DETAIL-REPLICATING SHAPE STRETCHING

by

Ibraheem A. Alhashim
B.Sc., Portland State University, 2008

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

© Ibraheem A. Alhashim 2011
SIMON FRASER UNIVERSITY
Spring 2011

All rights reserved. However, in accordance with the Copyright Act of Canada, this work may be reproduced without authorization under the conditions for Fair Dealing. Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.
APPROVAL

Name: Ibraheem A. Alhashim
Degree: Master of Science
Title of Thesis: Detail-Replicating Shape Stretching

Examining Committee: Dr. Torsten Möller
Chair

_____________________________________________
Dr. Hao Zhang, Senior Supervisor

_____________________________________________
Dr. Shahram Payandeh, Supervisor

_____________________________________________
Dr. Ligang Liu, Examiner,
Professor of Computing Science,
Zhejiang University

Date Approved: 11 February 2011
Declaration of Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website <www.lib.sfu.ca> at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada
Abstract

Mesh deformation methods are useful for creating shape variations. Existing deformation techniques work on preserving surface details under bending and twisting operations. Stretching different parts of a shape is also a useful operation for generating shape variations. Under stretching, texture-like geometric details should not be preserved but rather replicated. We propose a simple method that help create model variation by applying non-uniform stretching on 3D models. The method replicates the geometric details and synthesizes extensions by adopting texture synthesis techniques on surface details. We work on analyzing and separating the stretching of surface details from the stretching of the base mesh enabling the appearance of preserved details. The efficiency of our method is a result of defining a local parametrization with the help of curve skeletons. We show different results that demonstrate the usefulness of this intuitive and efficient stretching tool in creating shape variations.

Keywords: stretching; detail replication; detail-preserving deformation; geometry synthesis; mesh editing
To my parents and my wonderful siblings
“There is no capital more useful than intellect and wisdom, and there is no indigence more injurious than ignorance and unawareness.”

— Ali ibn Abi Talib, 660 CE
Acknowledgments

I would like to thank all the people whose support and encouragement have helped me reach this point. First, I would like to thank Dr. Hao Zhang, my senior supervisor, for his support and guidance during my research. Secondly, I would like to thank my sponsor for the opportunity to pursue research in my field of interest. I would also like to take this opportunity to thank my fellow students at the GRaphics, Usability, and VIsualization (GrUVi) lab for their help and valuable discussions and feedback especially Andrea Tagliasacchi, Oliver van Kaick and Yanzhen Wang. A special thank to my wonderful international friends that I have made throughout my stay at the Simon Fraser University. Finally I would like to thank my parents for their patience and support and my brother and sisters for their moral support and their love.
Contents

Approval ii
Abstract iii
Dedication iv
Quotation v
Acknowledgments vi
Contents vii
List of Tables x
List of Figures xi
Preface xiii

1 Introduction 1
  1.1 Overview of our approach 3
  1.2 Motivation and applications 3
  1.3 Contributions 7
  1.4 Thesis organization 7

2 Related Work 9
  2.1 2D detail preservation by replication 9
  2.2 Procedural modeling 11
  2.3 Shape deformation 11
## Table of Contents

2.4 Anisotropic resizing .................................................. 13  
2.5 Cut-and-paste ......................................................... 14  
2.6 Geometric texture synthesis .......................................... 15  

3 Background ................................................................. 17  
3.1 Curve skeletons ....................................................... 17  
3.2 Parametrization ......................................................... 18  
3.3 Detail representation ................................................ 19  
3.4 Texture synthesis ..................................................... 21  
3.5 Surface blending and hole filling .................................. 22  

4 Detail-replicating stretching .......................................... 25  
4.1 Base mesh .............................................................. 26  
4.2 Parametrization ......................................................... 28  
4.2.1 Cylindrical grid .................................................. 28  
4.2.2 Geometric texture representation ............................ 31  
4.3 Synthesis ............................................................... 33  
4.3.1 Texture synthesis ................................................ 33  
4.3.2 Geometry reconstruction .................................... 34  
4.4 Stitching patches ..................................................... 35  
4.5 Adding variation ...................................................... 37  

5 Implementation ............................................................. 39  

6 Results ....................................................................... 43  
6.1 Parameters ............................................................ 43  
6.2 Editing examples ..................................................... 44  
6.3 Limitations ............................................................ 45  

7 Conclusion and future work .......................................... 55  
7.1 Conclusion ............................................................ 55  
7.2 Future work .......................................................... 55  

A Mesh Smoothing ............................................................ 57
List of Tables

6.1 Statistics for our editing experiments. ........................................... 45
# List of Figures

1.1 Surface texture representation ................................................. 2  
1.2 Organic surface texture .......................................................... 3  
1.3 Overview of our detail-replicating stretching pipeline ....................... 4  
1.4 Extending the Stanford dragon .................................................. 5  
1.5 Examples of real objects containing essential surface details. .............. 6  
1.6 Stretching a snake model ........................................................ 7  

2.1 Image deformations with detail preservation .................................. 10  
2.2 Examples of resized images with different detected lattices using the summa-  
    rization method described in [66] ................................................. 10  
2.3 Procedural modeling .................................................................. 11  
2.4 Shape deformation ................................................................... 13  
2.5 Examples of anisotropic scaling [15] .......................................... 14  
2.6 Cut-and-paste editing .................................................................. 15  
2.7 Geometric texture synthesis ....................................................... 16  

3.1 Skeleton extraction: extracting a 1D skeletal shape by performing geometry  
    contraction using constrained Laplacian smoothing [4] ..................... 18  
3.2 Mesh parametrization ............................................................... 19  
3.3 Detail representation .................................................................. 20  
3.4 Texture synthesis ....................................................................... 22  
3.5 Patch stitching ........................................................................ 24  

4.1 Implicit fairing ........................................................................ 26  
4.2 Multi-layered surface texture pose a challenge when determining the smooth-  
    ing required for the base shape ....................................................... 28
Preface

I am interested in topics relating to modeling and shape generation. While researching different problems in mesh editing, I came across work done by Gal et al. [26] titled iWIRES that allowed for intuitive 3D shape edition operations. Their method performs well on man-made objects such as furniture, cars, etc. In their paper’s limitation section they state that organic shapes may not be easily manipulated by the set of wires defined in their method.

For operations such as bending and twisting, state of the art deformation methods preserve surface details on organic shapes very well. However, when applying non-uniform scaling it is impossible to preserve the shape and size of the details on the surface. This challenging problem led me to think about different approaches to the problem of resizing organic objects such as creatures or decorated furniture that contain complex surface details and variable curvature. As a first step, I explored the different approaches in the 2D image domain to identify the different techniques used. I came across the work done by Fang and Hart [22], in which their method seemed to resemble the non-uniform scaling problem in that the shapes they deal with are composed of several complex detailed parts. Applying these ideas from 2D images to 3D shapes turned out to be non-trivial. Images tend to have large amounts of high frequency details making it easier to perform blending operations with smaller building blocks (i.e. pixels) to achieve different stretching operations. Also, the added degree of freedom in 3D increases the complexity of the problem.

The challenge then was finding a method that allows a modeler to perform non-uniform scaling or stretching of certain mesh parts with minimal user interaction. The work done in this thesis serves as a step in that direction with some promising results. However, many challenges still remain for a robust and general tool that can perform visually pleasing non-uniform scaling of geometrically textured models.
Chapter 1

Introduction

Surface details are essential in generating realistic 3D models. The simplest form of surface detail representation is texture maps that can be composed of several images. Extending to parallax mapping adds an extra level of realism and are often used in real-time rendering systems for their efficient simulation of surface details. These texture representations, however, lack the capability to represent more complex surface details such as thorns, scales or bark that require more geometric primitives. In applications such as simulation of friction or shadow casting across multiple objects, a geometric representation of these details is required to achieve realistic results. Figure 1.1 shows a visual comparison of these different surface detail representations.

Editing meshes with existing complex surface details is a challenging task. State of the art mesh deformation methods work on preserving surface details under deformation operations such as small rotations and translation. Non-uniform scaling or stretching operations cause extreme distortion to these details limiting the amount of deformation possible for modelers when creating variation on existing models such as elongating a snake or resizing a patterned vase. Applying current shape manipulating scaling on organic models and meshes obtained with laser scanners often results in visual artifacts and loss of surface properties (Figure 1.2, middle). Another difficulty in dealing with organic models arises when the details are composed of several patterns or layers that make it challenging to identify and separate (Figure 1.2, right). The process of applying stretching to detailed meshes is then a time consuming process that involves the modeler introducing geometry to the model and synthesizing similar details. Being able to efficiently produce shape variations from exciting shapes by stretching would allow a modeler to populate a scene more quickly.
Deformations in the image domain have been widely researched [6, 7, 21, 22, 26, 32, 54]. A property of images that is often exploited by 2D deformation methods is that they are made up of large areas of similar pixels. When a deformation operation, such as stretching, is applied it results in a hole or stretching of a region which is then treated locally. These regions are often filled or partially replaced with information from their surroundings. Another property of images is that the amount of deformation applied is constrained by 2D transformations that is within a manageable problem space (i.e. a plane). For 3D shapes, deformations become more challenging with: the added degree of freedom, not being sampled over a regular domain, and the sparsity of geometric primitives that contribute to the surface detail. If a stretching operation is applied to extend certain parts of the mesh, it deforms the original shape of any surface detail or pattern in that part. Most of the current deformation methods are able to perform bending deformations that respect the surface area of the original shape. However, stretching a part beyond its volume would unequivocally deform the surface details. Resizing methods such as the one presented by [35] preserves the original shape of some details on the expense of largely deforming smooth parts of the shape. This, however, is not applicable for details that are known to naturally correspond
CHAPTER 1. INTRODUCTION

Figure 1.2: Organic surface texture: (left) input. (middle) Applying a FFD stretch with four control points results in interpolated details. (right) An example of multi-layered surface details (having both small and large scale details).

to a certain scale with regard to other parts of the shape such as scales on a fish. One way to deal with these challenges is to identify a set of constraints that help reduce the problem space for 3D shape stretching to that of the image domain.

