greiner \cite{Hormann2000} agree that $E_D$ alone does not fulfil the “no scaling” criterion (which angle preservation alone wouldn’t necessarily achieve).

In search for a perfect criterion, other authors have suggested various minimization expressions, some of whom, as Eck et al. \cite{Eck1995}, consider only $E_D$, while others still, like Maillot et al. \cite{Maillot1993}, define their own minimization criteria, in this case the Green-Lagrange deformation tensor matrix $I - I_2$ as the object of the minimization proposed. While this corroborates our result from \textsection{6.8}, it is still only a measure of conformality. Choosing a good minimization criterion to enforce area and angle preservation is still an open problem.

The Cotangent Formula

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{cotangent_formula.png}
\caption{Minimal surfaces transformation $f$.}
\end{figure}

Pinkall and Polthier \cite{Pinkall1993} propose a discrete method to compute minimal surfaces, which are characterized by the condition that they locally minimize area. Their result has influenced numerous studies on mapping. They compute iterative surfaces, in which each iteration minimizes not the Dirichlet energy of the 3D to 2D mapping, but rather of the 3D to 3D mappings that compare each iterated surface approximation to the original shape:

$$M_{i+1} = \min_{M} \frac{1}{2} \int_{M_i} |\nabla f : M_i \rightarrow M|^2.$$

\footnote{As before, $J$ is the block vector matrix $J = (u_1|u_2)$}
If \( f : \Omega \subset \mathbb{R}^2 \rightarrow f(\Omega) \subset \mathbb{R}^3 \) is a parametrization of a surface \( f(\Omega) \) over a 2D domain \( \Omega \subset \mathbb{R}^2 \), then the area of \( f(\Omega) \) is given by

\[
\text{Area}(f) = \int_\Omega J(f)
\]

with its associated Dirichlet energy defined as derived in our Section B.0.4. \( E_D(f) = \frac{1}{2} \int_\Omega \vert \nabla f \vert^2_g \) where \( g \) is the metric tensor representing \( f(\Omega) \). Then, the energy of an entire map between discrete surfaces is the sum of the energies of all linear triangular mappings \( f_i : (t_i, g) \rightarrow (T_i, h) \), where \( t_i \) is the \( i \)th triangle of the 2D triangulation, \( T_i \) its image under the 3D triangulation, \( f_i \) is the local mapping, and \( g, h \) are the metrics associated to the two triangulations. Correspondingly, the integrated energy over all individual mappings \( f_i \) are given by

\[
E_D(f_i) = \frac{1}{2} \int_{t_i, g} \vert \nabla g f_i \vert^2_h
\]

where \( \nabla g \) is the derivative operator with respect to \( g \) and \( \vert \cdot \vert_h \) is the norm in image space with respect to \( h \).

With these preliminaries, the main contribution of [Pinkall and Polthier 1993] is in proving the following formula:

\[
E_D(f) = \frac{1}{4} \sum_{i=1}^3 \cot \alpha_i \cdot \| a_i \|^2_h,
\]

(80)

where \( \alpha_i \) are the angles of \( t_i \) and \( \| \cdot \|^2_h \) is the vector norm applied to the sides of the image triangle \( T_i \).

**Proof:** Let \( \varphi, \psi \) two functions such that:

\[
\begin{align*}
(e_1, e_2) &\xrightarrow{\varphi} (v, w) & (v, w) &\xrightarrow{f} (a, b) \\
(e_1, e_2) &\xrightarrow{\psi} (a, b) & \uparrow & \uparrow \\
(v, w) &\xrightarrow{f} (a, b), & (e_1, e_2) &\quad\quad\quad (e_1, e_2)
\end{align*}
\]

(meaning that \( f = \psi(\varphi^{-1}) = \varphi^{-1} \circ \psi \)). With this function composition set up, our goal is to express \( E_D(f) = \int \text{trace}(J^T f \cdot J f) \) as \( E_D(\psi(\varphi^{-1})) \), in other words to express \( \text{trace}(J^T f \cdot J f) \) in terms of \( J\psi \) and \( J\varphi \):

