ILLUSTRATIVE DEFORMATION OF VOLUMETRIC OBJECTS AND OTHER GRAPHICAL MODELS

BY CARLOS D. CORREA

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Deborah Silver

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

May, 2007

ABSTRACT OF THE DISSERTATION

Illustrative Deformation of Volumetric Objects and Other Graphical Models

by Carlos D. Correa

Dissertation Director: Prof. Deborah Silver

The purpose of visualization is to gain understanding of 3D structures through images. Although many rendering techniques have been proposed for this purpose, the effective visualization remains a challenging task, due to occlusion, clutter, noise in the data, and acquisition pose. Recent solutions to this problem deal with transfer functions and other rendering techniques to enhance the visibility of certain parts of interest. At the core of these techniques is the assumption that the user's role is passive and that the data remains unchanged.

In this thesis, we explore a more active approach to visualization, where a scientist can manipulate a dataset as if deforming a real model. We call this type of manipulation *Illustrative Deformation*. Our approach draws the name from the types of deformations that are often depicted in scientific illustration, which are used to enhance visibility of certain features while providing context, or to abstract the structure of an object or procedure. Inspired by medical and surgical illustration, our approach was designed to reproduce some of their key properties: illustrations often contains cuts and dissections, they allow feature-sensitive operations, which can be applied to a semantic component of the object without affecting other parts of the object, and they enable virtual operations, which do not necessarily conform to reality, but are useful for understanding the structure of complex objects.

Our approach is based on a generalized notion of 3D displacement maps, which unifies

the specification of continuous and discontinuous deformations on both volumes and surface-based objects. We show how displacements can describe a wide range of transformations, including cuts and peels, how they can be extended to include feature-sensitive operations, and how can they be implemented to obtain high quality interactive rendering on commodity hardware. We also show how our approach can be extended to the deformation of surface-based models without the need for remeshing. Through a number of examples and quantitative results, we demonstrate the generality, flexibility and scalability of our approach, and we explore its applications in medical illustration, surgical planning and simulation, and as a general tool for visualization and computer graphics.

Acknowledgements

I would like to thank my advisor, Professor Deborah Silver, for her vision and encouragement throughout this research. This thesis would not have been possible without her support and opportune guidance. I also want to thank Professor Min Chen for his valuable input and his meticulous revision of notation and form.

I wish to thank Dr. Stanley Trooksin, Dr. Sid Roychowdhury and Dr. Marsha Jessup of the Robert Wood Johnson Medical School for valuable input on surgical and medical illustration. I am grateful to the National Library of Medicine for providing the Visible Human dataset, the AIM@Shape repository for the surface-based models and Stefan Roettger for providing and collecting a number of volume datasets. I wish to acknowledge the sources of the datasets in Stefan Roettger's collection.

I wish to thank my colleagues and friends from the Vizlab, who made these past years the best of my graduate studies, for their timely advice and proof-reading of my papers.

Finally, I want to thank my parents Guillermo and Elsa for all their support and encouragement during my studies and for giving me a thirst for knowledge and desired for intellectual achievement. Last, but not least, I want to thank Maria, for her constant support. This thesis would not have been possible without her.

Dedication

To

My parents, Guillermo and Elsa

To

Maria

Table of Contents

List of Tables

List of Figures

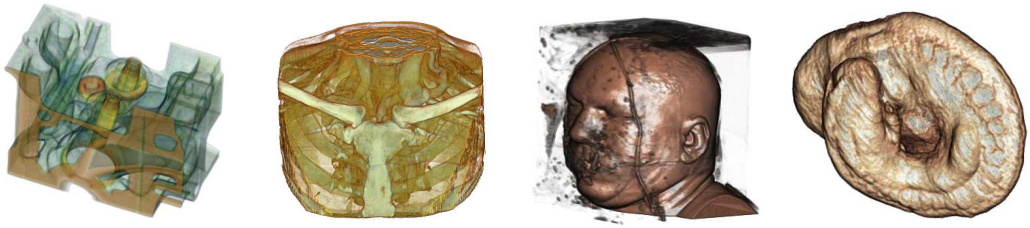
List of Notation

Chapter 1
Introduction

The purpose of visualization is to gain understanding of 3D structures through images. Although many rendering techniques have been proposed for this purpose, the effective visualization remains a challenging task, due to occlusion, clutter, noise in the data, and acquisition pose. Many recent solutions to this problem deal with transfer functions and other rendering techniques to enhance the visibility of certain parts of interest. At the core of these techniques is the assumption that the user's role is passive and that the data must remain unchanged. One of the ways in which people interact with complex objects in "real life" is by direct manipulation. In this thesis, we explore a more active approach to visualization, where a scientist can deform a dataset as if deforming a real model. We call this type of manipulation *Illustrative Deformation*. Our approach draws the name from the types of deformations that are often depicted in scientific illustration, which are used to enhance visibility of certain features while providing context, or to abstract the structure of an object or procedure.

1.1 Motivation

The work in this thesis draws inspiration from two observations about the visualization process. First, proper visualization of complex objects is sometimes difficult due to occlusion, clutter, noise or acquisition pose. 3D objects are obtained via a variety of methods. Surface models are often obtained artificially through a 3D modeling and sculpting software or scanned from a real object. These models represent only the exterior of the object. Volumetric models, on the other hand, represent also the interior of the object. These are acquired using computed tomography (CT) or magnetic resonance imaging (MRI), or obtained as the result of a simulation or procedural definition. Volumetric objects are very common in biomedicine and engineering since it is important that one can visualize the external and internal structure of objects. Figure 1.1



(a) Occlusion (b) Clutter (c) Noise (d) Pose

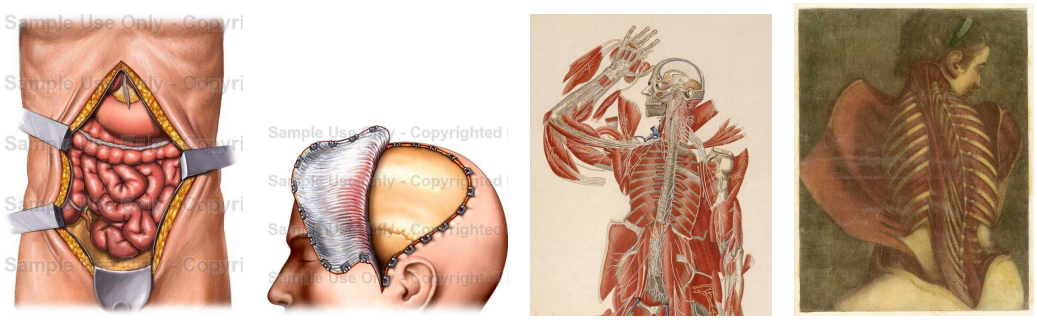
Figure 1.1: Common problems in the visualization of 3D datasets

shows examples of volumetric objects obtained via CT or MRI and some of the visualization problems. In the engine dataset (a), we can see that the proper visualization of the internal structure is complicated due to occlusion by a metallic plate. The neck dataset (b) is an MRI of a patient's neck. Visualization of the internal structures, e.g., the thyroid gland, is complicated not only due to occlusion by the skin and muscle layers, but also due to clutter, as many organs and glands are present in the region. The visman head dataset (c) is part of the Visible Human Project, a complete 3D representation of a male donated to the study of human anatomy, from the National Library of Medicine. In this case, noise prevents proper visualization of the object of interest. In Figure 1.1(d), we have the mouse fetus dataset. Note that acquisition pose, prevents some parts of interest, namely the head, from being visible.

The second observation is drawn from medical illustration. Medical and biological illustrators are trained to depict certain procedures and anatomical structures in such a manner that they have the most value in terms of communicating a certain idea, for education, legal or for surgical planning. In many of these illustrations, the illustrator is faced with the problem of making visible features that otherwise may not be visible, while still maintaining a contextual view of the surrounding object.

For this purpose, hand-drawn illustrations often include manipulating parts of an object to depict the stages and outcome of a procedure, uncover hidden features, or reveal the spatial relationship between different components of the objects. Such manipulation typically includes the following characteristics:

- It often contains *cuts and dissections*, which, for example, are commonly found in illustrations of surgical procedures as exemplified by Figures 1.2(a) and 1.2(b).



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

Figure 1.2: Example medical and anatomical illustrations. (a) Abdominal surgical operation (courtesy of Nucleus Inc.) (b) Craniotomy (courtesy of Nucelus Inc.) (c) Anatomical illustration with dissected "flaps", by Antonio Scrantoni and Paolo Mascagni, 1833 (U.S. National Library of Medicine) (d) The Flayed Angel, Jacques Gautier D'Agoty, 1746 (U.S. National Library of Medicine)

- It may allow *feature-sensitive operations*, which can be applied to a semantic component of the object, such as the skin in Figure 1.2(b), without affecting other parts of the object, such as the skull.

- It may enable *ubiquitous operations*, which can be applied to various parts of the object with different geometric transformations, as shown in Figure 1.2(c).

- It can facilitate *virtual operations*, which do not necessarily conform to the reality, such as the unreal flaps used to illustrate anatomical structure in Figures 1.2(c) and 1.2(d).

Another motivation of our work is the increasing requirement for deformation techniques for special effects and computer graphics in general. Rendering of special effects is often performed as an offline process, where a high resolution surface model of an object is progressively transformed to obtain the desired effect. A number of interactive techniques have been proposed to speed up the process or to work as a pre-visualization tool, where the artist can rapidly prototype a draft version of the special effect without incurring in long hours of rendering time. One of the problems in standard computer graphics is the requirement for re-meshing when large deformations or cuts need to be simulated on polygonal objects. One of the motivations for our approach was the possibility to treat polygonal objects as volumetric representations and allow the simulation of complex deformations without re-meshing.

1.2 Volume Rendering and Illustration

Volume datasets are object representations commonly obtained by sampling a volumetric object in a regular grid. Datasets are often acquired from CT or MRI scanners, defined procedurally, or as a product of a simulation. Rendering of volumetric datasets can be done indirectly by obtaining first an isosurface of interest and then rendering it with a traditional polygon renderer [68] or directly. With the increasing power of graphic processor units (GPU), it has been possible to implement direct volume rendering (DVR) on commodity hardware. One of the most common mechanisms for interactive volume rendering is via 3D textures, where a volume is sliced into view-aligned polygons and each slice is textured and composited along the view direction [125]. Exploration of the different features of the dataset is commonly done via the manipulation of transfer functions. With the advent of programmable GPUs, it is also possible to perform interactive raycasting, where each pixel generates a ray that is used to traverse the volume and perform the compositing of color and opacity values [96, 62]. Although slicing is equivalent to tracing uniform rays along a volume, GPU-based raycasting opens the possibility for techniques to improve the speed and quality of the rendering, such as adaptive sampling, antialiasing, pre-integrated volume rendering [36] and empty-space skipping [70].

The exploration of volume datasets via transfer functions is prevalent. A number of techniques have been proposed to improve its use [60], but it has been shown to be a complex task. One of the main challenges is the problem of visibility, where the user wants to have a clear view of a feature of interest and a clear view of contextual information at the same time. A number of solutions have been proposed, motivated by illustration techniques:

- Non photorealistic techniques allow the rendering of different parts using non traditional methods, resembling hand drawn illustrations [34, 116, 111, 12, 115, 107].

- Cutaway and ghosted views allow occluded objects to be rendered by fading [117], removing [30, 124, 14] or distorting [123] occluding parts.

- Selective rendering allows the rendering of semantic parts of the object in multiple modalities to improve the understanding of the structure of complex objects and enhance visibility of occluded parts. This can be obtained via two-level rendering of segmented

volumes [45], or through volume decomposition [104, 24]

The above-mentioned focus+context techniques rely on the manipulation of viewing attributes of the rendering engine and optical attributes of the data objects. Some solutions cannot effectively resolve the occlusion problem. Others can, but at the cost of decreasing the useful contextual information. Often the contextual information is completely suppressed. For example: (a) techniques based on distorted viewing often have difficulties to remove occlusive 'context' to reveal the intended 'focus' of the interior structures; (b) techniques based on reducing opacity of the occlusive objects can sometimes in fact remove the important 'context' that is relevant to the 'focus' to be revealed. Many of these techniques do not have a clear boundary between focus and context, it is often difficult for users to determine in a focus+context visualization whether a specific part of an object has the original geometry (or opacity) or the magically changed geometry (or opacity).

Another way of providing visibility is through interactive deformation, including the ability to cut and peel parts of an object. Deformation can be defined as the change in time of the shape of an object, and has been widely explored for surface meshes, but relatively unexplored for volumes. In this thesis, we explore the interactive deformation of volumetric objects.

1.3 Object Deformation

Deformation techniques have been commonly categorized as *non-physics-based* vs. *physics-based*. In the former, we find techniques that allow "free" deformation of an object's primitive elements, with little or no regard to the physical realism of such deformation. In the latter, we find techniques where deformation is driven as the result of applying a number of forces and solving for a subset of the physical equations governing the dynamics of the object. However, the line between the two categories is becoming blurred. This comes as the result of the introduction of constraints to otherwise non-physics-based deformation to account for desired properties found in "real-life" situations, such as volume conservation and prevention of self-intersection. In this thesis, we refer to non-physics-based deformation as those methods where deformation is not obtained explicitly as the product of applying forces to a mesh, volume or set of points. Furthermore, dynamic deformation effects, such as inertia and secondary forces,

are of little importance. Instead, deformations are defined empirically as a local or global transformation of an object, defined by the user through direct manipulation, procedurally, or as a combination of simpler deformations. Finally, here we further categorize *Illustrative Deformation* as a subset of non-physics-based deformation, where constraints such as self-intersection and volume conservation are loosely enforced, and emphasis is given to high quality rendering and interactivity. The term is analogous to the idea of *illustrative rendering*, where the appearance of rendered objects does not adhere necessarily to the physics of light transport, but it has an enhanced communicative value.

Another dichotomy in deformation is its definition as a modeling or rendering stage. Deformation as a modeling stage is aimed towards the creation of a new object which can be subsequently transformed and rendered independently. Most surface-based deformation techniques fall into this category. Volume deformation as a modeling process is problematic, since the deformed volume needs to be sampled at a suitable resolution. Re-sampling volumetric objects is more space demanding than for surface objects. In many cases, the size of the deformed volume exceeds the texture memory available for interactive rendering. As a rendering stage, deformation is obtained on-the-fly, without the need for an intermediate object representation. For volumetric objects, this becomes critical, as it does not require extra memory for intermediate frames, and the sampling resolution is obtained on-the-fly by the pixel resolution of the resulting image. In this thesis, we address the problem of interactive deformation as a rendering process.

Interactive deformation of volumetric objects is a difficult problem due to their limited geometric and topological information. Volumetric objects can be defined as a sampled representation of a continuous region in 3D space. In surface models, deformation can be reduced to the transformation of its vertices, since continuity is defined explicitly through a mesh. In contrast, volumetric objects do not have a explicit connectivity and deformation techniques cannot be simply reduced to the transformation of voxels. Because of the assumed continuity of volumetric objects, the simulation of cuts and breaks is also challenging. Dealing with cuts introduces additional challenges in deformation, namely, determining the shape and size of the cut, rendering smooth surfaces that appear as the cut or break is applied, and allowing the incorporation of semantics for meaningful deformation.

1.4 Contributions

In this thesis, we present a method for interactive deformation of volumetric objects, which unifies continuous and discontinuous deformations in a single framework. Our rendering approach enables the simulation of high quality rendering of volume deformation at interactive rates. The following contributions are made:

- A generalized method for modeling continuous deformations as 3D displacements, and a method for encoding discontinuities into 3D displacements suitable for volume deformation. This encoding preserves C^1 continuity in the regions near the cuts and breaks, required for high quality rendering of shaded volumes.

- A method for introducing volumetric constraints into deformation, for efficient rendering of illustration-inspired images. These constraints enable deformations that are feature *aligned*, so that parts of interest are preserved and made visible to the user or feature *anchored*, so that certain parts are deformed smoothly between a deformable region and a rigid region, such as fatty tissue between skin and bone tissues.

- A fast method for the estimation of normals for volume regions undergoing deformation and for regions in the vicinity of cuts.

- A novel method for surface-based deformation without re-meshing, which extends the notion of volume deformation to sampled layered representations of polygonal objects, such as depth maps, layered depth images and signed distance fields.

1.5 Organization of the Thesis

This thesis is organized as follows: Chapter 2 provides a comprehensive description of the state-of-the-art in the deformation of *sampled object representations*, including volumes and implicit surfaces.

Chapter 3 describes an overview of our approach.

Chapter 4 describes the basic deformation mechanism, which we dubbed *discontinuous displacement mapping*. We describe the general notion of deformation as well as a GPU-based implementation. A number of examples are used to demonstrate the capabilities of our

approach.

One of the challenges of this approach is to represent feature sensitive operations, which are required for illustration and visualization. