Abstraction of Man-Made Shapes

by

Qingnan Zhou

B.CS., The University of Waterloo, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

August 2009

Abstract

Man-made objects are ubiquitous in the real world and in virtual environments. While such objects can be very detailed, capturing every small feature, they are often identified and characterized by a small set of defining curves. Compact, abstracted shape descriptions based on such curves are often visually more appealing than the original models, which can appear to be visually cluttered. We introduce a novel algorithm for abstracting three-dimensional geometric models using characteristic curves or contours as building blocks for the abstraction. Our method robustly handles models with poor connectivity, including the extreme cases of polygon soups, common in models of man-made objects taken from online repositories. In our algorithm, we use a two-step procedure that first approximates the input model using a manifold, closed *envelope* surface and then extracts from it a hierarchical abstraction curve network along with suitable normal information. The constructed curve networks form a compact, yet powerful, representation for the input shapes, retaining their key shape characteristics while discarding minor details and irregularities.

Table of Contents

Abstract ii

Table of Contents iii

List of Tables v

List of Figures vi

Acknowledgments x

Dedication xi

Statement of Co-Authorship xii

1 Introduction 1
- 1.1 Man-made models 1
- 1.2 Abstraction of man-made models 2

2 Related Work 5
- 2.1 Artistic and human perception of shape 5
- 2.2 Non-realistic rendering and modeling 6
- 2.3 Shape analysis and reverse engineering 8

3 Algorithm Overview 11

4 Envelope 13
- 4.1 Initialization 13
- 4.2 Iterative fitting 18
  - 4.2.1 Matching 18
  - 4.2.2 Deformation 21
  - 4.2.3 Mesh regularization 22

5 Segmentation 31

5.1 Extracting network topology 31
5.2 Extracting network geometry 33
5.3 Network regularization and simplification 37

6 Reconstruction 40

7 Results 42

8 Conclusion 46

Bibliography 47

List of Tables

7.1 Abstraction statistics. 43

List of Figures

1.1 Commonly available man-made models often have multiple connected components, self-intersections, and triangles with bad aspect ratios. 1

1.2 (Left to right) Detailed 3D model, an artist's drawing, a crystal souvenir, and our abstraction of the Eiffel Tower. 2

1.3 (Left to right) Tourist map (courtesy of Arnaud Siminski) depicting landmark buildings, Picasso's famous abstraction of a bull (courtesy of the Museum of Modern Art), and a wooden toy (courtesy of NOVA68.com)designed by Calder all make use of distinctive attributes to generate minimalist, yet powerful, abstractions. 3

1.4 (Left to right) Input model, model simplified to 200 triangles, simplified envelope surface with 200 triangles. While the simplified envelope surface fares better, it does not preserve the core features of the Arc de Triomphe. 4

2.1 Eiffel tower rendered using crease lines (left), suggestive contours (middle), and crease lines on our 3D abstraction (right). 6

2.2 Comparison of common NPR line representations. Generated using the executable provided in [14]. For suggestive contour, ridges and valleys, and apparent ridges, the outer contours are overlaid on top as black lines. 7

2.3 Man-made objects may often contain large interior features, such as the interior wall of the famous Utah teapot. Left: the original Utah teapot. Right: the same teapot cut in the middle showing its interior wall. 10

3.1 (Left to right) Input model with 353 components, envelope surface, vector representation, and reconstructed model. . . . 12

4.1 Iterative envelope generation process: (Top left to bottom right) original mesh, initial envelope generated (grid size $9 \times 23 \times 9$ and subdivided twice), iteration 1 to 9, and iteration 11. 14
4.2 There might be multiple surfaces that separates the voxel hull with the non-intersecting voxels. (a): anatomic illustration of the Arc de Triomphe model which produces two separating surfaces. The exterior separating surface (yellow) encloses the input model (red), and the interior separating surface (green) is enclosed by the input model. (b): (left to right) exterior separating surface, input model, interior separating surface. . 15
4.3 Cross section of the voxel hull (red). Notice that the input model lies between the exterior separating surface and the interior separating surface. 15
4.4 A synthetic model with hidden surfaces that produces multiple interior separating surfaces. 16
4.5 Initial envelope generation. Given an input mesh (a), a grid of voxels is created to enclose it. (b) The intersecting voxels (gold) and the outer layer (blue) is computed. (c) The exterior region propagates inwards until it reaches the voxel hull. (d) Lastly, the initial envelope is extracted as the surface separating the exterior voxels from the voxel hull. 17
4.6 The effect of different grid sizes on the voxel hulls and the final envelopes. 19
4.7 Concavity not captured in the initial envelope (i.e. w is narrower than the grid resolution) might be captured during the matching step. Envelope is in red, and original mesh is in black. Left: the concave feature is captured because $d \leq 0.5w$, and $\mathbf{v}$ is mapped to the bottom of the concavity. Right: if $d > 0.5w$, $\mathbf{v}$ will map to one of the sides, thus the concavity is removed. 21
4.8 The deformation step, during envelope fitting, smooths out narrow concavities and suppresses details of the input model. 22
4.9 Effect of mesh regularization. 23
4.10 Local mesh smoothing. 24
4.11 Left: in locally flat regions, all vertex normals of a triangle are approximately equal. Right: in non-flat regions such as near a crease, the vertex normals of a triangle could be quite different. 25

4.12 Cross section view of mesh triangles (solid black lines) and the smooth surface (dashed black curves) they approximate. Before smoothing (left), the face normal $\mathbf{n}_f$ is close to the vertex normals $\mathbf{n}_0$ and $\mathbf{n}_1$. After smoothing (right), the new $\mathbf{n}_f$ might be very different from the original $\mathbf{n}_0$ and $\mathbf{n}_1$. . . . 25
4.13 Collapsing any of the 3-loop edges (red) will cause the mesh to be non-manifold. 26
4.14 The red edge is flipped if $\theta_1 + \theta_2 > 180°$. 26
4.15 Flipping an edge (red) that fails condition 4.3 may break the crease (center). Thus, we split such edge into two halves (right). 27
4.16 Left: degenerate triangle $(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2)$ was not fixed by mesh smoothing and edge flips because condition 4.3 failed. Also, since none of its edges are excessively long, edge splits did not fix it either. Center: during degenerate triangle removal step, edge $(\mathbf{v}_1, \mathbf{v}_2)$ is first split. Right: edge $(\mathbf{v}_0, \mathbf{v}_3)$ is then collapsed. 27
4.17 Eiffel Tower (left). Low resolution envelope (middle) is initialized with grid size $9 \times 23 \times 9$. The construction converged in 11 iterations. High resolution envelope (right) is initialized with grid size $27 \times 65 \times 27$. The construction converged in 7 iterations. 29
4.18 Arc de Triomphe (left). Low resolution envelope (middle) is initialized with grid size $21 \times 35 \times 33$. The construction converged in 12 iterations. High resolution envelope (right) is initialized with grid size $35 \times 61 \times 57$. The construction converged in 8 iterations. 29
4.19 Empire State Building (left). Low resolution envelope (middle) is initialized with grid size $7 \times 9 \times 7$. The construction converged in 6 iterations. High resolution envelope (right) is initialized with grid size $41 \times 141 \times 29$. The construction converged in 5 iterations. 30
4.20 Dome of the Rock (left). Low resolution envelope (middle) is initialized with grid size $37 \times 23 \times 37$. The construction converged in 10 iterations. High resolution envelope (right) is initialized with grid size $67 \times 43 \times 67$. The construction converged in 8 iterations. 30

5.1 Vectorization stages: (Left to right) VSA segmentation, segmentation after boundary improvement, smooth approximation geometry, extracted regularized curve network, surface after hierarchical simplification, regularized simplified curve network. 32
5.2 Left: unsmoothed model. Right: the same model rendered using smoothed normals. 34
5.3 Left: original partition. Middle: after one iteration of chart smoothing without boundary curve smoothing. Right: after one iteration of chart smoothing with boundary curve smoothing. Chart boundaries are indicated using yellow lines. 35
5.4 Each chart is indicated by a unique color. Top: the tower model before chart smoothing (left), after per-triangle solve (center), and after global assembly (right). Bottom: Zoomed view of the top of the tower before chart smoothing (left), after per-triangle solve (center), and after global assembly (right). 36
5.5 Input model, processed segments, vectorized curve network, and reconstructed abstraction. Zoom panels show section of curve network, and normals along the curves. Connectivity-only or virtual edges are marked in brown. For ease of visualization, normals from same curve loops are marked in identical colors. 39

7.1 Even for noisy input models, the regularization step allows creation of quality abstractions. We present abstractions with 2% and 5% (of bounding box diagonal length) noise added to the dome model. 42
7.2 (Left) Fine topological features are easily combined by our envelope construction stage. However, once such features are extracted by a finer grid resolution, we have no easy method to remove them, independent of their size. (Right) Some objects, perhaps those less familiar to us, have no obvious natural abstraction. 44
7.3 Result gallery showing various input models, extracted curve networks (with normals), and reconstructed abstractions. The high-resolution abstractions are rendered in yellow, while the low-resolution ones are in blue. 45

Acknowledgments

First of all, I would like to thank my supervisor Alla Sheffer, without whom this work would not have been possible. I would also like to thank my collaborators: Ravish Mehra, Jeremy Long, Niloy J. Mitra, and Amy Gooch. It was fun and rewarding to work with you guys. In addition, I want to thank my second reader, Robert Bridson, for his valuable feedback.

I would like to give special thanks to Tiberiu Popa, Vladislav Kraevoy, Xi Chen, Benjamin T. Cecchetto, Derek Bradley, Ian South-Dickinson and Cody Robson for their help and support. I also want to thanks all my friends for their companionship, support and encouragement. Thank you Jingwen, Xi, Xiaofei, Kaida, Yilan, Wei, Mianwei, Hongbo, Lan, Bo, Kwun Kit and anyone else I have forgotten.

Thank you to my father and mother for all their love, patience, understanding and support. Thank you to my cousin for her sharing and guidance.

Lastly, I would like to thank NOVA68.com, Arnaud Siminski, and Museum of Modern Arts for granting us to use their images in our figures.

Qingnan Zhou
August 2009

Dedication

I would like to dedicate this work to my parents: Jian Zhou and Hua Lu; to my cousin: Jiajia Li; and to my girlfriend.

Statement of Co-Authorship

The algorithm described in this paper was developed together with Prof. Alla Sheffer, Prof. Niloy J. Mitra, Prof. Amy Gooch, Ravish Mehra, and Jeremy Long. Prof. Sheffer and Prof. Mitra supervised the project, and Ravish and I performed research and implementation. Prof. Gooch and Jeremy helped us trying out various image-based approaches in the project. The implementation is mainly built on the CML code base and the Graphite code base.

Chapter 1

Introduction

1.1 Man-made models

Engineered objects constitute a large fraction of the models populating virtual environments such as games, movies, or simulations. In recent years, easy access to 3D modeling tools and the rapid growth of online modeling communities have resulted in large collections of such models. Most models of man-made objects present in such collections do not satisfy the notion of "good" geometry processing models [27]. They frequently consist of numerous disconnected components, have self-intersections, and lack accurate information about part junctions and interconnections. *From a processing point of view, such models can be considered as polygon soups (Figure 1.1).*

Models of man-made objects can be very detailed, capturing every hole and protrusion. However, such shapes are often characterized and identified by just a few defining features (Figure 1.2). Shape descriptions based on those features, commonly involving a handful of characteristic curves, potentially mimic the minimalist representations that we, as humans, possibly store and

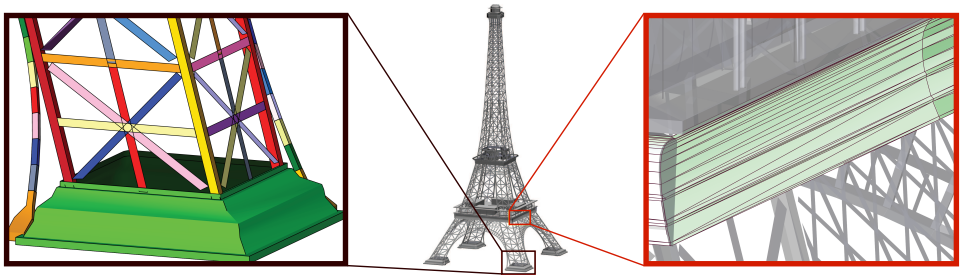


Figure 1.1: *Commonly available man-made models often have multiple connected components, self-intersections, and triangles with bad aspect ratios.*

use for our inference needs [38]. These compact, abstracted descriptions are visually more appealing than the detailed original ones, which may appear visually cluttered (Figure 1.2-left). Using this observation, artists frequently create recognizable images or icons of known objects by employing only a few brush strokes (see drawing in Figure 1.2). Such stylization is also common in tourist maps (Figure 1.3), where landmark buildings are depicted by just a few strokes that highlight their main features (see also [21]).

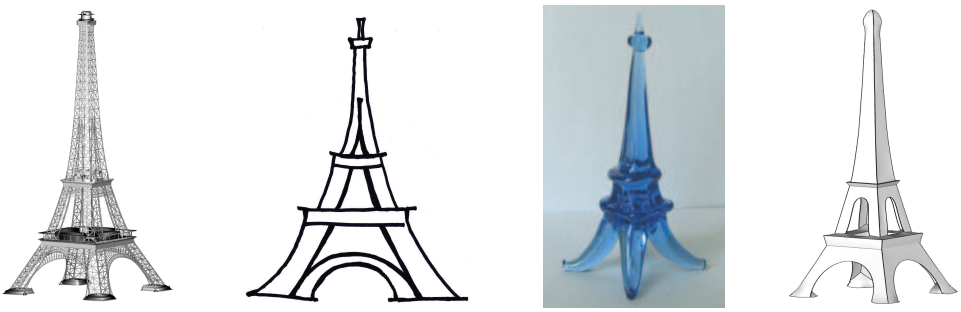


Figure 1.2: *(Left to right) Detailed 3D model, an artist's drawing, a crystal souvenir, and our abstraction of the Eiffel Tower.*

1.2 Abstraction of man-made models

In this thesis, we introduce a novel algorithm for abstracting three-dimensional shapes. Inspired by human shape perception literature [4, 13] and artistic techniques, we use characteristic curves or contours as building blocks for the abstraction. The choice is motivated by the observation that the shape of many man-made objects is clearly delineated by contour lines and can be faithfully modeled as a union of smooth patches welded together along these junction curves. Specifically, our method extracts a sparse network of space curves and associated normals as an abstraction of input models. This compact representation makes explicit the main features of the shape, which are challenging to identify from a polygon mesh or other low-level representations.