6 Sometimes the function composition is shown as \( f = \psi(\varphi^{-1}) = \psi \circ \varphi^{-1} \), although the feed line should be left to right, as in \( f = \psi(\varphi^{-1}) = \varphi^{-1} \circ \psi \), because the output of the argument function is evaluated first, and the outer function next. However, it is only a matter of convention.
\[ f = \varphi^{-1} \circ \psi = \psi(\varphi^{-1}) \]

\[ J_f = J_{\varphi^{-1} \circ \psi} = J_\psi \cdot J_{\varphi^{-1}} \]

\[ J_f^T = J_{\varphi^{-1} \circ \psi}^T = (J_\psi \cdot J_{\varphi^{-1}})^T = J_{\varphi^{-1}}^T \cdot J_\psi^T \]

\[ \implies J_f^T \cdot J_f = (J_{\varphi^{-1}}^T \cdot J_\psi^T) \cdot (J_\psi \cdot J_{\varphi^{-1}}) \]

In calculating \( \text{trace}(J_f^T \cdot J_f) \) we hope to group the Jacobian matrices of \( \psi \) and \( \varphi^{-1} \) and their respective transposes two by two, so we can calculate them according to their definitions — according to the respective changes of coordinates that they represent. The grouping is possible because the trace is a commutative operation (i.e. \( \text{trace}(AB) = \text{trace}(BA) \)):

\[ \text{trace}(J_f^T \cdot J_f) = \text{trace}[(J_{\varphi^{-1}}^T \cdot J_\psi^T) \cdot (J_\psi \cdot J_{\varphi^{-1}})] = \text{trace}(J_\psi^T \cdot J_\psi \cdot J_{\varphi^{-1}}^T \cdot J_{\varphi^{-1}}) \quad (81) \]

We therefore need to compute separately \( J_{\psi}^T \cdot J_\psi \) and \( J_{\varphi^{-1}} \cdot J_{\varphi^{-1}}^T \):

\[ J_{\psi}^T \cdot J_\psi = \left( \frac{\partial (a, b)}{\partial (e_1, e_2)} \right) \cdot \left( \frac{\partial (a, b)}{\partial (e_1, e_2)} \right) = \left( \begin{array}{c} \langle a, a \rangle \\ \langle a, b \rangle \\ \langle b, a \rangle \\ \langle b, b \rangle \end{array} \right), \quad (82) \]

For \( J_{\varphi^{-1}} \cdot J_{\varphi^{-1}}^T \), a few preliminary calculations are necessary:

\[ J_{\varphi^{-1}} \cdot J_{\varphi^{-1}}^T = J_{\varphi^{-1}}^{-1} \cdot (J_{\varphi^{-1}})^T = = J_{\varphi^{-1}}^{-1} \cdot (J_{\varphi^{-1}})^T = (J_{\varphi^{-1}} \cdot J_{\varphi})^{-1} \]

where we used the identities \((A^{-1})^T = (A^T)^{-1}\) and \(A^{-1} \cdot B^{-1} = (B \cdot A)^{-1}\).

Now, for \( J_{\varphi}^T \cdot J_\varphi \) can write an expression similar to \(82\):

\[ J_{\varphi}^T \cdot J_\varphi = \left( \frac{\partial (v, w)}{\partial (e_1, e_2)} \right) \cdot \left( \frac{\partial (v, w)}{\partial (e_1, e_2)} \right) = \left( \begin{array}{c} \langle v, v \rangle \\ \langle v, w \rangle \\ \langle w, v \rangle \\ \langle w, w \rangle \end{array} \right) \quad (83) \]

The inverse of the matrix from \(83\) is therefore:

\[ (J_{\varphi}^T \cdot J_\varphi)^{-1} = \frac{1}{\text{det}(J_{\varphi}^T \cdot J_\varphi)} \left( \begin{array}{c} \langle w, w \rangle & -\langle v, w \rangle \\ -\langle v, w \rangle & \langle v, v \rangle \end{array} \right) \quad (84) \]
Finally from equations 81, 82 and 84 we can compute:

\[
\text{trace}(J^T \cdot J) = \text{trace}\left[(J_{\varphi^{-1}}^T \cdot J_{\varphi^{-1}}^T) \cdot (J_{\varphi}^T \cdot J_{\varphi})^{-1}\right] = \frac{1}{\det J_{\varphi}^T \cdot \det J_{\varphi}} \cdot \text{trace}\left[\begin{pmatrix} a & a \\ a & b \end{pmatrix} \cdot \begin{pmatrix} \langle w, w \rangle & -\langle v, w \rangle \\ -\langle v, w \rangle & \langle v, v \rangle \end{pmatrix}\right] = \frac{1}{\det J_{\varphi}^2} \cdot \left(\langle a, a \rangle \langle w, w \rangle - 2\langle a, b \rangle \langle v, w \rangle + \langle b, b \rangle \langle v, v \rangle\right) = \ldots
\]