1.1 Overview of our approach

In our algorithm, we perform stretching via replication following the pipeline shown in Figure 1.3. A preprocessing step that computes the curve skeleton and a smoothed version of the input is needed once per shape. Our method starts by first extracting an approximation of the surface details from a user specified part of the model. Our detail extraction is similar to recent methods [2, 15] that defines details as displacements on the surface. We compute vertex displacements of the detailed mesh from a grid constructed using the smoothed base mesh. The region is parametrized by the grid reflecting the shape of the base. Details are then represented as a 2D image approximating a scalar height field of the details. Next, we use a texture synthesis algorithm, similar to patch-based texture synthesis, to replicate the details along the user specified direction of extension. Finally, we reconstruct and blend together the different patches corresponding to the synthesized 2D extension. Figure 1.4 shows a visual overview of our proposed pipeline for detail-replicating stretching of 3D shapes.

1.2 Motivation and applications

The ability to extend certain parts of a mesh allows for a more powerful editing tool. The idea of part resizing is primitive to designers and is often employed to add variation to
CHAPTER 1. INTRODUCTION

existing shapes. This task is possible if the building blocks of the shape are easily generated from simple building rules or grammar. This is the case in most architectural design and other CAD computer-aided design (CAD) software where the dimensions of a shape are easily modified by some parameters. However, this nice property is not available in triangular mesh based modeling frameworks.

When working on triangular meshes, a modeler might decide to adjust the length of a certain part of the shape. However, they are unfortunately restricted to the fixed triangulation of that mesh. This is the case for models designed using primitive polygon editing operations or in models scanned from real objects. The only options they have is either apply a local free form deformation (FFD) and risk distorting any surface details, or manually cut and paste different parts from the mesh that can be time consuming, tedious, and produces visible artifacts.

The objective is then to develop a tool that gives the modeler the ability to adjust the size of a detailed part of the mesh later on in the modeling process. This allows for an additional space of creativity to modify parts of the model either to adjust a shape to conform with a modified specification or to create varieties of the original model.

Such a tool can be useful in a number of different applications. Being able to create varying shapes from one source model in a matter of seconds can help reduce production times. If a virtual set designer or video game level editor needs to populate a scene with different objects of a similar class with some variety then scaling is an available operation to accomplish that. Given an input snake model, an intuitive operation to generate another snake is to elongate its body. Another example is a cave scene consisting of many speleothem columns (cave formations) where a handful of source shapes can be used to create a large number of columns with varying sizes, thus reducing the time required to build such scenes from scratch. Another situation where partial scaling is useful is fitting a model with other
Figure 1.4: Given the input shape, the user selects two points to specify the region to be extended (a). A base shape is then computed and the triangles corresponding to that area are then selected (b). The parameterization grid is then constructed and an approximating image of the surface details is computed (c). The user then specifies the intended extension by simply sketching a 2D curve in the screen space which is then projected along the curve skeleton of the part (d). The extension is synthesized in the parameter domain and the corresponding geometry is reconstructed following the extension curve (e). In (f) the result of the extension is shown from a different view.
models of different sizes. One example is using models from different sources that do not match in height or length such as Lego blocks, screws, building columns, venting hoses, roof shingles, patterned floors, etc (Figure 1.5). Such models contain surface details that should be respected according to aesthetics or physical constraints (e.g. screws follow a set of standards called the ISO metric screw thread).

Our method at this point is only able to handle 1D stretching along the shape’s curve skeleton. When considering 2D stretching of a roof for example, the process needs to be applied for each direction of the stretch. However, the extracted skeleton after applying a stretching operation in one direction will remain the same. Perhaps a better solution for 2D stretching is to identify skeleton sheets instead of 1D curve skeletons which can be considered in future work.

Figure 1.5: Examples of real objects containing essential surface details.
1.3 Contributions

In this work, we present an efficient algorithm for applying 1D stretching of mesh parts with minimal user interaction and aesthetically pleasing results. We encode the surface details of the input as a 2D image, therefore reducing the problem space to synthesis on a 2D plane. We also constrain the synthesis process such that the results blend well with the original mesh while minimizing distortions to the original surface details. The generated surface details on the extended area matches the frequency, scale, and topology of the source area. We show that separating surface details into different levels can help when dealing with complex or multi-layered details. The efficiency of our method allows the user to interactively test and modify different synthesis parameters on large and highly detailed meshes.

Our method performs detail-replication when applying a mesh stretching operation. We preserve the surface details with the help of 2D texture synthesis that guides the geometry replication process. The added extension is then connected to the source mesh while minimizing the stitching artifacts that result from the extension operation. Figure 1.6 shows an example of a stretching operation made using our method and compares it with output generated using current deformation methods (that would distort surface details).

1.4 Thesis organization

The remaining part of this work is organized as follows. In chapter 2, we survey some of the related editing techniques used in the past for generating shape variety and compare their
ideas to our method. Chapter 3 describes the concepts used in our algorithm and we will
discuss some details and the role of each technique in our work. In chapter 4 we present
our stretching algorithm in full detail including the process of texture representation, surface
replication, and the stitching method. A discussion about some of the implementation issues
is presented in chapter 5. Experimental results are then provided in chapter 6, and finally
in chapter 7 we conclude and discuss the limitations of the current method and present the
reader with some ideas for possible future work.
Chapter 2

Related Work

In this chapter we survey editing techniques used in the past for generating shape variety and compare some of their ideas to our method. We are also interested in other methods that address the issues of dealing with shapes containing high details.

2.1 2D detail preservation by replication

In the image domain, Fang and Hart [22] proposed re-synthesizing texture patches from the source image to preserve details around modified feature curves. This allows for free form deformations of a user defined feature curve on the image while preserving the frequency of the details. By replicating different source patches in accordance to the feature curve the appearance of stretched patterns caused by moving and/or bending the curve is reduced. Barnes et al. [6] introduced the PatchMatch method which allows the user to apply stretching and widened operations on an image while preserving the details in a similar way to [22]. The method produces high quality outputs due to the process used to randomly find good patch matches that preserve image coherence. Examples of deformations using these methods are shown in Figure 2.1.

However, the reliance on existing background details and the ability to blend pixel colors makes such methods non-trivial in the context of 3D shape deformations. In our 3D extension of the replication idea, the details are constrained by the boundaries of the mesh’s part, therefore, the idea of a background is analogous to a single colored background in an image. Perhaps more challenging to represent in 2D are inner details containing geometry of non-zero genus. Our method approximates surface details in a way that can help resolve problematic
Figure 2.1: Image deformations with detail preservation: (left) user specified feature curves are manipulated and a synthesis process described in [22] preserves texture frequency and orientation. (right) Detail preserving nonuniform scaling is applied to a user marked polygon using PatchMatch [6].

Figure 2.2: Examples of resized images with different detected lattices using the summarization method described in [66]

instances such as small handles or complex surface protrusions.

Wu et al. [66] introduced a method of resizing 2D images containing symmetric patterns by separating the process into two methods. First, they detect regions with translational symmetry and segment the image into symmetric and non-symmetric regions. Next, they resize the different regions using different techniques and then merge the results. When resizing the image, symmetric regions are resized using a summarization algorithm that removes or replicates cells from the extracted lattices in symmetric regions. The non-symmetric regions are resized using optimized wrapping and the two results are seamlessly merged, using graph cuts, to produce the final image. Figure 2.2 shows two examples of resizing images containing symmetric patterns with aesthetically pleasing results (i.e. no parts are distorted and no seams are visible). However, the method only works on symmetric patterns that can be identified using their symmetry detection algorithm (which is not always the case).
Figure 2.3: Procedural modeling: (left) an entire detailed city generated with 190 shape rules [49]. (right) The source shape, shown in red, is analyzed for symmetries and a set of rules are generated that allow for user defined and random constructions.

2.2 Procedural modeling

Designing objects using procedural modeling or CAD software allows us to incorporate surface details easily by adding rules that correspond to the geometric details as shown by [50, 49] (Figure 2.3, left). Bokeloh et al. [12] introduced an inverse procedural modeling framework that takes a piece of exemplar 3D geometry and extracts shape generation rules (Figure 2.3, right). Their method relies on finding symmetries by aligning salient feature lines on the input. This approach works well on shapes with highly symmetrical and distinct features. However, when considering organic shapes this approach might not be suitable since we may deal with small or continuously varying surface details and patterns. Furthermore, the extracted rules do not allow the freedom to merge different pieces having largely different orientations. Hence, procedural modeling is not general enough for describing organic shapes or applying free form edits. Our proposed method tries to deal with high frequency geometric details which is a hard problem in the context of inverse procedural modeling.