While shape abstraction can be attempted at the rendering level by developing suitable NPR tools, applying it directly to the models has a number of advantages. Model-level abstraction allows consistent rendering of a shape from a variety of views and is independent of the rendering resolution or

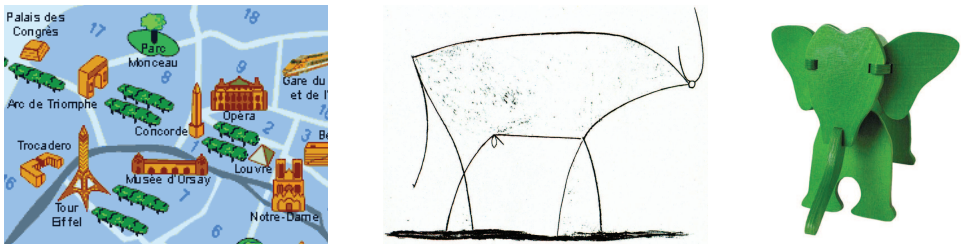


Figure 1.3: *(Left to right) Tourist map (courtesy of Arnaud Siminski) depicting landmark buildings, Picasso's famous abstraction of a bull (courtesy of the Museum of Modern Art), and a wooden toy (courtesy of NOVA68.com)designed by Calder all make use of distinctive attributes to generate minimalist, yet powerful, abstractions.*

zoom level. The set of extracted curves forms a minimalist representation, or an icon, of the modeled object. By maintaining the extracted curve network properties, subsequent processing can generate new models that automatically retain the defining characteristics of the original one [17]. Analogous to the diffusion curves for images [36], our curve network, along with suitable normal information, presents a *vectorized* representation of the input models.

Our abstraction method operates in two stages. First, it maps the given geometry to a voxel grid of suitable resolution, and extract a corresponding closed, manifold *envelope surface*, that wraps around the input model while smoothing out minor details (Figure 3.1 middle). This allows us to robustly handle non-manifold meshes and multiple component meshes, including the extreme case of polygon soups. In the second stage, the method extracts a network of curves or vectors from the envelope. After extracting the network connectivity using a mesh segmentation approach, it establishes the network geometry using a combination of regularization and approximation criteria (Figure 3.1 right).

An alternative approach for filtering out insignificant shape details is simplification, which, operating at triangle-vertex level, aims to reduce polygon count while controlling the deviation of the simplified object from the original one. Unfortunately, in the process, characteristic shape curves are likely to be disturbed, especially under extreme simplification [19], see Figure 1.4. Additional artifacts arise when the input is a polygon soup instead of a well-formed manifold mesh. In contrast, *abstraction* attempts to directly extract

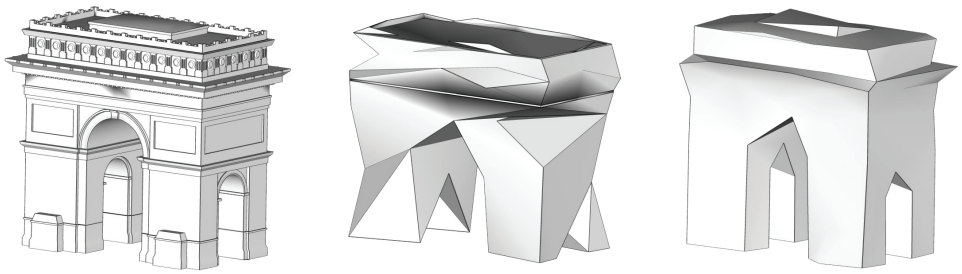


Figure 1.4: *(Left to right) Input model, model simplified to* 200 *triangles, simplified envelope surface with* 200 *triangles. While the simplified envelope surface fares better, it does not preserve the core features of the Arc de Triomphe.*

the high-level structure of objects, intentionally removing insignificant visual details and potentially allowing significant topological changes.

In addition to the main contribution of providing a method to perform abstraction of 3D geometric shapes, our two secondary contributions are: a novel vector-based representation of 3D geometry, which can be used for a variety of mesh editing tasks; and a simple yet robust mechanism for approximating polygon soup models by a manifold surface envelope.

Chapter 2

Related Work

Abstraction, the process of identifying characteristic properties and extracting their mutual relationships and topology [16], has been studied in many fields and disciplines including art, non-realistic rendering and modeling, and human perception, for purposes such as shape analysis, generation of compact descriptors, and recognition.

2.1 Artistic and human perception of shape

In the twentieth century, artists like Kandinsky, Mondrian, and Picasso pushed the boundaries of geometric abstraction in 2D representations of the surrounding world to the extremes. Alexander Calder ingeniously used wires in addition to sheet metal, wood, and bronze to create abstract 3D sculptures. While automatically generating abstraction levels like those created by Kandinsky and Mondrian is unrealistic, we draw motivation from the curve-based abstraction portrayed by Calder in his 3D wire sculptures.

Koenderink and Doorn [26] hypothesized that humans internally represent shapes as functions that measure the visual complexity of solid shapes. Later, Nackman and Pizer [33] differentiated between representation and description of an object where an object representation contains enough information to enable an approximate reconstruction, while an object description needs only to contain enough information to identify an object as a member of some object class, which is exactly what abstraction aims to do.

2.2 Non-realistic rendering and modeling

A major goal of non-photorealistic rendering (NPR) is to highlight or amplify defining object characteristics. Since low-level geometry does not provide a natural prioritization of the shape features (Figure 2.1), NPR techniques strive to identify view-specific important features, which should be rendered or exaggerated to convey form [12]. Significant research has been devoted to identifying candidate feature lines including contours, ridge or valley lines [32], and suggestive contours [14].

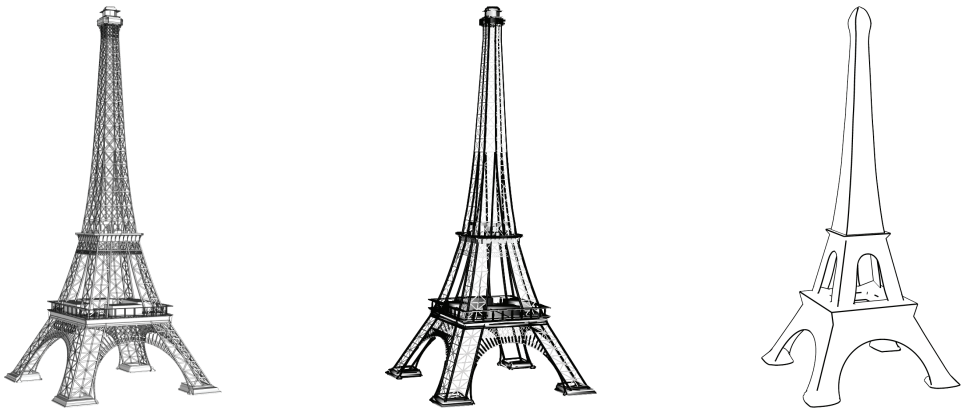
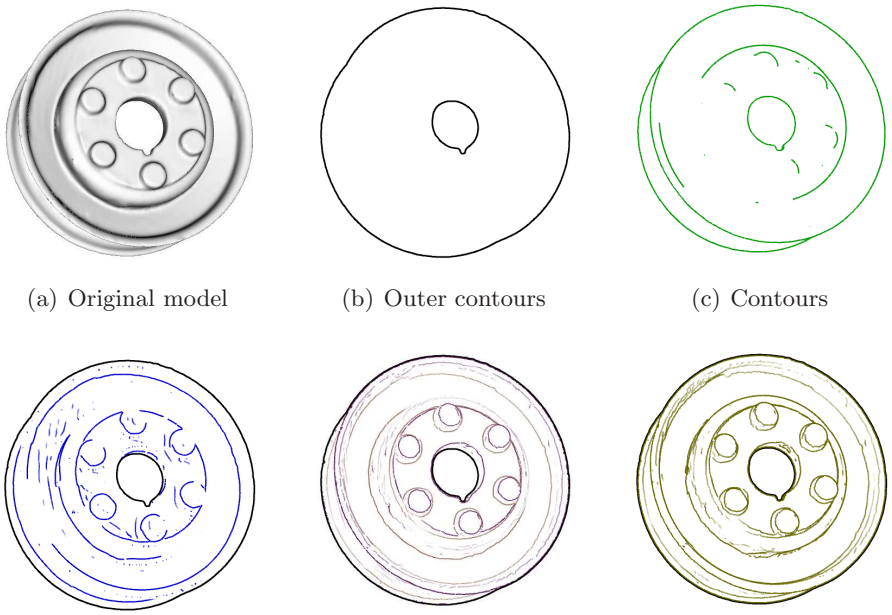


Figure 2.1: *Eiffel tower rendered using crease lines (left), suggestive contours (middle), and crease lines on our 3D abstraction (right).*

Contours [1] (points where the surface normals are orthogonal to the viewing direction) are the most commonly used feature lines for shape representation purposes [14, 20, 22, 26]. However, contours alone cannot fully convey even moderately complex shapes [12]. For instance, as shown in Figure 2.2(c), the cylindrical features on the front face of the pulley are not captured in the contour drawing. An important subset of contours, called outer contours or bounding contours (Figure 2.2(b)), are lines that separate the object from the background [24]. Outer contours are often combined with other NPR line representations to convey shape (Figure 2.2(d), 2.2(e), and 2.2(f)).

DeCarlo et al. [14] presented a new set of lines, the suggestive contours, which contains not only traditional contours but also features that would

[1] Also called occluding contours [24]. In certain literatures such as [20] and [22], the term "silhouette" is used instead of "contour". We will use the term "contour" in this thesis to avoid confusion.



(a) Original model (b) Outer contours (c) Contours

(d) Suggestive contours [14] (e) Ridges and valleys [35] (f) Apparent ridges [24]

Figure 2.2: *Comparison of common NPR line representations. Generated using the executable provided in [14]. For suggestive contour, ridges and valleys, and apparent ridges, the outer contours are overlaid on top as black lines.*

become contours in a nearby view as shape representations. As shown in Figure 2.2(d), suggestive contours extend the actual contours and contain lines that are almost contours. In the presence of noise, suggestive contours might be unstable and capture excess shallow surface features.

Ridges and valleys are another family of lines commonly used to convey shape. Conceptually, they capture places where the surface bends sharply. Formally, they are defined as the extrema of the principle curvatures in the corresponding curvature directions [35]. In certain situations such as in the absence of concavities, they yield better results than suggestive contours [12, 14]. However, due to the view-independent nature of ridges and valleys, they often cause the line representation to exaggerate the sharpness of features. In order to compensate for this, Na et al. [32] proposed to assign strength to each ridge and valley line based on the viewing angle. Only the features

with significant strengths are displayed in the final rendering.

Inspired by ridges and valleys, Judd et al. [24] introduced apparent ridges, which are positions where the surface normal changes at a locally maximal rate with respect to the image space. Apparent ridges essentially provide a reweighting scheme that favors regions where the projection foreshortening is most apparent, making the result view dependent. In the extreme case, contours have weights near infinity due to the big per pixel normal difference near those regions. Figure 2.2(e) and 2.2(f) show the result of rendering the pulley model using ridges and valleys as well as apparent ridges. Although they are able to represent the shape visually from the given view point, the lines are inconsistent between viewing directions.

All of the NPR algorithms mentioned above are illustrated in Figure 2.2. Notice that the lines extracted are often view-specific. While the view dependent nature of NPR line representations helps to convey the shapes visually, it is hard to use them for 3D abstract representations due to inconsistency across views. Moreover, input models often contain surface noise which causes excess small strokes in line representations and increase the difficulty for viewers to extract the underlying geometry. In contrast, the curve networks extracted by our abstraction method can be used to create view-independent NPR effects, which remain persistent across motion and animation and are robust against surface noises.

Little work exists for stylizing or creating iconic representations of 3D models. A notable exception is the research by Gal et al. [18], which creates 3D collages on top of target shapes using a database of objects as primitive building blocks. The resulting collages, though artistically powerful, are not intended for other uses.

2.3 Shape analysis and reverse engineering

Motivated by procedural modeling and constructive solid geometry (CSG), researchers have long proposed to approximate a given 3D model with parametric parts [3, 46]. Such parametric descriptions can indicate the structural composition of the given model, where different abstraction levels could be obtained by direct part manipulation such as removing some of the parts while preserving others. For example, in [3], the authors described a hierarchical segmentation algorithm based on fitting primitives such as planes, spheres, and cylinders. The paper shows very promising results for decom-

posing and denoising CAD models, where the input were primarily composed of those primitive shapes. However, due to the variety of man-made objects in general, it is hard to pick a pre-defined set of primitives as the basic building block. Thus, in general, reverse engineering structure and regularity from 3D geometry is still considered a difficult open problem [37], with algorithmic solutions unlikely to reach human performance levels in the near future. Instead, we present a method that uses low-level analysis of the models to automatically extract the main feature curves, providing a compact vector representation of the model at a desired abstraction level. Our approach bypasses the difficult reverse engineering task of detecting the global structure, while implicitly preserving the main characteristic features of the models.

In the domain of 2D shapes, it is well known that features that are stable over scale changes are significant [5, 47]. Since the extrema in a signal and its derivatives often provide important information (e.g. image edges [10]), Witkin et al. [47] proposed to detect and track signal extrema at different scales and introduced scale-space representation. The scale-space concept has been widely used in computer vision and image analysis. Bengtsson et al. [5] obtained abstractions by studying contours at different scales.

In 3D settings, finding features that are stable over scale changes is difficult. It would require to sample the input model at different density levels, reconstruct surfaces from these sampled point clouds, and analyze features that are consistent in all reconstructions. In theory, this approach may be sound, and the resulting stable features could be used for shape abstraction. However, given that the input could be non-manifold meshes of multiple components with self-intersections, even surface reconstruction from point clouds alone could be quite challenging because the points are unlikely to lie on a manifold surface. Even if the input model is manifold, the sampling randomness might still cause the final abstraction to miss certain important details while capturing other undesirable features. Moreover, such algorithm would rely on sampling to decide the importance of features, where large but hardly visible features (e.g. the interior wall of the teapot shown in Figure 2.3) are likely to be kept due to their large surface areas even though these features have very little contribution to the user perceived shape.