Now we introduce the substitution \( c \leftarrow b - a \), where \( ||c||^2 = (||a|| + ||b||)^2 \), and \( ||c||^2 - ||a||^2 - ||b||^2 = -\langle a, b \rangle \) will replace \(-\langle a, b \rangle\) in the second term in the parentheses above:

\[
\ldots = \frac{1}{\det J_{\varphi}^2} \cdot \left(\langle a, a \rangle \langle w, w \rangle + \langle (c, c) - (a, a) - (b, b) \rangle \langle v, w \rangle + \langle v, v \rangle \langle v, v \rangle\right) = \frac{1}{\det J_{\varphi}^2} \cdot \left[\langle a, a \rangle \left(\langle w, w \rangle - \langle v, w \rangle\right) + \langle b, b \rangle \left(\langle v, v \rangle - \langle v, v \rangle\right) - \langle c, c \rangle \langle v, w \rangle\right] = \ldots
\]

Expanding the dot products and denoting \( \gamma = \angle(v, w) \) we get:

\[
\ldots = \frac{1}{\det J_{\varphi}} \left[ ||a||^2 \left(||w||^2 - ||v|| ||w|| \cos \gamma\right) + ||b||^2 \left(||v||^2 - ||v|| ||w|| \cos \gamma\right) + ||c||^2 ||v|| ||w|| \cos \gamma\right] = \ldots
\]

We now develop the determinant of the Jacobian \( J_{\varphi} \):

\[
J_{\varphi} = \begin{vmatrix} \frac{\partial v}{\partial e_1} & \frac{\partial v}{\partial e_2} \\ \frac{\partial w}{\partial e_1} & \frac{\partial w}{\partial e_2} \end{vmatrix} = ||v \times w|| = ||v|| ||w|| \sin \gamma
\]

and insert it into the (now ongoing) equation:

\[
\ldots = \frac{1}{\det J_{\varphi}} \left[ a^2 \frac{w(w - v \cos \gamma)}{vw \sin \gamma} + b^2 \frac{v(v - w \cos \gamma)}{vw \sin \gamma} + c^2 \frac{vw \cos \gamma}{vw \sin \gamma}\right] = \ldots = \frac{1}{\det J_{\varphi}} \left[ a^2 \frac{w(1 - v \cos \gamma)}{v \sin \gamma} + c^2 \frac{\cos \gamma}{\sin \gamma}\right] = \frac{1}{\det J_{\varphi}} \left[ a^2 \frac{d_w}{h_w} + b^2 \frac{d_v}{h_v} + c^2 \cot \gamma\right],
\]

where

\( d_w = w - v \cos \gamma \) is the difference between the length of \( w \) and \( v \)'s projection onto \( w \)

\( h_w = v \sin \gamma \) is the length of the perpendicular from \( v \)'s tip onto \( w \)

\( d_v = v - w \cos \gamma \), same as above, with the roles of \( v \) and \( w \) reversed

\( h_v = w \sin \gamma \), idem.

161
We now introduce the other two angles in the triangle formed by \(v\) and \(w\): let \(\alpha \leftarrow \angle(-w, v - w)\) and \(\beta \leftarrow \angle(-v, w - v)\). With these notations, we have \(\frac{d_w}{h_w} = \cot \alpha\) and \(\frac{d_v}{h_v} = \cot \beta\), and replacing these quantities in our equation we get our cotangent formula:

\[
\text{trace}(J_T f \cdot J f) = \frac{1}{|J \phi|} [a^2 \cot \alpha + b^2 \cot \beta + c^2 \cot \gamma] \tag{85}
\]