2.3 Shape deformation

In recent work, proposed 3D deformation methods focus on protecting large scale features of the shape. A survey by Botsch and Sorkine [13] discussed different linear deformation methods that preserve details under bending and twisting deformations. Sorkine et al. [60] presented a method that apply shape deformation operations on an intrinsic surface representation that encodes each vertex by its relative neighborhood based on the Laplacian of the mesh. A user can then define a region of interest for editing and start to manipulate
the shape. The surface of the area affected by that editing operation is then reconstructed in such a way that the original details of the shape are preserved as much as possible (Figure 2.4a).

Botsch et al. [14] presented a method that emulates physical surface behavior where the surface of the mesh is embedded in a layer of volumetric prisms. The prisms are connected by non-linear elastic forces and during deformation they are transformed to satisfy the user's editing operation while minimizing the elastic energy. Such deformation methods also preserve surface details under bending and twisting operations (Figure 2.4b).

However, these methods would ignore any form of stretching or resizing of parts but they can incorporate such operations by allowing interpolation similar to the techniques used in [30, 18] (Figure 2.4c). Unfortunately, with interpolation surface details will be distorted regardless of the technique being used.

Another approach by Gal et al. [26] extracts a descriptive set of wires and their relations allowing for intuitive resizing while preserving major shape characteristics (Figure 2.4d). The wires allow for more intuitive editing operations analogous to real life armatures used in sculpture. The method work well on shapes with smooth surfaces but would interpolate complex surface details according to the resizing operation. Still the idea of using wires or other shape descriptors such as a medial axis or a curve skeleton allows for the possibility of an editing framework that handles stretching and at the same time be able to deal with surface details using a separate process. Our work in this thesis explores such an opportunity as explained in more details in chapter 4.

A non-homogeneous resizing method presented by Kraevoy et al. [35] protects some model features, particularly the distinct ones, during resizing based on a vulnerability map (Figure 2.4e). The vulnerability computation is based on a per-face metric that combines slippage (a measure of surface persistence under a transformation [27]) with normal curvature. The model is then embedded into a protective volumetric grid and grid-based space-deformations are applied during resizing. The contribution of each cell in the grid to the scaling transformation is then computed and the final transformations are carried back to the model by interpolation. The method allows shape-aware scaling of the entire model while partial scaling operations are also possible by adding hard constrains to a group of surrounding cells. This method is analogous to image retargeting methods [64] that work on preserving salient features rather than replicating them. Again, surface details will either be distorted by interpolation or can preserved in their original form on the expense of stretching other
2.4 Anisotropic resizing

Chen and Meng [15] proposed a method that performs anisotropic resizing on meshes with surface details. They extend the grid idea from Kraevoy et al. [35] to incorporate geometric textures by separating them from the underlying surfaces and reproducing them on the scaled surfaces using texture synthesis. Figure 2.5 shows some examples of their scaling method. Their method extracts the texture after a segmentation process which would not be suitable in shapes with non-homogeneous surface details as different patterns belong to different segments. Furthermore, the output of their geometric texture is sensitive to the mesh density, thus limiting the reproduction of the details when models have low triangle regions in the model.
Our method is similar in that we replicate the surface details to compensate for the extended area introduced by the stretching or scaling operation. Advantages of our approach over [15] can be seen in: dealing with low density triangular models as well as large scale meshes, working with non-homogeneous surface details, dealing with multi-layered surface details (i.e. large and small scale details of the surface), and the intuitive user interface that allows for free form stretching operations. Our experimental results show that our method is significantly faster, able to deal with more types of details, and produces higher quality outputs.

2.5 Cut-and-paste

Cut-and-paste methods are used to combine different parts of different models to generate new shapes. Biermann et al. [10] described a number of algorithms based on multi-resolution subdivision surfaces that achieve cut-and-paste edits at interactive rates (Figure 2.6a). One limitation of their method is that it only works with regions that are homeomorphic to a disk. Fu et al. [25] proposed a topology-free cut-and-paste editing method that deals with regions of non-zero genus while minimizing any possible distortions caused by incompatible geometry between source and target (Figure 2.6b). Their described surface parametrization is used to transfer the details onto the target surface by identifying a shared simple planer base surface. Sharf et al. [56] introduced another cut-and-paste tool that allows the user to drag one mesh part onto another with some overlap and the system would snap them together using their proposed Soft-ICP algorithm. Also, recent work done by Schmidt and Singh [55] explored shape reuse and composition in 3D mesh modeling by combining ideas...
CHAPTER 2. RELATED WORK

Figure 2.6: Cut-and-paste editing: (a) a selected feature is extracted from a base surface and pasted onto a target region. (b) a feature with non-zero genus is pasted onto a target mesh (Stanford bunny) using [25]. (c) combining different meshes using the meshmixer tool [55].

similar to cut-and-paste along with other techniques (Figure 2.6c).

Our proposed method is related to cut-and-paste in the sense that we automatically cut patches of the surface from the same model and paste them coherently in order to replicate the surface detail. The pasted elements cover the introduced surface area during the stretching of the mesh’s part. An advantage of our approach over cut-and-paste is that the extended part is seamlessly merged with the source mesh allowing for a more natural look.

2.6 Geometric texture synthesis

Bhat et al. [9] presented an example-based technique for synthesizing geometric details based on ideas from texture synthesis in the image domain. The method works on volumetric models and is computationally expensive. It also requires training of the specific detail along with user defined vector fields on the surface in the source and target model. 2.7a shows two examples of this synthesis process. Lagae et al. [37] presented a similar method that works on polygonal mesh elements inside a grid. The method represent the input by a hierarchical distance field allowing for more efficient matching between target and input field. The synthesis process is then performed using a multi-resolution approach. This method is still computationally expensive and suffers from noticeable artifacts caused by the simple reconstruction process used between neighboring synthesized blocks of elements.
CHAPTER 2. RELATED WORK

Figure 2.7: Geometric texture synthesis: (a) a volumetric example-based detail synthesis technique on the bunny model with the given details [9]. (b) synthesizing geometry from a given example by working directly on the geometry introduces artifacts when combining different elements [37]. (c) synthesizing complex geometry given a patch of the detail and (d) shows synthesis over curved surfaces using a low-distortion multi-chart texture atlas [69].

(Figure 2.7b). Zhou et al. [69] mesh quilting paper is another example of applying 2D texture synthesis analogies (Efros and Freeman [20]) into the 3D domain (Figure 2.7c). They also extend their algorithm to covering curved surfaces in 3D by using local surface parametrization (Figure 2.7d).

Most of these methods are designed to produce new shapes with added details rather than editing existing ones. One shared limitation in previous work done on geometric texture synthesis is the computational complexity. All of the previously mentioned methods would require at least several hours to produce a single synthesized model of average complexity making them unsuitable for interactive editing. An advantage of our method is the ability to produce the synthesized regions in a matter of seconds allowing the user to interactively modify and experiment with different stretching operations.
Chapter 3

Background

In this chapter we present a general overview of the different concepts used in our pipeline (Figure 1.3) and discuss the role of each one. For our texture representation we will use an approximate parametrization of the mesh with the help of an extracted curve skeleton (computed during the preprocessing stage). For the synthesis stage, an efficient patch-based texture synthesis method is used to generate the extensions in the parameter domain. The final stage reconstructs the patches and combines them by filling the region in between with appropriate geometry.

3.1 Curve skeletons

Curve-skeletons are 1D structures that approximate the topology and shape of a 3D object. These shape descriptors have many applications such as shape registration, retrieval, and deformation. By representing a complex shape in these abstract curves we are able to reduce the complexity in such applications. A survey by Cornea et al. [17] classifies different methods and analyze the advantages and limitations of each class. A geometric method introduced by Au et al. [4] uses the idea of mesh contraction in which the object is smoothed with different constraints resulting in a thinning of the original shape (3.1). The resulting curve skeleton have the advantage of being clean, topology-preserving and geometry-aware. Another notable volumetric method that can compute curve skeletons on-the-fly was presented by Sharf et al. [57].

Giesen et al. [28] introduced an extension of the medial axis transform that computes a hierarchy of simplified skeletons. Their method offers a promising approach of defining curve
skeletons at different levels of abstraction. Another nice feature is the requirement of only one parameter representing feature scale rather than the different weight constraints as in [4]. User sketched skeletons can also be used to minimize this preprocessing step as in the shape deformation framework presented by Blanco and Oliveira [11].

In our implementation we utilize extracted curve skeletons using the method in [4]. In the extraction process we specify parameters such that the resulting skeleton is smooth and nicely embedded inside the shape. We use these skeletons when constructing our approximate surface parametrization. The skeleton also simplifies the process of selecting mesh regions in which the user intend to apply stretching by using its segmentation.

3.2 Parametrization

Mesh parametrization is widely used in applications such as texture mapping, re-meshing, deformation, morphing, among other applications. When given a mesh and another surface with similar topology, there exists a bijective mapping between them which defines a parametrization (Figure 3.2a). A couple of surveys on mesh parametrization [24, 31] provide detailed information on the different techniques along with discussions on what method works best for what application.

State of the art non-linear parametrization techniques compute natural boundaries with minimal distortion. Zayer et al. [68] proposed a linear formulation of such methods based on the Angle Based Flattening (ABF) introduced by Sheffer and de Sturler [58]. Their method can efficiently compute parametrization of complex meshes onto a plane (Figure 3.2b). However, several cuts on the original mesh are needed in the process resulting in discontinuity and irregular borders. Furthermore, the result of the parametrization is dependent on the start point of the flattening process. Finding the best appropriate cuts in order to parametrize a mesh of high genus is therefore a challenging task. In our method we use an approximate
parametrization that helps us avoid these issues of parameterizing shapes of high genus. A
different non-linear mesh parametrization method presented by Tarini et al. [62] offers a nice
parametrization alternative that works on projecting the original shape onto a representa-
tive poly-cube (Figure 3.2c). Our method also works by projecting the original shape onto
a simpler parameter mesh.