While the general shape analysis problem is difficult even for manifold, connected meshes, we show that for man-made objects, due to their inherent regularity and structure, creating effective abstraction, even from polygon soups, is possible. Our abstraction comes in the form of a curve network,

Figure 2.3: *Man-made objects may often contain large interior features, such as the interior wall of the famous Utah teapot. Left: the original Utah teapot. Right: the same teapot cut in the middle showing its interior wall.*

which can be used as input for modeling and editing systems. One of such systems is iWires [17], where the authors used a collection of 1D wires and their mutual relationship to manipulate a given shape in a structure-preserving manner. These wires were either extracted from the model based on dihedral angles or defined by a user to reflect the input structure. The curve network generated by our abstraction algorithm captures similar structural information for the reconstructed surface. Therefore, the position of the curve network can be considered as iWires to allow further manipulation of the abstracted shape. Another curve based surface modeling system is FiberMesh [34], where users draw a collection of 3D control curves and a manifold surface is created to interpolates them. Although both the FiberMesh control curves and our curve network can be used to define a surface, they are very different and serve distinct goals. Specifically, FiberMesh control curves are designed to allow smooth surface interpolation and intuitive model manipulation. In particular, the default type of FiberMesh control curves requires the interpolating surface smoothly interpolate the curves, and there is little correspondence between these control curves and surface features. In contrast, our curve networks aims to capture those shape defining features of a given model.

Chapter 3

Algorithm Overview

Our goal is to extract a *vector* representation for three-dimensional shapes, targeted specifically toward abstraction of man-made objects, i.e., objects whose main features can be captured by a few smooth surfaces glued together along characteristic curves. Our representation satisfies the following properties:

Reconstruction: The network of curves, or vectors, combined with the normals prescribed along them, is sufficient to define the abstracted shape, encoding both the connectivity and the geometry of the reconstructed model (Chapter 6). To describe the connectivity, the network is required to be a connected *B-Rep*[2] *representation*, as this significantly simplifies the reconstruction step. To adequately reconstruct the geometry we define the surface normals across the model as weighed combinations of the curve normals.

Abstraction: Depending on the desired level of abstraction, our representation controls which geometric features are appropriate, while smoothing out the less significant ones.

Structure: Regularity and structure, which lead to simplicity of design, fabrication, and installation, are properties common to most man-made objects [29]. Viewers are known to be sensitive to breakup of regular structures present in the input models when those are processed [27], and they seem to remember or identify shapes based on symmetry and regularity, ignoring the deviations [2]. Thus we expect abstractions of man-made shapes to be as regular or structured as possible. This translates to the curves being locally as simple as possible, i.e., being planar, linear, circular arcs, etc., while satisfying global regularity requirements such as symmetry, parallelism, and orthogonality.

Our abstraction method generates such vector representations in two steps.

[2]Boundary representation, where a solid shape is represented using the surface boundaries.

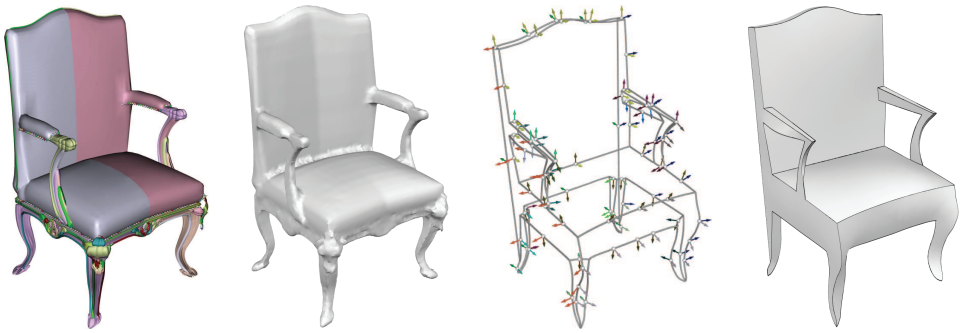


Figure 3.1: *(Left to right) Input model with* 353 *components, envelope surface, vector representation, and reconstructed model.*

First, it constructs an *envelope* surface, a closed, manifold surface approximating or "enveloping" the input model (Chapter 4). The surface provides a manifold approximation of the potentially poorly-connected data, enabling subsequent extraction of a meaningful network of curves. It also helps to approximate the input at a desired degree of abstraction, controlling the genus of the final model as well as smoothing out minor details (Figure 3.1).

The second step extracts the curve network that serves as the vector representation of the input model (Chapter 5) . First it extracts the network connectivity using mesh segmentation and then establishes the network geometry using a combination of regularization and approximation criteria. The geometry computation is performed using an optimization procedure that simultaneously computes the vector representation and an approximation of the reconstructed surface obtained from this representation. The combined computation controls the tradeoff between the smoothness and regularity of the reconstruction, and the level of approximation of the input.

This project is a *collaboration with* several other researchers. Therefore, in this thesis, I will mainly focused on the parts that I contributed. Namely, I will focus on the envelope generation process (Chapter 4) and the smooth surface approximation (Section 5.2). The other components are also present in this thesis for completeness.

Chapter 4

Envelope

The envelope construction stage defines a tight manifold approximation of the input models, providing both a first level of abstraction and a well-defined domain for further processing. This step can be skipped if the input model is *a priori* described by a single manifold surface. However, in this case the abstracted model will preserve the topology of the input.

While it is conceivable to have an implicit, distance function based approach followed by isosurface extraction for constructing the envelope [8], such an approach is ill-suited for our setup, as our input models often contain many self intersections and may have inconsistent triangle orientation, making distance computations problematic.

Instead, we start by constructing an initial envelope that contains the input model and loosely follows its geometry. Using a local refinement process similar to iterative closest point (ICP) [7], the envelope is attracted toward the model to obtain the desired approximation quality (Figure 4.1). The iterative fitting process helps satisfy the potentially conflicting goals of bringing the envelope mesh close to the input model while maintaining a quality triangulation of the envelope. The fitting process, which provides a tradeoff between envelope resolution and input approximation, smoothes out narrow concavities, yielding a first level of abstraction.

Section 4.1 provides details about initializing the envelope construction process. Section 4.2 explains the operations performed in each iteration. Envelopes of various input models are shown at the end of this chapter.

4.1 Initialization

The goal of the initialization step is to construct a surface that approximates the shape of the given model. We embed the input model in a regular grid and define the *voxel hull* of the model as the set of all grid voxels that

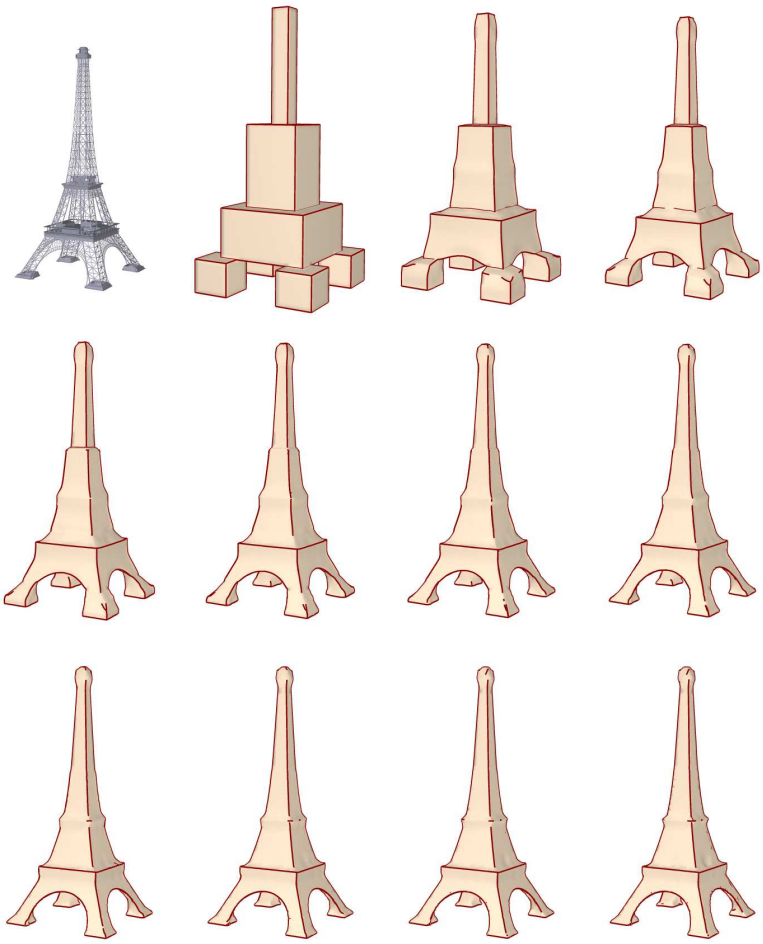
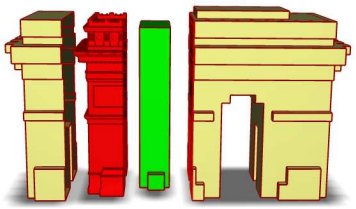


Figure 4.1: *Iterative envelope generation process: (Top left to bottom right) original mesh, initial envelope generated (grid size* $9 \times 23 \times 9$ *and subdivided twice), iteration 1 to 9, and iteration 11.*

intersect any of the model triangles. Often, the voxel hull is formed by a closed layer of grid cells which encompass a space made of non-intersecting voxels in the middle. As shown in Figure 4.2, such voxel hull may have multiple disconnected surfaces that separate the intersecting voxels from the non-intersecting ones. We use the term *interior separating surface* to denote the surface that separates regions surrounded by the voxel hull from the hull, and the term *exterior separating surface* for the surface between the



(a) Anatomic view

(b) Exterior, original, and interior surfaces

Figure 4.2: *There might be multiple surfaces that separates the voxel hull with the non-intersecting voxels. (a): anatomic illustration of the Arc de Triomphe model which produces two separating surfaces. The exterior separating surface (yellow) encloses the input model (red), and the interior separating surface (green) is enclosed by the input model. (b): (left to right) exterior separating surface, input model, interior separating surface.*

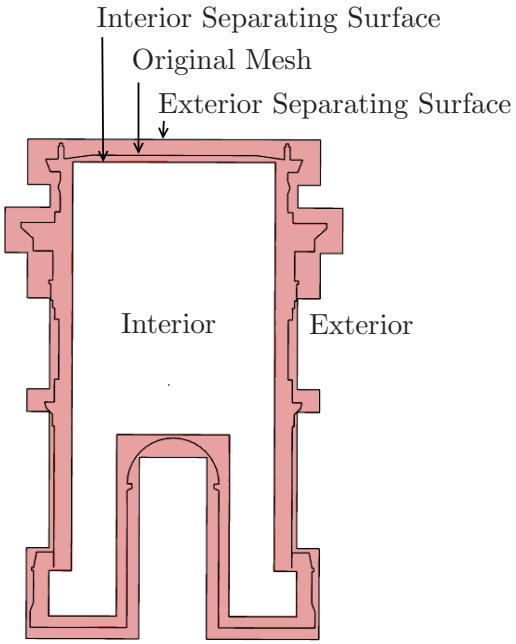
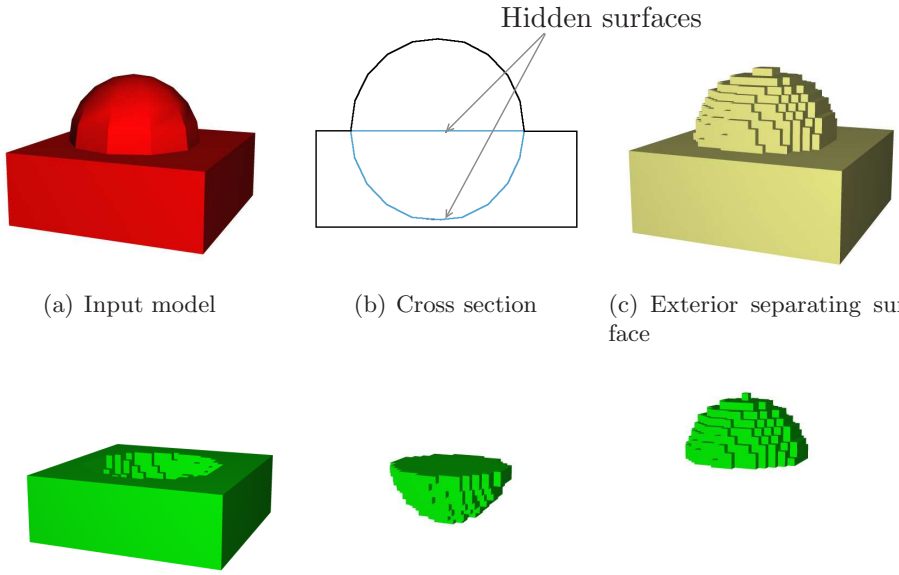


Figure 4.3: *Cross section of the voxel hull (red). Notice that the input model lies between the exterior separating surface and the interior separating surface.*

voxel hull and the region outside it (Figure 4.3). *We do not use interior separating surface in our algorithm because there could be several interior separating surfaces due to hidden surfaces in the input model (Figure 4.4). In contrast, the exterior separating surface is unique for each component of the voxel hull.* We use the exterior separating surface as the initial envelope because it loosely approximates the visible structure of the input model.

The initial envelope construction consists of three steps:



(a) Input model (b) Cross section (c) Exterior separating surface

(d) Interior separating surfaces

Figure 4.4: *A synthetic model with hidden surfaces that produces multiple interior separating surfaces.*

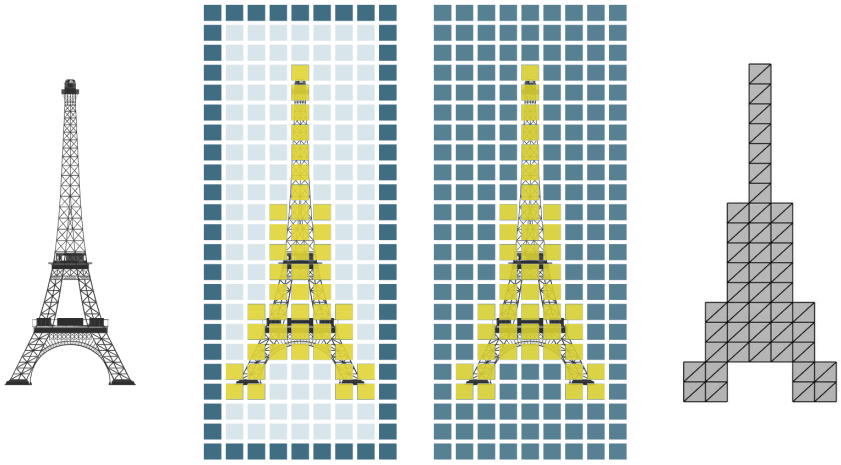
1. **Computing the Voxel Hull:** The goal of the computation is to determine all voxels intersecting the input model. The brute force way to achieve this is to perform intersection tests of all voxels with every edge and every triangle. The running time is $O\left(|V_h| \cdot (|E| + |F|)\right)$, where $|V_h|$, $|E|$, $|F|$ are the number of voxels, the number of edges, and the number facets respectively. Such method is very inefficient especially when there are many non-intersecting voxels (Figure 4.5(c)).