After insertion into the expression for the Dirichlet energy, this becomes:

\[
E_D = \sum_{t_i} E_D(f_i) = \frac{1}{2} \sum_{\Delta_i} \left(\text{trace}(J_{t_i}^T \cdot J_{f_i})\right) = \frac{1}{2} \sum_{t_i} \left[ a_i^2 \cot \alpha_i + b_i^2 \cot \beta_i + c_i^2 \cot \gamma_i \right] |J_{\phi_i}^{-1}| = \frac{1}{4} \sum_{t_i} \left[ a_i^2 \cot \alpha_i + b_i^2 \cot \beta_i + c_i^2 \cot \gamma_i \right],
\]

where triangles \(t_i\) as well as angles \(\alpha_i, \beta_i,\) and \(\gamma_i\) are in the source domain \(\Omega \subset \mathbb{R}^2\), while the segment lengths \(a_i, b_i,\) and \(c_i\) are the lengths of the transformed triangles in the image \(f(\Omega) \subset \mathbb{R}^3\). The area of triangle \(t_e = \Delta(e_1, e_2) = \det J_{\phi_i}^{-1} = \frac{1}{2}\) is asserted knowing that \((e_1, e_2)\) is an orthonormal unit vector system.

With the additional observation that each shared edge will appear twice over a domain, once for each adjacent triangle it belongs to, the same formula can be converted to minimize a Dirichlet energy over the edges, instead of triangles:

\[
E_D(f) = \frac{1}{4} \sum_{\text{edges } e_i} \left(\cot \theta_i + \cot \phi_i\right) \|e\|^2,
\]

where angles \(\theta_i, \phi_i\) are angles opposed to \(e\) in the two adjacent triangles sharing \(e\). In this representation, the non-existing triangle in case of a boundary edge will annul one of the two terms in the parentheses. The reduction to two terms reflects the edge-based as opposed to triangle-based formula, while the constant coefficient \(\frac{1}{4}\) remains unchanged.

For a point-based minimization formula, again observing that each edge will appear twice in the characterization of each point in a domain, the energy can be translated once more into:

\[
E_D(f) = \frac{1}{8} \sum_{v_i \in \Omega} \sum_{v_j \in N_i} \left(\cot \theta_i + \cot \phi_i\right) \|v_i - v_j\|^2 \tag{86}
\]
Here, the number of occurrences of each vertex were diminished exactly by one half, hence the coefficient was changed to $\frac{1}{8}$.

Under a discrete conformal parametrization, each vertex will suffer a transformation of the following type:

$$v'_i = \begin{cases} \arg \min_{v_j \in N_i} \left\{ \frac{1}{8} \sum_{v_j \in N_i} (\cot \theta_i + \cot \phi_i) \|v_i - v_j\|^2 \right\}, & \text{if } v_j \in N_i, \\ 0, & \text{if } v_j / \in N_i \end{cases}$$

The Mean Value Coordinates

![Figure 61: Mean value coordinates: the neighborhood around vertex $v_i$.](image)

In Floater [2003], the weights from Equation 92 are derived departing from the mean value property of harmonic functions. As a preliminary result, the author introduces the following

**Lemma B.0.1** If $f : T_i \rightarrow \mathbb{R}$ is a linear function and $v_j \in N_i$, then

$$\int_{\Gamma_j, v_j \in N_i} f(v) \, ds = r \alpha_i f(v_i) + r^2 \tan \frac{\alpha_i}{2} \left( \frac{f(v_j) - f(v_i)}{\|v_j - v_i\|} + \frac{f(v_{j+1}) - f(v_i)}{\|v_{j+1} - v_i\|} \right) \tag{87}$$

**Proof:** We represent $v \in \Gamma_j$ in polar coordinates w.r.t. $v_i$:

$$v = v_i + r (\cos \theta, \sin \theta), \quad r = \|v - v_i\|.$$  

In particular,

$$v_j = v_i + r_j (\cos \theta_j, \sin \theta_j),$$

$$v_{j+1} = v_i + r_{j+1} (\cos \theta_{j+1}, \sin \theta_{j+1})$$
are the points \( v_j, v_{j+1} \in N_i \), at distances from \( v_i \):  
\[ r_j = \| v_j - v_i \|, \quad r_{j+1} = \| v_{j+1} - v_i \| \]