In this work we focus on extending details along the curve skeleton. Vertices of a curve
skeletons are generally wrapped inside a sphere like area of the shape while edges cover
a tubular region. Therefore, we are interested in mapping tubular or cylindrical parts of
the mesh onto a similar curvilinear grid. This grid construct approximates the surface of
the mesh to a simple plain and is efficient to compute when given the curve skeleton. The
construction of our parametrization domain by sweeping is described in section 4.2.1. The
mapping between our constructed grid is then bijective with respect to a 2D image that
wraps around at one of the axis.

3.3 Detail representation

For the representation of surface details on triangular meshes, a large number of representa-
tions build on the concept of a height field on a smooth base. Lee et al. [38] introduced the
displaced subdivision surface in which details are represented as a scalar-valued displacement
over a smooth domain surface allowing for a compact representation beneficial for geometry
compression and editing. An initial base mesh is obtained by simplification and then de-
formed to accurately fit the original mesh. A displacement map is then sampled by shooting
CHAPTER 3. BACKGROUND

Figure 3.3: Detail representation: (a) a coarse control mesh is extracted from the input shape (middle) and the reconstruction process performs subdivision and apply the computed displacements (black lines). (b) the coating of the bunny is transferred onto the leg model using Laplacian detail encoding. (c) the displacement of a vertex can be described in terms of a displacement length from the base (red) and the rotation from the normal.

rays along the surface of the base and the intersection is recorded as the required signed displacement (Figure 3.3a).

Another approach is to define surface details in the gradient domain. A representation by Sorkine et al. [60] captures the intrinsic geometry of the surface based on the Laplacian of the mesh by encoding each vertex relative to its neighborhood. They formulate the Laplacian coordinates in a way that makes them invariant to rotation and twisting. They demonstrated several applications using their representation such as free-form deformations, detail transfer (Figure 3.3b), and transplanting surface meshes. Andersen et al. [2] proposed another intrinsic representation that extends height fields by incorporating displacements in the tangential plane as normal tilts (Figure 3.3c). This offers a compromise between the compactness of height fields and the generality of complex 3D textures. They also demonstrated the usage of their representation in texture transfer, editing, and animation. Generally, intrinsic representations are preferable as they preserve local geometry the most during deformations. However, they are computationally more involved and require additional cuts to be parametrized on an image for synthesis.

In this work we use a signed displacement from our approximating grid to generate an image representation of the surface details. The geometric representation is similar to [2] as we compute a scalar displacement and a shift factor from a local reference frame. The shift factor allows us to deal with more complex surface details which are not possible with simple height fields.
3.4 Texture synthesis

Example-based texture synthesis is a widely studied topic in computer graphics. Given an input example texture, we are interested in generating a larger texture image of similar appearance to the input. State of the art algorithms were described and compared in a survey by Wei et al. [65]. The prominent 2D texture synthesis algorithms are the pixel-based, patch-based, and the texture optimization approach (Figure 3.4). Pixel-based and patch-based algorithms are usually faster than texture optimization and in some cases produce better quality and are easier to implement.

Pixel-based texture synthesis work on copying pixels from the input example to the target by finding candidates from the source and picking the one that best matches the neighborhood on the target. Several methods have been proposed to avoid an exhaustive search for the best pixel match. One approach based on k-coherence analysis the input and constructs similarity sets that contain a list of pixels with a similar neighborhood [63]. Advances in graphics hardware gave rise to real-time on-the-fly synthesis as presented by Lefebvre and Hoppe [40].

Patch-based synthesis is an extension of pixel-based methods. Rather than copying one pixel at a time, we copy patches of pixels at each step. However, this introduces issues relating to visual discontinuities between existing patches and newly copied ones. Proposed solutions include patch blending [42, 51], finding optimal cuts in the overlapped region [20, 36], wrapping of patches to ensure pattern continuity [67], or a combination of these techniques [1, 59].

In this work, we are interested in synthesizing large continuous patches of the input in order to avoid expensive surface reconstruction or introducing large stitching artifacts between the patches. Pixel-based methods are not suitable for this goal since they tend to produce a large number of small disconnected areas, therefore, patch-based methods are the ideal choice. In our early experiments, we implemented a fast patch-based synthesis method that applies the k-coherence idea based on [53]. The synthesis process is controlled by two parameters: patch size and the overlap band size, both in pixels. These parameters are dependent on the the input where generally patch sizes that cover the prominent feature or pattern gives the best results. Using this method allows us to synthesize the extensions using different patches from the source which may help in creating more randomness in the output. However, the produced patches were still largely disconnected and in practice we
observed the best results are when the patch sizes increased to the maximum possible size.

As an editing method, we are interested in preserving the original shape and texture as much as possible. Splitting the input in half and copying each to the ends of the target can help guide the synthesis process such that the start and end regions of the target exactly match the input. This constrain, however, introduces the problem of finding the best parameters to avoid synthesizing an area where the target image contain previous patches on both sides of the patch’s neighborhood leading to obvious discontinuities. Also, surface details that contain non-homogeneous patterns pose another challenge as most texture synthesis algorithms produce an output more biased towards the predominant texture feature. Rosenberger et al. [52] presented a method that automatically generates control maps that allows synthesis of multi-layered patterns, however, it requires a prior segmentation step and is computationally expensive. Surprisingly, a much simpler method can deal with non-homogeneous patterns and is an order of magnitude faster. Our texture synthesis method simply replicate overlapping patches of the entire input texture. Then, appropriate cuts are made on the overlap between the patches in order to achieve a more natural blending. Details and a discussion about the performance and limitations are described in section 4.3.1.

3.5 Surface blending and hole filling

The resulting patches of the texture synthesis stage specify the regions from the input where the actually geometry is copied to the extension area. However, the region between the patches is not well defined and needs to be dealt with separately.
Sampling then blending along the seams is one approach and have also been used when filling large missing parts in the 2D image domain [29]. In our context, we can sample the area along the seams in the parameter domain and interpolate between the positions given by each patch. This approach, however, introduces two challenges. The first challenge is the need to define a sampling method and a weighting scheme that would gradually blend one patch to the other. We can compute a distance field [41] for the different patches on the 2D parameter domain and use it as the weights of a set of uniformly sampled points. Since our parameter domain is only approximate, we do allow folded triangles which unfortunately can lead to noisy samples using this sampling method. Figure 3.5a shows a simple example of a good sampling when the parameter domain does not contain any folds or self-intersections. A better sampling approach is to directly sample on the surface which would avoid these problems and can be computed more efficiently using cross-sections along the curve skeleton of each overlap. However, we still need to define a robust sampling method that should take into account the surface density.

The second challenge of sampling and blending is the process of reconstructing the surface of the overlap using the computed samples. This introduces a number of different difficulties inherited from the problem of surface reconstruction. Surface reconstruction methods belong to two categories: function fitting methods [33, 48]; and computational geometry based methods [8, 34]. The fact that we are interested in preserving as much as possible of the original geometry of the shape with our preference of efficiency makes function fitting methods an overkill. A computational geometry based reconstruction techniques such as the ball-pivoting algorithm (BPA) [8] would be sufficient. However, any reconstruction process will most certainly be the bottle neck of our editing framework. Furthermore, we are still faced with the problem of varying triangle density of the surface between the source mesh and the reconstructed part. To avoid this issue we could either resample the entire region, making the process computationally expensive, or devise an adaptive surface reconstruction method that accounts for the varying density of sample points.

In order to avoid re-sampling, we can construct the extensions based on the original connectivity of the mesh. This allows for an efficient and simple method that avoids challenges when dealing with surface details of high genus. Geometry inside each patch will contain the same structure and connectivity as the source including holes, degeneracy, or non-manifold faces. The final step is then to join the resulting pieces of geometry in an aesthetically pleasing manner. Since the regions missing geometry between the copied surface patches is
relatively small, we could formulate the last step as a hole-filling problem. A standard method for hole filling would identify the boundary then triangulate the hole and perform a last step of refinement and fairing [44]. Still, two patches may be significantly close, or even intersecting, which may lead to obvious artifacts. A more general approach is to apply a set of mesh repair techniques that can deal with both intersecting faces and holes such as the approach by Attene [3]. However, such a method would require finding a set of constraints that help avoid any destructive operations on the inner geometry of each patch especially on coarse meshes. Nonetheless, experiments with these two approaches yielded results with apparent artifacts along the missing regions. In Figure 3.5b we show the result of applying such mesh repair operations on multiple patches.

To avoid reconstruction while achieving a balance of visually pleasing results and efficiency, we formulate our hole filling problem as a boundary stitching one. Our simple stitching procedure is described in more details in section 4.4 along with a discussion its limitations.
Chapter 4

Detail-replicating stretching

Inspired by deformation techniques in the image domain [22], we apply the concept of detail-replication when performing a stretching operation. We replicate the surface details of the region on the detailed mesh we intend to stretch. We adopt 2D texture synthesis to help guide the geometry synthesis process. We first represent the surface details as a height-field like image that approximates the actual details. The second step is to synthesize the image and create more of the same pattern along the direction of the skeleton edges in the selected region. The synthesized image is then used to specify the placement of the new geometrical surface patches along the extension. The final step is to combine all the patches into one closed surface that starts and ends with the exact geometry of the source region. The output of the process is a single connected surface which is then trivially combined with the source and produces the final extended mesh.

Computing the surface details image first requires computing a base mesh that is used in the parametrization process. A curvilinear grid is then constructed from that base mesh and the geometry of the selected region is then mapped onto that grid. We construct the grid such that its cells represent squares, making them ideal to efficiently construct a 2D image with minimum distortion. The pixel values of the image represent the signed distance between the square’s center and the original detailed mesh. The image is then synthesized using our tilting method that guarantees coherence with the original mesh geometry. Surface patches are then reconstructed from the corresponding image patches along the user specified extension path. The patches are then stitched together using a simple boundary stitching algorithm that works on avoiding intersecting faces and degenerate triangles. An additional optional step is to add some variation on the surface that can be useful when synthesizing
organic patterns to avoid any artificial look of exact replication.

4.1 Base mesh

We start by computing a base shape $\mathcal{B}$ that defines the underlying geometry of the detailed input mesh $\mathcal{D}$. The typical approach is to apply a fairing process (mesh smoothing) that would gradually smooth the high frequencies of the surface texture. As suggested by Andersen et al. [2], applying few steps of mean curvature flow would result in a good vertex-to-vertex correspondence between $\mathcal{D}$ and $\mathcal{B}$ and minimizes vertex sliding that typically occur when fairing using the umbrella operator (more details in Appendix A).

Curvature flow smooths the mesh by iteratively moving vertices along the surface normal $\mathbf{n}$ with the speed of the mean curvature $\kappa$ [19]. We start the fairing process by computing the non-zero coefficients of the matrix $K$ representing the matrix of the curvature normals. We compute the entries of $K$ with the following discrete expression for the curvature normal at each vertex $x_i$:

$$-\kappa \mathbf{n} = \frac{1}{4A} \sum_{j \in N(i)} (\cot \alpha_j + \cot \beta_j)(x_j - x_i)$$

where $\alpha_j$ and $\beta_j$ are as described in Figure 4.1b, and $A$ is the sum of the areas of the adjacent triangles of $x_i$ as in Figure 4.1a.

We then iteratively solve the following linear system representing the implicit integration of the diffusion equation, when used for mesh smoothing, using the backward Euler method as described by Desbrun et al. [19]:

$$(I - \lambda dt K)X^{n+1} = X^n$$

where $\lambda dt$ is the smoothing time step and as shown in [19] increasing the value results in
smoother meshes. This linear system is sparse, thus, we are able to solve it efficiently using a preconditioned conjugate gradient (PCG) solver. For preconditioning we use the typical diagonal preconditioner $\tilde{A}$ where $\tilde{A}_{ii} = 1/A_{ii}$.

This process of smoothing generally introduces shrinkage of $B$. One possible way to compensate for this shrinkage is to scale $B$ back to its original volume. The correction factor given by [19] is $\beta = (V^0/V^n)^{1/3}$, where $V^0$ is the volume of $D$ and $V^n$ is the volume of $B$ at iteration $n$. Let $x^1_k$, $x^2_k$, $x^3_k$ be the three vertices of the $k$-th triangle of the mesh, then the volume computation for a discrete mesh is given by the following expression [43]:

$$V = \frac{1}{6} \sum_{k=1}^{nbFaces} g_k \cdot N_k$$

where $g = \frac{x^1_k + x^2_k + x^3_k}{3}$ and $N_k = \frac{x^1_k x^2_k}{x^1_k x^2_k} \wedge \frac{x^1_k x^3_k}{x^1_k x^3_k}$. Each vertex of $B$ is then multiplied by the computed volume correction factor $\beta$. Depending on the size of the surface detail relative to the entire shape, it might not be ideal to apply the volume correction step as it may introduce large vertex sliding. An adaptive volume correction method would be more suitable, however, to the best of our knowledge no such method exists.

The amount of smoothing is specified by the time step $\lambda dt$. The larger the parameter value the smoother the mesh gets. The optimal value depends on the surface details of the input. To automatically assign such a value would require a method that can accurately separate the pattern from the underlying surface. In general, the problem of computing pattern symmetries, regularity, and repetitive structure is a fundamental problem in computer graphics. Several attempts have been made that work on extracting structure from repeated patterns in 2D images [45, 46, 16]. These attempts, however, assume structure in the provided input which is not always the case with irregular patterns or multi-layered surface textures as in Figure 4.2.

In our framework we relay on a user specified value of $\lambda dt$. The efficiency of the fairing method and our parametrization technique allows for an interactive process in which the user can experiment with different $\lambda dt$ values and check the results almost immediately even with large complex models.
4.2 Parametrization

A parametrization is a bijective mapping from one surface onto a another provided they have the same topology. If the topology differs between the two surfaces, cuts or other topology changing operations are required on the source mesh. In our context, we are interested in a general parametrization that can handle any mesh regardless of the topology. With this relaxed condition we are able to construct an approximate parametrization mesh that is invariant to the topology of the input.

We will refer to our parametrization domain as the cylindrical grid as it is constructed by sweeping along the curve skeleton and sampling the cross-section of $\mathcal{B}$. This cylindrical grid $\mathcal{G}$ is used in computing the approximate geometric texture as an image and its geometry reflects the geometry of $\mathcal{B}$.

4.2.1 Cylindrical grid

Given a segment on the curve skeleton selected by the user, we construct a cylinder like surface by sweeping. The quality of the extracted curve-skeleton affects the construction of $\mathcal{G}$ and we may need to refine the edges on that segment. We do so by subdividing the edges and then perform a simple Laplacian smoothing of the entire segment. The refined segment nodes are then used as control points for a Bézier spline $C$ to ensure a desirable continuity. We then parametrize $C$ with $t \in [1, 0]$ and uniformly sample it based on the specified grid resolution $h$.

The ideal grid resolution is dependent on the surface texture. More detailed textures
would require a larger $h$ to ensure a more faithful representation in the resulting texture image. In most of our experiments we set $h$ in the range of 70 to 140 pixels, Figure 4.4 shows the result of different resolution values. Given the grid resolution $h$, we compute the size used for the grid cells $y$ with the following expression:

$$ y = \frac{2\pi r}{h} \quad (4.3) $$

where $r$ is the closest distance from the base mesh $B$ to the curve $C$ computed by projecting a sample of vertices onto the skeleton segment. We can now compute the grid width:

$$ w = \frac{L_C}{y} \quad (4.4) $$

where $L_C$ is the arc length of the spline $C$. The values $h$ and $w$ would respectively represent the height and width of the approximating texture image.

Next, we sample $w$ cross-sections by intersecting a plane $p_i$ with the base mesh $B$. The plane $p_i$ passes through $C$ at $t_i = \frac{i}{w}$ with a normal $n_i$ equal to the tangent of $C$ at $t_i$. The resulting cross-section $S_i$ forms a closed polygon. In the case of missing geometry or multiple curves we simply select the largest connected part and then close that polygon.

An arbitrary vector $a_0$ on the plane $p_0$ is computed and transported along $C$. We compute at each $t_i$ the closest point on the polygon $S_i$ to the curve $C$ along the direction of $a_i$ and denote this point as the start of $S_i$. These start vectors can be rotated around the point $t_i$ on the curve if the original pattern exhibits some twisting. We then resample $S_i$ by walking along the polygon in $h$ segments of the length:

$$ \Delta e = \frac{L_{S_i}}{h} $$

Lastly we orient $S_i$ such that its normal is in the same direction as $n_i$. The re-sampled $S_i$ can now be encoded as a vector of distances between $C(t_i)$ and the samples. This vector $Y_i$ help indicates the shape of the underlying structure at $t_i$. Figure 4.3c shows an overview of this grid construction via our sweeping process.

The geometric construction of the grid is done by connecting the corresponding sample points on $S_i$ with the points on $S_{i+1}$. The height value from equation 4.4 results in grid cells with an aspect ratio of a square-like cell minimizing possible distortions. The resulting shape of the grid represents an abstraction of $B$ with a resolution dependent on $h$. Figure 4.5 shows an example of our grid construction.
Figure 4.3: Grid construction by sweeping: (a) the source mesh. (b) the computed base mesh with the medial curve representing the provided skeleton. We sample along the curve using $w$ planes shown in green. (c) detailed view of the base mesh sampling process during grid construction. The green curves on each plane represent the cross-section computed from the base. The yellow polygon is an equidistant polygon sampled from the closest intersecting point to $a_i$ and it contains $w$ edges.

Figure 4.4: Grid resolution: the grid defined on the blue region of the input shape (left) and the resulting texture image at different resolutions (right).
4.2.2 Geometric texture representation

We use the constructed grid $\mathcal{G}$ to generate a 2D image that approximates the surface details of $\mathcal{D}$ and to parametrize their geometry by projecting the vertices to the nearest cell in $\mathcal{G}$.

Computing the texture image $I$ is a straightforward operation. Let $n^c$ be the normal of the grid cell $c_{xy}$ and $p_c$ is the centroid, where $x \in [1, w]$ and $y \in [1, h]$ then the pixel value $d$ at $I_{xy}$ is:

$$d = ||q - p_c||$$

$$I_{xy} = \begin{cases} 
  d & \text{if } p_c \text{ is below } \mathcal{D} \\
  -d & \text{otherwise}
\end{cases}$$

(4.5)

where $q$ is the intersection point on $\mathcal{D}$ computed by shooting a ray with origin $p_c$ and direction $n^c$. The sign of $d$ is determined by which side of the plane defined by the intersecting face on $\mathcal{D}$ does $p_c$ falls into. We perform these intersection tests efficiently using a space partitioning octree [61].