   Instead, for each triangle, we perform intersection test only on voxels within its bounding box, and for each edge, we find intersecting voxels by ray tracing from one endpoint to the other using the algorithm described in [1]. The running time for this step is $O(m_1|F| + m_2|E|)$, where m_1 is the maximum number of voxels in the bounding box of a facet, and m_2 is the maximum number of voxels that intersect an edge. In all models used in this thesis, both m_1 and m_2 can be bounded by a constant number c, which makes the running time $O(c|F| + c|E|) \in O(|E|)$. Figure 4.5(b) shows the voxel hull of the tower model.

2. **Computing the Exterior Voxels:** The next step is to find all non-intersecting voxels that are exterior. The outer most layer of voxels in the grid is exterior by construction, shown as blue boxes in Figure 4.5(b). Since any non-intersecting voxels adjacent to an exterior voxel are also exterior, we expand the exterior region inwards until it reaches the voxel hull. This operation is $O(|V_h|)$. Figure 4.5(c) shows the grid with exterior voxels colored blue.

3. **Surface Extraction:** The final step is to extract the surface that separates the exterior voxels from the voxel hull. This operation is $O(|V_h|)$. Figure 4.5(d) gives an example of such extracted surface.

The resulting surface is the initial envelope. Overall, the initialization step has running time $O(|E| + |V_h|)$. Once the envelope is constructed we refine the initial coarse mesh using one or two iterations of regular subdivision to enable better subsequent approximation.



(a) Input mesh (b) Intersecting voxels and exterior voxels (c) Exterior region propagation (d) Initial envelope

Figure 4.5: *Initial envelope generation. Given an input mesh (a), a grid of voxels is created to enclose it. (b) The intersecting voxels (gold) and the outer layer (blue) is computed. (c) The exterior region propagates inwards until it reaches the voxel hull. (d) Lastly, the initial envelope is extracted as the surface separating the exterior voxels from the voxel hull.*

The voxel grid resolution directly determines the topology of the resulting

abstraction. Therefore, we allow users to choose a suitable resolution for each given model. Figure 4.6 shows the effect of different voxel grid resolution for the Eiffel Tower model.

In addition, the voxel grid resolution plays an important role in determining the abstraction level of the final envelope. As shown in Figure 4.7, narrow concavities that are not captured in the initial envelope will remain abstracted in the final envelope as well. See Section 4.2 for detailed discussion.

Besides its effects on the final topology and the abstraction of narrow features, the initial envelope only needs to follow the input model geometry loosely. For instance, both the the center and right voxel hulls of Figure 4.6 capture the cavity in the center of the tower but produce different cavity shapes. Despite the distinct cavity shapes in these two initial envelopes, the corresponding final envelopes are very similar.

4.2 Iterative fitting

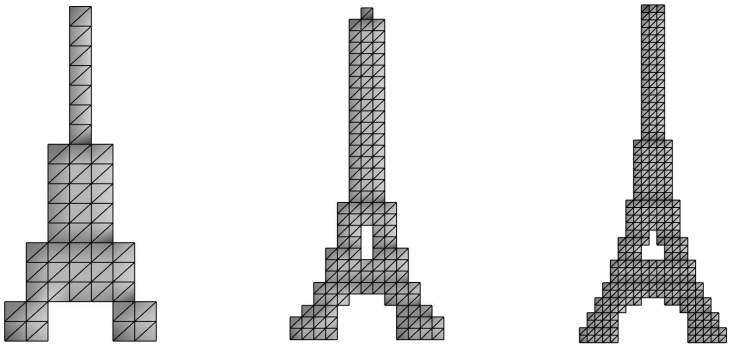
Once we have an initial envelope that loosely follows the input geometry, the iterative fitting process will progressively tighten the gap between the envelope and the input model. Similar to ICP [7], where a model is iteratively rotated and translated to align with the given shape, we iteratively deform the envelope towards the input model while preserving the overall shape and mesh quality. Each iteration consists of the following steps.

- *Matching*, which maps the vertices of the envelope to the closest positions on the input model;
- *deformation*, which deforms the envelope to better approximate the input based on the computed matches;
- and *mesh regularization*, which maintains the quality of the envelope mesh as it deforms.

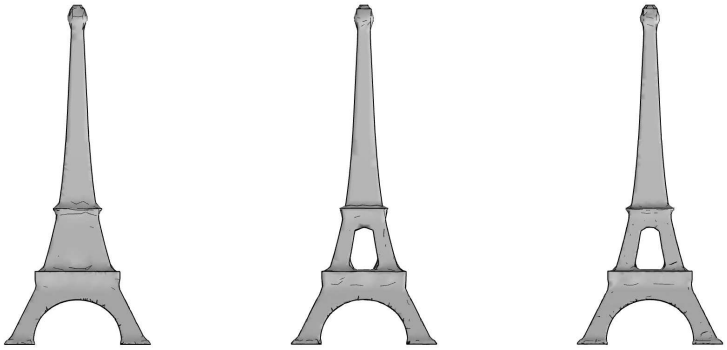
The process terminates when the envelope stabilizes or a maximum number of iterations is reached. We now describe the steps in detail.

4.2.1 Matching

Each vertex of the envelope is matched to the closest position on the input model. The straight forward matching method is to check the distance



(a) Left to right: grid size $(9 \times 23 \times 9)$, $(17 \times 41 \times 17)$, and $(27 \times 65 \times 27)$.



(b) The corresponding envelopes generated with two level of subdivision (left) and one level of subdivision (center, right).

Figure 4.6: *The effect of different grid sizes on the voxel hulls and the final envelopes.*

from an envelope vertex to every facet of the original mesh and find the shortest distance. This will take $O(|V_e| \cdot |F|)$, where V_e is the set of vertices of the envelope, and F is the set of facets in the original mesh.

To accelerate this, we observe that the input model is entirely contained within the voxel hull (Figure 4.3). Since the initial envelope is the exterior separating surface of the voxel hull, any envelope vertex is at most d_{cell} away from the input mesh, where d_{cell} is the diagonal length of a voxel. In addition, each iteration of the fitting step pulls the envelope towards the input model, so the distance between each individual envelope vertex and the input mesh is likely to decrease as well. Therefore, when searching for the

closest match of a target vertex, we eliminate the checks for facets known to be further than $2d_{cell}$ away. We use space subdivision to prune these unnecessary distance computations. The bounding box of the envelope is subdivided into a regular grid with the same grid size as in the initialization step (each cell's diagonal length is d_{cell}). For each envelope vertex, the grid cell that contains this vertex is calculated using its coordinates. To find the closest match, we only check the facets intersecting the same grid cell as the target vertex plus the neighboring grid cells, which contain all facets within the $2d_{cell}$ distance range. Thus, the running time becomes $O(|V_e| \cdot m)$, where m is the maximum number of facets intersecting a grid cell. Although, in the worst case, this algorithm is still $O(|V_e| \cdot |F|)$, each grid cell on average only intersect a constant number of facets. Therefore, the running time is roughly linear in $|V_e|$ for most of the input models.

Similar to ICP setup, we discard outlier matches. We use two simple but effective heuristics to validate each match. First, matched positions too far from the envelope are likely wrong. This check is implicitly enforced by the space subdivision algorithm described above, where matches with distance greater than $2d_{cell}$ are not considered. Second, since the initial envelope is strictly outside the input model, we force every envelope vertex to map inward. More specifically, for each envelope vertex, we enforce its matched point to be in the opposite direction with respect to its normal. This requirement works well for the earlier iterations. However, in latter iterations, the envelope might intersect the input mesh and the mappings could be in the direction of the normals. Therefore, we only apply this validation when the envelope is sufficiently far from the input mesh (average distance is greater than 0.01 times the diagonal of the bounding box).

The matching step plays a key role in removing narrow concavities. A concavity that is not captured by the initial voxel hull could be captured during the matching step if an envelope vertex is mapped to the bottom of the concavity. There are two factors that affect this possibility: the depth to width ratio of the concavity, and the position of the envelope vertex. For concavities with depth more than half of its width, it is impossible for an envelope vertex outside of the concavity to map to the bottom of the concavity because the boundary of the concavity will always be closer to the vertex than the bottom (Figure 4.7 right). For other cases, shallow concavity could be captured if there is an envelope vertex outside of the concavity that is closer to the concavity bottom than to the boundaries (Figure 4.7 left). Thus, such shallow concavities are more likely to be captured by dense envelopes than sparse envelopes.

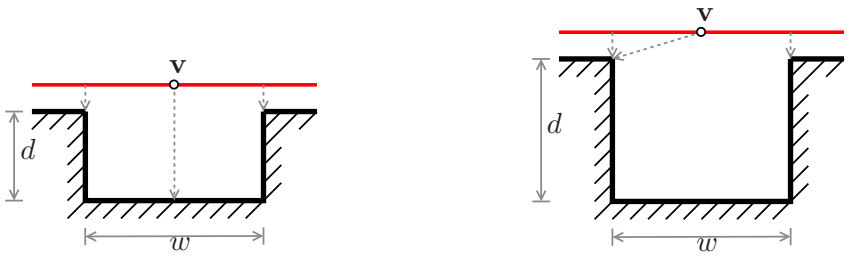


Figure 4.7: *Concavity not captured in the initial envelope (i.e. w is narrower than the grid resolution) might be captured during the matching step. Envelope is in red, and original mesh is in black. Left: the concave feature is captured because $d \leq 0.5w$, and $\mathbf{v}$ is mapped to the bottom of the concavity. Right: if $d > 0.5w$, $\mathbf{v}$ will map to one of the sides, thus the concavity is removed.*

4.2.2 Deformation

Despite filtering out outliers, the target positions provided by the correspondences established in the matching step can still sometimes be inconsistent, leading to self-intersections and other mesh degeneracies if strictly enforced. To avoid such artifacts we introduce a deformation formulation that provides a tradeoff between enforcing the matches and maintaining the shape and position of the current envelope. Specifically we optimize

$$\min_{\mathbf{v}_i} \sum_i c_1 T_l + c_2 T_o + c_3 T_n \tag{4.1}$$

$$\begin{aligned} \text{where } T_l &= \left\| \left(\mathbf{v}_i - \frac{1}{|(i,j)|} \sum_{e=(i,j)} \mathbf{v}_j \right) - \mathbf{l}_i \right\|^2, \\ T_o &= \|\mathbf{v}_i - \mathbf{v}'_i\|^2, \\ T_n &= \|\mathbf{v}_i - \mathbf{w}_i\|^2, \end{aligned}$$

$\mathbf{v}_i$ are the new positions of the mesh vertices, $\mathbf{v}'_i$ are the current vertex positions, $\mathbf{l}_i$ are the Laplacian vectors [42] given by $(\mathbf{v}'_i - \sum_{e=(i,j)} \mathbf{v}'_j/|(i,j)|)$ computed on the current envelope, and $\mathbf{w}_i$ are the positions proposed by the matching.

The term T_l is the objective function of many state of the art deformation algorithms [42], where it is often used to preserve the local details of the

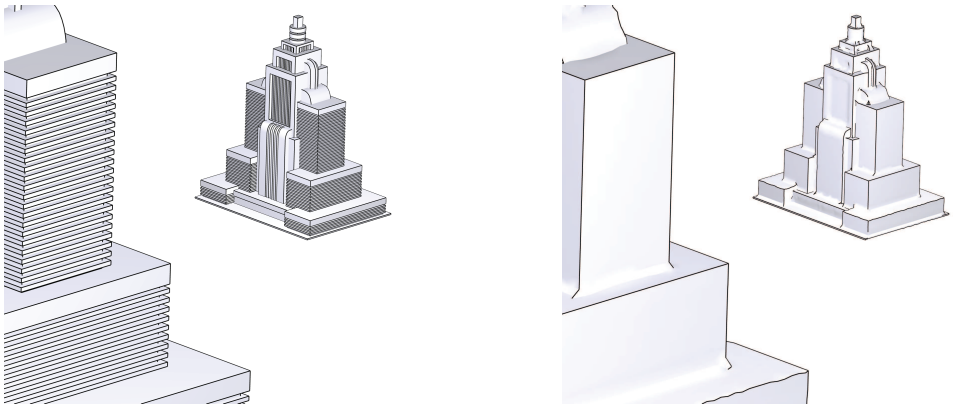


Figure 4.8: *The deformation step, during envelope fitting, smooths out narrow concavities and suppresses details of the input model.*

mesh. T_o and T_n pull the envelope towards its original position and its mapped position respectively. The weights determine the balance between the original shape and the target mesh shape as well as the speed of the convergence. We used $c_1 = 1$, $c_2 = 0.5$, $c_3 = 2$ for all the models shown in this thesis.

The shape preservation terms T_l and T_o combined with the regularization step further help smooth over narrow concavities in the input model where the size of the narrow features is smaller than the initial grid resolution (see Figure 4.8). This aids the abstraction process by removing potentially deep, but poorly visible, features.

The resulting linear system is solved using a sparse linear solver [45]. We choose to use Cholesky factorization to speed up the solving process. The following option is passed to the TAUCS linear solver.

```
"taucs.factor.LLT=true","taucs.factor.ordering=metis",NULL
```

4.2.3 Mesh regularization

As the envelope deforms, the quality of the mesh triangles can increasingly deteriorate as the initial connectivity may not accurately reflect the fitted geometry. Such mismatching connectivity can negatively affect the quality of the approximation (Figure 4.9). Our iterative fitting addresses this concern by performing mesh regularization after every iteration. The basic

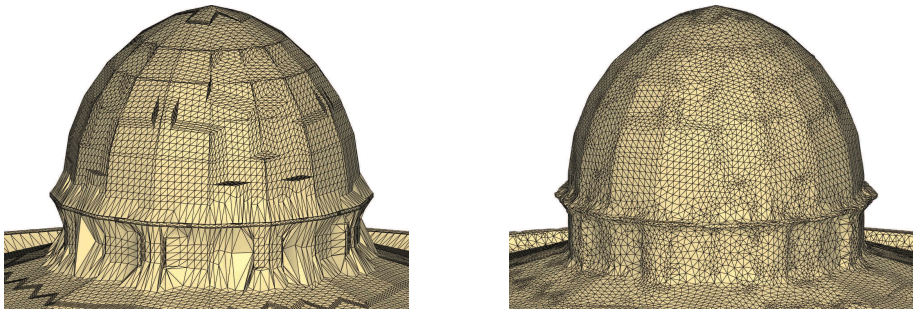


Figure 4.9: *Zooms of the mesh envelope, for the Dome of the Rock model, after four iterations of fitting without (left) and with (right) regularization.*

regularization step is based on [44] and involves a combination of local mesh smoothing, edge collapses, edge flips, edge splits, and degenerate triangle removal operations aimed at maintaining reasonable mesh quality.