Due to \( f \)'s linearity, we have \( f(v) \) as a linear combination of the surrounding points:
\[
f(v) = f(v_i) + \lambda_1 [f(v_j) - f(v_i)] + \lambda_2 [f(v_{j+1}) - f(v_i)], \tag{88}
\]

where
\[
\lambda_1 = \frac{A_1}{A} = \frac{\text{Area}(\triangle v_iv_{j+1})}{\text{Area}(\triangle v_iv_{j+1})} \quad \text{and} \quad \lambda_2 = \frac{A_2}{A} = \frac{\text{Area}(\triangle v_iv_j)}{\text{Area}(\triangle v_iv_j)}.\]

On the other hand, the value of \( f(v) \) on the arc \( \Gamma_j \) from \( \theta_j \) to \( \theta_{j+1} \) is expressible as
\[
\int_{\Gamma_j} f(v) \, ds = r \int_{\theta_j}^{\theta_{j+1}} f(v) \, d\theta. \tag{89}
\]

We can now expand the linear expression of \( f(v) \) from (88) into (89) including the area substitutions:
\[
\int_{\Gamma_j} f(v) \, ds = r \int_{\theta_j}^{\theta_{j+1}} \left\{ f(v_i) + \frac{A_1}{A} [f(v_j) - f(v_i)] + \frac{A_2}{A} [f(v_{j+1}) - f(v_i)] \right\} d\theta.
\]

We will now express the triangle areas using the Theorem of the Sines. The angles are taken relative to a reference line relative to which both \( |v_iv_j| \) and \( |v_i v_{j+1}| \) are on the same side. We denote \( \alpha_i = \theta_{j+1} - \theta_j \).

\[
\frac{A_1}{A} = \frac{rr_{j+1} \sin(\theta_{j+1} - \theta)}{2 r_j r_{j+1} \sin(\theta_{j+1} - \theta)} = \frac{r \sin(\theta_{j+1} - \theta)}{r_j \sin \alpha_i},
\]
\[
\frac{A_2}{A} = \frac{rr_j \sin(\theta - \theta_j)}{2 r_j r_{j+1} \sin(\theta - \theta_j)} = \frac{r \sin(\theta - \theta_j)}{r_{j+1} \sin \alpha_i}.
\]

As the only factors dependent on \( \theta \), we can calculate ahead of time the integrals:
\[
\int_{\theta_j}^{\theta_{j+1}} \sin(\theta_{j+1} - \theta) \, d\theta = \left[ \cos(\theta_{j+1} - \theta) + C \right]_{\theta_j}^{\theta_{j+1}} = 1 - \cos \alpha_i,
\]
\[
\int_{\theta_j}^{\theta_{j+1}} \sin(\theta - \theta_j) \, d\theta = \left[ \cos(\theta - \theta_j) + C \right]_{\theta_j}^{\theta_{j+1}} = 1 - \cos \alpha_i.
\]

164
We substitute the area ratios in the integral of $f(v)$ over $\Gamma_i$:

$$\int_{\Gamma_i} f(v) \, ds = \int_{\theta_i}^{\theta_{i+1}} f(v) \, d\theta = \sum_{j} \left\{ \int_{\Gamma_i} f(v) \, d\theta + \frac{(1 - \cos \alpha_i)r}{r_j \sin \alpha_i} [f(v_j) - f(v_i)] + \frac{(1 - \cos \alpha_i)r}{r_{j+1} \sin \alpha_i} [f(v_{j+1}) - f(v_i)] \right\} = \int_{\theta_i}^{\theta_{i+1}} f(v) \, d\theta = \sum_{j} \left\{ \int_{\Gamma_i} f(v) \, d\theta + \frac{(1 - \cos \alpha_i)r^2}{r_j \sin \alpha_i} [f(v_j) - f(v_i)] + \frac{(1 - \cos \alpha_i)r^2}{r_{j+1} \sin \alpha_i} [f(v_{j+1}) - f(v_i)] \right\}
$$

Using the trigonometric relation $\tan \frac{\alpha_i}{2} = \frac{1 - \cos \alpha_i}{\sin \alpha_i}$, we get:

$$\int_{\Gamma_i} f(v) \, ds = r \alpha_i f(v_i) + \frac{r^2}{r_j} \tan \frac{\alpha_i}{2} [f(v_j) - f(v_i)] + \frac{r^2}{r_{j+1}} \tan \frac{\alpha_i}{2} [f(v_{j+1}) - f(v_i)],$$

which is exactly the expression sought by Lemma B.0.1.

