UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FAUSTO RICHETTI BLANCO

A Technique for Interactive Shape Deformation of Non-Structured Objects

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Manuel Menezes de Oliveira Neto
Advisor

Porto Alegre, May 2007

CIP – CATALOGING-IN-PUBLICATION

Blanco, Fausto Richetti

A Technique for Interactive Shape Deformation of Non-Structured Objects / Fausto Richetti Blanco. – Porto Alegre: PPGC da UFRGS, 2007.

80 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2007. Advisor: Manuel Menezes de Oliveira Neto.

1. Object deformation. 2. Interaction techniques. 3. Curves and surfaces. 4. Object modeling. 5. Non-structured objects. I. Oliveira Neto, Manuel Menezes de. II. Title.

Live as if you were to die tomorrow;
learn as if you were to live forever.
Mohandas Karamchand Gandhi

ACKNOWLEDGEMENTS

A special thanks go to my advisor, Manuel Menezes de Oliveira Neto, who has always tried to help me raise my limits and never doubted I was able to do it. I have learned a lot by observating his passion for computer graphics and research in general.

I would like to thank my colleagues of the computer graphics group at UFRGS for providing an intellectually stimulating and enjoyable research environment. Especially to Francisco (Chico), who helped all group by enriching the discussions with his broad range of knowledge (sometimes useless) and his ideas for this thesis. Professors Comba, Carla, Luciana and Manuel for their great work at UFRGS, conducting the computer graphics group to a level of excellence.

Family members who passed away in the course of my studies: Alfredo and Luiz Alfredo. All my friends and my girlfriend, Tamara, for tolerating my absence during important events of their lives. All my family, especially my mother, Donatila, and my sisters Deise and Carla for motivating me during the bad days. My father João (*in memoriam*) who taught me the values I will carry with me for my entire life.

I would also like to thank Yutaka Ohtake, Doug DeCarlo, Thomas Lewiner, their colleagues and the people behind LAPACK for making their software available on the internet. The Stanford 3D scanning repository for providing the Armadillo, Buddha, Bunny and Dragon models. Cyberware, for the Dino model. Mario Botsch for the plane with bumps and Mark Pauly for the Octopus model. Vitor Pamplona, who modeled the Moebius strip and the non-manifold surface. A. Ben Hamza for making available the Cactus model and AIM@SHAPE project for other models.

Additionally, I would like to acknowledge the Brazilian Research Council (CNPq - process number 130817/2005-8) for supporting me during my graduate studies.

TABLE OF CONTENTS

LIST OF FIGURES 9

LIST OF TABLES 11

ABSTRACT 13

RESUMO 15

1 INTRODUCTION 17
1.1 Technique overview 19
1.2 Contributions 20
1.3 Structure of the thesis 21

2 RELATED WORK 23

3 DEFORMING NON-STRUCTURED 3D MODELS 27
3.1 Region Segmentation 27
3.1.1 Pre-filtering the Sketch 28
3.1.2 Region Affected by Sketch 29
3.1.3 Regions Not Affected by Sketch 32
3.2 Digest 34

4 CONSTRUCTING PARAMETRIC CURVES AND SKELETONS FROM SKETCHES 35
4.1 Surface Curve 35
4.2 Skeleton Curve 37
4.3 Creating a Structured Skeleton 38
4.4 Digest 39

5 MESH DEFORMATION 41
5.1 Defining Local Frames 41
5.2 Associating a Curve to a Mesh 43
5.3 Twisting and Scaling 45
5.4 Blending Between Curves 46
5.5 Avoiding Local Self-Intersections 48
5.6 Digest 50

6 SUGGESTIVE CONTOURS EDITING 51
6.1 Motivation 51
6.2 Technique Overview 52
6.3 Line Identification 53
6.4 Blending Function 53
6.4.1 Geodesic Distance Approximation 54
6.5 Digest 56

7 RESULTS 57
7.1 Digest 62

8 CONCLUSIONS AND FUTURE WORK 63
8.1 Pros 63
8.1.1 Interactivity and Robustness 63
8.1.2 Locality 64
8.1.3 Smoothness 64
8.1.4 Local Feature Preservation 64
8.1.5 Avoidance of Local Self-intersections 64
8.2 Discussion 64
8.3 Future Work 65

REFERENCES 67

APPENDIX A CATMULL-ROM SPLINE 73

APPENDIX B UMA TÉCNICA PARA DEFORMAÇÃO INTERATIVA DE OBJETOS NÃO ESTRUTURADOS 75
B.1 Deformação de objetos 3D não estruturados 75
B.2 Contornos e contornos sugestivos 78
B.3 Discussão 80

LIST OF FIGURES

Figure 1.1: Dancing Armadillo 18
Figure 1.2: Technique overview step one 19
Figure 1.3: Technique overview step two 20
Figure 1.4: Technique overview step three 20

Figure 3.1: Smooth 2D sketch 28
Figure 3.2: Mesh segmentation based on user sketch 30
Figure 3.3: Mesh segmentation when two parts of the object project to the same portion of the image plane 30
Figure 3.4: The effect of not providing a smooth transition between the regions affected by different curves 32

Figure 4.1: Sketch discretization 37
Figure 4.2: Skeleton curve creation step one 38
Figure 4.3: Skeleton curve creation step two 38
Figure 4.4: A skeleton created by the presented technique 39

Figure 5.1: The creation of a local frame 42
Figure 5.2: The creation of a frame field along the curve 43
Figure 5.3: Plane parameterization 43
Figure 5.4: Creation of the modified set of local frames 44
Figure 5.5: Mesh artifacts due to discretization 45
Figure 5.6: Twisting and bending operations performed with our technique 46
Figure 5.7: Twisting and scaling operation on the Bunny model 47
Figure 5.8: Blending curves 47
Figure 5.9: A smooth deformation achieved by using the blending function . . . 48
Figure 5.10: Avoiding local self-intersections 49
Figure 5.11: Local self-intersection avoidance 50

Figure 6.1: Contours and suggestive contours 52
Figure 6.2: Mesh deformation using suggestive contours 53
Figure 6.3: Geodesic distance approximation 54

Figure 7.1: Deformation of the Buddha model 57
Figure 7.2: Deformations of the cactus model 58
Figure 7.3: Deformations of the octopus model 59
Figure 7.4: Deforming a horse leg 59
Figure 7.5: Deformations of the dragon model 60
Figure 7.6: Local feature preservation 60

Figure 7.7: Deformation of a mesh with multiple components 61
Figure 7.8: Deforming non-orientable and non-manifold surfaces 61

LIST OF TABLES

Table 7.1: Times obtained using our technique for deforming several models . . 62
Table 7.2: Comparison among the various mesh deformation techniques 62

ABSTRACT

This work presents a technique for interactive shape deformation of unstructured 3D models, based on 2D sketches and interactive curve manipulation in 3D. A set of lines sketched on the image plane over the projection of the model can be combined to create a skeleton composed by parametric curves, which can be interactively manipulated, thus deforming the associated surfaces.

Free-form deformations are performed by interactively moving around the curves' control points. Some other interesting effects, such as twisting and scaling, are obtained by operating directly over a frame field defined on the curve. An algorithm for mesh local self-intersection avoidance during model deformation is also presented. This algorithm is executed at interactive rates as is the whole technique presented in this work.

The presented technique naturally handles both translations and large rotations, as well as non-orientable and non-manifold surfaces, and meshes comprised of multiple components. In all cases, the deformation preserves local features.

The use of skeleton curves allows the technique to be implemented using a very intuitive interface, and giving the user fine control over the deformation. Skeleton constraints and local self-intersection avoidance are easily achieved. High-quality results on twisting and bending meshes are also demonstrated, and the results show that the presented technique is considerably faster than previous approaches for achieving similar results. Given its relatively low computational cost, this approach can handle meshes composed by hundreds of thousand vertices at interactive rates.

Keywords: Object deformation, interaction techniques, curves and surfaces, object modeling, non-structured objects.

Uma Técnica para Deformação Interativa de Objetos Não Estruturados

RESUMO

Este trabalho apresenta uma técnica para deformação interativa de objetos 3D não estruturados que combina o uso de *sketches* em 2D e manipulação interativa de curvas. Através de *sketches* no plano de imagem, o usuário cria curvas paramétricas a serem usadas como manipulares para modificar a malha do objeto. Um conjunto de linhas desenhadas sobre a projeção do modelo pode ser combinado para criar um esqueleto composto de curvas paramétricas, as quais podem ser interativamente manipuladas, deformando assim a superfície associada a elas.

Deformações livres são feitas movendo-se interativamente os pontos de controle das curvas. Alguns outros efeitos interessantes, como torção e escalamento, são obtidos operando-se diretamente sobre o campo de sistemas de coordenadas criado ao longo da curva. Um algoritmo para evitar inter-penetrações na malha durante uma sessão de modelagem com a técnica proposta também é apresentado. Esse algoritmo é executado a taxas interativas assim como toda a técnica apresentada neste trabalho.

A técnica proposta lida naturalmente com translações e grandes rotações, assim como superfícies não orientáveis, não variedades e malhas compostas de múltiplos componentes. Em todos os casos, a deformação preserva os detalhes locais consistentemente.

O uso de curvas esqueleto permite implementar a técnica utilizando uma interface bem intuitiva, e provê ao usuário um controle preciso sobre a deformação. Restrições sobre o esqueleto e deformações sem inter-penetrações são facilmente conseguidos. É demonstrada grande qualidade em torções e dobras nas malhas e os resultados mostram que a técnica apresentada é consideravelmente mais rápida que as abordagens anteriores, obtendo resultados similares. Dado seu relativo baixo custo computacional, esta abordagem pode lidar com malhas compostas por centenas de milhares de vértices a taxas interativas.

Palavras-chave: deformação de objetos, técnicas de interação, curvas e superfícies, modelagem de objetos, objetos não estruturados.

1 INTRODUCTION

The realism of computer-generated images depends heavily on the ability to accurately represent geometric details of the objects in the scene. Modeling, however, is a labor-intensive task that can be significantly accelerated with the use of 3D scanners. Although 3D scanners provide a fast solution for the problem of sampling geometrically complex shapes, algorithms for surface reconstruction from point clouds tend to produce non-structured models consisting of a single polygonal mesh. Surface reconstruction from point clouds has received a lot of attention in the recent years (LAGA; TAKAHASHI; NAKAJIMA, 2003; OHTAKE et al., 2003; CARR et al., 2001; OHTAKE; BELYAEV; SEIDEL, 2004a; HOPPE et al., 1992; TOBOR; REUTER; SCHLICK, 2003, 2004; CARR et al., 2003; XIE; MCDONNELL; QIN, 2004; ZHAO; OSHER; FEDKIW, 2001; OHTAKE; BELYAEV; SEIDEL, 2004a; YU, 1999; TERZOPOULOS; METAXAS, 1991; SCLAROFF; PENTLAND, 1991; AMENTA; BERN; KAMVYSSELIS, 1998; DEY; GIESEN, 2001; DEY; GOSWAMI, 2004; WANG; OLIVEIRA; KAUFMAN, 2005; WANG et al., 2005; BíSCARO, 2005) and 3D representations of geometrically complex objects are becoming widely available. However, providing the user the means for changing pose and animating the resulting representations is key for using these models in applications such as computer movies and games. The challenge is to provide an intuitive interface and interactive feedback, while preserving surface details and accommodating user-defined constraints.

In the recent years, there has been a growing interest in approaches to facilitate the manipulation of 3D objects in intuitive ways, while assuring interactive rates. A large number of techniques for performing mesh deformation have been proposed (ALEXA, 2003; BOTSCH; KOBBELT, 2004; GUSKOV; SWELDENS; SCHRÖDER, 1999; HUANG et al., 2006; KOBBELT et al., 1998; LIPMAN et al., 2004, 2005; SORKINE et al., 2004; YU et al., 2004; ZAYER et al., 2005; ZHOU et al., 2005; ZORIN; SCHRÖDER; SWELDENS, 1997). However, most of these techniques are based on complex formulations, requiring the solution of large linear systems, which becomes prohibitive for large meshes. In general, these linear systems must be solved for each deformation, which introduces undesirable delays in interactive modeling sessions. These techniques are capable of producing very nice results, but none of them seem to simultaneously satisfy the requirements of interactiveness and interactivity for meshes containing a few hundred thousand vertices or more.

Recently, Kho and Garland (KHO; GARLAND, 2005) presented a technique for mesh deformation based on 2D sketches, in which the user specifies a deformation by sketching two curves on the image plane. The interface is very intuitive and easy to use. However, if the user does not like the resulting deformation, there is no provision to perform fine adjustments. Also, noisy sketched curves tend to cause local jaggedness on the model that

must be smoothed by solving a linear system.

Interaction techniques based on 2D sketches are commonly used to create easy interfaces for user input, especially for creation and manipulation of 3D objects (IGARASHI; MATSUOKA; TANAKA, 1999; IGARASHI; HUGHES, 2001; CHERLIN et al., 2005; KHO; GARLAND, 2005). Objects acquired with 3D scanners tend to be rich in local details and global smooth deformations are often useful for this kind of objects because, in general, the user is not interested in modifying local details when performing a global deformation.

Global smooth deformations must preserve the local details of the objects in a consistent orientation during user manipulation. Ideally, the technique must achieve interactive rates and the artist must be capable of experimenting with different object poses during a modeling session. Based on these observations, some characteristics must be prioritized when developing a technique for 3D object editing, as listed below:

- **Interactivity**: The technique must be able to perform at interactive rates, providing immediate visual feedback;
- **Robustness**: Given the growing geometric complexity of the 3D objects used in computer graphics, the ability to deal with such models is essential;
- **Locality**: Deformations applied to a specific region of the object mesh must be kept local, not affecting "distant" parts of the mesh;
- **Smoothness**: Global deformations should be smooth;
- **Feature preservation**: Object' local details must be preserved during a global transformation. The features must be transformed in an intuitive way.

Figure 1.1: Dancing Armadillo. A set of dancing poses created using our model-deformation technique. The original pose is shown on the left.

In this work, a new interactive technique for geometric shape deformation of non-structured 3D objects based on the use of 2D sketches is presented. It was inspired by Kho and Garland's oversketching metaphor (KHO; GARLAND, 2005), but this new approach is significantly different from theirs. Parametric curves created from 2D sketches are used to deform, twist and scale the associated mesh. The resulting deformations are visually pleasing, provide local control and feature preservation under global deformations. Our approach also supports some user-defined constraints, such as the specification of rigid segments for deforming articulated figures. Moreover, it handles arbitrary

meshes, including multiple connected component ones, non-orientable and non-manifold surfaces, and can be implemented around a simple and intuitive user interface. Figure 1.1 shows the Armadillo model in some dancing poses obtained using our technique. Note the high-frequency details properly preserved on the deformed models.

1.1 Technique overview

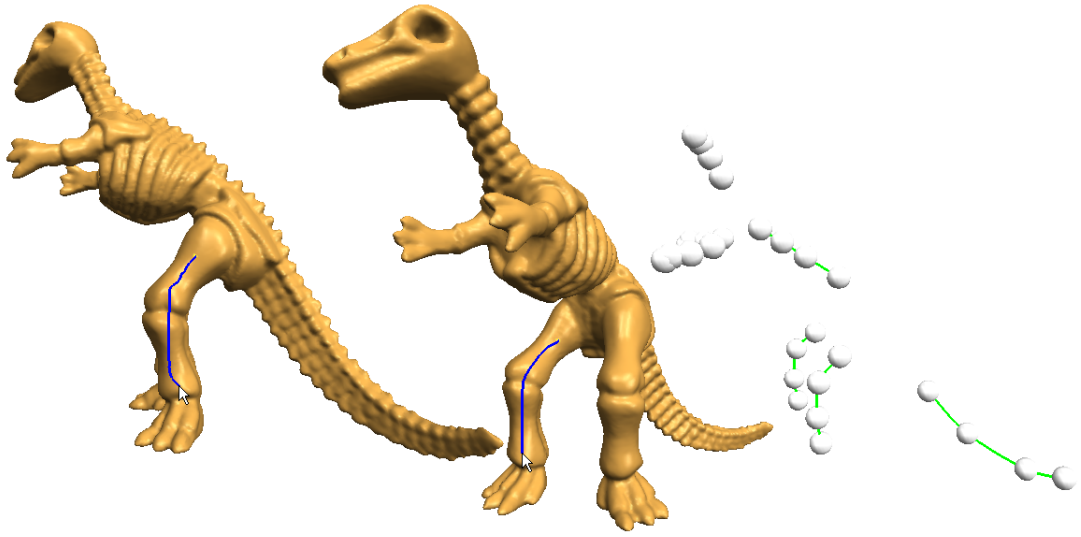


Figure 1.2: Technique overview step one. The user oversketches the parts of the model he/she would like to deform and is able to change camera's viewpoint (left and center). Each sketched line becomes a parametric curve in 3D that will be used as a handle for deforming the model (right). The small white balls represent the curves' control points.

A modeling session based on the technique presented in this work consists of three main steps:

- *Sketching 2D curves over the 3D model representation* (Figure 1.2) to produce a set of parametric curves in 3D that will be used as handles for deforming the model. Thus, by interactively modifying a curve, the deformation is automatically transferred to the parts of the model linked to it. A curve is modified by moving its control points, which are represented in Figure 1.2 (right) as small white balls;

- *Connecting individual curves to form skeletons*, as shown in Figure 1.3. Note that there is no need to have a single skeleton per object. In fact, the user may even want to keep all curves separate (disconnected) from each other;

- *Deforming the model by selecting individual control points and moving them in 3D*. Figure 1.4 illustrates this with two different deformations. The two characters on the left show the before and after states of a twist on the neck of the Dino model. The pair of images on the right provides a similar illustration for a deformation applied to the object's tail.

Similar to the technique described in (KHO; GARLAND, 2005), the user defines a deformation by sketching 2D curves on the image plane. Unlike their technique, however, we filter the user's noisy 2D sketches by creating a parametric curve in 3D. The creation

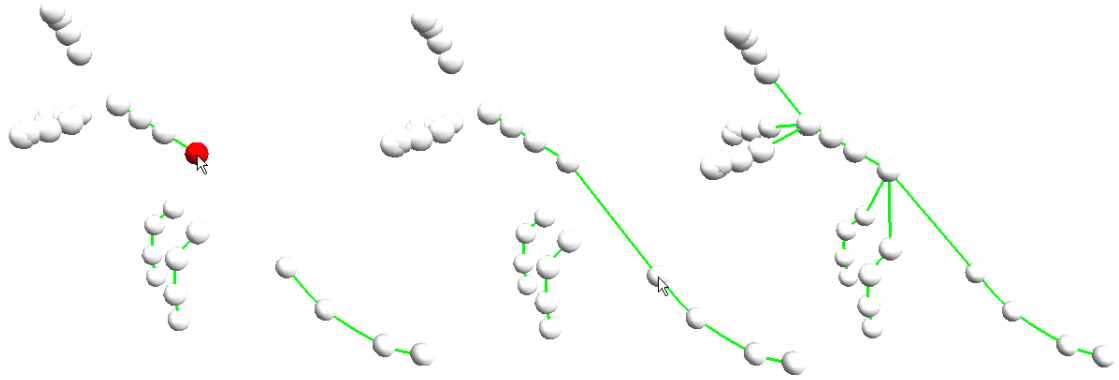


Figure 1.3: Technique overview step two. The user is able to combine parametric curves in order to create complex skeletons.

of parametric curves allows the user to specify skeletons for the objects and perform deformation while changing the camera's viewpoint.

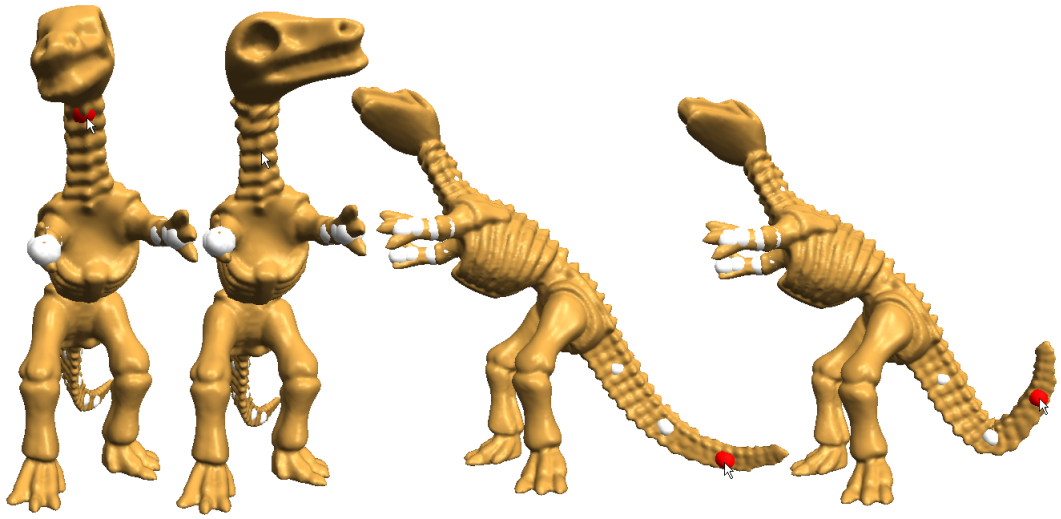


Figure 1.4: Technique overview step three. The user modifies a 3D curve and the technique deforms the mesh. Different viewpoints can be used.