Mesh Smoothing. As the envelope deforms, multiple envelope vertices often maps to the same or close by positions on the input mesh, which may produce poorly shaped or tiny triangles. The smoothing step improves the triangle quality by sliding vertices along the surface. Specifically, for each vertex, we compute the average of its neighbors and use the projection of this average onto the tangent plane as the new position. As shown in Figure 4.10, the new position $\mathbf{v}_i$ of vertex i is

$$\begin{aligned} \mathbf{v}_i &= \mathbf{v}'_i + \mathbf{t} \\ \text{where } \mathbf{t} &= (\mathbf{p}'_i - \mathbf{v}'_i) - \big((\mathbf{p}'_i - \mathbf{v}'_i) \cdot \mathbf{n}\big)\, \mathbf{n}, \end{aligned}$$

$\mathbf{v}'_i$ is the position before smoothing, $\mathbf{p}'_i = \sum_{e=(i,j)} \mathbf{v}'_j / |(i,j)|$ is the average of neighbors of vertex i, and $\mathbf{n}_i$ is the vertex normal. Note that all vertex normals are computed as the area weighted average of the facets normals adjacent to them. Since each vertex stays on its tangent plane, the shrinkage problem associated with standard mesh smoothing is largely alleviated.

To avoid smoothing out sharp details on the envelope, the following two metrics [44] are used to determine the flatness of the local neighborhood around a triangle and the fidelity of the resulting mesh:

$$E_{dist} = \max_{i=\{0,1,2\}} \arccos(\mathbf{n}_i \cdot \mathbf{n}_{i+1}) < \theta_{dist} \tag{4.2}$$

$$E_{smth} = \max_{i=\{0,1,2\}} \arccos(\mathbf{n}_f \cdot \mathbf{n}_i) < \theta_{smth} \tag{4.3}$$

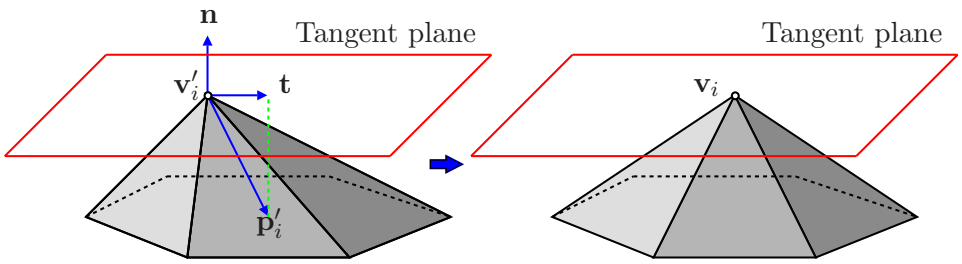


Figure 4.10: *Mesh smoothing: each vertex is moved to the projection of* $\mathbf{p}'_i$ *onto the tangent plane, where* $\mathbf{p}'_i$ *is the average of neighbors.*

where $\mathbf{n}_i$ are the vertex normal before smoothing, and $\mathbf{n}_f$ is the face normal after smoothing. All indices are modulo 3. A vertex is smoothed if and only if both E_{dist} and E_{smth} satisfy the corresponding thresholds.

The term E_{dist} measures how flat the surface is locally before smoothing, and it is defined over the local neighborhood of a triangle (i.e. all triangles sharing a vertex with the given triangle). Figure 4.11 shows two triangles along with their local neighborhoods. In relatively flat neighborhoods (left), all vertices of a triangle have similar normals, and hence E_{dist} is small. On the other hand, in a non-flat neighborhood (right), E_{dist} is big due to large normal difference between the triangle vertices. We set the threshold θ_{dist} to 45° for all examples shown in this thesis.

In addition to the smoothness criterion, the term E_{smth} serves as a heuristic to preserve shape. Consider each envelope triangle as a linear local approximation of a smooth surface, where the vertex normal is the surface normal at the corresponding vertex, and each triangle approximately coincides with the local tangent plane. Notice that the triangle normal is an interpolation of the three vertex normals. Thus, E_{smth} requires the resulting triangle normal to remain close to the vertex normals in order to prevent deviation from the original shape. As shown in Figure 4.12, whenever the resulting triangle normal differs from any of the vertex normals by a large amount, θ_{smth}, the smoothing operation is not carried out. For all examples shown in this thesis, θ_{smth} is set to 30°.

Edge Collapses. Although the smoothing step greatly improves the triangle shapes, tiny triangles often remain tiny. To eliminate these, we collapse edges shorter than a given tolerance. Each collapse operation removes an edge by moving one of its end points (call it $\mathbf{s}$) to the other end point (call

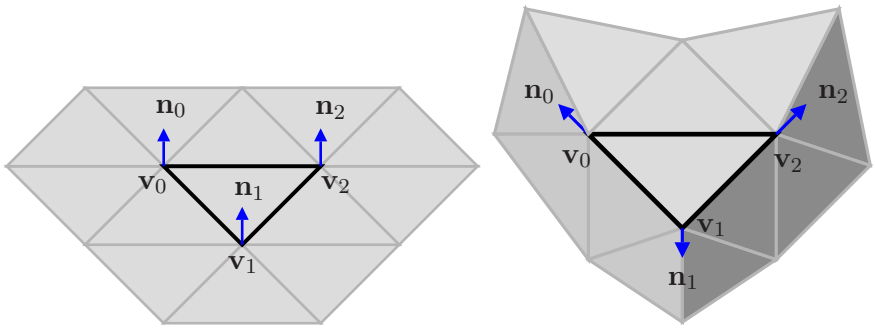


Figure 4.11: *Left: in locally flat regions, all vertex normals of a triangle are approximately equal. Right: in non-flat regions such as near a crease, the vertex normals of a triangle could be quite different.*

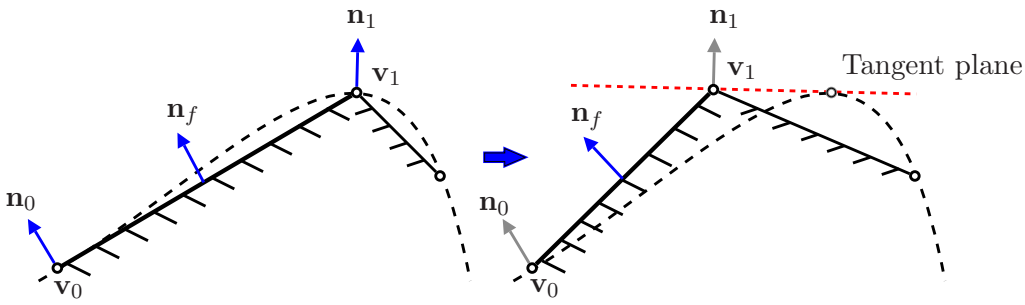


Figure 4.12: *Cross section view of mesh triangles (solid black lines) and the smooth surface (dashed black curves) they approximate. Before smoothing (left), the face normal $\mathbf{n}_f$ is close to the vertex normals $\mathbf{n}_0$ and $\mathbf{n}_1$. After smoothing (right), the new $\mathbf{n}_f$ might be very different from the original $\mathbf{n}_0$ and $\mathbf{n}_1$.*

it $\mathbf{t}$). $\mathbf{s}$ and $\mathbf{t}$ are chosen to minimize the shape difference introduced by this operation. Specifically, we require

$$\sum_{e=(\mathbf{v},\mathbf{s})} 180^\circ - A(\mathbf{v},\mathbf{s}) <= \sum_{e=(\mathbf{v},\mathbf{t})} 180^\circ - A(\mathbf{v},\mathbf{t})$$

where $A(\mathbf{a},\mathbf{b})$ is the dihedral angle across the edge $(\mathbf{a},\mathbf{b})$.

As usual [44], we avoid producing non-manifold meshes. Specifically, as shown in Figure 4.13, edges that are part of a 3-loop are not collapsed regardless of their length. Otherwise, the collapse will cause non-manifold mesh where an edge could connect more than two triangles. We used one

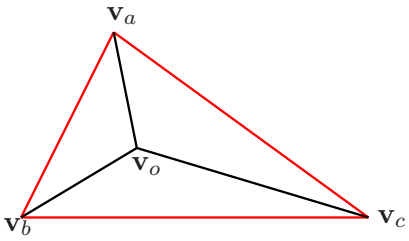


Figure 4.13: *Collapsing any of the 3-loop edges (red) will cause the mesh to be non-manifold.*

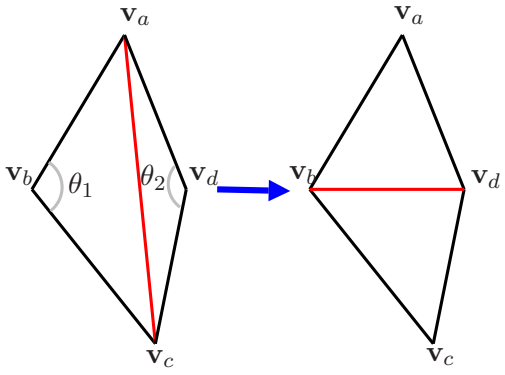


Figure 4.14: *The red edge is flipped if* $\theta_1 + \theta_2 > 180°$.

fifth of the average edge length as the tolerance for edge collapse operation.

Edge Flips. Mesh smoothing improves the geometry of the envelope while the connectivity is left unchanged. Bad connectivity, such as high or very low valence vertices, can negatively affect the mesh quality. We therefore use edge flipping to improve mesh connectivity. In 2D triangulation, edge flips are used to achieve the Delaunay property [6]: the circumcircle must contain no other mesh vertices except on its border. In 3D settings, edge flips are also the basic operation used for generating high quality meshes such as Delaunay mesh [15]. In our remeshing algorithm, as shown in Figure 4.14, edges with sum of opposite angles bigger than 180° are flipped. Once again, we use formula (4.3) to check if the result preserves the original shape. In Figure 4.14, the algorithm will compute E_{smth} for triangle $(\mathbf{v}_a, \mathbf{v}_b, \mathbf{v}_d)$ and triangle $(\mathbf{v}_b, \mathbf{v}_c, \mathbf{v}_d)$. If either result exceeds the threshold, the flipping operation will not be carried out. As for smoothing, θ_{smth} is set to 30°. As before, to avoid introducing non-manifold regions, we check that the flipped edge does not exist before performing the flipping operation. For instance, in the case shown in Figure 4.14, we check that there should not be an edge connecting $\mathbf{v}_b$ and $\mathbf{v}_d$ before the flipping.

Edge splits. Sometimes, an edge become excessively long, but it cannot be flipped because condition 4.3 fails. Such case typically happen on the creases of the mesh, where the flipping operation will interrupt the crease

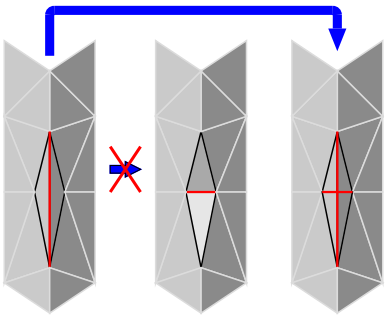


Figure 4.15: *Flipping an edge (red) that fails condition 4.3 may break the crease (center). Thus, we split such edge into two halves (right).*

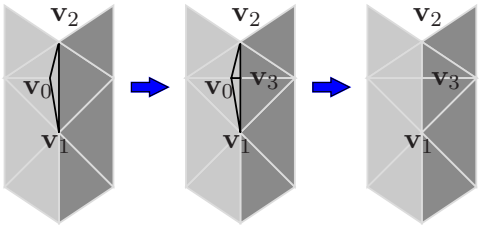


Figure 4.16: *Left: degenerate triangle* $(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2)$ *was not fixed by mesh smoothing and edge flips because condition 4.3 failed. Also, since none of its edges are excessively long, edge splits did not fix it either. Center: during degenerate triangle removal step, edge* $(\mathbf{v}_1, \mathbf{v}_2)$ *is first split. Right: edge* $(\mathbf{v}_0, \mathbf{v}_3)$ *is then collapsed.*

(Figure 4.15). We split such edge into two halves. We use 1.5 times the average edge length as the threshold.

Degenerate triangle removal. Even with all the operations described above, the resulting mesh may still contain some very skinny but obtuse triangles (Figure 4.16), where each edge is longer than the edge collapse threshold and shorter than the edge split threshold. These triangles are typically located near the creases where the smoothness requirement (formula 4.3) fails so that mesh smoothing and edge flip operations cannot be applied. Therefore, as the last step of the regularization, we remove these degenerate triangles. To be conservative, we define every triangle with the largest angle greater than 120° as a degenerated triangle. In such case, we first split the longest edge of the triangle even if its length does not exceeds the threshold (Figure 4.16 center). The split introduces four new edges, and a regular edge collapse operation is performed on these newly formed edges (Figure 4.16 right). Notice that the edge collapse operation is designed to minimize shape changes, thus the algorithm always collapses the vertex not on the crease toward the vertex on the crease. For instance, in Figure 4.16, $\mathbf{v}_0$ is collapsed to $\mathbf{v}_3$, thus the crease is preserved.

Other operations. In addition to remeshing for improving the mesh quality, in later iterations of the envelope contraction we use heuristic remeshing

operations to improve the approximation quality. Specifically, we flip edges if the mid-point of the flipped edge is closer to the input model than that of the current edge. If the flip creates poorly shaped triangles, we split the flipped edge at the midpoint. The order of flipping is determined by the approximation improvement amount, i.e., the edge for which the improvement is largest gets flipped first.

Results. The resulting envelope surfaces are manifold triangle meshes with fair mesh quality that approximate both the topology and the geometry of the input model up to a desired resolution. The envelope generation step fills up small through-holes, smooths out narrow concavities, and removes self-intersections. Figure 4.17, 4.18, 4.19, and 4.20 demonstrate envelope results for four different models, each with two distinct resolution.