Now, according to the mean value theorem, we can calculate the weights $\lambda_{ij}$ to compute $f(v_j)$ as a linear combination of its neighbors $v_j \in N_i$:

$$f(v_i) = \frac{1}{2\pi r} \int_{\Gamma_i} f(v) \, ds = \sum_j \left[ \frac{1}{2\pi r} \int_{\Gamma_i} f(v) \, ds \right]$$

The summation takes place over $j$ arcs, throughout the neighborhood $N_i$ of $v_i$. With the appropriate substitutions in this latest form, we get:

$$f(v_i) = \sum_j \frac{1}{2\pi r} r \alpha_i f(v_i) + \sum_j \frac{1}{2\pi r} \frac{r^2}{r_j} \tan \frac{\alpha_i}{2} [f(v_j) - f(v_i)] + \sum_j \frac{1}{2\pi r} \frac{r^2}{r_{j+1}} \tan \frac{\alpha_i}{2} [f(v_{j+1}) - f(v_i)] = f(v_i) + \frac{r}{2\pi} \sum_j \tan \frac{\alpha_i}{2} \frac{f(v_j) - f(v_i)}{r_j} + \frac{r}{2\pi} \sum_j \tan \frac{\alpha_i}{2} \frac{f(v_{j+1}) - f(v_i)}{r_{j+1}}$$
We now isolate \( f(v_i) \), after reducing, in the remaining instances from the neighbor formulas:

\[
\sum_j \tan \frac{\alpha_i}{2} \frac{f(v_j) - f(v_i)}{r_j} + \sum_j \tan \frac{\alpha_i}{2} \frac{f(v_{j+1}) - f(v_i)}{r_{j+1}} = 0
\]

\[
\sum_j \tan \frac{\alpha_i}{2} \left( \frac{f(v_i)}{r_j} + \frac{f(v_i)}{r_{j+1}} \right) = \sum_j \tan \frac{\alpha_i}{2} \left( \frac{f(v_j)}{r_j} + \frac{f(v_{j+1})}{r_{j+1}} \right)
\]

Finally, \( f(v_i) \), which does not depend on \( j \), can be expressed as:

\[
f(v_i) = \frac{\sum_j \tan \frac{\alpha_i}{2} \left[ \frac{f(v_j)}{r_j} + \frac{f(v_{j+1})}{r_{j+1}} \right]}{\sum_j \tan \frac{\alpha_i}{2} \left[ \frac{1}{r_j} + \frac{1}{r_{j+1}} \right]}
\]  

At the numerator, we subtract and add, in the sum, the amount \( \tan \frac{\alpha_i}{2} \frac{f(v_{j+1})}{r_{j+1}} \):

\[
\sum_j \left[ \tan \frac{\alpha_i}{2} \frac{f(v_j)}{r_j} - \tan \frac{\alpha_i}{2} \frac{f(v_{j+1})}{r_{j+1}} \right] + \sum_j \left[ \tan \frac{\alpha_i}{2} + \tan \frac{\alpha_{i+1}}{2} \right] \frac{f(v_{j+1})}{r_{j+1}}
\]

At the denominator, we repeat the artifice with the amount \( \tan \frac{\alpha_i}{2} \frac{1}{r_{j+1}} \):

\[
\sum_j \left[ \tan \frac{\alpha_i}{2} \frac{1}{r_j} - \tan \frac{\alpha_{i+1}}{2} \frac{1}{r_{j+1}} \right] + \sum_j \left[ \tan \frac{\alpha_i}{2} + \tan \frac{\alpha_{i+1}}{2} \right] \frac{1}{r_{j+1}}
\]