1.2 Contributions

The main contribution of this work is a new technique for 3D model deformation that produces visually pleasing results at interactive rates while preserving the surface's local features (Chapter 7). The inherent smoothness of parametric curves is transferred to the induced global deformations, while a set of frames along the curves guarantees that the local features, expressed with respect to these frames, are preserved under global transformations. As such, the presented approach is related to recent mesh deformation techniques based on multiresolution (BOTSCH; KOBBELT, 2004; KOBBELT et al., 1998), Laplacian coordinates (ALEXA, 2003; LIPMAN et al., 2004, 2005; SORKINE et al., 2004) and gradient domain (HUANG et al., 2006; YU et al., 2004; ZAYER et al., 2005; ZHOU et al., 2005). It will be demonstrated that, despite its conceptual simplicity, the

proposed approach is quite general, not suffering from some limitations found in most previous techniques, outperforming them in terms of speed.

1.3 Structure of the thesis

Chapter 2 reviews some related work and dicusses their main characteristics and limitations. Chapter 3 introduces the technique proposed by this work and the steps needed for perfoming mesh deformation using it are detailed in Chapters 3 to 5. Chapter 6 explores the use of suggestive contours (DECARLO et al., 2003) as handles for mesh deformation. Chapter 7 presents the results, with extensive examples of mesh deformation using the presented technique. Finally, Chapter 8 discusses the conclusions and presents some ideas for future work.

2 RELATED WORK

There has been a considerable amount of work on model deformation in recent years. Free-Form Deformation (FFD) and its variations perform object deformation indirectly by manipulating a set of control points (handles) that deform the space containing the object. Such handles can be defined as 3D lattices (SEDERBERG; PARRY, 1986; COQUILLART, 1990; MACCRACKEN; JOY, 1996), a set of curves (CHANG; ROCKWOOD, 1994; SINGH; FIUME, 1998), or points (HSU; HUGHES; KAUFMAN, 1992). By performing the deformation indirectly, FFD techniques lack precise control over the deformation and tend to be somehow unintuitive.

Multiresolution techniques (ZORIN; SCHRÖDER; SWELDENS, 1997; KOBBELT et al., 1998; GUSKOV; SWELDENS; SCHRÖDER, 1999; BOTSCH; KOBBELT, 2004) decompose the surface into a smooth base representation (low frequency) and the surface details (high frequencies). Mesh deformation is applied directly to the base representation. The resulting mesh is then obtained by adding back the details (as displacement vectors) to the deformed base representation. Since the details are preserved uniformly over the entire surface, artifacts tend to become apparent in highly-deformed regions (LIPMAN et al., 2005).

Some techniques avoid factoring the base surface representation by directly applying the deformation to the original mesh. These techniques try to preserve some differential properties of the mesh, such as discrete Laplacian coordinates (ALEXA, 2003; SORKINE et al., 2004; LIPMAN et al., 2004, 2005) or gradient functions over the mesh (YU et al., 2004; ZAYER et al., 2005; ZHOU et al., 2005). All these techniques treat mesh deformation as a minimization problem, where the energy function to be minimized contains both a detail preservation term and some position constraints (HUANG et al., 2006). The detail preservation term is nonlinear as it also depends on the position constraints (*i.e.*, the mesh coordinates and local frames are defined with respect to a global coordinate system and the deformation may cause a rotation in these local frames used as reference for computing the differential properties). For efficiency reasons, the non-linear term has often been approximated by a linear one using various strategies, such as local linearization (SORKINE et al., 2004), heuristic approximations for the local rotations (LIPMAN et al., 2004), propagation of user-defined transformations (YU et al., 2004), and interpolation from handles (ZAYER et al., 2005; ZHOU et al., 2005). The use of these approximations, however, introduces artifacts in the orientation of the reconstructed surface details. For instance, the technique described in (SORKINE et al., 2004) has trouble handling large rotations (LIPMAN et al., 2005). Although gradient-based methods tend to handle rotations well, they face difficulties when adding translations to a given deformation, as translations preserve the gradient (BOTSCH et al., 2006). As pointed out by Botsch et al. (BOTSCH et al., 2006), a similar translation insensitivity can be observed

in the rotation-invariant technique described by Lipman et al. (LIPMAN et al., 2005). A nice and illustrated characterization of the limitations of these algorithms can be found in (BOTSCH et al., 2006).

Botsch et al. (BOTSCH et al., 2006) presented a physically-plausible approach for mesh deformation that uses regions of the mesh as manipulation handles. While the interaction metaphor is very intuitive and the method produces physically plausible deformations, it is also time consuming and, therefore, not suitable for interactive modeling sessions.

The approach presented in this work avoids the need of performing a non-linear minimization by using a simple, although very effective, solution: *the coordinates of the mesh vertices in the regions of interest are represented in terms of a set of frames associated with some parametric curve used to control the deformation*. Since such frames are instantly updated as the curves are deformed, so are the coordinates of the mesh. This solution naturally handles both rotations and translations and is computationally efficient. By using external, as opposed to local frames, our approach provides very general framework for mesh deformation. For instance, it naturally handles surfaces that should pose some difficulties to differential methods, such as non-orientable surfaces, non-manifolds, and surfaces composed by multiple connected components.

The use of handle manipulators has been exploited by some researchers (SORKINE et al., 2004; BOTSCH; KOBBELT, 2004, 2005) as a way to specify mesh deformations. This kind of modeling metaphor requires simple user interactions and provides a good interface for producing pleasant results with smooth deformations. However, the use of handles limits the amount of fine control over the deformation. Although complex deformations can be split into a sequence of simpler ones (SORKINE et al., 2004; BOTSCH; KOBBELT, 2005), the time required to setup each deformation makes this approach prohibitive for large meshes.

Nealen et al. (2005) proposed a sketch-based interface where the user defines a sketch curve which is transformed in a suggestive contour line. The deformation is carried on by adding new constraints to the work of (SORKINE et al., 2004) and thus this technique is not able to preserve mesh details when large rotations occur. Moreover, the work of Nealen et al. (NEALEN et al., 2005) focused on mesh local details; the technique presented in this work, in contrast, deals with both local and global deformations, which are not possible to perform using the technique proposed in (NEALEN et al., 2005).

In (ZHOU et al., 2005), Zhou et al. propose the creation of a volumetric graph extended from inside to outside of the mesh and perform the deformation using the Laplacian framework on this graph. This approach is able to preserve the volume of the mesh and avoid self-intersections. A WIRE (SINGH; FIUME, 1998) is used to deform a sequence of vertices that are used as constraints in the Laplacian system. However, as the graph has more vertices than the mesh, it compromises the overall performance of the system.

Skeletons are often used for performing mesh deformation (BLOOMENTHAL, 2002; DU; QIN, 2004; LIEN; KEYSER; AMATO, 2006; YAN; HU; MARTIN, 2006; YOSHIZAWA; BELYAEV; SEIDEL, 2003; BLOOMENTHAL; LIM, 1999). However, traditional skeleton-based methods require a labor-intensive task of weight selection to obtain satisfactory results (YAN; HU; MARTIN, 2006). To avoid this task, one can use the medial axis of an object mesh, which greatly simplifies the process of mesh deformation. Since the mesh can be represented with respect to the medial axis by using some distance field, one can perform the deformation on the medial axis and then reconstruct a deformed shape

using the distance field (YOSHIZAWA; BELYAEV; SEIDEL, 2003; BLOOMENTHAL; LIM, 1999; BLOOMENTHAL, 2002; DU; QIN, 2004). Unfortunately, medial axis extraction is a computationally expensive task (LIEN; KEYSER; AMATO, 2006; DU; QIN, 2004; YOSHIZAWA; BELYAEV; SEIDEL, 2003) and its use has not been applied to large meshes. One way of dealing with this limitation is to use multi-resolution techniques, as in the work of Yoshizawa et al. (YOSHIZAWA; BELYAEV; SEIDEL, 2003) but this may lead to a poor representation of details.

Recent work on geometric modeling has focused on ways to perform mesh deformation while maintaining some specific characteristics of the mesh, like volume preservation (FUNCK; THEISEL; SEIDEL, 2006; ZHOU et al., 2005; HUANG et al., 2006). The technique presented in this thesis does not handle volume preservation which is beyond the scope of this work. Some aspects of these techniques, however, will be discussed and used for comparison with the presented approach in Chapter 7.

3 DEFORMING NON-STRUCTURED 3D MODELS

This chapter introduces the main steps required for the deformation technique presented in this work, which is based on 2D user sketches and curve manipulation in 3D. User sketches are used to define curves and associate regions of the mesh to them so that these regions can be deformed as the user interacts with the curves. The user can also define skeletons by linking individual curves. Such skeletons give the user more control over a deformation and provides information about the global structure of the object. Twisting and scaling operations can be performed by operating directly on a frame field defined on the curves. In order to deform non-structured models at interactive rates, all the operations required by the technique must be performed as efficiently as possible.

The presented technique is composed by three main steps:

- **Segment the mesh into regions.** This step is responsible for recognizing the parts of the mesh under the user sketches, which constitute the regions of interest (ROI) to be deformed. This step is detailed in Section 3.1;
- **Construct parametric curves and skeletons to be used as handles.** Parametric curves are created from the user sketches, and used as handles for mesh deformation. Several parametric curves can be combined to form skeletons for the objects. This step is explained in Chapter 4;
- **Transfer the deformation from curves to meshes.** When the user manipulates the handles, the deformation performed on parametric curves are directly transferred to the mesh. This step is detailed in Chapter 5.

The rest of this chapter explains the region segmentation step.

3.1 Region Segmentation

This Section explains how the region segmentation is performed in order to assure interactive response even with meshes in the order of hundreds of thousands vertices. As this segmentation step is performed each time the user sketches on the screen (to create a new handle curve), it must be done efficiently to assure interactive response to the whole system. The use of 2D sketches is often adopted to create intuitive interfaces (IGARASHI; MATSUOKA; TANAKA, 1999; IGARASHI; HUGHES, 2001; CHERLIN et al., 2005; KHO; GARLAND, 2005). The technique presented in this work make use of this approach in a similar way as done by Kho and Garland (KHO; GARLAND, 2005).

Firstly, each sketched curve is filtered in order to remove the noise from the user input (Section 3.1.1). Points from the filtered sketch are then projected into the object surface

and a region growing algorithm is applied to find the region of the mesh under the user sketch (Section 3.1.2). The remainder of the mesh is also identified by a region-growing algorithm (Section 3.1.3).

Mesh vertices in the region under the user sketch will be directly transformed by a parametric curve (Chapter 5 explains how the deformation is done). In order to guarantee the continuity between the regions as the user interacts with a curve, two approaches can be considered. First, if the curve is not connected to any other curve, the regions associated to the extremities of the curve are rigidly transformed. Alternatively, the user can link several curves in order to create a skeleton to the object (Section 4.3). In this case, a blending function is used to smoothly stop the deformation between each curve (see Section 5.4 too see how this is done).

3.1.1 Pre-filtering the Sketch

A typical user sketch is defined by a set of connected points on the 2D image plane, captured from mouse movements (Figure 3.1 left). The noise present in a sketch usually results from lack of precision from the input device or user abilities. So, a filtering algorithm is executed on the set of input points in order remove the undesired noise. This situation is illustrated in Figure 3.1.

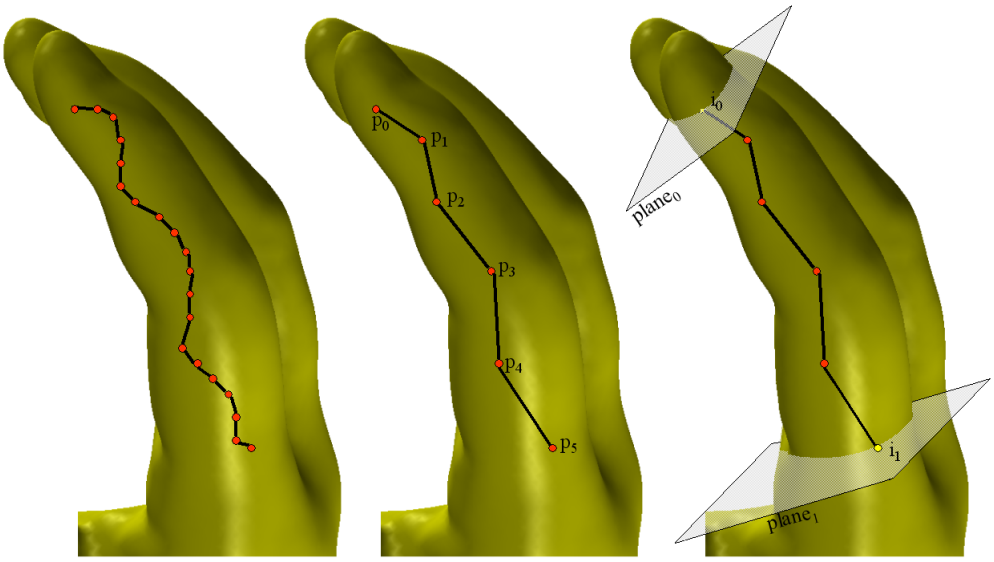


Figure 3.1: Smooth 2D sketch. Left: A typical user sketch, composed by a set of connected points. Center: The same sketch after the filtering algorithm. Right: Cutting planes created at the start and at the end of the sketch.

This filtering process is the same done in the work of Kho and Garland (KHO; GARLAND, 2005): the idea is to just take into account the segments of the sketch that represents a minimum movement δ of the mouse. Note that this process is done on the 2D space, so δ is measured in pixels. A pseudo-code for the algorithm used to filter the sketch is presented in Algorithm 1.

Algorithm 1 Filtering of 2D user sketch

Require: $points$ {List of points that define the sketch}
Require: δ {Minimum length of a sketch segment}

```
1:
2:  p0 ← points[0]
3:  newPoints.add(p0) {Add point 0 as the start of the filtered sketch}
4:
5:  for each point p from points, starting at points[1] do
6:     added ← 1
7:     pointSum ← p
8:
9:     p1 ← p
10:    sz ← distance(p0,p1)
11:
12:    for each point p from points not already visited do
13:       if sz >= δ then
14:          break
15:       end if
16:       added++
17:       pointSum ← pointSum + p
18:       p1 ← pointSum/added
19:       sz ← distance(p0,p1)
20:    end for
21:
22:    newPoints.add(p1)
23:    p0 ← p1
24:
25: end for
```

Using this approach, only segments with a minimum length δ are considered. A smooth version of the noisy user sketch in the left image of Figure 3.1 is shown at the center image of the same figure.

3.1.2 Region Affected by Sketch

Using the filtered sketch (center image of Figure 3.1), the first and the last points of the sketch (*i.e.* p_0 and p_5) are projected onto the object mesh, and two cutting planes are defined in the 3D space, as shown in the right image of Figure 3.1.

Let d be the camera view direction and let p_0 be the position of the mouse where the sketch started. A 3D ray is created by using p_0 and d and the first intersection point of this ray with the mesh is calculated using the approach described in (MOLLER; TRUMBORE, 1997). In the example of Figure 3.2, the intersection is shown as a small yellow triangular region on the left image. This procedure is also used to find the point on the mesh where the sketch has finished (*i.e.* p_5 on Figure 3.1). This leads to two intersection points on the mesh, which we call i_0 and i_1.

The intersection points and the directions of the first and the last sketch segment, used as normals, define two cutting planes for limiting the region of interest of the deformation. In Figure 3.1, plane $plane_0$ is defined at origin i_0 and has normal $p_1 - p_0$; plane $plane_1$'s origin is at i_1 and has normal $p_4 - p_5$. In the example of Figure 3.2, the two cutting planes

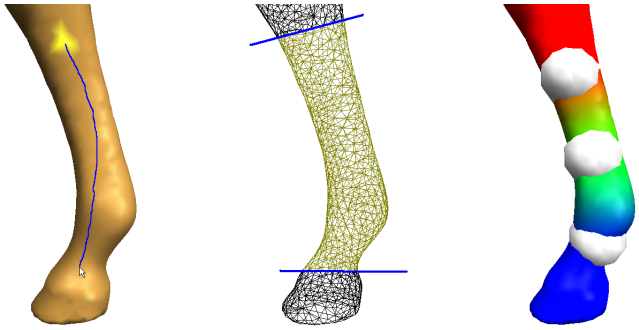


Figure 3.2: Mesh segmentation based on user sketch. Left: User sketch over the 3D model. Center: Two cutting planes (indicated by the blue lines) are created at the start and at the end of the sketch by calculating its intersection points from camera's viewpoint on the mesh and using the direction of the sketch segments as orientation. Right: Mesh segmentation. The colors represent the parameterization of the mesh according to the curve. Red and blue regions represent the start and the end of the curve while the in-between region is associated to the curve.

are shown as blue lines in the center image.

To find the vertices between $plane_0$ and $plane_1$, one has to evaluate the planes' implicit equations using the coordinates of the mesh vertices. If the signs of both evaluations are greater than zero the vertex is classified as being in the ROI (the region under the user sketch). In the example of Figure 3.2, these vertices are shown in yellow on the center image. A curve will be created using the user sketch (see Chapter 4), and these vertices will be directly transformed by this curve as the user interacts with it. The control points of the curve for the example of Figure 3.2 are shown on the right image as white spheres.

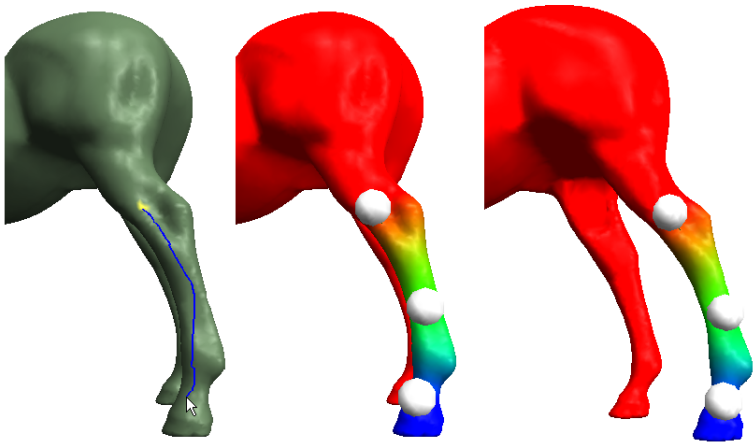


Figure 3.3: Mesh segmentation when two parts of the object project to the same portion of the image plane. Left: Two legs of the Horse model project to the same portion of the image plane. Center: The mesh segmentation algorithm can deal with such cases. Right: The same scene seen from a different viewpoint after region segmentation. Note that just the leg that was nearest to the view plane has been selected and used to create the curve.

Note that, by performing region segmentation based only on the cutting planes, parts of the object that project to the same portion of the image plane could be selected, even if they are disconnected within the region of interest. This case can be seen in the left image

of Figure 3.3, where the two legs of the horse model would be merged into a single region. To avoid this problem, a region-growing algorithm guided by the mesh topology is used. Since the legs of the horse are not topologically connected inside the region defined by the two cutting planes, only the leg closer to the camera is selected (as shown on the right image of Figure 3.3).

Algorithm 2 Region-growing algorithm to find the vertices affected by the user sketch

Require: t_0 {Triangle where i_0 lie}
Require: t_1 {Triangle where i_1 lie}
Require: $plane_0$ {First plane created from the user sketch}
Require: $plane_1$ {Last plane created from the user sketch}
{Create a stack $trianglesToExaminate$ to store the triangles where the test of planes' equations will be done}

$trianglesToExaminate.push(t_0)$ {Push t_0 into the stack}
$trianglesToExaminate.push(t_1)$ {Push t_1 into the stack}

while $trianglesToExaminate.size() > 0$ **do**
  $currentTriangle \leftarrow trianglesToExaminate.pop()$

  for each vertex v of the $currentTriangle$ **do**
    if $plane_0(v) >= 0$ and $plane_1(v) >= 0$ **then**
      Classify v as affected by the curve

      for each triangle t that uses vertex v **do**
        if t not in $trianglesToExaminate$ and t not already tested **then**
          $trianglesToExaminate.push(t)$
        end if
      end for

    end if
  end for
  {Identify boundaries to be returned}
  if there is a vertex from $currentTriangle$ classified and at least one not classified **then**
    for each unclassified vertex v **do**
      if $plane_0(v) < 0$ **then**
        Add vertex v as associated to $plane_0$ to the list of boundary vertices
      else
        Add vertex v as associated to $plane_1$ to the list of boundary vertices
      end if
    end for
  end if

end while
return List of boundary vertices and associated planes

Instead of testing all the vertices of the mesh against the planes' equations, only those

that are connected within the region of interest are tested. Consider $plane_0$ and $plane_1$ the cutting planes created at points i_0 and i_1, respectively, as depicted in Figure 3.1. Algorithm 2 finds the vertices of the mesh within the region defined by $plane_0$ and $plane_1$ avoiding the problem of classifying different parts of the object that project to the same portion of the image plane as shown in Figure 3.3. Note, however, that when dealing with meshes composed by multiple connected components, all vertices must be tested since although they may not be connected by the mesh topology within the region of interest they must be deformed as a single region.

3.1.3 Regions Not Affected by Sketch

The set of vertices around a tight neighborhood of the two cutting planes define the boundaries of the region of interest. Algorithm 2 identifies the vertices that will be directly affected by the parametric curve, *i.e.* the vertices of the mesh that were under the user sketch. The remaining vertices must be classified in order to associate them to the start or to the end of the curve. This classification is needed to keep continuity of the mesh during the deformations.