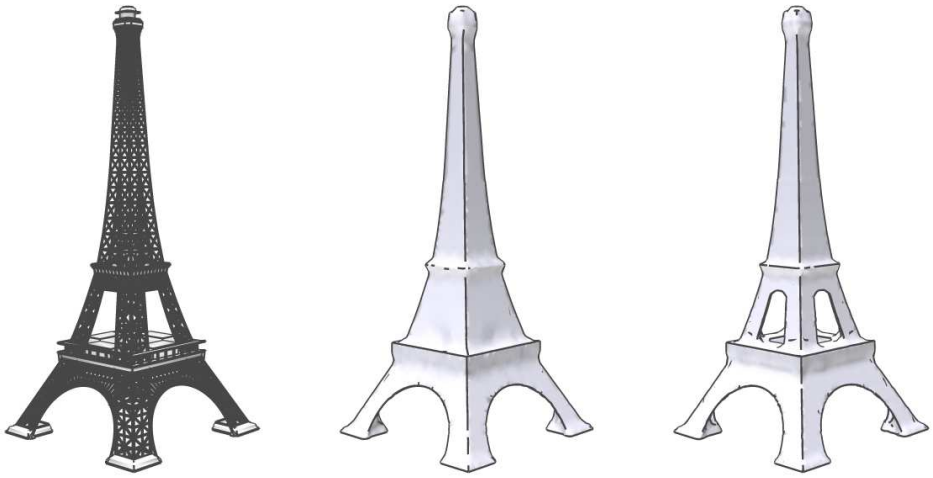


Figure 4.17: *Eiffel Tower (left). Low resolution envelope (middle) is initialized with grid size* $9 \times 23 \times 9$*. The construction converged in 11 iterations. High resolution envelope (right) is initialized with grid size* $27 \times 65 \times 27$*. The construction converged in 7 iterations.*

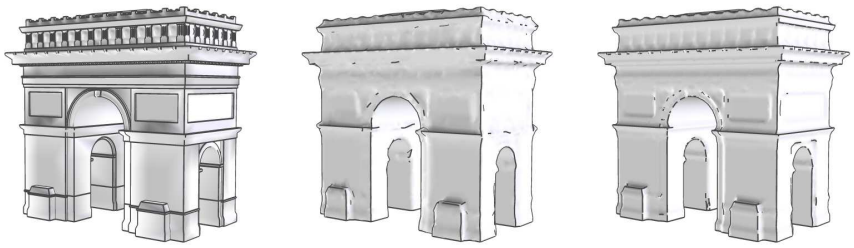


Figure 4.18: *Arc de Triomphe (left). Low resolution envelope (middle) is initialized with grid size* $21 \times 35 \times 33$*. The construction converged in 12 iterations. High resolution envelope (right) is initialized with grid size* $35 \times 61 \times 57$*. The construction converged in 8 iterations.*

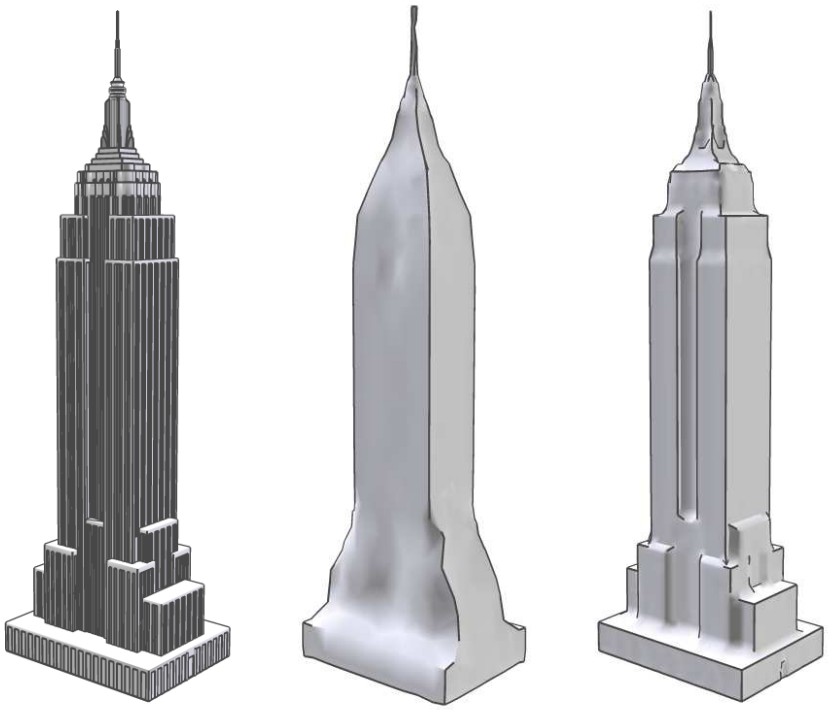


Figure 4.19: *Empire State Building (left). Low resolution envelope (middle) is initialized with grid size* $7 \times 9 \times 7$*. The construction converged in 6 iterations. High resolution envelope (right) is initialized with grid size* $41 \times 141 \times 29$*. The construction converged in 5 iterations.*

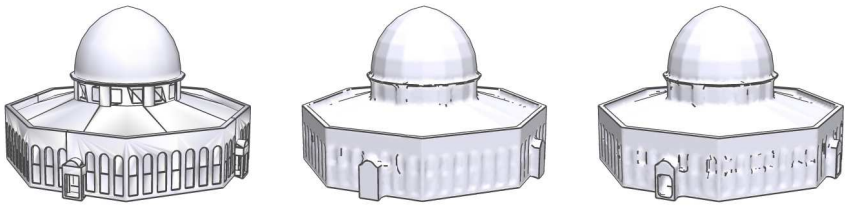


Figure 4.20: *Dome of the Rock (left). Low resolution envelope (middle) is initialized with grid size* $37 \times 23 \times 37$*. The construction converged in 10 iterations. High resolution envelope (right) is initialized with grid size* $67 \times 43 \times 67$*. The construction converged in 8 iterations.*

Chapter 5

Segmentation

Having generated an envelope surface we now proceed to extract the network of curves and associated normals sufficient to reconstruct an abstracted replica of the envelope and hence the input shape. To avoid storing unnecessary geometric information, we distinguish between two types of curves in our network: *regular* curves that have a geometric definition (positions and normals) along the curve, and *virtual* or *connectivity-only* curves that have no such information and which are used solely to define the connectivity of the network.

The network is extracted in two steps. We first extract the network connectivity using a mesh segmentation mechanism. We then finalize the curve geometry using smoothing and regularization enforcing spatial, relational, and metric constraints (Figure 5.1).

5.1 Extracting network topology

We recast the network extraction problem as one of mesh segmentation, with the curve network defined as the set of chart boundaries. Explicitly aiming at a segmentation that satisfies the reconstruction and approximation criteria in Chapter 3 is likely to be rather time consuming, since even evaluating a chart quality would be computationally expensive. Instead we opt for a simpler, conservative approach, requiring charts to be relatively planar. Clearly any chart that satisfies a near-planarity requirement would satisfy the other two. Though alternative criteria such as chart developability [25] are likely to work as well, the advantage of the planarity criterion is the simplicity and speed of the resulting method. While this approach may initially result in an over-segmented network, the number of charts is later reduced when the network is hierarchically simplified (Section 5.3).

We employ the Variational Segmentation Algorithm (VSA) [11] to obtain

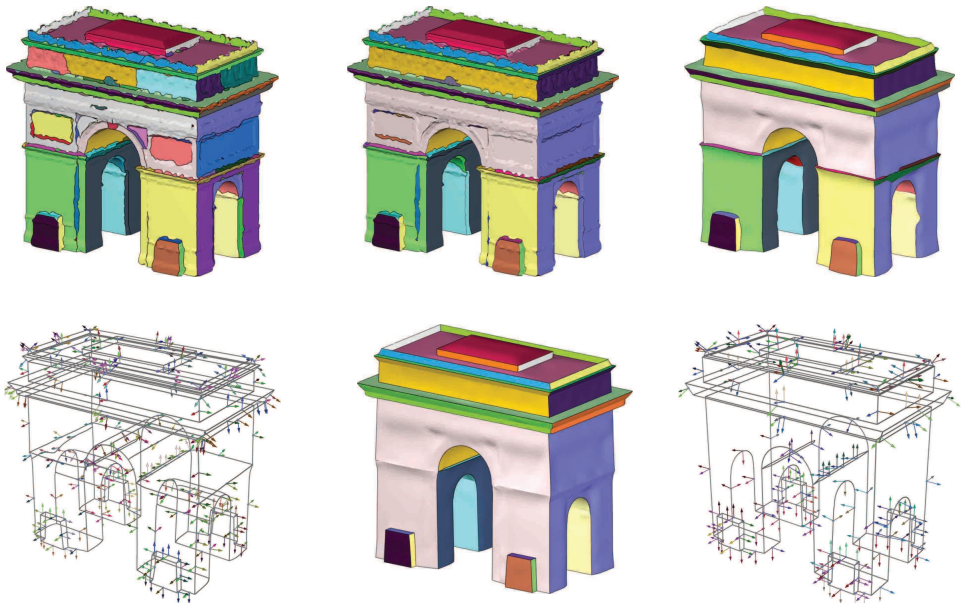


Figure 5.1: *Vectorization stages: (Left to right) VSA segmentation, segmentation after boundary improvement, smooth approximation geometry, extracted regularized curve network, surface after hierarchical simplification, regularized simplified curve network.*

the near-planar segmentation, using the approximation error to control the number of charts. We start VSA with an initial number of charts (typically just one) and measure the total approximation error. If the error is above the user-prescribed threshold we add another chart, using the triangle with the maximal error as the seed and repeat the process. The segmentation is improved using a standard post-processing step [25] of straightening boundaries and merging small charts with their neighbors, while bounding the per-chart error (we allow up to 10% increase in per-chart error for straightening and 20% for merging).

In the last step, to create a connected network, we split charts with multiple boundary loops into simple ones using a bottom-up triangle clustering. The new boundaries are defined as connectivity-only as they are unnecessary from a geometry point of view.

5.2 Extracting network geometry

We expect the reconstructed surface to consist of smooth charts bounded by the prescribed curve network. Hence, our goal is to define positions and normals along the curves, that enable such a reconstruction. We achieve this using an optimization that simultaneously edits the chart boundaries and the chart geometry such that the optimized charts reflect the reconstructed geometry. The optimization balances the following aspects:

- **Surface Smoothness:** The normals across each surface chart should change smoothly. This ensures that the reconstruction using the normals along the chart boundaries will approximate the optimized charts.
- **Approximation:** The optimized and hence the reconstructed surface should remain close to the original one.
- **Curve Smoothness:** The boundary curves should be smooth. This helps to regularize the curve network and thus the subsequent reconstruction result.

Normals being a highly non-linear function of the vertex positions, an optimization functional aiming to simultaneously satisfy the above goals would be challenging to minimize. Instead we use a solution that decouples the normals and the positions, splitting the solution into three steps: normal solve, per-triangle vertex positioning, and, finally, global assembly.

Normal Solve: We first compute new triangle normals which provide a tradeoff between smoothness and normal-level approximation of the envelope surface,

$$\min_{\{\mathbf{n}_i\}} \sum_i \|\mathbf{n}_i - \frac{1}{|N(i)|} \sum_{j \in N(i)} \mathbf{n}_j\|^2 + \omega_1 \sum_i \|\mathbf{n}_i - \mathbf{n}_i'\|^2 \tag{5.1}$$

where i indexes mesh triangles, $N(i)$ denotes the set of triangles adjacent to i that belong to the same chart, and $\mathbf{n}_i$ and $\mathbf{n}_i'$ are the new and current triangle normals, respectively. We set ω_1 to 0.25 for interior chart triangles, and to 0.1 for triangles adjacent to chart boundaries. We use a smaller weight for boundary triangles since their envelope normals often deviate from those of the input models and hence are less critical to preserve (Figure 5.1 top, center). Figure 5.2 compares the input envelope with the normal smoothed envelope, where the effect of normal smoothing is visualized through rendering each facet using the smoothed normal. Notice the new normals yield a much smoother shading.

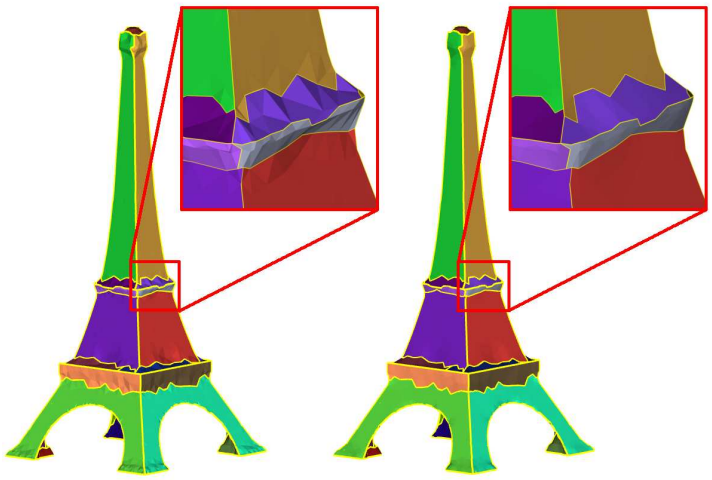


Figure 5.2: *Left: unsmoothed model. Right: the same model rendered using smoothed normals.*

Per-Triangle Solve: Next we compute the vertex positions that generate the desired normals while staying close to the original positions. The computation is performed on a per-triangle basis.

- For interior triangles we solve

$$\min_{\{\mathbf{v}_k\}} \sum_k \|\mathbf{v}_k - \mathbf{v}'_k\|^2 + \omega_2(\mathbf{n}_i\mathbf{v}_k + d_i)^2,$$

  where k indexes the three triangle vertices and d_i is the unknown distance component of the triangle's plane equation (normal form). The weight ω_2 is set to 1000, effectively ensuring that the new positions $\mathbf{v}_k$ satisfy the desired normal $\mathbf{n}_i$.