Noticing that the first sums vanish in each case, we reassemble the fraction into the final Mean Value Coordinates formula:

\[
f(v_i) = \frac{\sum_j \left[ \tan \frac{\alpha_i}{2} + \tan \frac{\alpha_{i+1}}{2} \right] \frac{f(v_j)}{r_j}}{\sum_j \left[ \tan \frac{\alpha_i}{2} + \tan \frac{\alpha_{i+1}}{2} \right] \frac{1}{r_j}}
\]
Let $f_1(x) = \sin^2 x$ and $f_2(x) = \sin^4 x$, where $x \leftarrow \frac{\alpha - \beta}{2}$. We calculate the first and second derivatives of these functions. In the case of $E_{Cohen-Steiner}$:

$$f'_1(x) = (\sin^2 x)' = 2 \sin x \cos x = \sin 2x = \sin(\alpha - \beta)$$

$$f''_1(x) = (\sin 2x)' = 2 \cos 2x = 2 \cos(\alpha - \beta)$$

In the case of $E_{Wise}$:

$$f'_2(x) = (\sin^4 x)' = 4 \sin^3 x \cos x = (2 \sin x \cos x) \cdot 2 \sin^2 x =$$

$$= 2(\sin 2x) \sin^2 x = 2(\sin 2x) \frac{1 - \cos 2x}{2} = \sin 2x(1 - \cos 2x) =$$

$$= \sin(\alpha - \beta)[1 - \cos(\alpha - \beta)]$$

$$f''_2(x) = (\sin 2x)'(1 - \cos 2x) + \sin 2x(1 - \cos 2x)' =$$

$$= 2 \cos 2x(1 - \cos 2x) + \sin 2x(2 \sin 2x) = 2(\cos 2x - \cos^2 2x + \sin^2 2x) =$$

$$= 2(\cos 2x - \cos 4x) = 2[\cos(\alpha - \beta) - \cos 2(\alpha - \beta)]$$

To find $f_1$'s inflexion point, we set $f''_1 = 0$:

$$2 \cos(\alpha - \beta) = 0 \Rightarrow \alpha - \beta = \frac{\pi}{2} + k\pi, \ k \in \mathbb{Z} \quad (93)$$

Proceeding in the same fashion for $f_2$, and keeping the expression for $f''_2$ only in terms of $\cos 2x$, we obtain:

$$f''_2(x) = -2(2 \cos^2 2x - \cos 2x - 1) = 0$$

By making the change of variables $u \leftarrow \cos 2x$, we get the second degree equation:

$$2u^2 - u - 1 = 0$$

with solutions:

$$u_{1,2} = \frac{1 \pm \sqrt{(-1)^2 - 4 \cdot 2 \cdot (-1)}}{2 \cdot 2} = \frac{1 \pm 3}{4}$$

$$u_1 = -\frac{1}{2}, \ u_2 = 1$$
We get the cases:

\[(i) \cos(\alpha - \beta) = -\frac{1}{2} \Rightarrow \alpha - \beta = \pm \frac{2\pi}{3} + 2k\pi, \ k \in \mathbb{Z} \]

\[(ii) \cos(\alpha - \beta) = 1 \Rightarrow \alpha - \beta = 2k\pi, \ k \in \mathbb{Z} \]  

After setting \(k = 0\) for convenience, we find from Equations 93 and 94 that the slope of \(f_1\) grows up to \(x = \frac{\pi}{4}\) (or \(\alpha - \beta = \frac{\pi}{2}\)), while the slope of \(f_2\) keeps growing up to \(x = \frac{\pi}{2}\) (or \(\alpha - \beta = \frac{2\pi}{3}\)). Since on \(x \in [0; \frac{\pi}{2}]\) \(f_1\) is consistently greater than \(f_2\), it follows that \(f_2\)'s slope stays slower longer than \(f_1\)'s (from \(f_2' \leq f_1'\) we get \(\alpha - \beta = 2x \in [-\frac{\pi}{2} + 2k\pi; \frac{\pi}{2} + 2k\pi]\)), therefore \(f_2\) grows more slowly for smaller angle differences, which gives our error its smoother segmentation characteristic we found useful in curvy and noisy models.


David Cohen-Steiner and Mathieu Desbrun. **Hindsight: LSCM and DNCP are one and the same**, 2002.


Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. **Multiresolution Analysis of Arbitrary Meshes.** In *SIGGRAPH ’95:...*
Bibliography


Bibliography


Bibliography


Bibliography