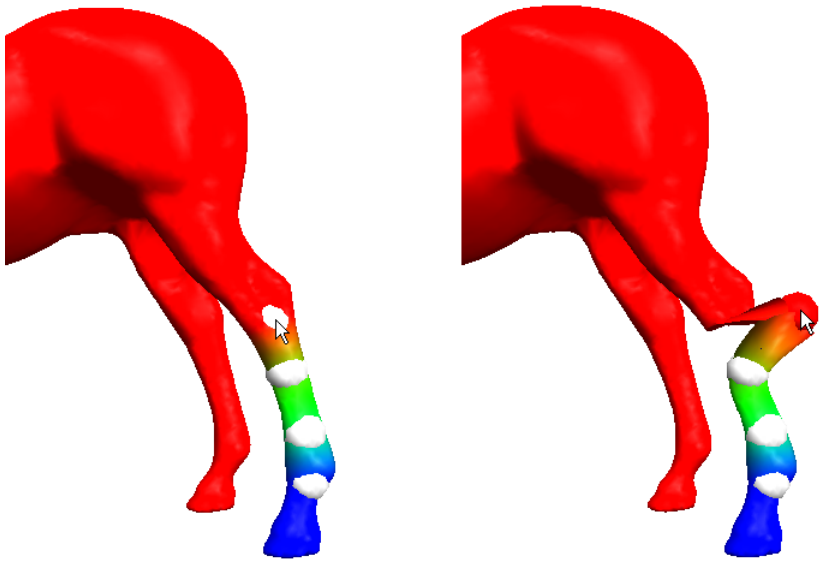


Figure 3.4: The effect of not providing a smooth transition between the regions affected by the curve and regions not affected by the curve. Left: Unmodified model. Right: The user interacts with the curve and only the vertices associated to it are transformed, introducing some discontinuity artifacts.

As the user interacts with the curve, the vertices of the segmented region of mesh are transformed. In order to avoid the occurrence of some discontinuity artifacts, the remainder of the mesh must also be transformed. Figure 3.4 illustrates this problem. To avoid this, a blending function should be used to smooth the transition between the regions (Section 5.4 presents details of the blending function used for this work).

The vertices that define a boundary and their respective cutting planes, are returned by Algorithm 2. Starting from them, another region-growing algorithm is performed to associate the remaining vertices to the start or to the end of the curve. We follow the mesh topology adding the vertices not in the ROI to the region created by each cutting plane.

The region-growing algorithm used in this step is simpler since it just needs to propagate the association of planes along the mesh. A pseudo-code for it is shown in Algorithm 3.

Algorithm 3 Region-growing algorithm to classify the remaining vertices

Require: *boundaries* {List of boundary vertices and associated planes}
{Create a stack *trianglesToExaminate* to store the triangles to examinate}
{Create a stack *associatedPlanes* to store the planes associated to *trianglesToExaminate*}

{Push into *trianglesToExaminate* the triangles that share any of the vertices in *boundaries*}
associatedPlanes.push(*boundaries.planes*) {Push associated planes into the stack}

while *trianglesToExaminate.size*() > 0 **do**
  currentTriangle ← *trianglesToExaminate.pop*()
  currentPlane ← *associatedPlanes.pop*()

  for each vertex *v* of the *currentTriangle* **do**
    if *v* is not already classified **then**
      Associate *v* to *currentPlane*

      for each triangle *t* that uses vertex *v* **do**
        if *t* not in *trianglesToExaminate* and *t* not already tested **then**
          trianglesToExaminate.push(*t*)
          associatedPlanes.push(*currentPlane*)
        end if
      end for

    end if
  end for

end while

In Figure 3.2, the first region-growing algorithm (Algorithm 2) is responsible for finding the vertices drawn using a predominant green color scale, which stops at the cutting planes. The second region-growing procedure (Algorithm 3) is responsible for creating the blue and red regions. This is done by following the mesh topology and adding any unclassified vertex to the region defined by the vertices on the outer sides of the cutting planes.

The above strategy will segment the object mesh in one region affected by the curve and two other regions, each one associated to one of the cutting planes (Figure 3.2). This association is used to define which parts of the mesh are to be modified by the parametric curve and which parts are to be rigidly modified by the orientation of the start or the end of the curve. This can be seen in Figure 3.2 where three regions were segmented: the upper part of the Horse leg (shown in red), the region of the mesh actually affected by the curve (color scale with predominant green) and the remainder of the leg (shown in blue).

Note that a rigid transformation is done only when the curve is not connected to other curves. As skeletons can be created by linking individual curves (see Section 4.3), a

blending function (explained in Section 5.4) is used to smooth the transition between regions controlled by different curves.

3.2 Digest

This chapter introduced the steps of the non-structured object deformation technique presented in this work and detailed the region segmentation step. When the user sketches over the image plane, a filtering algorithm is executed to reduce the noise that may be present in the user input. Using the filtered sketch, the mesh is segmented in three regions, one of them defining the region of interest (ROI) for deformation. Two region-growing algorithms, guided by the mesh topology, allow this segmentation to be performed efficiently.

The filtered sketch will be used to create parametric curves and skeletons in the next chapter. The segmented mesh regions will be deformed as the user interacts with these parametric curves.

4 CONSTRUCTING PARAMETRIC CURVES AND SKELETONS FROM SKETCHES

In this work, object deformation can be performed using two types of curves: *surface curves* and *skeleton curves*, that differ from each other based on the place where they are instantiated. Surface curves (Section 4.1) are positioned on the object surface. They are useful for deforming thick regions of an object because they can stay visible during deformations. Skeleton curves (Section 4.2) are placed inside the mesh and are useful to represent global structures for objects. This chapter explains how these curves are created from 2D user sketches and discusses the differences between them.

Individual curves can also be linked in order to create skeletons for the models (Section 4.3). The use of skeletons provides an alternative way of guaranteeing the continuity of the mesh during deformations. Instead of rigidly transforming the regions of the mesh not in the ROI, a blending function is used to transition the deformation of each curve restricted to its ROI.

4.1 Surface Curve

When the user draws a sketch on the image plane the following steps are performed to produce a curve on the object's surface:

- Filter the 2D sketch to smooth the noisy user input (Section 3.1.1);
- Get equally-spaced points in the sketched curve and project them on the mesh;
- Use the projected points as control points for creating a parametric curve.

Firstly, the sketch is filtered as described in Section 3.1.1. A parameterization is used to get equally-spaced points from the filtered sketch. This parameterization is based on the length of each segment and a pseudo-code for it is given by Algorithm 4.

Algorithm 4 Parameterizing 2D sketches

Require: *points* {List of points from the pre-filtered sketch}

```
1:
2: distances[0] ← 0
3: totalDistance ← 0
4: for i ← 1 to points.size() do
5:    distances[i] ← distance(points[i−1].position,points[i].position)
6:    totalDistance ← totalDistance + distances[i]
7: end for
8:
9: accumulatedDistance ← 0
10: for i ← 0 to points.size() do
11:    accumulatedDistance ← accumulatedDistance + distances[i]
12:    points[i].t ← accumulatedDistance/totalDistance
13: end for
```

Let n be the number of points the user wants to use to approximate the sketch. To retrieve n equally-spaced points from the sketch, one has to linearly discretize the interval [0-1] with n values and use the Algorithm 5, which returns the desired points. Each equally-spaced point is projected onto the mesh using the method described in (MOLLER; TRUMBORE, 1997). The resulting points on the mesh are used to create an interpolating parametric curve. Note that the first and the last points of the sketch can be used without this sketch parameterization step because their projected positions on the mesh were already calculated, as described in Section 3.1.2.

Algorithm 5 Getting equally-spaced points from the sketch

Require: *points* {List of points from the pre-filtered sketch}
Require: t {Parameter t at which to return a point}

```
1:
2: for i ← 0 to points.size() − 1 do
3:
4:    if t >= points[i].t and t <= points[i+1].t then
5:
6:       localt ← (t − points[i].t)/(points[i+1].t − points[i].t)
7:
8:       return points[i].position ∗ (1 − localt) + points[i+1].position ∗ localt)
9:    end if
10:
11: end for
12:
```

The proposed approach to parameterize the sketch works well in practice, leading to an effective way to approximate the user sketch with few control points in the parametric curve. Figure 4.1 shows a few examples of user sketched curves and the respective approximating parametric curves. Note that the control points are equally-spaced in the parameter space and the resulting splines approximate well the user sketches.

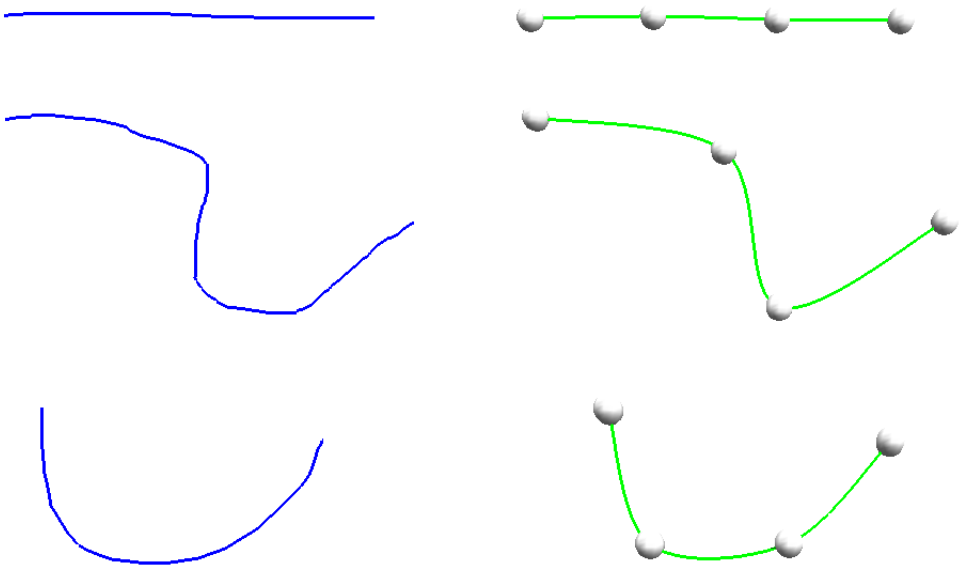


Figure 4.1: Sketch discretization. Left: examples of typical user sketched curves. Right: Parametric curves generated from the corresponding user sketches. Note that the control points are equally-spaced in parameter space and the resulting splines approximate well the 2D sketched curves.

4.2 Skeleton Curve

Skeleton curves can represent the structure of an object and be used to globally deform it. To create a skeleton curve, a surface curve is first created as described in Section 4.1. Thus, let k be the number of control points of a surface curve C. A skeleton curve S is obtained from C using the following steps:

1. Create $k-1$ planes perpendicular to C, each one halfway two consecutive control points. The planes are obtained by evaluating C and its first derivative at corresponding parameter values;

2. For each of the k subspaces delimited by the $k-1$ planes, compute the centroid of the vertices of the region of interest falling in that subspace;

3. Use these centroids as the control points for a Catmull-Rom skeleton curve.

Note that the test needed to find the vertices between each pair of planes is very efficient because one just needs to evaluate the vertex position in the equations of the planes, which reduces to two dot products. Note that only the vertices previously segmented (see Section 3.1) are tested. This is to prevent the centroid calculation to take into account separate regions of the object that project to the same portion of the image plane and also helps to improve the overall performance of the technique. Figure 4.2 shows a surface curve (left) and the distribution of mesh vertices for the planes created between each pair of control points (right). The light blue regions are not taken into account because they were not under the original user sketch.

Figure 4.3 shows the skeleton curve created using the surface curve showed in Figure 4.2. Note that the control points are inside the object mesh and the curve is a reasonable approximation for the skeleton of that region of the mesh (Figure 4.3 right). The control

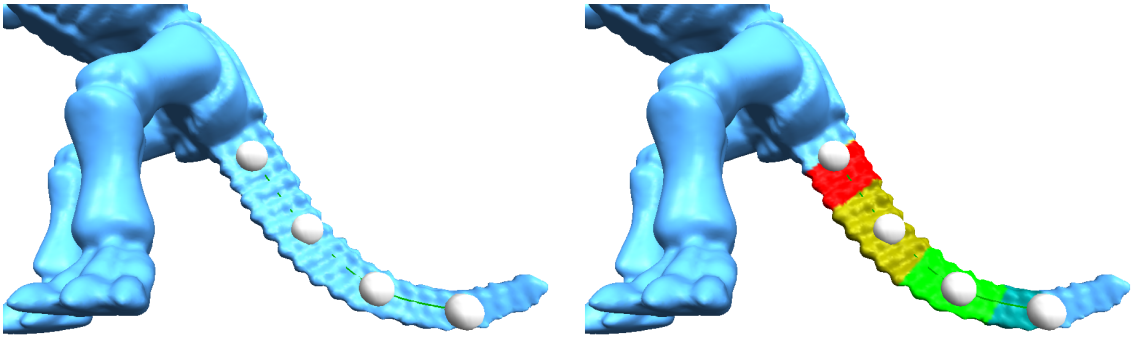


Figure 4.2: Skeleton curve creation step one. Left: Surface curve. Right: Planes are created between each pair of control points in order to create a skeleton curve. The color scale shows the distribution of the mesh vertices between each pair of planes.

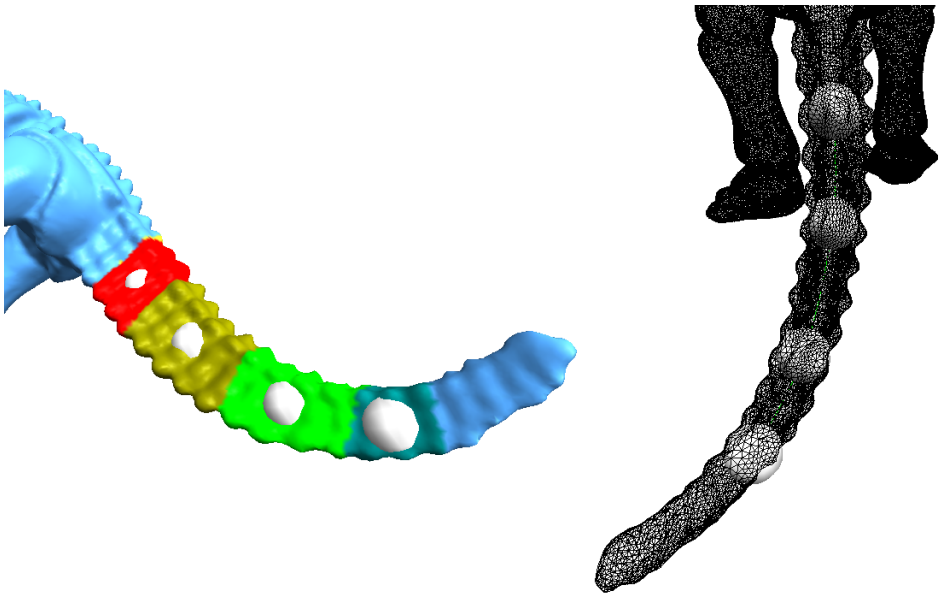


Figure 4.3: Skeleton curve creation step two. Left: Control points are created by calculating the centroid of the vertices between each pair of planes. Right: The created skeleton curve viewed using wireframe rendering. Note that the curve is inside the object mesh.

points were calculated as the centroid of their respective colored subspaces. Note that the first and last subspaces are half the size of the in-between subspaces. This occurs because they comprise just one half, while other subspaces comprises two halves of the space between each control point of the surface curve.

4.3 Creating a Structured Skeleton

Several parametric curves can be linked by simply clicking on their control points. Curves can also be merged using the same interaction approach. This leads to a simple but effective interface for skeleton creation. The skeleton of Figure 4.4 was created in a few seconds using this technique. The creation of skeletons is useful because they represent the global "structure" of an object. For this reason they are often used to interpolate between different poses in an animating system. Moreover, it provides the user the possibility to edit only the skeleton without even visualizing the object mesh.

Figure 4.4: Skeleton creation. Left: Original model. Right: A skeleton created using the technique presented in this work. Several parametric curves are created from 2D user sketches and then linked by simply clicking on control points.

When creating a skeleton, the user is free to mix surface curves and skeleton curves, choosing the one he or she judges more appropriate for each part of the mesh. The curves' effect remain local to the region where the original sketch was done and a blending function is used to keep the mesh continuity along parts of the mesh controlled by different curves (see Section 5.4).

The use of skeletons helps an artist to concentrate on the deformations he or she wants to perform. After the creation of the skeleton, the mesh can be interactively modified. This explains why skeletons are often used for other applications such as deformation or animation retargeting.

In this work, the use of skeletons introduces a new treatment for the vertices not in the ROI. Using the mesh topology, a blending function is used to keep the mesh continuity by smoothly transitioning the deformation from inside the ROI to outside it.

4.4 Digest

This chapter has explained how to create parametric curves from 2D sketches. The process of creating surface curves and skeleton curves was detailed. Surface curves are created by projecting equally-spaced points from the filtered sketch and using their projections as curves' control points. Skeleton curves are created from surface curves by computing the centroids of the vertices falling between each pair of surface curves' control points.

Skeletons are created by linking individual curves and provide good representations for the global structures of objects. By creating skeletons, a user can modify the object while treating its parts individually.

Next chapter explains how the curves and skeletons created in this chapter can be used as handles to perform mesh deformation.

5 MESH DEFORMATION

The use of parametric curves as manipulation handles provide an intuitive interface for specifying deformations. A lot of information about a deformation can be given in simple ways, either by moving the curve control points or by specifying parameters on them (see Section 5.3). Moreover, the use of a 3D curve instead of another 2D sketch, like was done by Kho and Garland (KHO; GARLAND, 2005), allows the user to make fine adjustments to the deformations and adds the possibility to change the camera's viewpoint during an editing session.

This chapter explains how the deformation of the curve can be transferred into mesh deformations in an easy and efficient way. Briefly, local frames are created along the curve and the coordinates of the mesh vertices are represented with respect to these local frames. As the user modifies the curve, the local frames are modified causing the associated surfaces to be deformed.

The simplicity of the formulation adopted by the presented technique allows the construction of an efficient algorithm to avoid local self-intersections of the mesh during deformations.

5.1 Defining Local Frames

In order to transfer the deformation of a parametric curve to the mesh, the curve is instrumented with a set of local frames, whose orientation must be kept consistent with the other frames along the curve all the time. This consistency is important because the set of local frames define the mesh of the object and any lack of consistency on them would be directly transferred to the mesh. Frenet frames (CARMO, 1976) are quite intuitive and can be computed analytically. Unfortunately, Frenet frames are not defined at inflection points or along straight segments. Moreover, at inflection points, Frenet frames can undergo some violent twists (BLOOMENTHAL, 1990).

In order to avoid these problems, local frames are propagated along the curve using the following strategy: Let $\vec{u_0}$ be the curve's unit tangent vector at parameter value $t = 0$ (point p_0). The second vector ($\vec{v_0}$) of the frame at $t = 0$ is obtained by projecting $\vec{u_0}$ onto the world XZ plane and then normalizing and rotating the projection $\vec{u_{0p}}$ by 90 degrees (Figure 5.1). The third vector, $\vec{w_0}$, is obtained as the cross product $\vec{u_0} \times \vec{v_0}$. If $\vec{u_0}$ coincides with the vector $(0,1,0)$ in the world coordinate system, this procedure would fail because the projection $\vec{u_{0p}}$ is null. To avoid this problem, the components of $\vec{u_0}$ are inspected and the projection is performed on a world plane that would not lead to a null vector, by appropriately choosing one of the planes XY or XZ. This strategy is shown in Algorithm 6. Note that there is no need to use the YZ plane because there is no case in the 3D space where a vector has null projection on both XY and XZ planes.

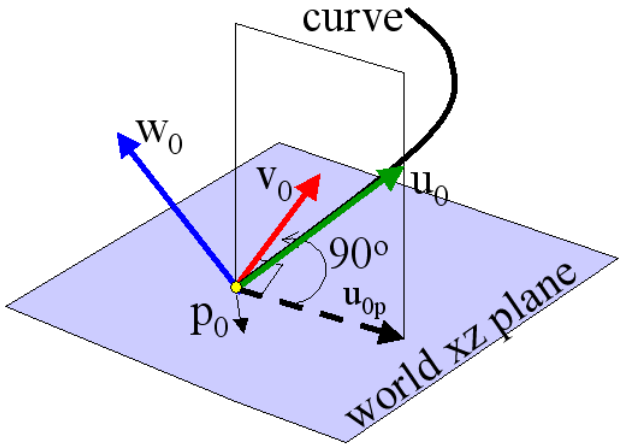


Figure 5.1: The creation of a local frame at $t = 0$. The vector $\vec{u_0}$ is the curve's unit tangent vector at $t = 0$. $\vec{v_0}$ is obtained by normalizing and rotating by 90 degrees the projection $\vec{u_{0p}}$ of $\vec{u_0}$ onto the XZ plane. $\vec{w_0} = \vec{u_0} \times \vec{v_0}$.