- For boundary triangles, we incorporate an additional boundary curve smoothness requirement. For every boundary edge $\mathbf{e}_b = \overrightarrow{\mathbf{v}_b^1\mathbf{v}_b^2}$ of the triangle, we add an additional component to the minimization:

$$\min_{\{\mathbf{v}_k\}} \sum_k \|\mathbf{v}_k - \mathbf{v}'_k\|^2 + \omega_2(\mathbf{n}_i\mathbf{v}_k + d_i)^2 + \omega_3 \sum_b \|\mathbf{e}_b - \mathbf{s}_b\|^2,$$

  where $\mathbf{s}_b$ is the smoothed boundary edge. More specifically, $\mathbf{s}_b$ has the same magnitude as $\mathbf{e}_b$ but its direction aligns with the local average boundary direction. Such average boundary direction is obtained by

tracing boundary edges to each side of $\mathbf{e}_b$, treating each edge as a vector whose direction is consistent with $\mathbf{e}_b$, and compute their weighted average. The tracing stops if either the midpoint of a boundary edge is further than r away from the midpoint of $\mathbf{e}_b$ or an intersection is reached where three or more charts meet at that point. The weight of an edge vector $\mathbf{e}$ is $G_{(0,0.5r)}(d)$, where $G_{(0,0.5r)}$ is a Gaussian distribution with mean 0 and standard deviation $0.5r$, and d is the distance from the midpoint of $\mathbf{e}$ to the midpoint of $\mathbf{e}_b$ along the chart boundaries. In this thesis, we chose r to be $1.5\times$ the average edge length of the mesh, and ω_3 is set to 2 in all our examples. Figure 5.3 shows the effect of boundary smoothing.

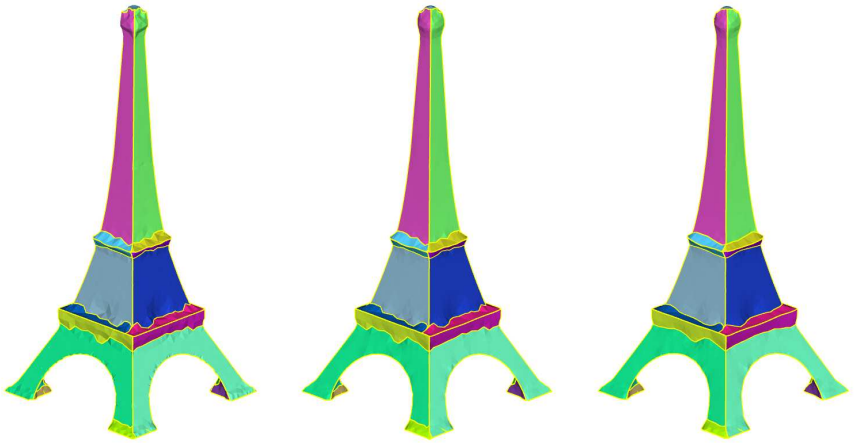


Figure 5.3: *Left: original partition. Middle: after one iteration of chart smoothing without boundary curve smoothing. Right: after one iteration of chart smoothing with boundary curve smoothing. Chart boundaries are indicated using yellow lines.*

As shown in Figure 5.4 (top and bottom middle images), the original mesh is disconnected due to per-triangle solve. Notice that when the original surface is smooth (along the sides of the tower), triangles approximately stay at their original position. When the surface is rough (near the tip of the tower), large gaps are created due to large normal changes.

Global Assembly: Finally, to obtain a connected mesh we reconcile the different per-triangle positions computed for each vertex. At this stage the shape and orientation of each individual triangle can be viewed as optimal, hence we aim for new vertex positions such that the per-triangle transforma-

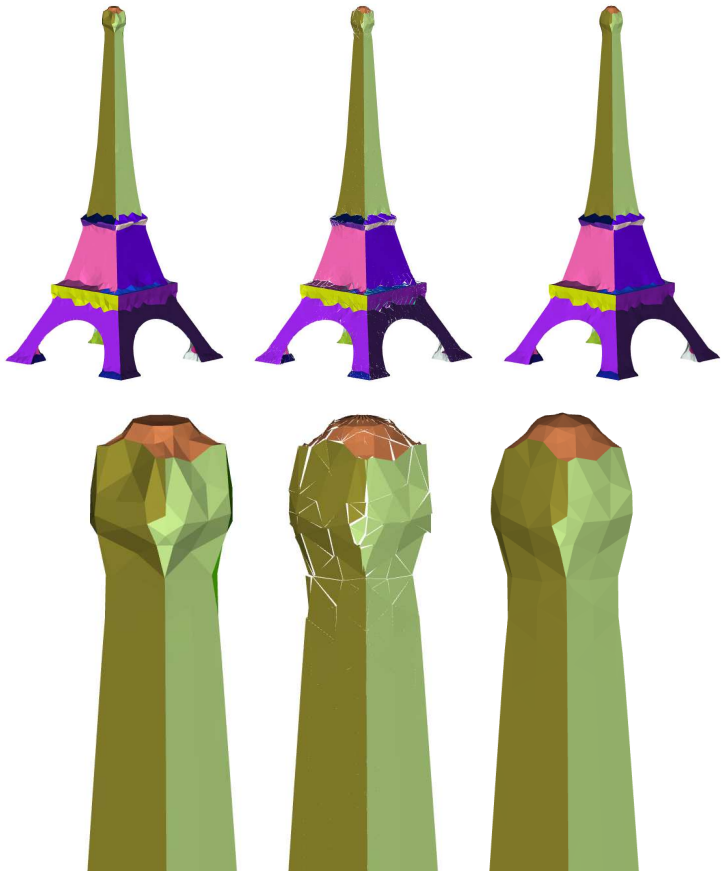


Figure 5.4: *Each chart is indicated by a unique color. Top: the tower model before chart smoothing (left), after per-triangle solve (center), and after global assembly (right). Bottom: Zoomed view of the top of the tower before chart smoothing (left), after per-triangle solve (center), and after global assembly (right).*

tion gradient is close to identity. We express this requirement using a similar formulation to [43]. For each triangle we define $V_k = [\mathbf{v}_3 - \mathbf{v}_0, \mathbf{v}_3 - \mathbf{v}_1, \mathbf{v}_3 - \mathbf{v}_2]$ where $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ are the three vertices of the triangle computed in the previous step and $\mathbf{v}_3 = \mathbf{v}_0 + \mathbf{n}_i$. We define W_k similarly but using the unknown shared vertex positions $\mathbf{w}$. We optimize over $\mathbf{w}$ using

$$\min_{\{\mathbf{w}_j\}} \sum_i ||W_i V_i'^{-1} - I||_F^2 + \omega_4 \sum_j \|\mathbf{w}_j - \mathbf{v}'_j\|^2$$

where F denotes the Frobenius norm, j indexes mesh vertices, and ω_4 set to 10. We iterate the steps until the positions and normals converge, typically in three to four iterations. After each iteration we perform topological cleanup, reapplying the segmentation post-processing step (Section 5.1).

Finally, we associate per-chart normals with curve vertices, using the average normal of the adjacent triangles around the vertex in each chart (see Figures 5.1 and 5.5).

5.3 Network regularization and simplification

The obtained curve network satisfies the abstraction and reconstruction criteria (Chapter 3). We now further regularize the network to capture structure, and then simplify it by removing appropriate curves, producing a sequence of abstractions.

Regularization. The vector representation we obtain at this stage is sufficient to faithfully reconstruct an approximation of the input. However, our goal is to create abstractions for man-made objects, highlighting regularity and structure, while ignoring minor deviations or inconsistencies. Hence, we regularize individual curves and loops of the network, and enhance their mutual relations.

Our *local regularization* replaces near-regular geometric features by perfectly regular ones, in an effort to retain the essence of shapes while ignoring small variations. The process considers both positions and normals across the network. First near-planar curves are made exactly planar using a least squares plane fit. Curves, once optimized, are kept fixed during the later iterations, as side constraints for least squares optimization. Subsequently, near-linear curves are converted to line segments. Similarly, curves that fit well to circular arc segments are regularized to perfect circular arcs. Such tests are incremental, where once a curve or a sequence of curves fit into one of these categories, we incrementally check if consecutive curves sharing the same boundary loop with the current sequence also fit the same plane, line, or arc. We use a similar incremental approach to detect sequences of curves with nearly identical normal values along a shared face loop, indicating a locally near planar face region and make the normals identical.

In man-made or engineered objects, symmetry and regularity play a dominant role due to aesthetic considerations, as well as convenience in design

and manufacturing. A good abstraction should capture such relations, making them explicit. In the *global regularization* step we enforce mutual relations between curve pairs that local processing may have missed. First, planar loop pairs that are nearly parallel (or orthogonal) are made exactly so. Such an approach clearly depends on the order in which the loops are processed. As a (heuristic) solution, we process the loop pairs in a greedy fashion (i.e., the loop-pair, which is closest to being parallel, is processed first), taking small steps towards making the loops parallel (or orthogonal), and iterate until convergence. Most of the examples presented in the paper converged in less than five rounds. Finally, we detect global reflective and (discrete) rotational symmetry in the input model [30]. When detected, we enforce symmetry in the curve network, minimally moving the curves to make them exactly symmetric using a symmetrization approach [31]. In the regularization steps, we only update positions and normals without changing connectivity or topology of the network.

Hierarchical Simplification. Our original method for network topology extraction is fairly conservative, often resulting in more charts than strictly necessary for reconstruction purposes. Hence it is often possible to simplify the network further without compromising the abstraction quality. Simplification can also be used to create higher levels of abstraction. To define the hierarchical simplification mechanism we reuse the smoothness and approximation criteria targeted during network geometry extraction.

We compute a per-curve simplification error measuring the impact of removing the curve from the network using two terms. We use integral dihedral angle along a curve to measure smoothness error. To quickly estimate the approximation or reconstruction error, we integrate the deviation of the average normal along the curve from the arc-length interpolation of the curve end-point normals at that point. This measure provides a rough estimate of the difference between the reconstructed surfaces with and without the curve in question. To control the simplification process we set separate thresholds on the two terms.

During simplification, rather than removing curves completely, we simply demote them to connectivity-only discarding all the geometric information associated with them (Figure 5.5). Leaving the network topology untouched, helps the reconstruction process while the overhead of storing connectivity-only curves is negligible. In order to maintain reconstruction consistency, once a curve is demoted, we update the curve network, re-assigning nor-

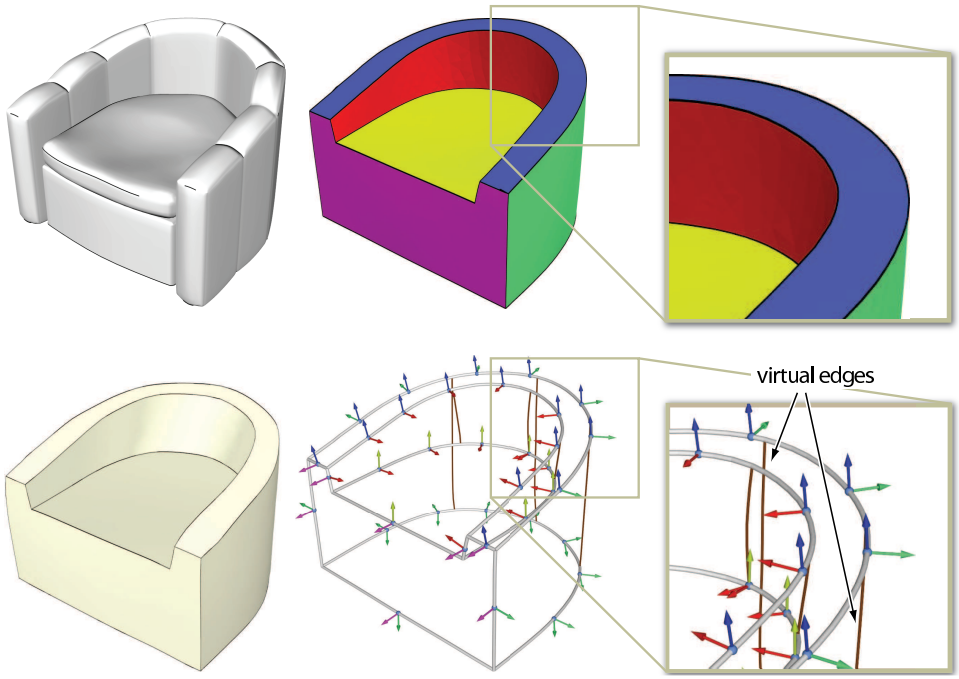


Figure 5.5: *Input model, processed segments, vectorized curve network, and reconstructed abstraction. Zoom panels show section of curve network, and normals along the curves. Connectivity-only or virtual edges are marked in brown. For ease of visualization, normals from same curve loops are marked in identical colors.*

mals along the affected edges using our chart optimization (see Section 5.2) and regularization procedures. Recomputing the surrounding geometry after each edge removal is expensive. However, the number of curves in our curve networks is small, allowing such expensive iterations. Thus we get multiple curve networks and associated reconstructions or abstractions of the input (see Figure 5.1).

Chapter 6

Reconstruction

The reconstruction process constructs the abstracted model shapes from the vector representation (Figure 3.1-right), establishing both the underlying mesh connectivity and the actual mesh geometry.

Initial Reconstruction. The reconstruction starts by triangulating the faces of the curve network, aiming to construct a connectivity that reflects the target geometry. It first embeds the boundary of each face in the plane, using Multi-Dimensional Scaling (MDS) [28] aiming to preserve the shape of the boundary. The MDS embedding best preserves the Euclidean distances between all pairs of vertices on the boundary, which in case of planar boundary loops results in preservation of the boundary shape and even for non-planar boundaries generates a reasonable embedding. As MDS allows for self intersections, we explicitly check for them. When an intersection is observed, we embed the loop on a circle using arc-length parameterization. Once a planar boundary is computed, its interior is triangulated [41], providing the desired connectivity reconstruction.

To generate the geometry of the abstracted model we deform the computed planar meshes, enforcing the boundary vertex positions and normals prescribed by the input vector representation. We use the shape preserving deformation method of Popa et al. [39] whose rotation propagation approach is well suited to our setting. The deformation is applied simultaneously to clusters of faces connected with connectivity-only network curves as we expect the normal impact to propagate across them.

Improvement. Although for many surfaces the initial reconstruction produces satisfactory results, it is not always the case. First, if the normal differences across a reconstructed face are fairly large, our deformation may "under-rotate" the triangles involved as it satisfies the normal constraints only in a least-squares sense. This problem is easily resolved by iterating

through the deformation step using the current surface as the undeformed model.

Second, if the initial embedding of the boundary differs significantly from the "natural" parameterization of the corresponding face, the mesh triangles can undergo significant deformation leading to visible artifacts. This is resolved by repeating the connectivity and geometry construction steps. Since we now have an interior triangulation, we use standard mesh parameterization methods to find the optimal planar domain, e.g ABF++ [40]. We then retriangulate the obtained boundary and repeat the deformation as above.

Chapter 7

Results

We tested our abstraction algorithm on a variety of man-made models mostly described by polygon soups. The abstracted models are generated using a combination of local processing with subsequent consolidation of global relations common in man-made forms. The abstracted shapes, having highlighted the mutual relations of parts or curves, are significantly more compact and lightweight compared to any low level representation. Here we take advantage of the observation that engineered forms are often defined by a few 1D curves, having relatively low information content compared to organic objects. The resultant models are extremely concise, while explicitly encoding information about relations between parts (see table). For the models shown in the paper, the average time taken for generating an abstraction was less than 2 minutes on a 3GHz machine with 2GB RAM, with the envelope surface creation phase taking the majority of computing time.