We then parametrize the geometry of $\mathcal{D}$ onto the domain of $\mathcal{G}$ using the following steps: for each vertex $v$ of the selected region on $\mathcal{D}$, we search for the closest grid cell using ray-triangle intersections. The ray $r$ has the origin $v$ and normalized direction $-d$, where $d$ is the normal of the corresponding vertex on the base $\mathcal{B}$. For each hit, we encode the location of $v$ with two quantities: a scalar height value; and a shift quantity with respect to a local frame on the intersected grid cell. The main idea of this process is to project the source onto the grid in a flattening manner.

In order to compute the height value, we first compute the weights of the mean value coordinate (MVC) [23] of the intersection point inside the cell. Let $\alpha_i$ be the angle at the intersection point $v_0$ in the virtual triangle $[v_0, v_i, v_{i+1}]$ as shown in (Figure 4.6) where $v_{i>0}$
CHAPTER 4. DETAIL-REPLICATING STRETCHING

are the corners of the cell, the mean value coordinates are computed using the weights:

\[ \lambda_i = \frac{w_i}{\sum_{j=1}^{k} w_j}, \quad w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|v_i - v_0\|} \]

and in our cells \( k \) is always equal to 4 corners. It is now possible to reconstruct the intersection point \( v_0 \) using:

\[ v_0 = \sum_{j=1}^{4} \lambda_j v_i \]

When the intersection point falls on the boundary of the cell we only need to linearly interpolate on that edge.

The height value \( q \) at \( v_0 \) is then defined as:

\[ q = n^e \cdot (v - v_0) \] (4.6)

The shift factor depends on the local frame of the grid cell and it is defined by the two scalar quantities \( m \) and \( \theta \) using the following equations:

\[ p' = v_0 + (qn^e) \]
\[ s = v - p', \quad k = v_4 - v_1 \]

\[ m = \|s\| \] (4.7)
\[ \theta = \arccos\left(\frac{s \cdot k}{\|s\| \|k\|}\right) \] (4.8)

The shift factor helps compensate for any vertex sliding that occurs during the smoothing operation and allows us to encode points that do not constitute simple displacements from \( B \).

With these three quantities we can reconstruct the vertex \( v \) given a new cell by the following expression:

\[ v = v'_0 + qn^e + m\tilde{R} \] (4.9)

where \( v'_0 \) is the intersection point computed in the new cell and \( \tilde{R} \) is the rotated vector \( k' \) along the axis \( n^e \) with angle \( \theta \) using a the shortest quaternion rotation. The entire vertex displacement encoding is detailed in Figure 4.6.
4.3 Synthesis

The synthesis process starts after the user specifies the extension operation by simply drawing a curve \( E' \) that extends on the original length of the selected segment of \( D \). As discussed in section 3.4, the most practical approach of texture synthesis in our context is the one that produces the largest connected patches. We experimented with various 2D texture synthesis methods [36, 39, 53] and the most seamless results were produced by the one that uses the entire input as the patch.

4.3.1 Texture synthesis

Given the input texture \( I \) with dimensions \( w \) and \( h \) defined by our grid, we are interested in synthesizing \( I' \) such that its width is equal to:

\[
    w' = w + e, \quad e = \frac{L(E') - L(C)}{y}
\]

where the function \( L(x) \) is the arc length of the curve \( x \), and \( y \) is the segment length as in Eq.4.3. We perform the synthesis by copying the entire patch \( I \) overlapped by \( b \) pixels \( n \) times where \( n = \lceil \frac{w'}{w} \rceil \). The amount of overlap \( b \) is highly dependent on the source texture. Complex patterns may require larger overlap to allow for more room to circumvent some distinct features of the pattern.

In the overlap region we compute the minimum error boundary cut by computing a weighted \( L^2 \)-norm of differences between the two overlapping patches \( A \) and \( B \) at each pixel \( p \):

\[
    D_p(A, B) = \|A_p - B_p\|_2
\]
We then find the minimum cut on the graph $M$ in which each adjacent pixels $s, t$ in the overlap have an arc with the following cost:

$$M_{s,t}(A,B) = D_s(A,B) + D_t(A,B)$$

(4.10)

We then connect the top row with a start node and the bottom row with an end node with arcs of zero cost and find the shortest path $c$ given by Dijkstra’s algorithm. The computed path $c$ defines a binary mask that we use to copy the new patch $B$ to $I'$. In order to preserve the initial surface details in $I$ we start the synthesis with a simple copy of $I$ as the first step. To add some variety on the output we can slightly randomize the band size $b$ or the source of subsequent patches. The output of the entire synthesis process is the matrix $T$ that represents the corresponding pixel indices of $I'$ with respect to the source $I$. Figure 4.7 shows an overview of the tiling process. Using this method, the complexity of our texture synthesis algorithm is bounded by the complexity of solving Dijkstra’s algorithm for each patch. One drawback of the method is when synthesizing highly random patterns where the repetition of the patches may become more visible.

### 4.3.2 Geometry reconstruction

We use the indices matrix $T$ to construct the actual geometry of the extended area. First, we combine the two curves $C$ and $E'$ into one curve $E$. To extend the grid $G$ to the new width of $E$ we start by splitting $G$ in half and copy the exact structure of its first half to the target grid $G'$. We then go over the middle section of $E$ and generate interpolated $S_i$ cross-sections until we reach $i = w' - \frac{w}{2}$ where we switch back to simply copying from the end half of $G$. The idea behind this split is to preserve as much as possible of the geometry in the original
shape which helps minimize any distortions resulting from our different approximations. Interpolating the cross-sections of the middle region might not always be the best option. If the input pattern corresponds to both local and global changes on the surface then a better approach is to correspond these cross-sections with the patches given by $T$. Figure 4.8 shows an example where cross-section interpolation is preferable.

We now have the extended grid $G'$ with the new width specified by the user’s extension operation. The next step is to reconstruct the geometry of each patch in $T$ by finding complete faces $f \in F$ belonging to that patch. A face $f$ is complete if all of its vertices are located inside the patch. We do this by constructing a hash table of all the indices of the vertices collected from the patch’s cells and then simply test against the set of faces in the selected region of $D$. We can then reconstruct the geometry of the vertices in the set $F$ using Eq. 4.9. The result of this process is a set of separated patches of geometry representing the synthesized image $I'$.

### 4.4 Stitching patches

The final step in our framework is the stitching of the synthesized patches. As discussed in section 3.5, we formulate the problem as that of hole-filling. Between the patches $A$ and $B$ there exists a small empty region that separates the geometry of the two. Our objective is to fill this region such that $B$ blends well with $A$ resulting in the least amount of visual distortion.
artifacts. Aesthetically pleasing stitches are the ones that triangulate the area while avoiding triangles that stand out by having long edges or sharp curvature.

The general approach to hole-filling is to compute the minimum area triangulation of the hole, refine the new triangles to match the surrounding ones, and finally apply a surface fairing operation [44]. We apply a similar three steps method where we: treat the boundary of the two consecutive patches; zip the two boundaries $B_A$ and $B_B$ by simply advancing to the best adjacent vertices; then apply a local fairing operation on bad triangles.

The geometry reconstruction process results in missing geometry on the boundary of each patch. Our objective in the boundary treatment step is to produce a more flat and smooth boundary. For every adjacent edges $e_1$ and $e_2$ we add a new boundary triangle between them if their angle $\phi < \frac{5\pi}{9}$ and the dihedral angle between their respective two boundary faces is $\omega < \frac{\pi}{4}$. These angle conditions can be seen in Figure 4.9. The first condition $\phi$ ensures a flatter boundary and the second condition $\omega$ help preserve sharp edges of the original shape. We iteratively apply this refinement stage until no new faces are added to the boundary. The resulting edges on the final boundary are more well adjusted for the second zipping stage.

We start the zipping stage by first aligning the boundaries $B_A$ and $B_B$ and assigning the start vertices on each boundary to the closest pair $v^A_i$ and $v^B_j$. We traverse the two boundaries in a greedy manner using the following set of heuristic checks. If $|v^A_i v^A_{i+1}| + |v^B_j v^A_{i+1}| < |v^A_i v^B_{j+1}| + |v^B_j v^B_{j+1}|$ then we consider advancing the boundary $B_A$ [5]. If $\alpha < \theta < \beta$ where $\alpha$ is the min-angle of the triangle resulting from advancing on $B_A$ and $\beta$ is the min-angle of the triangle from advancing $B_B$ we swap our choice. The threshold angle $\theta = \frac{\pi}{18}$ produced the best results as per our experiments. The last check is for self intersections which is done by checking whether or not the next vertex lies inside the considered triangle. The two boundaries do not necessarily have the same number of vertices, therefore, we stop advancing once we reach the end of one boundary and then fill the remaining hole with a similar method. Figure 4.10 shows the result of our boundary stitching approach.

The last stage of stitching is the fairing process which helps blend new faces with existing ones. In some situations, the triangulated faces might contain extreme angles resulting in sharp edges and other visual artifacts. We test the angles $\lambda_i$ of each new face and mark vertices with angles outside the range $10^\circ < \lambda_i < 150^\circ$ as bad vertices. We then go over the list of bad vertices and apply Laplacian smoothing with uniform weights while avoiding flips. Other local quality measures can be further applied as described by Attene [3] since our simple stitching process does not guarantee that the stitched regions do not contain
CHAPTER 4. DETAIL-REPLICATING STRETCHING

4.5 Adding variation

A common feature of organic patterns is randomness in both structure and frequency. In such patterns, our synthesized parts may exhibit an artificial look of repetition, for instance, tree trunks do not have the exact cross-section as they grow. Our method allows the user to apply some random appearance on the resulting cross-sections. These fluctuations on the surface can follow a pattern themselves, as in wavy pattern, or can be randomly generated from gradient noise as shown in Figure 4.11. A nice feature of this randomness effect is that

self-intersecting triangles.
high frequency details are not equally effected as the base surface due to the separation in our synthesis process.
Chapter 5

Implementation

We have implemented a simple editing interface that we used to produce the results in chapter 6. The system was implemented and tested on a 3.0 GHz AMD Phenom II quad core computer using C++ and OpenGL. We used the Qt application framework to develop the graphical user interface which allowed for rapid development and excellent portability. We also used the SparseLib++ iterative solver for the base extraction process using the preconditioned conjugate gradient method.

Figure 1.4 shows the overall process followed in our implementation. In an editing session, the user starts by loading an arbitrary triangular mesh. Our method support editing of meshes containing boundaries and can be of non-zero genus. We assume the availability of a previously extracted curve skeleton that contain both the skeleton graph and a basic segmentation of the mesh. The user then simply selects a point on the surface and the corresponding node on the skeleton is set as the start. Next the user clicks on a different part of the surface that would specify the end. We assign each edge on the skeleton a weight of one and run Dijkstra’s algorithm to obtain the shortest path from the start to the end. We then collect all the branches and loops between the two selected nodes in this path. This step is necessary for surface details that are complex enough to produce branching on the skeleton. The result of this user interaction is a set of faces and their major medial curve.

There are three parameters that control the texture extraction and synthesis: the smoothing step size, grid resolution, and the overlap size. All three parameters are highly dependent on the input pattern. The smoothing parameter $\lambda dt$ specifies the amount of smoothing needed to obtain a base mesh. In most of our examples we set $\lambda dt = 0.001$ by default and for objects with more complex surface details we increase it to around 0.01. The second
parameter is the grid resolution $h$ which we set to 70 pixels by default. For more regular structured patterns a lower grid resolution would produce good results with low computational cost. The last parameter is the overlap $b$ that is used by the texture synthesis process. The best overlap size is one that would cover the largest feature of the pattern. We set the default value as 20\% of the width of the input texture image.

The next step is drawing the curve defining the extension operation. We allow the user to draw a curve on the screen that would specify the extension direction and path during the synthesis. We project the cursor’s position in screen space onto the focal plane defined by the camera. For each different stretching operation the user needs to rotate the camera such that the viewing direction is almost orthogonal to their desired direction of stretching (Figure 5.1). After the curve is drawn, we embed this curve in 3D in a way that ensures better continuity with the original shape. Let $v_a$ be a unit vector representing the starting tip of the user drawn curve and let $v_b$ be the tangent vector on the curve skeleton at the middle of the selected input region, an example is shown in Figure 5.2b. We compute the rotation that would align $v_a$ with $v_b$ and apply it to the drawn curve. We then translate the drawn curve so that its starting point at $a$ is on the mid-point $b$ of the selected part of the curve skeleton.

The final step is tweaking the parameters previously mentioned to achieve aesthetically
pleasing extensions. The efficiency of our method allows for an interactive editing experience. The synthesis process is almost instant and would usually take two seconds to finish on medium sized objects (about 50K vertices). Perhaps the most challenging parameter to automate is the overlap width which is still an open problem in 2D texture synthesis as it relates to texture and pattern recognition. In our experiments we usually test a couple of overlap values before obtaining a desirable result as discussed in Section 6.1.

We realize that the nature of our synthesis process allows for a large number of opportunities of parallel optimizations that can lead to near real-time editing. Such parallel opportunities are during grid construction, the tile-based synthesis, and the patch stitching. We exploited some of these opportunities of parallelism using OpenMP, for example when smoothing, and we have witnessed a considerable performance increase.
Figure 5.2: Details of the embedding of user drawn extension curves. In order to ensure continuity, user drawn curves are transformed to align with the existing curve skeleton of the source shape. This is done by transforming the start of the drawn curve (green curve) to match the curve skeleton at the middle, point $b$, of the selected input region (dashed blue line).
Chapter 6

Results

In this chapter we present examples of editing operations performed using our method. First, we show one example that demonstrate the effect of our parameters on the extension process. We then show examples of stretching different shapes while preserving surface details allowing for an unprecedented editing tool. We then discuss the current limitations of our method and propose some possible solutions for each case.

6.1 Parameters

There are three parameters needed during the extension process: the step size value $\lambda dt$ that signifies the amount of smoothing; the grid resolution $h$ that specifies the quality of the parametrization domain; and the size of the overlap $b$ used in the synthesis process. In our implementation the parameters can be easily tuned by the user using a simple spin box (or slider).

We set the parameter $\lambda dt$ depending on the complexity of the surface details. In most of our experiments we set it between 0.001 and 0.01 depending on the size of the detail. In Figure 6.1 we show the effect of choosing different values of $\lambda dt$ on extending the neck of the raptor model. As expected, the larger the value the more global displacements are considered as part of the detail. In this example we can see that lower values give the best results as the surface details on the neck region are relatively small.

The parameter $h$ determine the grid size and resolution of the representative image of the surface texture. For surface details with distinct features, a larger grid is more suitable as it ensures the feature is not trimmed during synthesis. In Figure 6.2 we notice that the
CHAPTER 6. RESULTS

resolution had a small effect on the output due to the simplicity of the details. However, the quality of the output is better on the larger grid since the reconstruction process is more accurate as we work on a finer grid.

The last parameter is the overlap size $b$ used in the texture synthesis stage. In Figure 6.3 we experiment with three values of $b$ using a grid resolution of 180. Finding the best minimum cut is a challenging problem especially when considering patterns with variable size features as in Figure 6.3. The efficiency of our method allows for an interactive selection of the most visually pleasing value. We notice that setting the value too high or too low would result in a repetitive appearance of the extended area.

In Figure 6.4 we show an example for a set of good parameters for the raptor model using $\lambda dt = 0.0001$, $h = 180$, and $b = 7$.

6.2 Editing examples

We present results of editing sessions that demonstrate generating shape variety using our detail replicating stretching. Table 6.1 list the processing timing of our method for these examples (in multiple outputs we consider the first shape). Note that the preprocessing base extraction stage is not included in our total time as this process need to be computed only once per model. The second column in the table indicates the number of vertices in the region selected for extraction. The third column represents the amount of extension performed on that region. The total timing results in the last column in Table 6.1 demonstrate the efficiency of our method in performing interactive stretching operations.

The first set of results in Figure 6.5 shows four cylindrical shapes with different patterns. For the Knot model we made the knot curve in another modeling software. The Smooth model demonstrate the usefulness of our tool in generating extensions that preserve the density of the geometry for models with no details. We are able to extend the pattern on the Twist model by rotating the start vector as described in 4.2.1.

The second set of results in 6.6 demonstrate stretching operations on characters and shapes with non-circular cross-sections. For most models we use linear interpolation for the cross-sections as described in 4.3.2. Additional results are presented in Figure 6.7 and Figure 6.8. Figure 6.9 and Figure 6.10 show larger views of some selected models generated by our method.
### Table 6.1: Statistics for our editing experiments.

<table>
<thead>
<tr>
<th>Model</th>
<th>Base (ms)</th>
<th>Vertices considered</th>
<th>Ext. factor</th>
<th>Projection</th>
<th>Texture</th>
<th>Synthesis</th>
<th>Recon.</th>
<th>Stitching</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capsule</td>
<td>296</td>
<td>3360</td>
<td>2.5</td>
<td>87</td>
<td>50</td>
<td>5</td>
<td>71</td>
<td>109</td>
<td>322</td>
</tr>
<tr>
<td>Knot</td>
<td>1265</td>
<td>9380</td>
<td>13.4</td>
<td>273</td>
<td>60</td>
<td>96</td>
<td>726</td>
<td>1406</td>
<td>2561</td>
</tr>
<tr>
<td>Smooth</td>
<td>1164</td>
<td>8260</td>
<td>4.2</td>
<td>212</td>
<td>45</td>
<td>8</td>
<td>225</td>
<td>406</td>
<td>896</td>
</tr>
<tr>
<td>Twist</td>
<td>553</td>
<td>5529</td>
<td>2.5</td>
<td>173</td>
<td>114</td>
<td>11</td>
<td>116</td>
<td>167</td>
<td>581</td>
</tr>
<tr>
<td>Cream</td>
<td>138</td>
<td>1936</td>
<td>9.0</td>
<td>52</td>
<td>16</td>
<td>18</td>
<td>112</td>
<td>355</td>
<td>553</td>
</tr>
<tr>
<td>Dino</td>
<td>321</td>
<td>301</td>
<td>14.4</td>
<td>11</td>
<td>77</td>
<td>54</td>
<td>93</td>
<td>73</td>
<td>308</td>
</tr>
<tr>
<td>Asian Dragon</td>
<td>1828</td>
<td>3023</td>
<td>6.7</td>
<td>127</td>
<td>114</td>
<td>46</td>
<td>213</td>
<td>502</td>
<td>1002</td>
</tr>
<tr>
<td>Dragon</td>
<td>1582</td>
<td>1647</td>
<td>8.9</td>
<td>62</td>
<td>47</td>
<td>26</td>
<td>143</td>
<td>332</td>
<td>610</td>
</tr>
<tr>
<td>Necklace</td>
<td>294</td>
<td>3297</td>
<td>24.2</td>
<td>112</td>
<td>92</td>
<td>170</td>
<td>581</td>
<td>3366</td>
<td>4321</td>
</tr>
<tr>
<td>Hand</td>
<td>4758</td>
<td>1481</td>
<td>25.9</td>
<td>59</td>
<td>60</td>
<td>49</td>
<td>371</td>
<td>1005</td>
<td>1544</td>
</tr>
<tr>
<td>Multi</td>
<td>1309</td>
<td>2125</td>
<td>3.6</td>
<td>61</td>
<td>25</td>
<td>5</td>
<td>61</td>
<td>92</td>
<td>244</td>
</tr>
<tr>
<td>Armadillo</td>
<td>1454</td>
<td>1549</td>
<td>2.2</td>
<td>55</td>
<td>42</td>
<td>18</td>
<td>38</td>
<td>66</td>
<td>219</td>
</tr>
<tr>
<td>Organic</td>
<td>1635</td>
<td>15272</td>
<td>3.6</td>
<td>353</td>
<td>159</td>
<td>4</td>
<td>459</td>
<td>776</td>
<td>1751</td>
</tr>
<tr>
<td>Handle</td>
<td>13</td>
<td>402</td>
<td>2.7</td>
<td>60</td>
<td>42</td>
<td>3</td>
<td>21</td>
<td>17</td>
<td>143</td>
</tr>
<tr>
<td>Voronoi</td>
<td>2004</td>
<td>11610</td>
<td>7.4</td>
<td>578</td>
<td>606</td>
<td>69</td>
<td>653</td>
<td>4603</td>
<td>6509</td>
</tr>
<tr>
<td>Sheet</td>
<td>1448</td>
<td>41522</td>
<td>3.9</td>
<td>1509</td>
<td>791</td>
<td>107</td>
<td>1242</td>
<td>2659</td>
<td>6398</td>
</tr>
</tbody>
</table>

### 6.3 Limitations

**Parametrization:** the parametrization used in our method requires a curve skeleton that might not be well defined for shapes such as sheets or spheres. One way of dealing with these objects is to define a different parametrization scheme based on the topology. Another possible method is to allow for user assisted skeleton generation by sketching curves on the surface and computing approximate medial axis. A more challenging case is when the selected region does not split the mesh into two separate parts as in a regular torus. Extending along the skeleton would require a global deformation of the shape that may cause large distortion.

**Texture Synthesis:** limitations related to texture synthesis are inherited in our method. For highly irregular patterns most texture synthesis techniques fail to produce plausible outputs. What might be a trivial pattern for humans can be extremely challenging to replicate given a limited amount of the input. Also, complex patterns that are composed of several layers or different sub-patterns may not be easy to replicate without user interaction or precomputed control maps as in current 2D texture synthesis methods.

**Boundary Stitching:** stitching and blending different patches is the most challenging aspect in our method. For instance, our simple boundary stitching process will not be able to deal with cases where the patch cuts produce multiple boundaries. These cases could
be avoided by adding constraints in the texture synthesis process that penalizes any cuts
that produce multiple boundaries. Also the boundaries of the computed patches do not
necessarily match in terms of shape or geometry resulting in obvious artifacts when blending
without any preprocessing. Finding a better shape for the neighborhood of the blend area
results in a problem with a large search space that may not help us converge on a good
solution especially for mixed patterns.

In Figure 6.8 we use a simple cylinder as the grid for the Handle model to produce better
results. Using our regular grid construction method (Figure 4.5) instead of the cylinder
results in large distortions near the ends of the handle due to the cross-sections resulting
from the changed topology. The Voronoi and Sheet models demonstrate drawbacks of using
approximations in our method when dealing with very sharp or very smooth patterns. In the
case of the Voronoi model, the sharp features are not synthesized probably in the texture
synthesis stage resulting in visible seams. In the Sheet model, the smoothness of the original
pattern is not encoded well since we use a discretized grid.
CHAPTER 6. RESULTS

Figure 6.1: Smoothing parameter

Figure 6.2: Grid resolution parameter

Figure 6.3: Overlap parameter
Figure 6.4: Selecting the best parameters for a particular model results in seamless extensions.
Figure 6.5: Results set A
CHAPTER 6. RESULTS

Figure 6.6: Results set $B$
Figure 6.7: Results set $C$
Figure 6.8: Results set $D$
Figure 6.9: Extensions applied on two dragon models
Figure 6.10: Results from the Voronoi and Necklace models
Chapter 7

Conclusion and future work

7.1 Conclusion

In this thesis we have presented an intuitive mesh editing tool that preserves surface details during 1D stretching. Instead of interpolating details during an extension operation, we replicate the surface by synthesizing and stitching appropriate patches from the original mesh. Separating the synthesis of large scale details from small ones allows us to apply such global operations on the shape with minimum distortions and minimum artifacts on the seams. We demonstrated our efficient technique with a variety of synthesized examples that are generated in a matter of seconds.

7.2 Future work

In the following we present a number of possible extensions on this work that would allow for a more practical and more general editing tool. Some of these ideas are based on previous work and can be easily incorporated in our framework.

Automatic segmentation of textured regions: being able to segment the mesh based on the patterns of each region can help avoid the user selection step in our method. This allows us to extend parts with multiple patterns more easily.

Skeleton generation: our method depends on the quality of the extracted curve skeleton. Current skeleton methods suffer from drawbacks when generating skeletons for flat or highly detailed surfaces. Two possible approaches can help avoid issues relating to curve skeletons. The first approach is to develop an adaptive mesh skeletonization method that
can treat challenging cases appropriately. Another idea is to replace our current grid construction process with abstract representations of the overall shape as in [47]. Using shape descriptors that are more than simple curves may also help with our current 1D stretching limitation.

**Extending groups:** it is common for both man-made and organic shapes to contain branches or limb like elements that share similar geometric properties. For example, a table can contain four legs that contain similar surface details. It is ideal that once the user selects a region in one leg that the other regions are automatically selected as well. Therefore, performing a stretching operation on one element in the group should propagate on the entire or partial elements of the set. An example of this idea is demonstrated in [26].

**Robust boundary blending:** with a more robust boundary blending method we would be able to allow for more powerful editing applications such as detail transfer. It would also allow us to achieve more randomness in our synthesized parts and possibly allow for coherent an-isometric synthesis of geometric texture using ideas presented by Lefebvre and Hoppe [40].

**Integration:** this method can be easily incorporated into existing state of the art mesh deformation techniques that preserve surface details during bending and twisting. This results in increasing the tool set and flexibility for 3D modelers. For instance, at the end of an extension operation, the shape can be bent with preserved properties such as volume using an existing method in a post-processing step.

**Parallel opportunities:** being able to perform stretching operations in real-time can help compensate for the limitations faced by texture synthesis methods as it gives the users an opportunity to experiment with different parameters to help them decide on the settings for the best looking detail replication.

**Learning & Editing patterns:** being able to map surface details from 3D geometry to the image domain would make it easy to utilize most techniques used in the 2D literature onto 3D geometric textures such pattern segmentation, classification, grid detection, watermarking, super-resolution, etc.
Appendix A

Mesh Smoothing

The general idea behind mesh smoothing is to reduce noise in a mesh through a diffusion process. In the context of triangular mesh smoothing, we displace mesh vertices in the direction of the Laplacian vector (2nd spatial derivative), thus smoothing surface disturbances of high frequency.

The common method for smoothing meshes is the discrete diffusion flow. Each vertex \( x_i \) on the mesh is iteratively updated by adding the scaled discrete Laplacian \( \Delta x_i \)

\[
x'_i = x_i + t \Delta x_i
\]  \hspace{1cm} \text{(A.1)}

and by assigning scalar weights \( w_{ij} \) to each neighbor of \( x_i \) such that \( \sum_j w_{ij} = 1 \), the discrete Laplacian is defined as

\[
\Delta x_i = \sum_{j \in N(i)} w_{ij} (x_j - x_i)
\]

The simplest approximation of the Laplacian uses the equal weights

\[
w_{ij} = \frac{1}{m}
\]

where \( m \) is the number of neighbors of \( x_i \). This is known as the umbrella operator which in effect defines the Laplacian as the vector from \( x_i \) to the barycenter of its one-ring neighbors. Another approximation of the Laplacian assigns weights based on the distance between \( x_i \) and its neighbors.
\[ w_{ij} = \frac{1}{|e_{ij}|} \]

This operator is known as the scale-dependent umbrella operator. Both these weighting schemes introduce vertex sliding that could deform the original mesh with each smoothing iteration especially with the typical umbrella operator. Desbrun et al. [19] proposed a solution that uses intrinsic properties of the surface. The weighting scheme they used eliminates the tangential component of the Laplacian

\[ w_{ij} = \cot \alpha_j + \cot \beta_j \]

where \( \alpha_j \) and \( \beta_j \) are as described in Figure 4.1b. Using these weights in Eq.(A.1) would result in curvature flow. Curvature flow smoothes the surface by moving along the surface normal with a speed equal to the mean curvature.
Bibliography


[67] Qing Wu and Yizhou Yu. Feature matching and deformation for texture synthesis.