Algorithm 6 Finding which world plane is to be used for projection

Require: y {y vector of the local frame to be created}

1: $xc \leftarrow$ x component of y
2: $yc \leftarrow$ y component of y
3:
4: **if** $|xc| \geq |yc|$ **then**
5: **return** world XZ plane
6: **else**
7: **return** world XY plane
8: **end if**

Once a frame has been created at $t = 0$, one can vary t in the interval [0,1] to define new local frames along the curve. The curve position p_i and unit tangent vector $\vec{u_i}$ at $t = t_i$ define an implicit plane at p_i. We then project the vector $\vec{v}_{i-1}$ (from the previous frame) onto this new plane, obtaining $\vec{v_i}$, after the projection has been normalized. $\vec{w_i}$ is again obtained as $\vec{w_i} = \vec{u_i} \times \vec{v_i}$. An additional operation consists in switching the sign of $\vec{w_i}$ in case the angle between $\vec{w_i}$ and $\vec{w}_{i-1}$ is bigger than 90 degrees.

This approach leads to a set of local frames along the curve that minimizes the occurrence of undesirable twists. Figure 5.2 compares the results obtained by analytically computing a Frenet frame field along the parametric curve (top) with the results produced by the presented approach (bottom). Note that the results obtained by the propagation algorithm are much smoother, avoiding discontinuities in the field as a result of torsion. Let dc and ddc be the first and second derivatives of the spline curve, respectively. The Frenet frames shown in Figure 5.2 were computed using Equations (5.1)-(5.3), where $\times$ is the cross product operator.

$$x(t) = \frac{ddc(t)}{|ddc(t)|} \tag{5.1}$$

$$y(t) = \frac{dc(t)}{|dc(t)|} \tag{5.2}$$

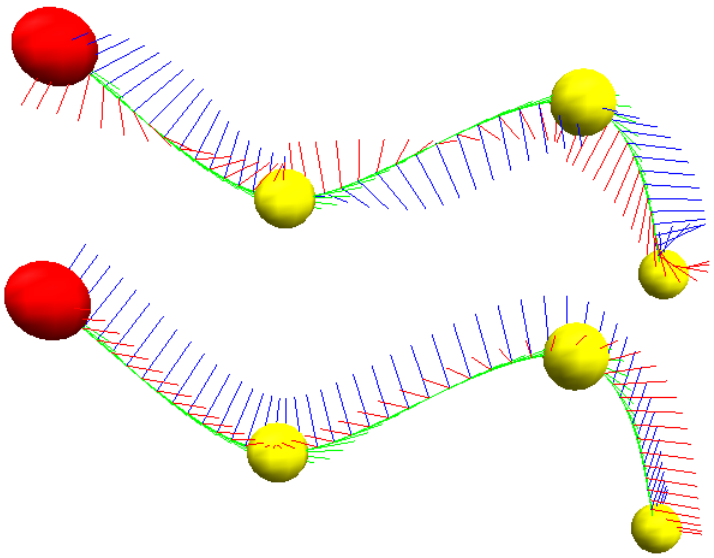


Figure 5.2: The creation of a frame field along the curve. Top: Frenet frame. Bottom: The approach presented in this work. Note that the frames do not suffer from sudden rotations as in the case of Frenet frames.

$$z(t) = \frac{x \times y}{|x \times y|} \tag{5.3}$$

5.2 Associating a Curve to a Mesh

Let v be a mesh vertex. Given the set of planes along the reference curve, one can associate each mesh vertex v to the pair of planes delimiting the space region where v lies. The color scale of the mesh in Figure 3.2 represents the plane association distribution. The identification of these regions can be performed very efficiently since one needs to test only the vertices of the mesh that were previously classified as being affected by the curve (Section 3.1.2).

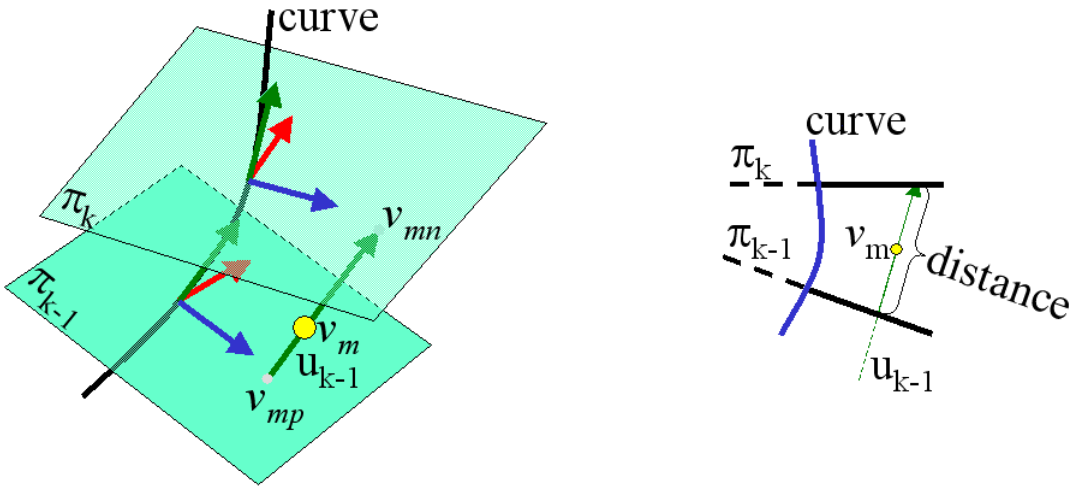


Figure 5.3: Plane parameterization. Left: The vertex v_m is associated to planes π_{k-1} and π_k. During deformations, the position of vertex v_m is maintained in the same relative position along $\vec{u_{k-1}}$ with respect to its projections v_{mp} and v_{mn} on π_{k-1} and π_k, respectively. Right: The same configuration in an orthographic view.

Given the set of frames along a reference curve C, one can represent the coordinates of each vertex in the region of interest of C in terms of its frames. Thus, let π_{k-1} and π_k be the planes spanned by the pairs of frame vectors $(\vec{v}_{k-1}, \vec{w}_{k-1})$ and $(\vec{v}_k, \vec{w}_k)$, respectively (Figure 5.3). All vertices v_j delimited by π_{k-1} and π_k will have their coordinates expressed with respect to the frame $F_{k-1} = (\vec{u}_{k-1}, \vec{v}_{k-1}, \vec{w}_{k-1})$.

Let $(\alpha_m, \beta_m, \gamma_m)$ be the coordinates of vertex v_m expressed in terms of frame F_{k-1} (*i.e.*, $v_m = p_{k-1} + \alpha_m \vec{u}_{k-1} + \beta_m \vec{v}_{k-1} + \gamma_m \vec{w}_{k-1}$). Also let v_{mp} and v_{mn} be the projections of v_m onto the planes π_{k-1} and π_k, respectively. Such projections are obtained as the intersections of the line defined by v_m and the vector $\vec{u}_{k-1}$ with the planes π_{k-1} and π_k, respectively. We then associate to v_m the ratio r_m as defined by Equation 5.4, computed based on the undeformed curve (Figure 5.3 right). *dist* is the Euclidean distance between two points in 3D. This ratio will be preserved during deformations.

$$r_m = dist(v_m, v_{mp}) / dist(v_{mn}, v_{mp}) \tag{5.4}$$

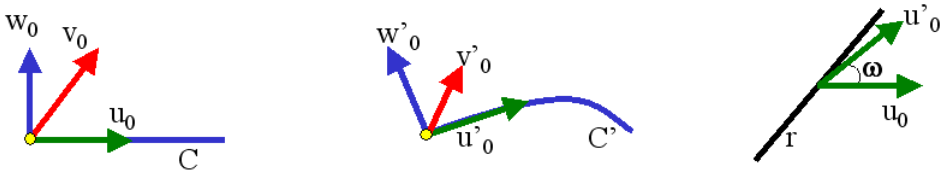


Figure 5.4: Creation of the modified set of local frames. Left: a parametric curve C with the initial local frame defined as axes $\vec{u}_0$, $\vec{v}_0$ and $\vec{w}_0$. Center: The user interaction defines a modified curve C', and a new initial local frame defined by axes $\vec{u'}_0$, $\vec{v'}_0$ and $\vec{w'}_0$ must be computed. Right: $\vec{u'}_0$ is created from the spline first derivative. The $\vec{v'}_0$ and $\vec{w'}_0$ axes are created by rotating $\vec{v}_0$ and $\vec{w}_0$ by ω degrees around the r axis defined by a cross product of $\vec{u}_0$ and $\vec{u'}_0$.

When the user modifies a parametric curve C, its set of frames are recomputed for the same values of the parameter t. Let C' be such a deformed curve. The first frame of C' cannot be created using the same approach used for C, because C''s tangent vector at $t = 0$ could require a different world plane for projection, (see Section 5.1), causing the sets of frames from C and C' to completely diverge. Thus, let $(\vec{u}_0, \vec{v}_0, \vec{w}_0)$ be the coordinate frame of C at $t = 0$ and let $\vec{u'}_0$ be the unit tangent vector of C' at $t = 0$. Also, let ω be the angle between the vectors $\vec{u}_0$ and $\vec{u'}_0$ and let $\vec{r} = \vec{u}_0 \times \vec{u'}_0$. Thus, the vectors $\vec{v'}_0$ and $\vec{w'}_0$ are obtained from $\vec{v}_0$ and $\vec{w}_0$, respectively, by simply rotating them around $\vec{r}$ by ω degrees. Once the first frame of C' has been defined, its remaining frames are obtained using the same procedure defined for the subsequent frames of the original curve.

Note that, as the curve differs from the initial curve, the planes and coordinate systems will also differ. Using this updated set of coordinate systems, modified positions for the vertices can be retrieved by positioning them in the same relative position on the new coordinate system. Given the set of frames of curve C', the new 3D coordinates of v_m after deformation could be computed simply as $v'_m = p'_{k-1} + \alpha_m \vec{u'}_{k-1} + \beta_m \vec{v'}_{k-1} + \gamma_m \vec{w}_{k-1}$, where p'_{k-1} is the point on C' for which $t = t_{k-1}$. This, however, would not take into account possible stretching applied to the curve when control points are moved apart. In order to transfer the stretching to the mesh, we scale v'_m's component along the u'_0 direction according the current distance between π'_{k-1} and π'_k times the ratio r_m (Equation 5.4). Thus, let v'_{mp} and v'_{mn} be the projections of the deformed vertex onto the planes π'_{k-1} and

π'_k, respectively. v'_{mp} is given by Equation 5.5. v'_{mn} is found by calculating the intersection of the line defined by v'_{mp} and $\vec{u'}_{k-1}$ with π'_k (Figure 5.1). The new 3D coordinates of v_m are then recomputed as v'_m, given by Equation 5.6. This scaling factor also plays an important role in the self-intersection avoidance algorithm that will be presented in Section 5.5.

$$v'_{mp} = p'_{k-1} + \beta_m \vec{v'}_{k-1} + \gamma_m \vec{w}_{k-1} \tag{5.5}$$

$$v'_m = v'_{mp} + \alpha_m(r_m) dist(v'_{mp}, v'_{mn}) \vec{u'}_{k-1} \tag{5.6}$$

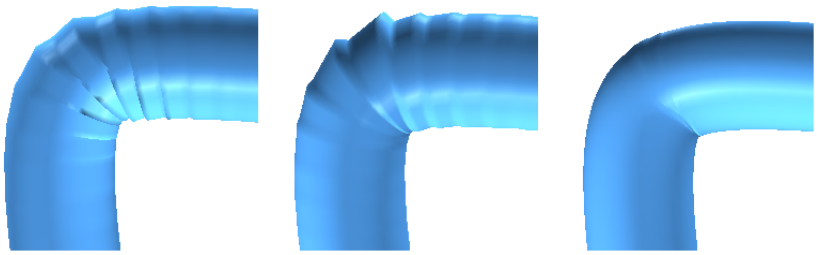


Figure 5.5: Mesh artifacts due to discretization. Left: Mesh artifacts created by using few (20) frames along the curve. Center: A similar deformation using the same number of frames but keeping the same distance ratio. The use of the scaling factor results in a smoother deformation. Right: The use of several planes (200) and the scaling factor results in a very smooth deformation while still achieving interactive rates.

The number of local coordinate systems used along the curve is very important to guarantee the smoothness of the deformation. If very few planes are used, the curve behavior may not transfer well to the deformed mesh. The scaling factor is helpful to avoid the creation of discontinuity features along the mesh deformation due to discretization. However, as the operations described above can be computed efficiently, several local frames can be used while still achieving interactive rates. Figure 5.5 shows the effect of using few local frames (left), the smoother version achieved by using the same number of local frames but using the scaling factor (center) and the smooth deformation achieved by using more coordinate systems.

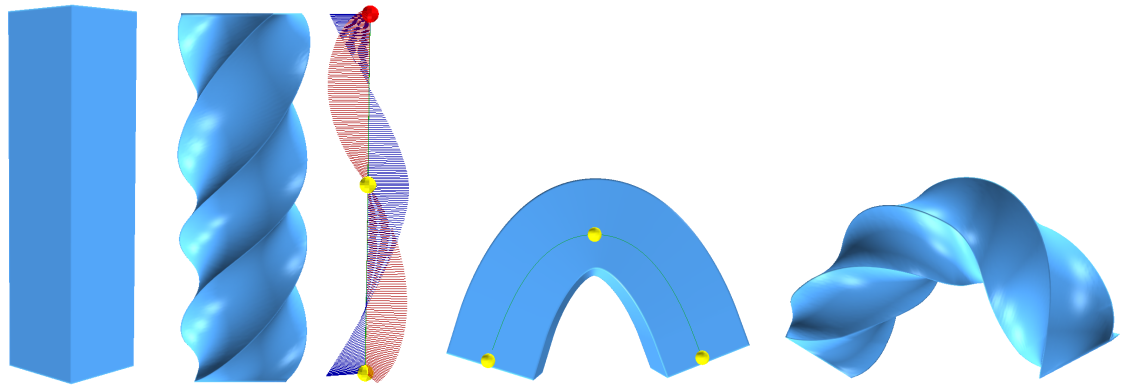
In fact, the number of coordinate systems could be calculated taking into account the size of the triangles in the region of interest. Note that the curve is discretized by a set of frames. The vertices falling between each pair of frames are deformed according to one frame and the relation to the other one is maintained by using the scaling factor. Optimally, each vertex should lie in a plane spanned by a local frame. This would guarantee all the vertices to be transformed by using only the curve, and thus avoid possible artifacts due to the discretization used. However, in all examples shown in this work (except for the left and center images of Figure 5.5), 200 coordinate systems were used for each curve, which provides very smooth deformations even for the very small triangles found in large meshes.

5.3 Twisting and Scaling

Some interesting effects, such as twisting and scaling, are obtained by operating directly over the frame field of the curve. For instance, by interpolating, along a segment

of C, a local rotation of the frames vectors $\vec{v}_i$ and $\vec{w}_i$ around $\vec{u}_i$ produces some twisting effect. Non-uniform and localized scaling effects can be achieved by independently scaling the vectors that form the frames. All these operations can be combined to create more complex deformations.

The user can interactively specify rotation angles for the local coordinate systems at each curve control point. The angle specified at a control point is linearly distributed along the curve to the neighbors control points. The user can also select more neighbors control points to specify an area of influence for the rotation. The rotation angle is then linearly interpolated along the curve to the selected neighbors control points, resulting in a twist effect. Scale factors can also be specified by the same interaction approach.



(a) (b) (c) (d)

Figure 5.6: Twisting and bending operations performed with our technique. A reference block (a). Twisted block obtained by interpolating a rotation along the curve's frame field (b). (c) Bent block obtained by moving the curve's control points (shown over). (d) Twisted and bent block combining the transformations (b) and (c).

As the rotation is always performed around the $\vec{u}_i$ axis of the local coordinate system, this operation results in a twisting deformation. By scaling the axes of the local frames, the effect of inflating or deflating the mesh is achieved. Figure 5.6 illustrates the results produced by twisting and bending operations. Figure 5.6 (a) shows a block as the original mesh. A twisted version of the block is shown in (b) and was obtained simply by interpolating a rotation along the frame field of the handle curve, which is shown to its right. Figure 5.6 (c) shows the result of bending the block, obtained by moving the curve's control points (shown over). In (d) one sees the combined result of twisting and bending the block. Note the smooth results.

The Figure 5.7 shows a deformation of the Bunny model where twisting and scaling were applied to the right ear. Note that these operations are simple but can produce very pleasingly results yet operating at interactive rates.

5.4 Blending Between Curves

To avoid discontinuity along mesh regions affected by different curves, a blending function is used to transition among the deformations defined by each parametric curve. The skeleton connectivity is used to define such a function, where a zero-mean Gaussian function ($b(x) = e^{-x^2/(2\sigma^2)}$) is associated to each control point linked to another curve (note that each control point can be linked to more than one curve). The standard deviation σ of the Gaussian controls the width of the function. Let p be a control point where a

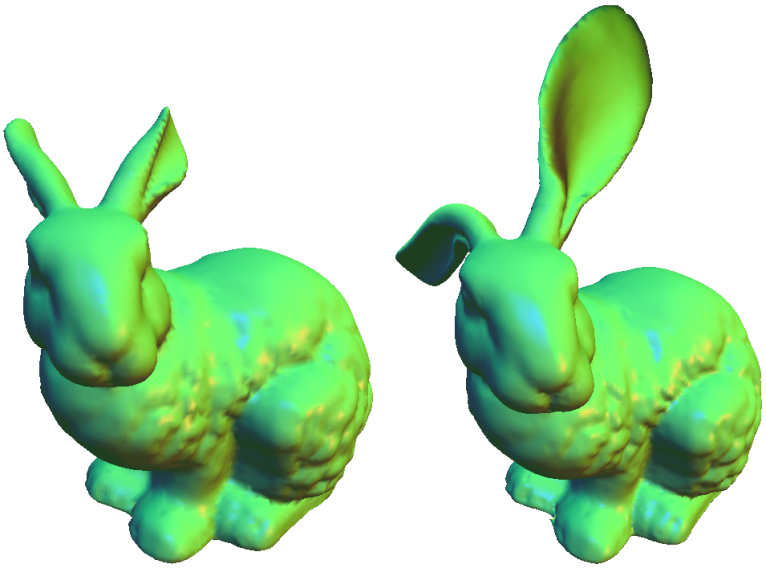


Figure 5.7: Twisting and scaling operation on the Bunny model. Left: Original model. Right: The left ear was modified and twisting and scaling operations were applied to the right ear.

Gaussian function is to be defined. Let L be the set of control points linked to p and let a be the average position of L. In the left image of Figure 5.8, one can think of p as the red control point and L as a set containing the white control points linked to the red one. In our experiments, we found that a Gaussian standard deviation $\sigma = \sqrt{dist(p,a)}$ works fine, where *dist* is the Euclidean distance function. The reason for this setting is to provide a smooth transition from the deformation defined by an interaction with the control point p to the regions of the mesh affected by other curves.

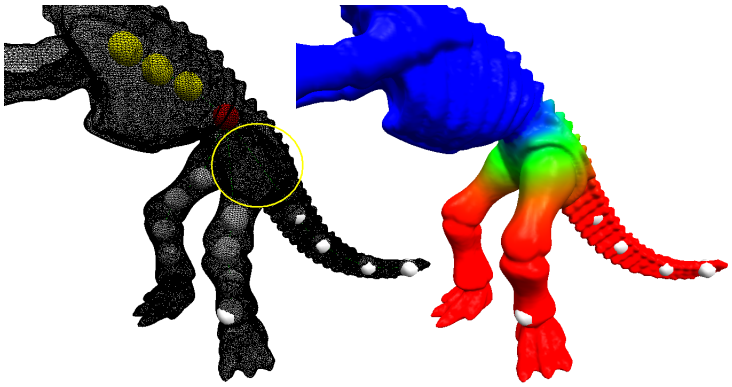


Figure 5.8: Blending curves. Left: The skeleton. Note that the region inside the yellow circle is not controlled by any curve. Right: The blending function centered at the red point of the highlighted curve shown on left. The color scale represents the smooth parameter obtained with the Gaussian function, where 1 is mapped to blue and 0 is mapped to red. Note the smooth transition from 1 to 0 between the highlighted curve (yellow and red points on the left) and the other curves (white control points).

Let v_m be a vertex outside the ROI of curve C. In order to guarantee a smooth transition between the deformed and non-deformed regions of the mesh, we compute the influence of p over v_m as the weight $w_m = b(dist(v_m, p))$. Thus, during the deformation induced

by C, the new coordinates of v_m are given by Equation 5.7, where v_{rm} are the coordinates of v_m if it had been rigidly transformed according to the transformation applied to plane π_j (defined by the frame vectors $\vec{v}_j$ and $\vec{w}_j$ at p_j). Since the function $b(dist(v_m, p))$ quickly approaches zero as the distance between v_m and p increases, we achieve a smooth transition between the two regions.

$$v_m = (1 - w_m)v_m + w_m v_{rm} \tag{5.7}$$

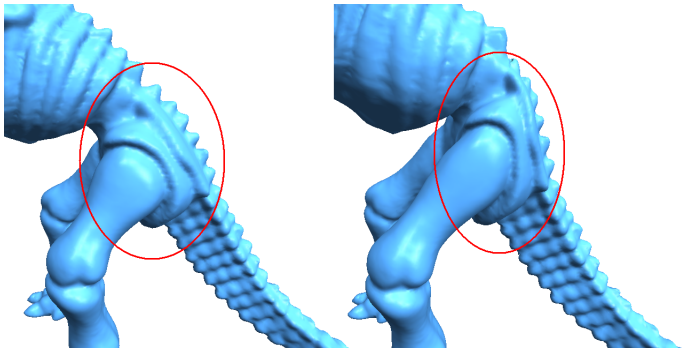


Figure 5.9: A smooth deformation achieved by using the blending function. Left: The original model. The region inside the red ellipse is not controlled by any curve. Right: The model is deformed by modifying the highlighted curve shown on Figure 5.8. Note that the deformation defined by the curve smoothly decreases for the vertices far from it. The region inside the red ellipse, despite of not being controlled by any curve, is deformed to keep the mesh continuity by using the blending function.