Figure 7.1 indicates how the abstraction result degrades with increasing noise and ambiguity in the data. Since the input models consist of many disconnected components, little is achieved by local smoothing. In contrast, the abstraction results, which enforce local and global regularization, are stable and capture the characteristic features.

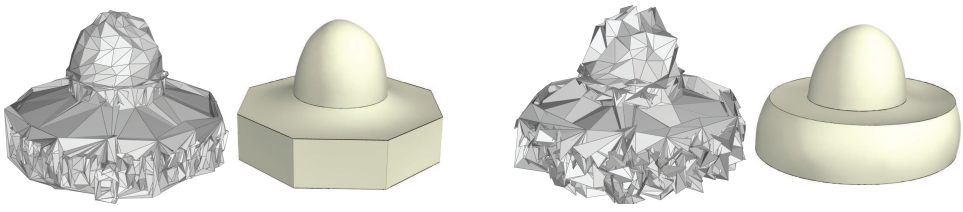


Figure 7.1: *Even for noisy input models, the regularization step allows creation of quality abstractions. We present abstractions with* 2% *and* 5% *(of bounding box diagonal length) noise added to the dome model.*

Model	# triangles	# components	level	#lines	# arcs	# curves
Empire State	16k	17	low	34	0	4
			high	146	2	4
Eiffel Tower	15.6k	2417	low	47	19	19
			high	85	31	24
building	3.7k	89	low	70	0	12
			high	288	2	9
Arc-Triomphe	13k	8	low	127	2	10
			high	163	16	14
Pagoda	37k	1173	low	64	0	1
			high	54	0	22
dome	3.8k	2	low	24	2	0
			high	116	23	6
rocking chair	32k	22	low	26	33	39
			high	55	35	32
baroque chair	164k	353	low	25	25	16
			high	20	33	42

Table 7.1: *Abstraction statistics.*

It has been hypothesized that abstract shapes are easier to recognize and classify compared to detailed ones [23]. Besides adding highlights for NPR applications, curves or feature lines are also used for detecting partial symmetries [9], for shape editing [17], etc. Such methods usually work on manifold, connected geometric models or rely on edges with sharp dihedral angles to identify feature edges, and thus cannot be readily applied to polygon soups or noisy models. Our abstraction procedure, which robustly generates a curve network capturing the high-level features of the model while ignoring finer details or deviations, provides an abstraction network that can used as auxiliary information by the above mentioned algorithms.

Limitations. Our method's ability to remove topological details is currently strongly linked to the resolution of the envelope. Once the envelope separates features like the fine diagonal bars in the water tower in Figure 7.2, they persist throughout the algorithm stages, independent of size. Topology level abstraction, removing such features is an important topic to address in future research.

On a more perceptual note, not all objects around us have well-defined, recognizable abstract representations. For instance, the Burj Al-Arab building in Figure 7.2 is not yet well-known enough for its abstract representation to

be recognizable.

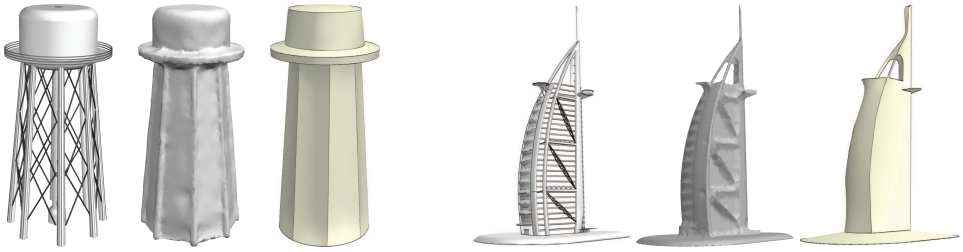


Figure 7.2: *(Left) Fine topological features are easily combined by our envelope construction stage. However, once such features are extracted by a finer grid resolution, we have no easy method to remove them, independent of their size. (Right) Some objects, perhaps those less familiar to us, have no obvious natural abstraction.*

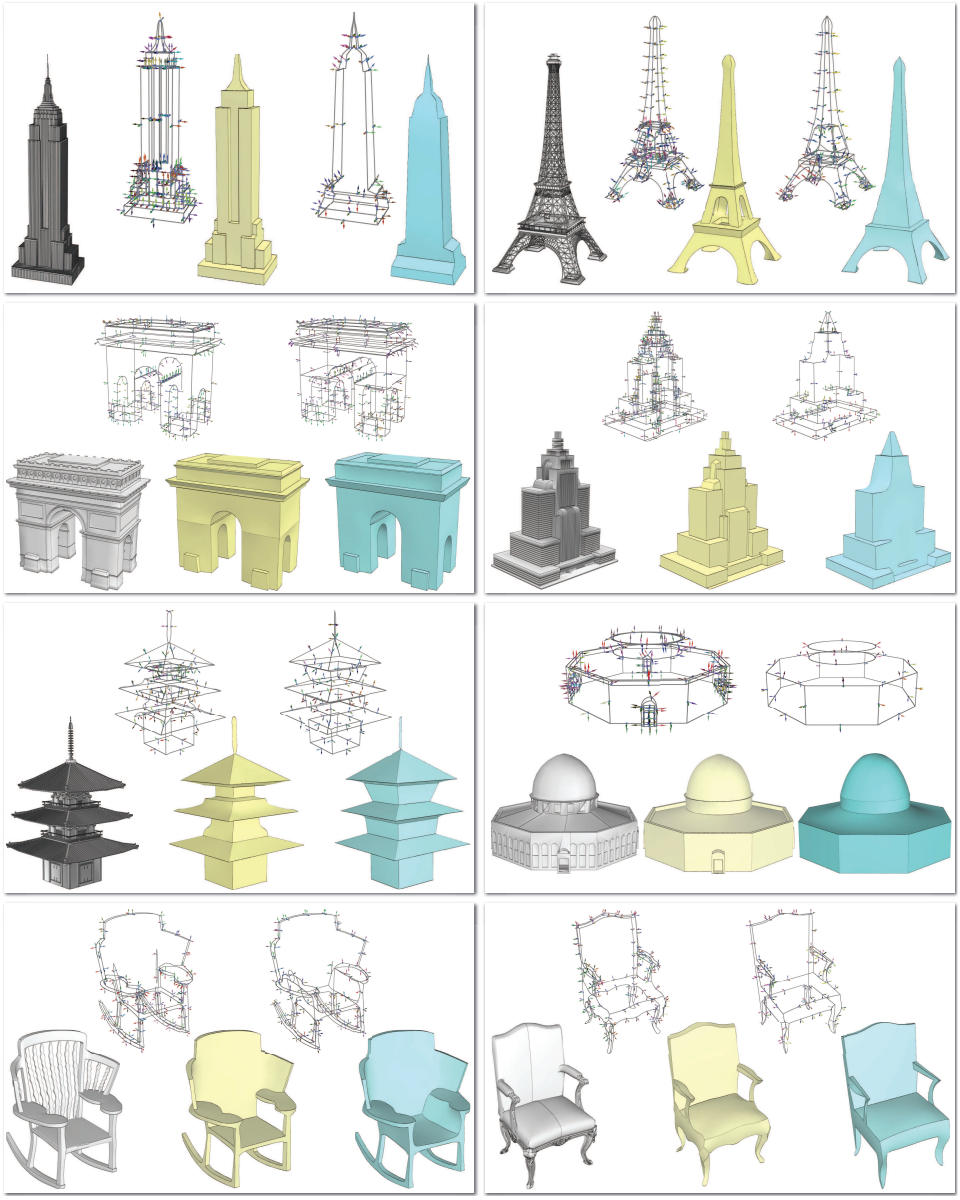


Figure 7.3: *Result gallery showing various input models, extracted curve networks (with normals), and reconstructed abstractions. The high-resolution abstractions are rendered in yellow, while the low-resolution ones are in blue.*

Chapter 8

Conclusion

We presented an algorithm for creating abstractions of 3D geometric shapes, specifically targeting man-made objects, using hierarchical curve networks that capture the defining characteristics of the inputs. Our method is robust, designed to handle models with disconnected components, self-intersections and noise, and uses a regularization step to make specific both individual and mutual curve relations. The curve networks, along with the associated normal information, can be used to reconstruct clean manifold meshes retaining the *essence* of the input forms. We believe that the extracted curve network provides a powerful handle to the input shape, since the representation contains specific and explicit global information about the shape of object parts and the relations between them.

Bibliography

[1] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10, 1987.

[2] Rudolf Arnheim. *Art and Visual Perception: A Psychology of the Creative Eye*. Faber and Faber, 1956.

[3] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22(3):181–193, 2006.

[4] Fred Attneave. Some informational aspects of visual perception. *Psychological review*, 61(3):183–193, 1954.

[5] A. Bengtsson and J.-O. Eklundh. Shape representation by multiscale contour approximation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(1):85–93, Jan 1991.

[6] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. *Computing in Euclidean geometry*, 1:23–90, 1992.

[7] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. PAMI*, 14(2):239–256, 1992.

[8] Stephan Bischoff, Darko Pavic, and Leif Kobbelt. Automatic restoration of polygon models. *ACM Trans. Graph.*, 24(4), 2005.

[9] Martin Bokeloh, Alexander Berner, Michael Wand, Hans-Peter Seidel, and Andreas Schilling. Symmetry detection using line features. *Computer Graphics Forum (Proc. EUROGRAPHICS)*, 28(2):697–706, 2009.

[10] John Canny. A computational approach to edge detection. PAMI-8(6):679–698, 1986.

[11] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Transactions on Graphics. Special issue for SIGGRAPH conference*, pages 905–914, 2004.

[12] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *ACM Trans. on Graphics*, pages #88, 1–11, 2008.

[13] Luciano Costa and Roberto Marcondes Cesar. *Shape Analysis and Classification: Theory and Practice*. CRC Press, 2001.

[14] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics*, 22(3):848–855, July 2003.

[15] R. Dyer, H. Zhang, and T. Möller. Delaunay mesh construction. In *ACM Symposium on Geometry Processing*, pages 271–282, 2007.

[16] B. Falcidieno and M. Spagnuolo. A shape abstraction paradigm for modelling geometry and semantics. pages 646–656, 1998.

[17] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. iwires: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics*, 28(3), 2009. to appear.

[18] Ran Gal, Olga Sorkine, Tiberiu Popa, Alla Sheffer, and Daniel Cohen-Or. 3d collage: expressive non-realistic modeling. In *NPAR '07: Proceedings of the 5th international symposium on non-photorealistic animation and rendering*, pages 7–14, New York, NY, USA, 2007. ACM.

[19] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH*, pages 209–216, 1997.

[20] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive technical illustration. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 31–38, New York, NY, USA, 1999. ACM.

[21] Floraine Grabler, Maneesh Agrawala, Robert W. Sumner, and Mark Pauly. Automatic generation of tourist maps. *ACM Trans. Graph.*, 27(3):1–11, 2008.

[22] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[23] Suyu Hou and Karthik Ramani. Structure-oriented contour representation and matching for engineering shapes. *Comput. Aided Des.*, 40(1):94–108, 2008.

[24] Tilke Judd, Frédo Durand, and Edward Adelson. Apparent ridges for line drawing. *ACM Trans. Graph.*, 26(3):19, 2007.

[25] Dan Julius, Vladislav Kraevoy, and Alla Sheffer. D-charts: Quasi-developable mesh segmentation, 2005.

[26] J. J. Koenderink and A. J. van Doorn. The internal representation of solid shape with respect to vision. *Journal Biological Cybernetics*, 32(4):211–216, 1979.

[27] Vladislav Kraevoy, Alla Sheffer, Ariel Shamir, and Daniel Cohen-Or. Non-homogeneous resizing of complex models. *ACM Trans. Graph.*, 27(5):1–9, 2008.

[28] J. B. Kruskal and M. Wish. Multidimensional scaling. *Sage University Paper series on Quantitative Application in the Social Sciences*, 07-011, 1978.

[29] Paul Merrell and Dinesh Manocha. Continuous model synthesis. *ACM Trans. Graph.*, 27(5):1–7, 2008.

[30] N. J. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3d geometry. In *ACM Transactions on Graphics*, volume 25, pages 560–568, 2006.

[31] N. J. Mitra, L. Guibas, and M. Pauly. Symmetrization. In *ACM Transactions on Graphics*, volume 26, pages #63, 1–8, 2007.

[32] Kyung Na, Moon Jung, Jongwan Lee, and Chang Geun Songa. Redeeming valleys and ridges for line-drawing. pages 327–228, 2005.

[33] L.R. Nackman and S.M. Pizer. Three-dimensional shape description using the symmetric axis transform. *IEEE PAMI*, 7(2):187–201, 1985.

[34] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph.*, 26(3):41, 2007.

[35] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.*, 23(3):609–612, 2004.

[36] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. In *ACM Trans. Graph.*, volume 27, 2008.

[37] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. Guibas. Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics*, 27(3):#43, 1–11, 2008.

[38] Tomaso Poggio, Vincent Torre, and Christof Koch. Computational vision and regularization theory. *Nature*, 317:314 – 319, 1985.

[39] Tiberiu Popa, Dan Julius, and Alla Sheffer. Material-aware mesh deformations. In *SMI*, page 22, 2006.

[40] Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, and Alexander Bogomyakov. Abf++: fast and robust angle based flattening. *ACM Trans. Graph.*, 24(2):311–330, 2005.

[41] Jonathan Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148, pages 203–222. Springer-Verlag, 1996.

[42] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Symposium on Geometry Processing*, pages 179–188, 2004.

[43] Robert W. Sumner and Jovan Popović. Deformation transfer for triangle meshes. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 399–405, 2004.

[44] Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In *Symposium on Geometry Processing*, pages 17–28, 2003.

[45] S. Toledo. Taucs: A library of sparse linear solvers, version 2.2, http://www.tau.ac.il/ stoledo/taucs, 2003. http://www.tau.ac.il/ stoledo/taucs.

[46] T. Várady and R. R. Martin. Reverse engineering. In G. Farin, J. Hoschek, and M. S. Kim, editors, *Handbook of Computer Aided Geometric Design*, pages 651–681. Springer, 2002.

[47] A.P. Witkin et al. Scale-space filtering, April 14 1987. US Patent 4,658,372.