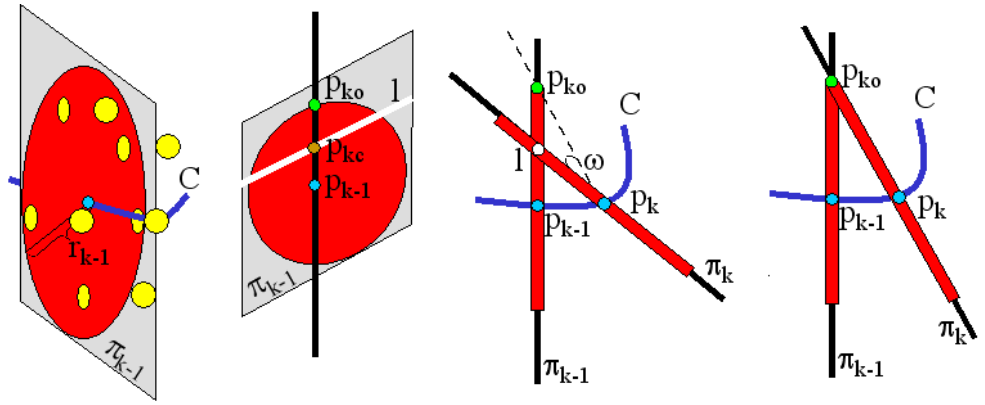
The smoothing parameter is calculated when the user links a control point to another curve so that when the user performs a deformation, the smoothing effect can be achieved by a simple linear interpolation avoiding the need to evaluate a Gaussian function for each frame.

The color scale shown on the right part of Figure 5.8 illustrates the smoothing parameter obtained by the Gaussian blending function for the curve shown with red and yellow control points on Figure 5.8 (left). Figure 5.9 shows a deformation on the Dino model where the smoothing parameter plays an important role. The curve controlling its torso was modified and the mesh between the torso and the legs (inside the yellow circle on Figure 5.8) was smoothly modified to keep the mesh continuity.

5.5 Avoiding Local Self-Intersections

Self-intersection is a common problem in mesh deformation and several approaches for trying to avoid it have been proposed, especially in the context of character skinning (MOHR; TOKHEIM; GLEICHER, 2003). We take advantage of the curve's frame field to devise a simple but effective way of avoiding local self-intersections. Let V_k be the set of vertices in the ROI of curve C that are represented in the coordinate system of C's frame ($\vec{u}_k$,$\vec{v}_k$, $\vec{w}_k$). The projection of V_k onto π_k defines a circle of influence for V_k (Figure 5.10). By guaranteeing that the circles of influence of all such planes do not intersect, self-intersections on the deformed mesh are avoided.

Let π_{k-1} and π_k be two such planes with origins at p_{k-1} and p_k created along the curve at parametric values t_{k-1} and t_k, respectively. Let l be the intersection line between them.



(a) (b) (c) (d)

Figure 5.10: Avoiding local self-intersections. (a) The projection of the vertices associated to a plane π_{k-1} defines a circle of influence with radii r_{k-1}. (b) l is the line of intersection of π_{k-1} and π_k. p_{kc} is the point on l closest to p_{k-1}, the point on C evaluated at $t = t_{k-1}$. p_{ko} is a point outside the circle of influence of π_{k-1}. (c) The plane π_k is rotated by ω degrees around the line parallel to l passing through p_k. (d) After rotation, where self-intersections no longer occur.

Also, let r_{k-1} and r_k be the radii of the circles of influence at π_{k-1} and π_k, respectively. If $dist(l,p_{k-1}) < r_{k-1}$ and $dist(l,p_k) < r_k$ then an intersection is assumed. To avoid self-intersections, we rotate the plane π_k so that the circles of influence in π_{k-1} and in π_k do not self-intersect anymore. Thus, let p_{kc} be the point on l that is closest to the p_{k-1}. Also, let p_{ko} be another point along the segment connecting p_{k-1} and l passing through p_{kc}, such that p_{ko} is the closest point to p_{k-1} outside the circle of influence of π_{k-1} (Figure 5.10 b). Note that the norm of the vector $p_{kc} - p_{k-1}$ is the $dist(l,p_{k-1})$. If this vector is scaled so that its norm equals r_{k-1}, a valid point p_{ko}, which is outside the circle of influence of π_{k-1} is found. This is done analytically by Equation 5.8.

$$p_{ko} = p_{k-1} + (p_{kc} - p_{k-1}) * \frac{r_{k-1}}{|p_{kc} - p_{k-1}|} \tag{5.8}$$

Let ω be the angle between the vectors $\vec{a}_c$ and $\vec{a}_o$, defined as $\vec{a}_c = (p_{kc} - p_k)$ and $\vec{a}_o = (p_{ko} - p_k)$. Self-intersection is avoided by rotating the frame of π_k by ω degrees around the direction of line l passing through p_k (Figures 5.10 c and d). Note that in order to avoid self-intersection, we have forced the vector $\vec{u}_k$ of the local frame not to coincide with the curve's tangent direction at $t = t_k$.

In order to guarantee that local self-intersections do not occur in the mesh regions attached to a curve, all coordinate systems must be tested. So, for a plane π_k, the test must be done $\forall \pi_j, j < k$.

Note that in order to avoid self-intersections, we have forced the vector $\vec{u}_k$ of the local frame not to coincide with the curve's tangent direction at $t = t_k$. This may become a problem when large rotations on the frames are needed to avoid self-intersections, which would cause the mesh to be unintuitively deformed. However, if the curve is edited again to a position where self intersections do not occur anymore, $\vec{u}$ will match the curve's tangent again.

Figure 5.11 shows two similar deformations. The deformation on the left does not use the presented algorithm to avoid local self-intersections, while the deformation on the right shows the benefits of using it. The details (center) show that the deformation

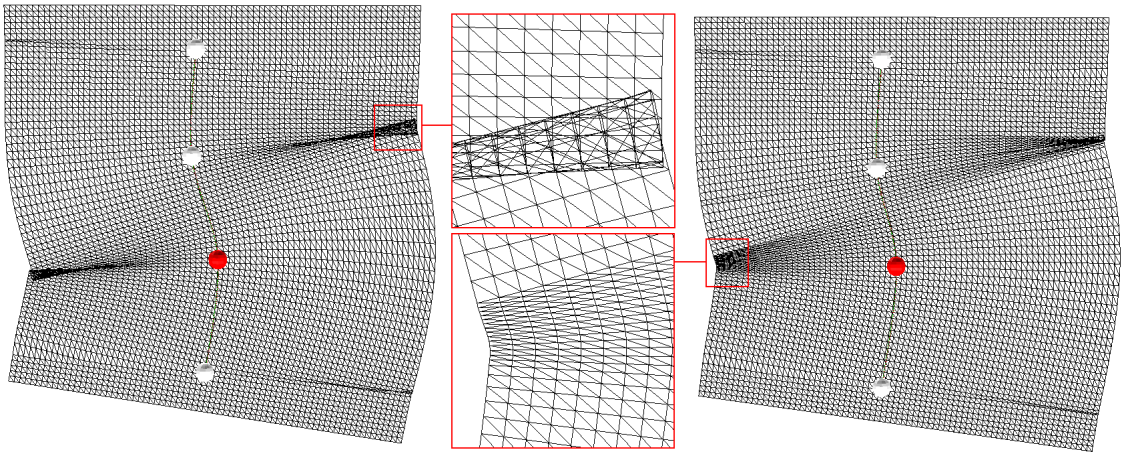


Figure 5.11: Self-intersection avoidance. Left: A deformation on a plane performed without local self-intersection avoidance. Center: Details of critical parts of the meshes, for comparing the occurrence of self-intersections. Right: A similar deformation using the presented algorithm, where local self-intersections are successfully avoided.

produced with the use of the presented algorithm indeed avoid local self intersections. Note that the presented algorithm for local self-intersection avoidance runs at interactive rates as the whole technique presented in this work.

5.6 Digest

This chapter presented a strategy for transferring curve deformations to meshes. A consistent frame field is created along each curve and the mesh is parameterized with respect to these frame fields. When the user interacts with the curve, the frame field changes and the mesh is deformed accordingly. Free-form deformation is achieved by interactively moving the curves' control points. Interesting effects, such as twisting and scaling, are easily achieved by operating directly on the frame field.

A blending function is used in order to keep the mesh continuity while a skeleton is deformed. Blending functions are responsible for smoothly transitioning the deformation among the vertices controlled by the different curves.

An interactive algorithm to avoid local self-intersection is proposed. The algorithm works by rotating some frames along the curve. The next chapter describes the use of the presented framework to deform meshes guided by suggestive contours automatically extracted from the meshes.

6 SUGGESTIVE CONTOURS EDITING

Contours and suggestive contours are lines on the surface of an object that convey important information about its shape. For this reason, giving the user the ability to edit objects by changing these lines seems quite intuitive. This chapter extends the technique described in the previous chapters to transfer deformations applied to contours and suggestive contours into actual mesh deformations.

Parametric curves are created from the lines extracted based on contours and suggestive contours, in the same way as done with user sketches in Chapter 4. The deformation of curves are thus transferred to mesh deformations as described in Chapter 5. However, a blending function based on an approximation of geodesic distances is proposed in order to achieve better results when editing local details.

6.1 Motivation

3D objects can be represented using a relatively small number of lines. In fact, in (OHTAKE; BELYAEV; SEIDEL, 2004b) Ohtake et al. show that is possible to reconstruct the entire shape of an object by using just ridge and valley lines extracted from the original surface.

Contour lines (KOENDERINK, 1984) were defined to retrieve the most important information from an object, based on human perception. Let p be a point on the surface S of an object equipped with normal vector $n(p)$. When viewed from a perspective camera positioned at point c, let $v(p) = p - c$. Contour lines are defined as the points on the surface where $\vec{n} \cdot \vec{v} = 0$.

Suggestive contours (DECARLO et al., 2003) give a lot of information about the shape of an object. They are defined as points on the surface that are not contours but would become contours with a slightly change in the camera's view position. Informally, this is equivalently to locations at which $\vec{n} \cdot \vec{v}$ is a positive local minimum rather than zero.

Together, contours and suggestive contours convey the shape of an object quite well and that explains why these lines are often used in non-photorealistic rendering. Figure 6.1 shows a rendered image of the Homer model (left) its contours lines (center) and the contours combined with suggestive contours (right). In the rest of this work, both suggestive contours and contours will be referred as suggestive contours for the sake of simplicity.

As these lines are simpler than the mesh itself but still represent the major details of the object, it follows that they provide a good interface for an artist to express his ideas when modifying a 3D object. Allowing the user to edit suggestive contours, thus modifying the associated mesh, seems to be a promising approach for deforming complex objects because the user can abstract some parts of the mesh concentrating on features that

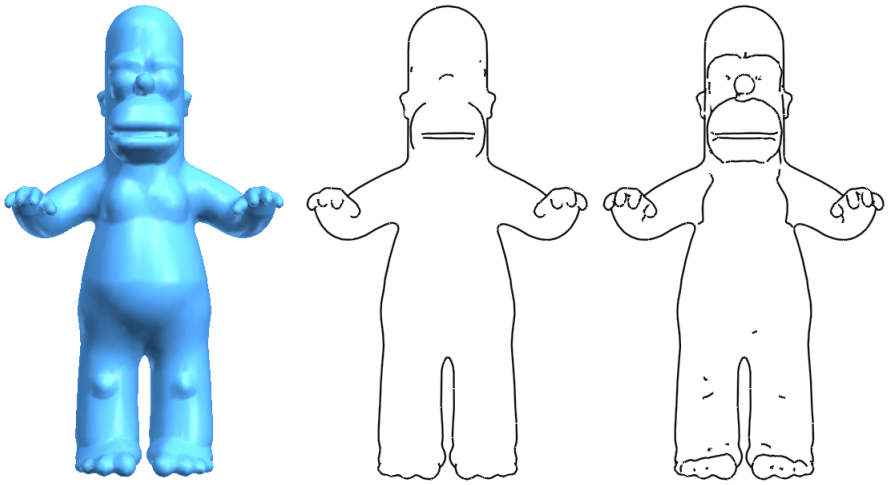


Figure 6.1: Contours and suggestive contours. Left: original model. Center: The contours as seen from the same viewpoint. Right: Suggestive contours and contours. Note that contours and suggestive contours combined conveys the shape of the object quite well.

convey more information. In fact, sketching a shape or editing suggestive contours can be seen as inverse non-photorealistic rendering, as already pointed out in (NEALEN et al., 2005). However, the only work that makes use of suggestive contours to perform mesh deformation we are aware is that of Nealen et al. (NEALEN et al., 2005).

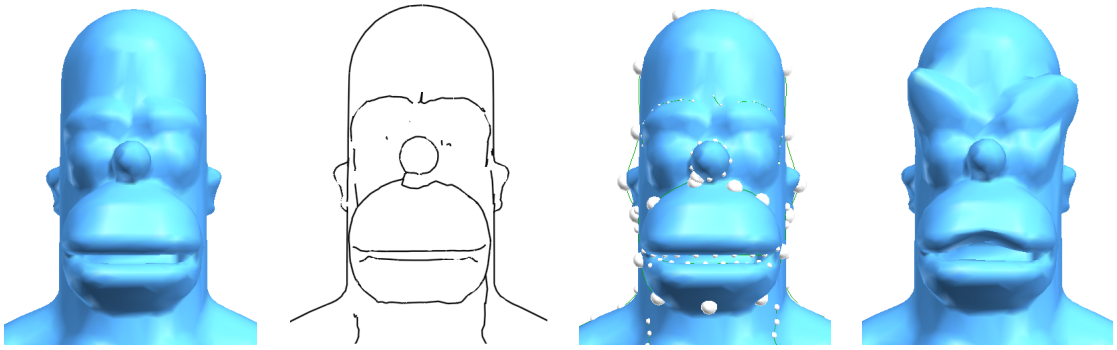
In that work, the user selects a suggestive contour line and sketches a destination line to where the suggestive contour should appear. The constraints defined by the new sketch are used in a Laplacian system (SORKINE et al., 2004) to perform the deformation. The technique presented in this chapter makes use of parametric curves in the 3D space, which provides fine control over the deformation, and directly transfer the curve deformation to the mesh, as discussed in the previous chapters.

6.2 Technique Overview

Suggestive contours can be efficiently computed for objects represented by 3D meshes. The extracted lines can then be used to create parametric curves which are associated to the object mesh as described in Chapter 5. So, a typical modeling session using the technique presented here consists of two main steps:

- Choosing a viewpoint where the rendering of suggestive contours shows a line representing a feature of interest to be edited;
- Modifying the feature by moving the control points of the parametric curve fitted to the parametric contour.

Figure 6.2 shows an example of this technique. Given a mesh (a), its suggestive contours are automatically calculated for each viewpoint (b). To modify the mesh at the current viewpoint, we fit parametric curves to the suggestive contour lines and use them as handles (c). By moving the control points of these handles, the mesh is appropriately deformed (d). This technique seems particularly interesting for editing mesh details, like the eyebrows of the homer model.



(a) (b) (c) (d)

Figure 6.2: Mesh deformation using suggestive contours. Parametric curves (c) are fitted to the suggestive contour lines shown in (b). By moving the parametric curves' control points, the user can deform the object (d). The mouth was opened and the eyebrows were lifted up. The original model is shown in (a) for comparison.

The way the curves are attached to the mesh are exactly the same as described in Section 3.1. However, the blending function is calculated using an approximation of the geodesic distance to the suggestive contour lines.

6.3 Line Identification

To efficiently calculate suggestive contours for a 3D mesh, each triangle must be processed independently. This approach facilitates the implementation of a rendering system for lines defined on meshes, since one just has to test if a line segment is to appear in each triangle and draw it, avoiding any storage. A suggestive contour line will be defined as a set of line segments crossing individual triangles. However, to fit a parametric curve to a suggestive contour line, the entire suggestive contour line is needed instead of its individual segments.

Algorithm 7 groups individual segments into a single structure. This algorithm follows the path of a suggestive contour starting from a line segment and following the mesh topology until the line ends. By using the mesh topology, this approach works very efficiently.

After the execution of Algorithm 7, each line can be treated as a sketch and thus the region segmentation step is identical to previously discussed (see Section 3.1). However, in order to emphasize big mesh features while helping to ensure more efficiency, only lines with a minimum length l are considered. In fact, significant details, which would worth editing, cannot be well represented by lines too small and are often identified as long lines in the set of suggestive contour lines retrieved. Empirically, l is set as $1/4$ of the radius of the bounding sphere of the object, but this can be manually adjusted by the user.

6.4 Blending Function

Let v_i be a vertex of the mesh and let $gdistance(v_i,C)$ be the geodesic distance of v_i to a curve C. A zero-mean Gaussian function ($b(x) = e^{-x^2/(2\sigma^2)}$) is associated to each curve. The standard deviation σ of the Gaussian controls the width of the function and is set empirically to $\sigma = \sqrt{gdistance(v_i,C)}$. This yields the weighting values to be used

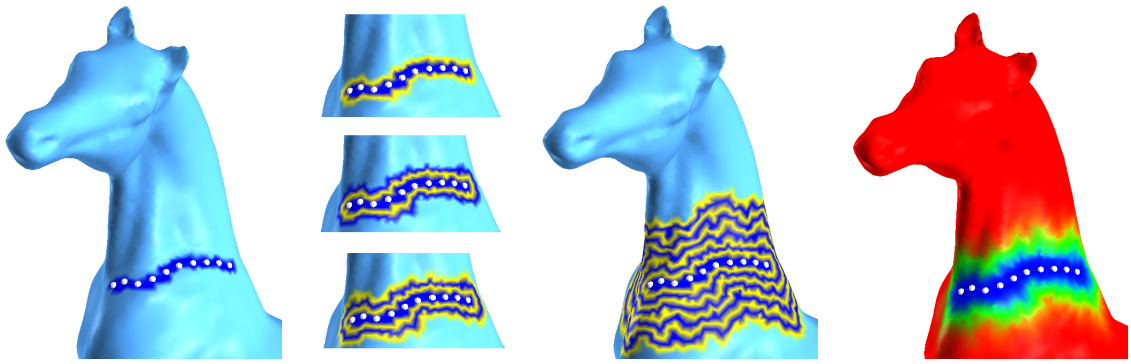
in the same way as described in Section 5.4. Figure 6.3 d shows the calculated weighting values for a curve on the horse model using a color scale, where 1 is mapped to blue and 0 is mapped to red. Note that there is a smooth transition as the vertices are geodesically far from the curve.

The geodesic distance approximation described in the next section is calculated very efficiently and works well especially for meshes with regular-sized triangles.

6.4.1 Geodesic Distance Approximation

In order to ensure interactive rates to the technique, geodesic distances are not exactly calculated. Instead, a rough approximation is obtained using a simple region-growing algorithm.

Since a suggestive contour line is identified as segments in each individual triangle, a list of triangles where the line passes is already available (shown in dark blue in Figure 6.3 a). The vertices of these triangles are assumed to be at distance 0 from the suggestive contour line. The next ring of triangles neighboring the triangles in dark blue is assumed to be at distance 1 from the suggestive contour line (shown in yellow on the top image of Figure 6.3 b) and so on. The maximum number of rings is defined by the user. However, when a ring touches a neighboring suggestive line the algorithm stops. This approach makes the region of interest of a given curve to be defined automatically, based only on the suggestive contours of the mesh at a given viewpoint. The motivation for this strategy is that suggestive lines represent surface details and a curve must control only the details of the mesh under its region of influence. For this reason, the user is able to delete suggestive contour lines that do not represent details he or she wants to edit, thus enlarging the area of interest to be deformed.



(a) (b) (c) (d)

Figure 6.3: Region of influence of a suggestive contour line. Starting from a suggestive contour line (a), a region-growing algorithm is executed and a set of rings on the mesh is identified. In each step of the region-growing algorithm, the rings become one unit further from the seed (*i.e.* the suggestive contour line) (b). The region-growing algorithm stops after user-defined number of steps (c). A Gaussian function is then evaluated using the approximations of geodesic distance to the curve at each ring, resulting in smooth weighting values for a blending function (d).

Algorithm 7 Algorithm to group line segments of suggestive contours

Require: *segments* {List of suggestive contours segments}

```
{Organize a lookup table}
for each segment s from segments do
    for each vertex v of the triangle where segment s lies do
        lookup[v] ← s
    end for
end for
done ← 0 {Number of segments tested}
totalDone ← segments.size() {Number of segments to test}

while done != totaldone do
    currentSegment ← nextsegmentnotalreadytested
    { Group all segments which triangles are neighbors in the mesh topology}
    neighborsSegments.pushBack(currentSegment)
    neighborsToAdd.pushBack(currentSegment)
    while neighborsToAdd.size() > 0 do
        currentNeighbor ← neighborsToAdd.pop()
        for each vertex v of the triangle where segment currentNeighbor lies do
            for each entry e of v on the lookup table do
                if e not already added as a neighbor then
                    neighborsToAdd.pushBack(e)
                    neighborSegments.pushBack(e)
                end if
            end for
        end for
    end while
    {Connect all segments of the line starting from currentSegment}
    line.pushBack(currentSegment.point[0]) { Push the geometric positions of the line
    segment}
    line.pushBack(currentSegment.point[1])
    done++
    while TRUE do
        sz ← line.size()
        for each segment s of neighborSegments do
            if geometric position of a defining point of line segment s coincides with last
            point added to line then
                line.pushBack(s)
                done++
            else if geometric position of a defining point of line segment s coincides with
            the point in the first position of line then
                line.pushFront(s)
                done++
            end if
        end for
        if sz != line.size() then
            BREAK
        end if
    end while
    {Add line to the list of suggestive contours }
end while
```

6.5 Digest

This chapter introduced an extension to the mesh deformation technique presented in this thesis, where suggestive contours are used as handles for editing details of geometrically complex meshes. A geodesic distance approximation was used in order to ensure interactive rates to the technique. The use of suggestive contour lines to perform mesh deformation seems an attractive area for further exploration.

7 RESULTS

This chapter presents the results obtained with the mesh deformation technique presented in this thesis. Several examples are shown, discussing key points related to other techniques.

The algorithms described in this thesis were implemented using C++, with OpenGL used for visualization. We have applied our technique to deform several models. The performance measurements were carried on an AMD Athlon 64 3700+ processor running a 32-bit version of Microsoft Windows XP with 2GB of RAM memory.

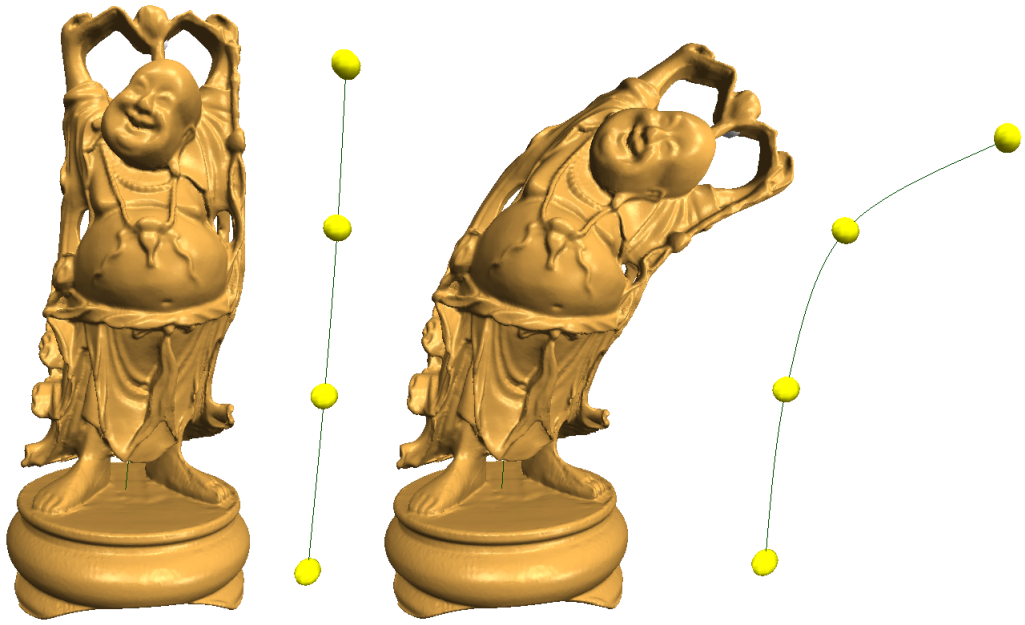


Figure 7.1: Deformations of the Buddha model. Left: original model and the original control curve. Right: Curve bent to the right and the corresponding deformed model.

The examples shown in this work stress the capabilities of the presented approach and compare it to current techniques. Figure 1.1 shows the Armadillo model in several poses and demonstrate the approach's ability to perform large deformations using a simple interface, while preserving local features. The legs and arms were modified by translating some control points. In the second example of Figure 1.1 the Armadillo's body was scaled and bent.

Figure 5.6 shows a block twisted and bent using a handle curve with 3 control points. Note the smoothness of the twisting and the bending. As has been demonstrated by Zhou et al. (ZHOU et al., 2005), Poisson meshes (YU et al., 2004) and Laplacian surfaces (LIP-

MAN et al., 2004, 2005; SORKINE et al., 2004) fail to produce such large twist and bend deformations. Also, as observed by (BOTSCH et al., 2006), the techniques described in (BOTSCH; KOBBELT, 2004) and (BOTSCH; KOBBELT, 2005) cannot perform smooth bendings, like the ones shown in Figure 5.6, on single large step rotations.

The Buddha model was bent to the right in Figure 7.1 using a single handle curve, which is shown to its right. Note that the smooth properties of the curve are directly transferred to the deformation. A similar kind of deformation is shown on Figure 7.2, where a cactus model (left) has been deformed using a single curve sketched over its trunk (center). Note how the branches nicely follow the deformation. As pointed out by Lipman et al. (LIPMAN et al., 2005), this kind of deformation cannot be performed using a single handle by Poisson Editing (YU et al., 2004), as the strength of the gradient field gets weaker with geodesic distance. Figure 7.2 (right) illustrates the versatility of our technique, which also allows the branches to be deformed independently, using a skeleton that mimics the structure of the cactus. This capability gives animators freedom to express themselves artistically.

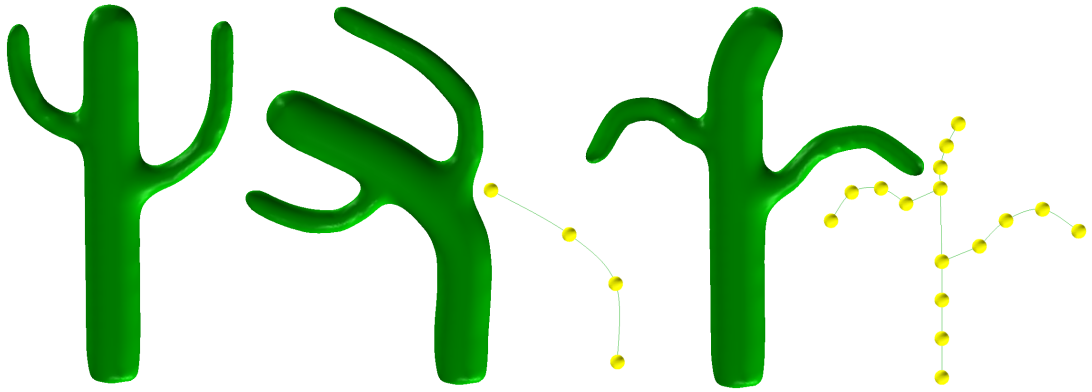


Figure 7.2: The cactus model (left) was deformed using a single handle curve sketched over its trunk (center). Independent deformations can be applied to the individual branches by building a skeleton and deforming the curves associated to the branches (right).

Figure 7.3 (center) shows a large and smooth deformation of one tentacle of the octopus model (left). This kind of result cannot be achieved using the technique describe in (SORKINE et al., 2004) as shown by the authors in their subsequent paper (LIPMAN et al., 2005). Figure 7.3 (right) shows a close-up view of the deformed tentacle seen from the back to illustrate that the surface's original features were preserved by the transformation.

Figure 7.4 illustrates the use of deformation with skeleton constraints (*i.e.*, preserving the rigidity of limb segments in articulated figures). While Huang et al. (HUANG et al., 2006) enforce such constraints via non-linear least-squares optimization and require that each articulated segment be a closed mesh, our approach imposes no restriction on the mesh topology. We implement skeleton constraints by simply treating each segment between two adjacent control points as a Hermite curve (MORTENSON, 1997) with small tangent vectors. This produces straight articulated segments with slightly bent endings, as desired. Figure 7.4 (left) shows a horse leg in its original position with a handle curve superimposed. The resulting deformed leg is shown on the right. Huang et al. (HUANG et al., 2006) report that their system takes about 20 minutes to perform the deformation of the four legs of the horse model. In our system, the user can deform each leg at a time

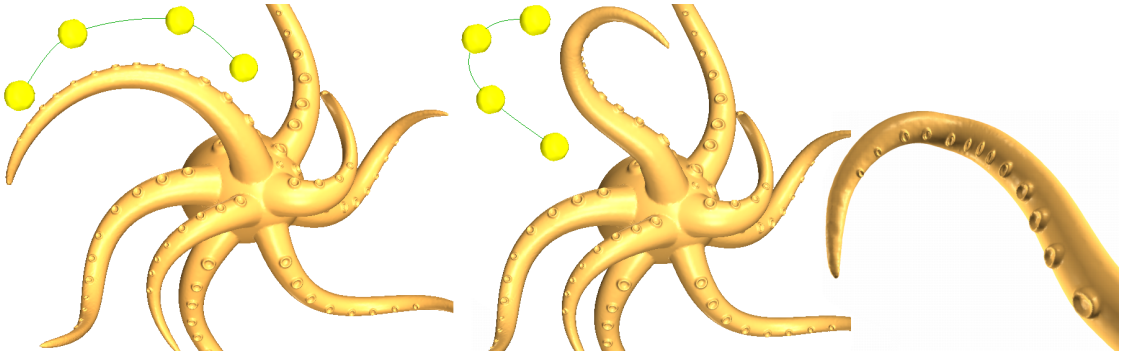
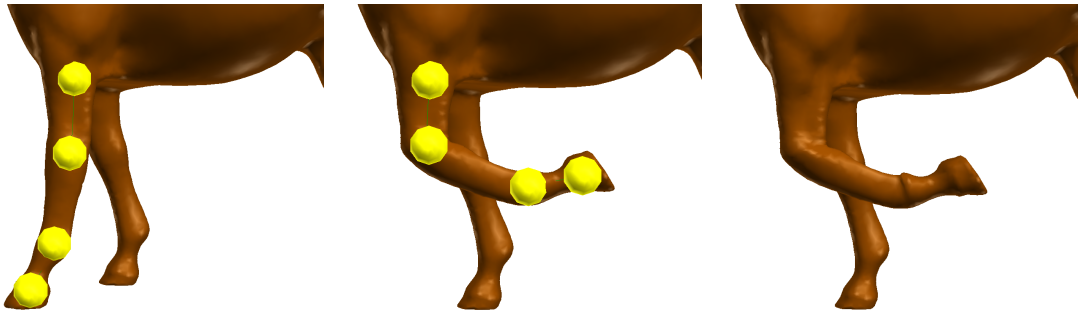


Figure 7.3: Deformation including large rotations. One of the tentacles of the original octopus model (left) was deformed (center). A close-up view of the deformed tentacle, seen from the back, shows its smooth profile and reveals its nicely preserved features (right).

in real time.



(a) (b) (c)

Figure 7.4: Deforming a horse leg (left) using skeleton constraints. Each segment between two adjacent control points in the handle is a Hermite curve with small tangent vectors. The resulting deformation (right) preserves the rigidity of the limb segments.

Figure 7.5 shows the dragon model with its mouth opened in two different ways. The original model is shown on the left for comparison and the handle curves are shown on top. Our approach produces visually pleasing global deformations while consistently preserving local features. As pointed by Botsch et al. (BOTSCH et al., 2006), local feature preservation is a challenge not met by most recent mesh deformation techniques (BOTSCH; KOBBELT, 2004, 2005; LIPMAN et al., 2004, 2005; SORKINE et al., 2004; YU et al., 2004; ZAYER et al., 2005). Figure 7.6 illustrates again the effectiveness of our technique to preserve local features. The result shown on the right was obtained with a one-step translation of a single control point of a handle curve associated to the model on the left.

Multiple-component meshes, non-manifold and non-orientable surfaces pose some challenge to differential-based approaches, which require computing and/or propagating a discrete frame field based on the mesh local properties. As the frames used in our approach live on the handle curve, all these configurations are treated in a natural and uniform way. Huang et al. (HUANG et al., 2006) handle non-manifold meshes by "ignoring the non-manifold vertices in the surface detail energy term" (HUANG et al., 2006). It is not clear, however, what the effects of such procedure on the feature details of the deformed mesh are.

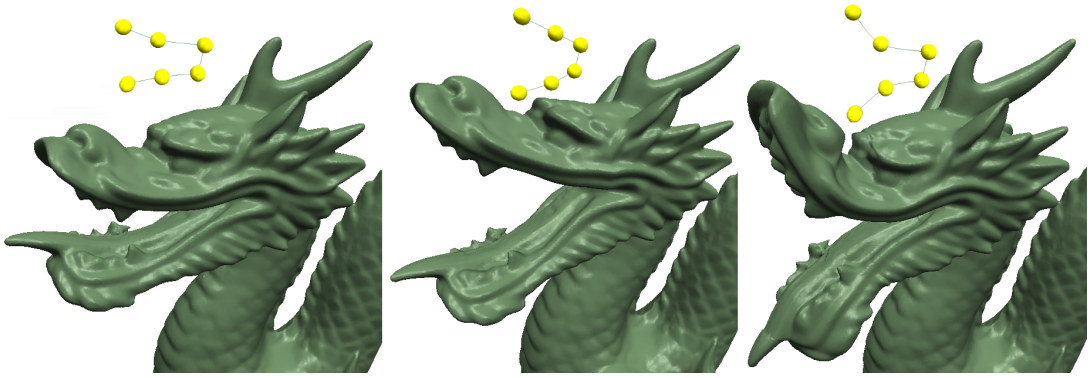


Figure 7.5: Deformations of the dragon model. Left: original model. Center and right: dragon mouth deformed in different ways. Handle curves shown on top.

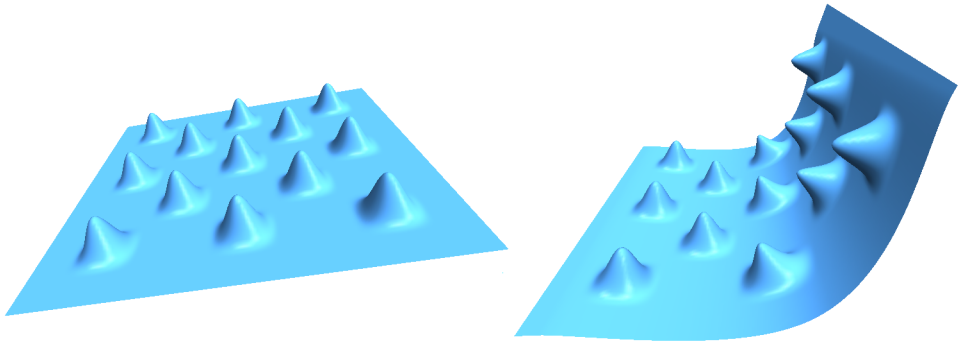


Figure 7.6: Local feature preservation. Original model (left). A deformation performed using our technique by translating a single control point (right). Note how the local features are correctly re-oriented on the resulting surface.

Figure 7.7 (left) shows a horse model after a ring of polygons have been removed to create two disconnected components. A handle curve with four control points has been attached to the back of the horse. On the right, the deformed model. Note that, although disconnected, the two parts align nicely after the deformation.

Figure 7.8 demonstrates the power of our metaphor by performing a deformation on non-orientable and non-manifold surfaces. On the left, one sees a Moebius strip (top) and a non-manifold surface. The images on the center and on the right show the deformed models after moving some control points in each of their corresponding handles.

Table 7.1 shows the performance of our technique. The setup time corresponds to the time required for region segmentation (Section 3.1), which is performed only when a new curve is created. For these measurements, we have transformed all vertices of the mesh every time the user modified the curve. So, the last column of Table 7.1 shows the worst case for any deformations made by our technique on these models. These numbers indicate that it can operate at interactive rates, even with meshes composed by hundreds of thousand vertices. Depending on the size of the ROI, our approach can handle meshes containing millions of polygons in real time. These numbers compare favorably to all previously known mesh deformation techniques supporting local feature preservation. For instance, in a recent work, Huang et al. (HUANG et al., 2006) report that their technique cannot handle the full Armadillo model (172,974 vertices) at interactive rates. By using a simplified version of the model (30k vertices) and using some multiresolution techniques described in (GUSKOV; SWELDENS; SCHRÖDER, 1999), the authors report achieving

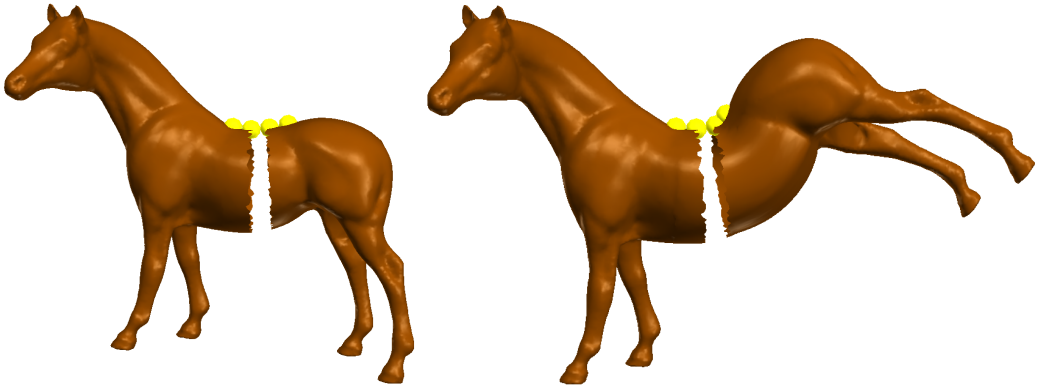


Figure 7.7: Deformation of a mesh with multiple components. Horse model with a ring of polygons removed (left). Deformed model obtained by moving some control points. Note how the parts still align nicely after the deformation.

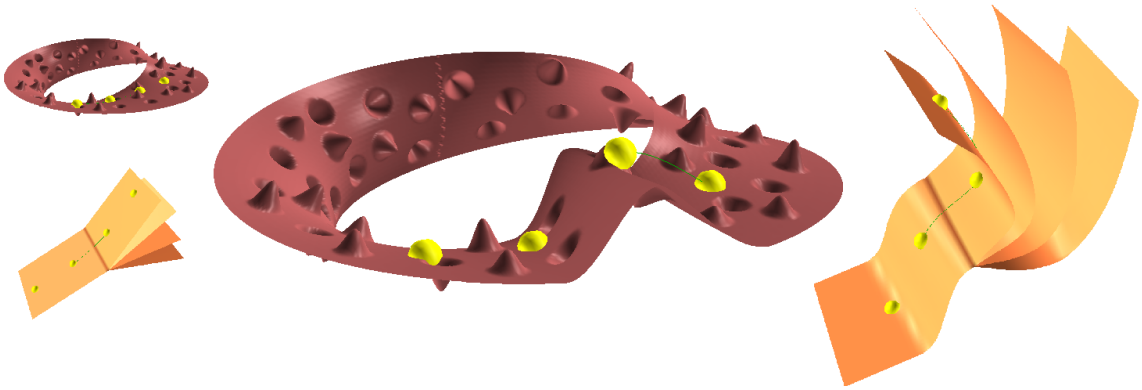


Figure 7.8: Deforming non-orientable (center) and non-manifold (right) surfaces.

9.1 fps. We should point that the approach by (HUANG et al., 2006) can preserve volume, which ours cannot.

Another technique supporting volume preservation under deformation was presented by Zhou et al. (ZHOU et al., 2005). The authors report that, for the example of the full Armadillo model, their technique would require solving a system containing approximately 1,020,000 variables, which currently cannot be done interactively.

The technique developed by Botsch et al. (BOTSCH et al., 2006) produces physically plausible deformations, which the current version of our technique does not do. According to the authors, their system can deform a model with 180k triangles at 1 fps on a 3.4 GHz P4 processor.

Table 7.2 compares various mesh deformations techniques according to three criteria: (i) does the technique preserve local features under translation (*Transl*)?, (ii) does the technique support large rotations as a single step deformation (*Large Rot.*)? and (iii) does it support skeleton constraints (*Skel.*)? Since the technique described in (HUANG et al., 2006) is based on Laplacian coordinates, detail preservation should be insensitive to translation, as are other techniques based on differential coordinates.

The technique presented in this work can handle both large rotations and translations as the frames are created on a curve rather than on the surface of the models. Botsch et al. (BOTSCH et al., 2006) has achieved this by enclosing the mesh with a set of rigid prisms with physical behavior, which has introduced a computational expensive physical

Table 7.1: Times obtained using our technique for deforming several models. The right column shows the performance of the deformation (in fps) when deforming all vertices of the model.

Model	**Vertices**	**Triangles**	**Setup (s)**	**Deform. (fps)**
Horse	19,851	39,698	0.0545	59.52
Dino	56,194	112,384	0.1550	34.60
Bunny	76,000	151,996	0.1339	29.85
Armadillo	172,974	345,944	0.2914	14.94
Dragon	437,645	871,414	1.0697	7.12
Buddha	543,652	1,087,716	3.4130	4.50

Table 7.2: Comparison among the various mesh deformation techniques according to the following criteria: (i) preserve local features under translation (*Transl*), (ii) support large rotations as a single step deformation (*Large Rot.*), (iii) support skeleton constraints (*Skel.*). [1]Rotations are approximated heuristically.

Method	**Transl.**	**Large Rot.**	**Skel.**
Laplacian Editing (SORKINE et al., 2004)	no	no	no
Differential Coordinates (LIPMAN et al., 2004)	no	yes[1]	no
Rotation Invariant (LIPMAN et al., 2005)	no	yes	no
Poisson-based (YU et al., 2004)	no	yes	no
Harmonic Guidance (ZAYER et al., 2005)	no	yes	no
Intuitive Framework (BOTSCH; KOBBELT, 2004)	yes	no	no
RBF (BOTSCH; KOBBELT, 2005)	yes	no	no
Volumetric Laplacian (ZHOU et al., 2005)	yes	yes	no
PriMo (BOTSCH et al., 2006)	yes	yes	no
Gradient Domain (HUANG et al., 2006)	no	yes	yes
Our approach	yes	yes	yes

simulation step. Zhou et al. (ZHOU et al., 2005) can also handle with both large rotations and translation, achieved by creating a graph which is more complex than the mesh itself. While this leads to a deformation technique that can preserve the volume of the models and the work of Botsch et al. (BOTSCH et al., 2006) produces visually plausible physical deformations, none of them can outperform the technique presented in this work in terms of speed. Moreover, they do not allow the creation of skeleton constraints, which can be done using the presented approach. As Table 7.2 shows, no other technique can handle both large rotations and translation while providing support for skeleton constraints.

7.1 Digest

This chapter has shown the results obtained with the deformation technique presented in this thesis. The examples and comparison tables show that it can indeed be used to interactively deform geometrically complex non-structured objects, preserving the small details commonly found in these models in a consistent way.

8 CONCLUSIONS AND FUTURE WORK

3D models used in computer graphics applications are constantly evolving in terms of geometric complexity and the process of modeling them is becoming increasingly complex. This has motivated the popularization of the use of 3D scanners to reduce the amount of time an artist has to spend to create such models. However, for movies, games and computer graphics applications, artists must be able to change the poses, deform and animate the digitized models, which is done by manipulating control structures on the models. Models reconstructed from point clouds do not have such structures and artists must construct them manually, which is a labor-intensive task.

A technique for deforming non-structured objects that combines high quality mesh deformation with an intuitive interface was presented in this work. The approach is based on the metaphor of transferring smooth deformations from parametric curves to complex 3D models, which does not require solving large linear systems as most of recent works do. Fast solvers have been proposed to reduce the time needed for solving these linear systems (SHI et al., 2006). However, a delay is created because this solving step must be done for each deformation region the user selects, and this is not desired in interactive systems. The technique presented in this thesis minimizes this delay by avoiding the solving step. A small delay, however, is still present due to the steps required by the presented approach.

The use of parametric curves allows the technique to be implemented using very intuitive user interfaces. As opposed to other techniques (SEDERBERG; PARRY, 1986; COQUILLART, 1990; SORKINE et al., 2004; LIPMAN et al., 2005; BOTSCH; KOBBELT, 2005; ZAYER et al., 2005; BOTSCH; KOBBELT, 2004), the presented approach provides local control over the deformation through the use of skeletons and individual curves. The simple formulation also avoids local self-intersections and preserves local features under global deformation, an important but hard to achieve goal, at interactive rates.

8.1 Pros

This Section discusses the results of this work by analyzing the characteristics presented in Chapter 1 and the avoidance of local self-intersection obtained by the presented technique, which is also an important goal.

8.1.1 Interactivity and Robustness

The ability to work with models composed by hundreds of thousand polygons (robustness), is essential to model deformation systems due to the increasingly complexity of the models used in computer graphics. Interactive feedback about the deformation is also

crucial for an effective technique because the user must be able to perform experiments with the object during and editing session. As shown in Chapter 7, the technique remains interactive for objects composed by a million polygons.

8.1.2 Locality

When the user is editing a restricted region of the object, other parts of the mesh must not be deformed. The technique presented provides such characteristic by using a blending function (Section 5.4) to guarantee the continuity of the mesh while keeping the deformation restricted to a specific area. The locality of the deformations allows the user to concentrate on the region he or she is deforming.

8.1.3 Smoothness

The quality of the mesh deformation is achieved implicitly by the smooth properties of the parametric curves used in the presented technique, as the properties of the curves are directly transferred to the mesh deformation. Despite its simplicity, this strategy works very well in practice, avoiding the need for pre computation steps and the time consuming process of solving large linear systems commonly found in recent works.

8.1.4 Local Feature Preservation

Local feature preservation is achieved by the inherent rotation of the local frames created along the curve. As the surface is entirely encoded with respect to these local frames, the problems of other variational minimization techniques do not occur.

8.1.5 Avoidance of Local Self-intersections

The avoidance of local self-intersections is also an important characteristic of the presented technique. Given the whole formulation of this work, a simple algorithm to avoid local self-intersections (see Section 5.5) could be incorporated to it while still allowing for interactive performance.

8.2 Discussion

Despite the visually pleasing results presented in Chapter 7 and the pros discussed in Section 8.1, some considerations must be made about the use of the presented technique.

The use of curves provides local control over the deformation so that the user can perform fine adjusts as shown in the example of Figure 7.5. However, this freedom is not always wanted. Sometimes the user may want a more abstract way of specifying the deformations as just changing the position or orientation of some parts of the mesh. In these cases, a click & drag metaphor as proposed in (BOTSCH et al., 2006) would be more efficient.

Because vertices close to each other in Euclidean space may be far when considering geodesic distances (which accounts for paths on the shape of the object), it is easy to note that a blending function based on the geodesic distance, as proposed in Chapter 6, would result in a more intuitive deformation. The choice to use a blending function based on Euclidean distance was made in order to minimize the computation time and thus provide interactive rates to the technique.

The local self-intersection avoidance algorithm (Section 5.5) may produce unintuitive results when large rotations of the frames are needed. If a frame is exaggeratedly rotated,

the associated mesh may become undesirably thin. Moreover, the more the frames are rotated, the more they move from the frame field of the curve. This causes the deformation to be somewhat unrelated to the curve and thus unintuitive. However, the recognition of self-intersections is still good and efficient in these cases and, instead of rotating the frame field, one could block user interactions that would cause intersections.

8.3 Future Work

The use of other types of parametric curves seems a promising area for future exploration, as the properties of the curve are directly transferred to the mesh. One can explore the use of the presented approach to create physically plausible deformations at interactive rates using splines for physical simulation as described in (LENOIR et al., 2005). Moreover, the use of Finite Element Method (CELNIKER; GOSSARD, 1991) allows the specification of different properties along the curve. This characteristic is very interesting in the context of mesh deformation as it provides a mean to simulate inverse kinematics at interactive rates.

The presented technique is also suitable for a GPU implementation, since each vertex is deformed independently from the others. After the creation of the frame field, all deformation could be done in parallel using the GPU, which would increase the overall performance of the technique.

By using motion capture data to animate the skeletons produced with our technique, animations with realistic movements could be achieved. Using 3D scanners and motion capture data, the time needed to create animations could be significantly reduced.

REFERENCES

ALEXA, M. Differential coordinates for local mesh morphing and deformation. **The Visual Computer**, [S.l.], v.19, n.2, p.105–114, 2003.

AMENTA, N.; BERN, M.; KAMVYSSELIS, M. A New Voronoi-Based Surface Reconstruction Algorithm. **Computer Graphics**, [S.l.], v.32, p.415–421, Aug. 1998.

BLOOMENTHAL, J. Calculation of reference frames along a space curve. In: **Graphics gems**. Boston: Academic Press Professional, 1990. p.567–571.

BLOOMENTHAL, J. Skining: medial-based vertex deformation. In: ACM SIGGRAPH SYMPOSIUM ON COMPUTER ANIMATION, 2002, San Antonio, Texas. **Proceedings...** New York: ACM, 2002. p.147–151.

BLOOMENTHAL, J.; LIM, C. Skeletal Methods of Shape Manipulation. In: INTERNATIONAL CONFERENCE ON SHAPE MODELING AND APPLICATIONS. SMI, 1999. **Proceedings...** [S.l.]: IEEE Computer Society, 1999. p.44.

BOTSCH, M. et al. PriMo: coupled prisms for intuitive surface modeling. In: EUROGRAPHICS SYMPOSIUM ON GEOMETRY PROCESSING, 2006. **Proceedings...** [S.l.: s.n.], 2006. p.11–20.

BOTSCH, M.; KOBBELT, L. An intuitive framework for real-time freeform modeling. **ACM Trans. Graph.**, New York, NY, USA, v.23, n.3, p.630–634, 2004.

BOTSCH, M.; KOBBELT, L. Real-Time Shape Editing using Radial Basis Functions. In: EUROGRAPHICS, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.611–621.

BíSCARO, H. H. **Reconstrução a partir de nuvem de pontos com a utilização funções de Morse discretas**. 2005. Tese (Doutorado em Ciência da Computação) — Instituto de Ciências Matemáticas e de Computação - ICMC-USP.

CARMO, M. P. do. **Differential Geometry of Curves and Surfaces**. [S.l.]: Prentice Hall, 1976.

CARR, J. C. et al. Reconstruction and representation of 3D objects with radial basis functions. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 28., 2001. **Proceedings...** New York: ACM Press, 2001. p.67–76.

CARR, J. C. et al. Representation: smooth surface reconstruction from noisy range data. In: INTERNATIONAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES IN AUSTRALASIA AND SOUTH EAST ASIA, 1., 2003. **Proceedings...** New York: ACM Press, 2003. p.119–126.

CATMULL, E. E.; ROM, R. J. A class of local interpolating splines. **Computer Aided Geometric Design**, [S.l.], p.317–326, 1974.

CELNIKER, G.; GOSSARD, D. Deformable curve and surface finite-elements for free-form shape design. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 18., 1991. **Proceedings...** New York: ACM Press, 1991. p.257–266.

CHANG, Y.-K.; ROCKWOOD, A. P. A generalized de Casteljau approach to 3D free-form deformation. In: SIGGRAPH, 1994. **Proceedings...** [S.l.: s.n.], 1994. p.257–260.

CHERLIN, J. J.; SAMAVATI, F.; SOUSA, M. C.; JORGE, J. A. Sketch-based modeling with few strokes. In: SPRING CONFERENCE ON COMPUTER GRAPHICS, SCCG, 21., 2005. **Proceedings...** New York: ACM Press, 2005. p.137–145.

COQUILLART, S. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In: SIGGRAPH, 1990. **Proceedings...** [S.l.: s.n.], 1990. p.187–196.

DECARLO, D. et al. Suggestive contours for conveying shape. **ACM Trans. Graph.**, New York, NY, USA, v.22, n.3, p.848–855, 2003.

DEY, T. K.; GIESEN, J. Detecting undersampling in surface reconstruction. In: ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY, SCG, 17., 2001. **Proceedings...** New York: ACM Press, 2001. p.257–263.

DEY, T. K.; GOSWAMI, S. Provable surface reconstruction from noisy samples. In: ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY, SCG, 20., 2004. **Proceedings...** New York: ACM Press, 2004. p.330–339.

DU, H.; QIN, H. Medial axis extraction and shape manipulation of solid objects using parabolic PDEs. In: ACM SYMPOSIUM ON SOLID MODELING AND APPLICATIONS, SM, 9., 2004, Aire-la-Ville, Switzerland, Switzerland. **Proceedings...** [S.l.]: Eurographics Association, 2004. p.25–35.

FUNCK, W. von; THEISEL, H.; SEIDEL, H.-P. Vector field based shape deformations. **ACM Transactions on Graphics**, New York, NY, USA, v.25, n.3, p.1118–1125, 2006.

GUSKOV, I.; SWELDENS, W.; SCHRÖDER, P. Multiresolution signal processing for meshes. In: SIGGRAPH, 26., 1999. **Proceedings...** New York: ACM, 1999. p.325–334.

HOPPE, H. et al. Surface reconstruction from unorganized points. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 19., 1992. **Proceedings...** New York: ACM Press, 1992. p.71–78.

HSU, W. M.; HUGHES, J. F.; KAUFMAN, H. Direct manipulation of free-form deformations. In: SIGGRAPH, 19., 1992. **Proceedings...** New York: ACM Press, 1992. p.177–184.

HUANG, J. et al. Subspace gradient domain mesh deformation. **ACM Transactions on Graphics**, New York, v.25, n.3, p.1126–1134, 2006.

IGARASHI, T.; HUGHES, J. F. A suggestive interface for 3D drawing. In: ANNUAL ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, UIST, 14., 2001. **Proceedings...** New York: ACM Press, 2001. p.173–181.

IGARASHI, T.; MATSUOKA, S.; TANAKA, H. Teddy: a sketching interface for 3d freeform design. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 26., 1999. **Proceedings...** New York: ACM Press/Addison-Wesley, 1999. p.409–416.

KHO, Y.; GARLAND, M. Sketching mesh deformations. In: SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES, I3D, 2005. **Proceedings...** New York: ACM Press, 2005. p.147–154.

KOBBELT, L. et al. Interactive multi-resolution modeling on arbitrary meshes. In: SIGGRAPH, 25., 1998. **Proceedings...** New York: ACM Press, 1998. p.105–114.

KOENDERINK, J. What Does the Occluding Contour Tell Us About Solid Shape? **Perception**, [S.l.], v.13, p.321–330, 1984.

LAGA, H.; TAKAHASHI, H.; NAKAJIMA, M. Boundary Constraints Based Implicit Representation and Reconstruction of Complex 3D Objects. **Image Information and Television Engineers (ITE) Technical Report**, [S.l.], v.27, n.46, p.29–32, July 2003.

LENOIR, J. et al. Adaptive resolution of 1D mechanical B-spline. **Computers & Graphics**, New York, v.29, n.5, p.395–403, 2005. Trabalho apresentado no GRAPHITE, 2005.

LIEN, J.-M.; KEYSER, J.; AMATO, N. M. Simultaneous shape decomposition and skeletonization. In: ACM SYMPOSIUM ON SOLID AND PHYSICAL MODELING, SPM, 2006. **Proceedings...** New York: ACM Press, 2006. p.219–228.

LIPMAN, Y. et al. Differential Coordinates for Interactive Mesh Editing. In: SHAPE MODELING INTERNATIONAL, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.181–190.

LIPMAN, Y. et al. Linear rotation-invariant coordinates for meshes. **ACM Trans. Graph.**, [S.l.], v.24, n.3, p.479–487, 2005.

MACCRACKEN, R.; JOY, K. I. Free-form deformations with lattices of arbitrary topology. In: SIGGRAPH, 1996. **Proceedings...** [S.l.: s.n.], 1996. p.181–188.

MOHR, A.; TOKHEIM, L.; GLEICHER, M. Direct manipulation of interactive character skins. In: SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES, I3D, 2003. **Proceedings...** New York: ACM Press, 2003. p.27–30.

MOLLER, T.; TRUMBORE, B. Fast, minimum storage ray-triangle intersection. **J. Graph. Tools**, Natick, MA, USA, v.2, n.1, p.21–28, 1997.

MORTENSON, M. **Geometric Modeling**. 2nd ed. [S.l.]: John Wiley and Sons, 1997.

NEALEN, A.; SORKINE, O.; ALEXA, M.; COHEN-OR, D. A sketch-based interface for detail-preserving mesh editing. **ACM Trans. Graph.**, New York, NY, USA, v.24, n.3, p.1142–1147, 2005.

OHTAKE, Y.; BELYAEV, A.; ALEXA, M.; TURK, G.; SEIDEL, H.-P. Multi-level partition of unity implicits. **ACM Trans. Graph.**, New York, NY, USA, v.22, n.3, p.463–470, 2003.

OHTAKE, Y.; BELYAEV, A.; SEIDEL, H.-P. 3D Scattered Data Approximation with Adaptive Compactly Supported Radial Basis Functions. In: SHAPE MODELING INTERNATIONAL, SMI, 2004. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p.31–39.

OHTAKE, Y.; BELYAEV, A.; SEIDEL, H.-P. Ridge-valley lines on meshes via implicit surface fitting. **ACM Trans. Graph.**, New York, NY, USA, v.23, n.3, p.609–612, 2004.

SCLAROFF, S.; PENTLAND, A. Generalized implicit functions for computer graphics. In: ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH, 18., 1991. **Proceedings...** New York: ACM Press, 1991. p.247–250.

SEDERBERG, T. W.; PARRY, S. R. Free-form deformation of solid geometric models. In: SIGGRAPH, 1986. **Proceedings...** [S.l.: s.n.], 1986. p.151–160.

SHI, L.; YU, Y.; BELL, N.; FENG, W.-W. A fast multigrid algorithm for mesh deformation. **ACM Transactions on Graphics**, New York, NY, USA, v.25, n.3, p.1108–1117, 2006.

SINGH, K.; FIUME, E. Wires: a geometric deformation technique. In: SIGGRAPH, 25., 1998. **Proceedings...** [S.l.: s.n.], 1998. p.405–414.

SORKINE, O. et al. Laplacian surface editing. In: SGP, 2004. **Proceedings...** New York: ACM Press, 2004. p.175–184.

TERZOPOULOS, D.; METAXAS, D. Dynamic 3D Models with Local and Global Deformations: deformable superquadrics. **IEEE Trans. Pattern Anal. Mach. Intell.**, Washington, DC, USA, v.13, n.7, p.703–714, 1991.

TOBOR, I.; REUTER, P.; SCHLICK, C. **Efficient reconstruction of large scattered geometric datasets using the partition of unity and radial basis functions**. [S.l.: s.n.], 2003. (RR-1301-03).

TOBOR, I.; REUTER, P.; SCHLICK, C. Efficient reconstruction of large scattered geometric datasets using the partition of unity and radial basis functions. **Journal of WSCG: an International Journal of Algorithms, Data Structures and Techniques for Computer Graphics and Visualization**, Plzen, v.12, p.467–474, 2004.

WANG, J.; OLIVEIRA, M. M.; KAUFMAN, A. Reconstructing Manifold and Non-Manifold Surfaces from Point Clouds. In: IEEE VISUALIZATION, 2005, Minneapolis, MN, USA. **Proceedings...** New York: IEEE Computer Society, 2005. p.415–422.

WANG, J.; OLIVEIRA, M. M.; XIE, H.; KAUFMAN, A. Surface Reconstruction Using Oriented Charges. **Computer Graphics International**, Stony Brook, NY, p.122–128, June 2005.

XIE, H.; MCDONNELL, K. T.; QIN, H. Surface Reconstruction of Noisy and Defective Data Sets. In: CONFERENCE ON VISUALIZATION, VIS, 2004. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p.259–266.

YAN, H.-B.; HU, S.-M.; MARTIN, R. Skeleton-Based Shape Deformation Using Simplex Transformations. In: COMPUTER GRAPHICS INTERNATIONAL, 2006. **Proceedings...** [S.l.: s.n.], 2006. p.66–77.

YOSHIZAWA, S.; BELYAEV, A. G.; SEIDEL, H.-P. Free-form skeleton-driven mesh deformations. In: ACM SYMPOSIUM ON SOLID MODELING AND APPLICATIONS, 2003. **Proceedings...** New York: ACM Press, 2003. p.247–253.

YU, Y. Surface reconstruction from unorganized points using self-organizing neural networks. In: IEEE VISUALIZATION CONFERENCE, 1999. **Proceedings...** [S.l.: s.n.], 1999. p.61–64.

YU, Y. et al. Mesh editing with poisson-based gradient field manipulation. **ACM Trans. Graph.**, [S.l.], v.23, n.3, p.644–651, 2004.

ZAYER, R. et al. Harmonic Guidance for Surface Deformation. In: EUROGRAPHICS, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.601–609.

ZHAO, H.-K.; OSHER, S.; FEDKIW, R. Fast Surface Reconstruction Using the Level Set Method. In: IEEE WORKSHOP ON VARIATIONAL AND LEVEL SET METHODS, VLSM, 2001. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. p.194.

ZHOU, K. et al. Large mesh deformation using the volumetric graph Laplacian. **ACM Trans. Graph.**, [S.l.], v.24, n.3, p.496–503, 2005.

ZORIN, D.; SCHRÖDER, P.; SWELDENS, W. Interactive multiresolution mesh editing. In: SIGGRAPH, 1997. **Proceedings...** [S.l.: s.n.], 1997. p.259–268.

APPENDIX A CATMULL-ROM SPLINE

Let p_0 and p_1 be points to be interpolated by a Catmull-Rom spline (CATMULL; ROM, 1974). The interpolating curve must have derivatives r_0 and r_1 at p_0 and p_1, respectively. Let $p(t)$ be the point in the interpolating curve at position t. Since a Catmull-Rom spline is a cubic interpolatory curve:

$$p(t) = at^3 + bt^2 + ct + d \tag{A.1}$$

For a Catmull-Rom spline, the coefficients a, b, c and d are given by:

$$a = -d_0 + 3p_0 - 3p_1 + d_1 \tag{A.2}$$

$$b = 2d_0 - 5p_0 + 4p_1 - d_1 \tag{A.3}$$

$$c = -d_0 + p_1 \tag{A.4}$$

$$d = 2p_0 \tag{A.5}$$

As noted, derivatives at first and last points are needed in order to construct a Catmull-Rom curve. One could let the user explicitly defines vectors d_0 and d_1 but defining such vectors can be somewhat cumbersome. Since their direction and magnitude must be controlled, this could lead to a more complicated interface. Instead, by duplicating the first and last control point, the curve interpolates all of the user control points and there is no need to explicitly control the tangents.

To allow the use of more than four control points, a piecewise curve must constructed. So, for the i-th segment, a direct application of Equations A.6 to A.9 yields the desired curve.

$$a = -p_{i-1} + 3p_i - 3p_{i+1} + p_{i+2} \tag{A.6}$$

$$b = 2p_{i-1} - 5p_i + 4p_{i+1} - p_{i+2} \tag{A.7}$$

$$c = -p_{i-1} + p_{i+1} \tag{A.8}$$

$$d = 2p_i \tag{A.9}$$

By varying t from 0 to 1 in the Equation A.1, using coefficients calculated by Equations A.6 to A.9, a Catmull-Rom spline is obtained between the i-th segment of the curve. However, to use a global parameter t that varies from 0 to 1 along the whole curve, a map scheme is required. This map scheme is often referred to as curve parameterization.

Let t be the global parameter that varies from 0 to 1 along the whole curve. Let n be the total number of points to be interpolated by the curve. A map scheme must return a local segment at which the global t value must be located and the local value to be used

in the Equation A.1. Let s be the segment index and let t' be the local parameter. The segment index s is used to find the proper coefficients, as calculated by Equations A.6 to A.9, to be used. Equation A.11 returns the local t' value and Equation A.10 returns the segment index s. This map scheme gives a uniform interval of t values along the curve segments, *i.e.* the global t is uniformly distributed for each curve segment.

$$s = \min\left(\lfloor (n-1)t \rfloor, n-1\right) \tag{A.10}$$

$$t' = (n-1)t - \lfloor (n-1)t \rfloor \tag{A.11}$$

This type of reparameterization is known as *uniform parameterization*, which is a fast approximation to *linear parameterization*. In order to have equally-spaced points along the curve's path, an *arc-length parameterization* should be used. However, this type of parameterization requires a pre-integration step, which is more expensive. Moreover, as the curve will be discretized with very small steps, this parameterization would not bring any visible advantage to the technique.

The first derivative of the Catmull-Rom spline is needed to create local frames along the curves (see Section 5.1) and is given by Equation A.12, using the coefficients calculated by Equations A.6 to A.8.

$$p'(t) = 3at^2 + 2bt + c \tag{A.12}$$

APPENDIX B UMA TÉCNICA PARA DEFORMAÇÃO INTERATIVA DE OBJETOS NÃO ESTRUTURADOS

Resumo da Dissertação em Português

O realismo das imagens geradas por computador é altamente dependente da habilidade de representar com fidelidade os detalhes geométricos dos objetos em cena. A modelagem destes objetos, entretanto, é uma tarefa que exige trabalho intensivo e que pode ser significativamente acelerada com o uso de scanners 3D. Apesar destes equipamentos se apresentarem como uma solução rápida para o problema de amostrar formas geometricamente complexas, algoritmos de reconstrução de superfícies a partir de nuvens de pontos tendem a produzir modelos não estruturados constituídos de uma única malha poligonal. A área de reconstrução de superfícies a partir de nuvens de pontos tem recebido bastante atenção recentemente (LAGA; TAKAHASHI; NAKAJIMA, 2003; OHTAKE et al., 2003; CARR et al., 2001; OHTAKE; BELYAEV; SEIDEL, 2004a; HOPPE et al., 1992; TOBOR; REUTER; SCHLICK, 2003, 2004; CARR et al., 2003; XIE; MCDONNELL; QIN, 2004; ZHAO; OSHER; FEDKIW, 2001; OHTAKE; BELYAEV; SEIDEL, 2004a; YU, 1999; TERZOPOULOS; METAXAS, 1991; SCLAROFF; PENTLAND, 1991; AMENTA; BERN; KAMVYSSELIS, 1998; DEY; GIESEN, 2001; DEY; GOSWAMI, 2004; WANG; OLIVEIRA; KAUFMAN, 2005; WANG et al., 2005; BíSCARO, 2005) e representações 3D de objetos geometricamente complexos estão se tornando amplamente difundidas. Entretanto, possibilitar ao usuário maneiras de modificar a pose e animar as representações resultantes é um ponto chave para utilizar estes modelos em aplicações como filmes e jogos de computador. O desafio é prover uma interface intuitiva, resposta interativa às ações do usuário e, ao mesmo tempo, preservar os detalhes de superfície e satisfazer restrições definidas pelo usuário.

B.1 Deformação de objetos 3D não estruturados

Neste trabalho, uma nova técnica interativa para deformação geométrica de objetos 3D não estruturados baseada no uso de rabiscos (*sketches*) 2D é apresentada. Ela foi inspirada pela metáfora de *oversketching* de Kho e Garland (KHO; GARLAND, 2005), mas esta nova abordagem é significativamente diferente. Curvas paramétricas criadas a partir de rabiscos 2D são usadas para deformar, torcer e escalar a malha associada. As deformações resultantes são visualmente agradáveis, provêem controle local e preservam os detalhes da malha em deformações globais. Esta nova abordagem também suporta algumas condições definidas pelo usuário, como a especificação de segmentos rígidos para

deformar figuras articuladas. Além disso, ela lida com malhas arbitrárias, incluindo malhas de múltiplos componentes conectados, superfícies não orientáveis e não-variedades, e pode ser implementada sob uma interface simples e intuitiva.

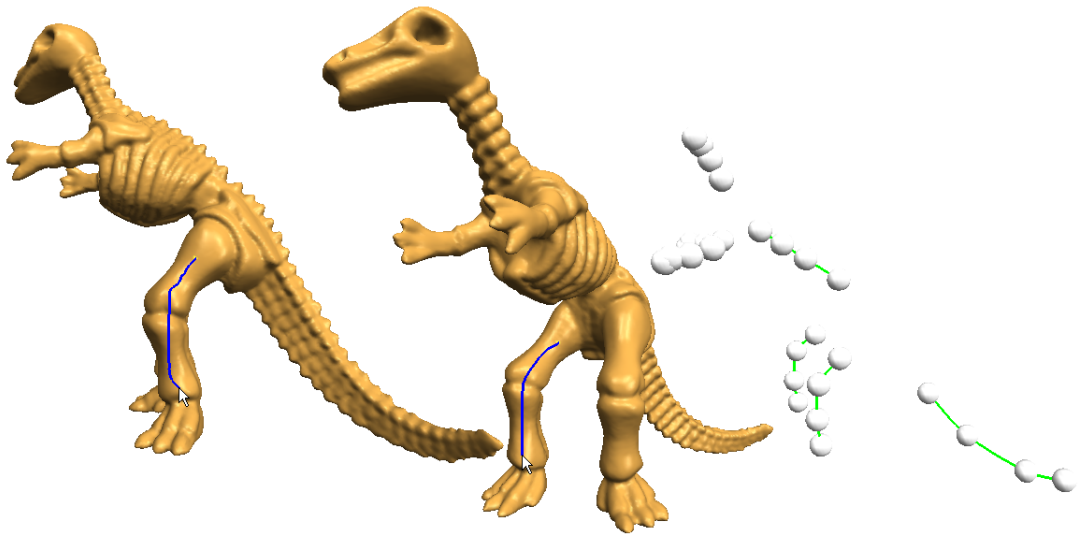


Figura B.1: Passo um da técnica. O usuário rabisca sobre as partes do modelo que ele gostaria de deformar e tem a possibilidade de mudar o ponto de vista da câmera (esquerda e centro). Cada linha rabiscada se torna uma curva paramétrica em 3D que será usada como um manipulador para deformar o modelo (direita). As pequenas esferas brancas representam os pontos de controle das curvas.

Uma sessão de modelagem baseada na técnica apresentada neste trabalho consiste em três passos principais:

- *Rabiscar curvas 2D sobre a representação tridimensional do modelo* (Figura B.1) para produzir um conjunto de curvas em 3D que serão usadas como manipuladores para deformar o modelo. A partir daí, modificando interativamente uma curva, a deformação é automaticamente transferida para as partes do modelo que estão ligadas a ela. A curva é modificada ao mover seus pontos de controle, que são representados na Figura B.1 (direita) como pequenas esferas brancas;

- *Conectar diferentes curvas para formar esqueletos* como mostrado na Figura B.2. Note que não há necessidade de ter apenas um esqueleto por objeto. De fato, o usuário pode até mesmo optar por deixar todas as curvas separadas (desconectadas) umas das outras;

- *Deformar o modelo selecionando pontos de controle da curva e movendo-os em 3D*. A Figura B.3 ilustra isso com duas diferentes deformações. Os dois personagens à esquerda mostram o antes e depois de uma torção no pescoço do modelo Dino. O par de imagens à direita provê uma ilustração similar para uma deformação aplicada à cauda do personagem.

Para que a deformação de modelos não estruturados geometricamente complexos seja feita a taxas interativas, todas as operações requeridas pela técnica devem ser feitas o mais eficientemente possível. Este critério foi prioritariamente levado em consideração na formulação das soluções apresentadas neste trabalho.

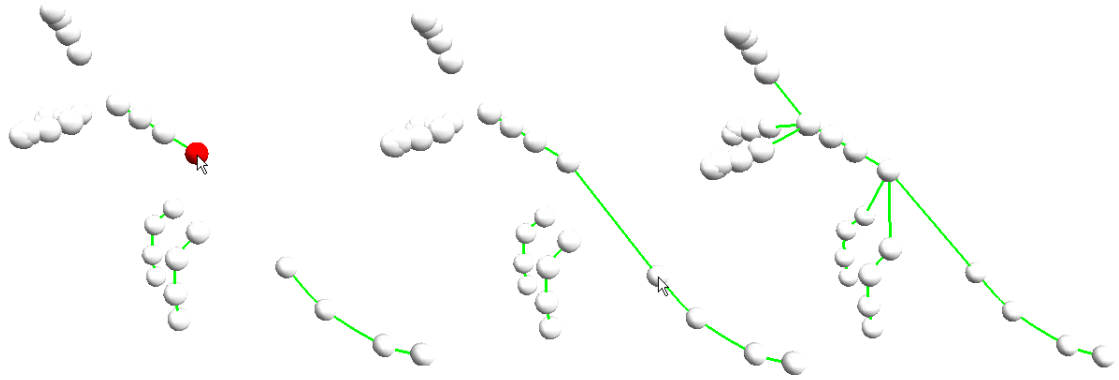


Figura B.2: Passo dois da técnica. O usuário tem a possibilidade de combinar curvas paramétricas para criar esqueletos complexos.

Inicialmente, a curva rabiscada pelo usuário é filtrada para remover o ruído decorrente da possível imprecisão contida na entrada do usuário. Pontos do rabisco filtrado são então projetados na superfície do objeto e um algoritmo de crescimento de regiões é aplicado para encontrar a região da malha que está sob o *sketch* do usuário. O restante da malha é também identificado por um algoritmo de crescimento de regiões. O rabisco também é utilizado para criar o manipulador usado pelo usuário nas deformações, ajustando uma curva paramétrica aos pontos do rabisco projetados na superfície do objeto.

As deformações impostas à curva são transferidas para a malha através de um conjunto de sistemas de coordenadas criado ao longo da curva. A abordagem para a criação deste conjunto de sistemas de coordenadas foi elaborada para respeitar certas condições que não estão presentes no Frenet *frame* (CARMO, 1976). Os vértices da malha são representados com respeito a este conjunto de sistemas de coordenadas e, quando o usuário interage com a curva, um novo conjunto de sistemas de coordenadas pode ser obtido na curva deformada. Assim, é possível reconstruir a malha utilizando o novo conjunto o que resultará em uma malha deformada de acordo com a deformação que o usuário impôs à curva.

Os vértices encontrados na região sob o rabisco serão transformados diretamente por uma curva paramétrica. A fim de garantir a continuidade entre as regiões enquanto o usuário interage com a curva, duas abordagens podem ser consideradas. Primeiro, se a curva não está conectada a nenhuma outra curva, as regiões associadas às extremidades da curva são rigidamente transformadas. Alternativamente, se o usuário conectou várias curvas a fim de construir um esqueleto para o objeto, uma função de mistura é usada para suavemente parar a deformação entre cada curva.

Como os vértices da curva estão representados com respeito ao conjunto de sistemas de coordenadas criados ao longo da curva, efeitos interessantes como torção e escalamento podem ser obtidos com operações simples diretamente sobre os eixos destes sistemas de coordenadas. A Figura B.4 mostra uma torção sendo realizada no modelo de um bloco e os sistemas de coordenadas como foram rotacionados para obter o efeito de torção.

Um algoritmo para evitar que a malha deformada sofra de interpenetrações também foi desenvolvido com operações simples sobre os eixos dos sistemas de coordenadas. Este algoritmo pode também ser executado a taxas interativas mesmo para modelos geometricamente complexos, assim como toda a técnica apresentada neste trabalho.

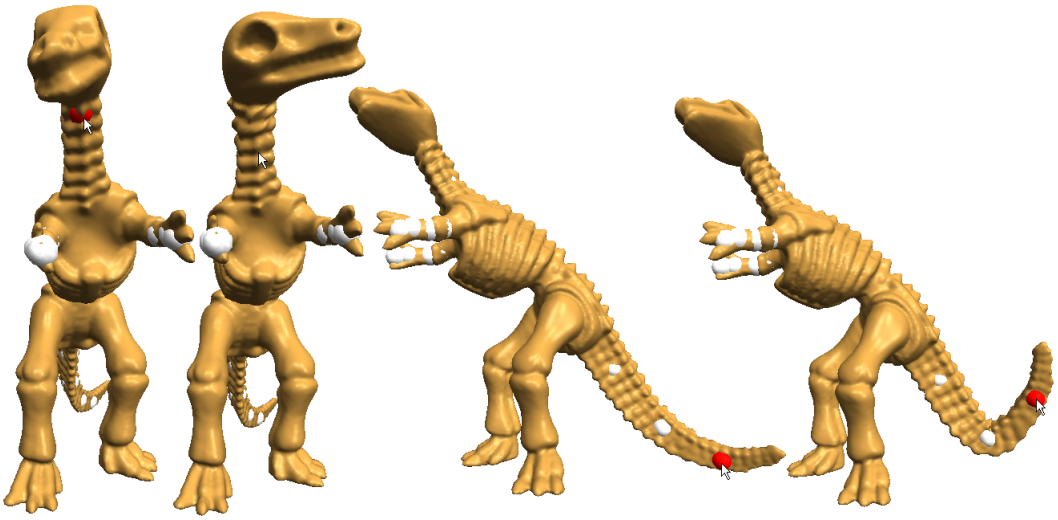
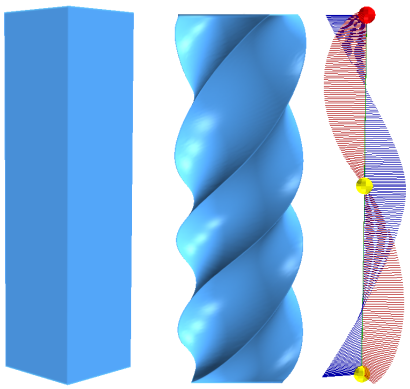


Figura B.3: Passo três da técnica. O usuário modifica uma curva 3D e a técnica deforma a malha. Pontos de vista diferentes podem ser usados.



(a) (b)

Figura B.4: Operação de torção obtida com a técnica apresentada. Um bloco de referência (a). Bloco torcido (b) obtido interpolando-se uma rotação ao longo do conjunto de sistemas de coordenadas da curva.

B.2 Contornos e contornos sugestivos

Contornos e *contornos sugestivos* são linhas na superfície de um objeto que carregam importantes informações sobre a sua forma. Por essa razão, dar ao usuário a possibilidade de editar objetos modificando estas linhas parece bem intuitivo. Por essa razão, a técnica descrita neste trabalho foi estendida para transferir deformações aplicadas aos *contornos* e *contornos sugestivos* em deformações de malha.

Curvas paramétricas são criadas a partir de linhas extraídas com base em *contornos* e *contornos sugestivos*, da mesma forma que é feito com os rabiscos criados pelo usuário. A deformação das curvas é então transferida para deformação de malhas da mesma forma que feito anteriormente. Entretanto, uma função de suavização baseada em uma aproximação de distâncias geodésicas é proposta de forma a obter resultados melhores ao se editar detalhes locais.

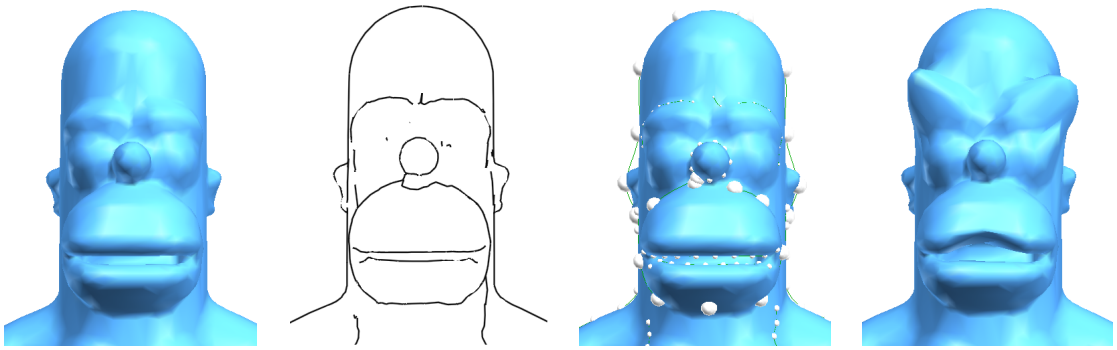
Como essas linhas geralmente são mais simples do que a malha mas ainda representam bem os detalhes do objeto, é fácil perceber que elas podem se tornar uma boa interface

para um artista expressar suas idéias quando modifica um modelo 3D. Dar ao usuário a possibilidade de editar contornos sugestivos, e dessa forma modificar a malha associada, parece uma abordagem promissora para deformar objetos complexos porque o usuário pode abstrair algumas partes da malha concentrando-se nos detalhes que carregam mais informação. De fato, rabiscar uma forma ou editar contornos sugestivos pode ser viso como o inverso do *rendering* não foto-realístico, como já foi apontado por Nealen et al. (NEALEN et al., 2005). Entretanto, o único trabalho conhecido que faz uso de contornos sugestivos para realizar deformação de malhas é o de Nealen et al. (NEALEN et al., 2005).

Contornos sugestivos podem ser eficientemente computados para objetos representados por malhas 3D. As linhas extraídas podem então ser usadas para criar curvas paramétricas que são associadas à malha do objeto como descrito anteriormente. Assim, uma sessão de modelagem típica usando a técnica apresentada nesta seção consiste em dois passos principais:

- Escolher um ponto de vista onde os contornos sugestivos mostrem uma linha representando um detalhe de interesse a ser modificado;
- Modificar este detalhe movendo os pontos de controle da curva paramétrica ajustada ao contorno sugestivo.

A Figura B.5 mostra um exemplo desta técnica. Dada uma malha (a), seus contornos sugestivos são automaticamente calculados para cada ponto de vista (b). Para modificar a malha no ponto de vista corrente, curvas paramétricas são ajustadas às linhas de contornos sugestivos e usadas como manipuladores (c). Movendo os pontos de controle destes manipuladores, a malha é apropriadamente deformada (d). Essa técnica mostrou-se particularmente interessante para editar detalhes da malha, como as sobrancelhas do modelo do Homer.



(a) (b) (c) (d)

Figura B.5: Deformação de malha usando contornos sugestivos. Curvas paramétricas (c) são ajustadas às linhas dos contornos sugestivos mostrados em (b). Movendo os pontos de controle das curvas paramétricas, o usuário pode deformar o objeto (d). A boca foi aberta e as sobrancelhas levantadas. O modelo original é mostrado em (a) para comparação.

A forma como as curvas são associadas à malha é exatamente a mesma usada para o caso dos rabiscos. Entretanto, a função de suavização é calculada usando uma aproximação da função de distância geodésica das linhas de contorno sugestivo.

B.3 Discussão

A principal contribuição deste trabalho é uma nova técnica para deformação de modelos 3D que produz resultados visualmente interessantes a taxas interativas preservando os detalhes locais de suas superfícies. A suavidade inerente às curvas paramétricas é transferida às deformações globais, enquanto que um conjunto de sistemas de coordenadas ao longo das curvas garante que os detalhes locais, representados com respeito a estes sistemas de coordenadas, são preservados sob transformações globais. Dessa forma, a abordagem apresentada é relacionada às recentes técnicas de deformação de malhas baseadas em modelos multi-resolução (BOTSCH; KOBBELT, 2004; KOBBELT et al., 1998), coordenadas Laplacianas (ALEXA, 2003; LIPMAN et al., 2004, 2005; SORKINE et al., 2004) e no domínio gradiente (HUANG et al., 2006; YU et al., 2004; ZAYER et al., 2005; ZHOU et al., 2005). É demonstrado que, apesar de sua simplicidade conceitual, a abordagem proposta é bem geral, e não sofre de algumas limitações encontradas na maioria das técnicas anteriores, superando-as em termos de velocidade como mostra a Tabela B.1.

Tabela B.1: Tempos obtidos usando a técnica apresentada neste trabalho para deformar diversos modelos. A coluna "Curva" refere-se ao tempo de segmentação da malha em regiões e criação de uma curva paramétrica. A coluna da direita mostra a performance da deformação (em fps) ao deformar todos os vértices do modelo.

Modelo	**Vértices**	**Triângulos**	**Curva (s)**	**Deform. (fps)**
Horse	19,851	39,698	0.0545	59.52
Dino	56,194	112,384	0.1550	34.60
Bunny	76,000	151,996	0.1339	29.85
Armadillo	172,974	345,944	0.2914	14.94
Dragon	437,645	871,414	1.0697	7.12
Buddha	543,652	1,087,716	3.4130	4.50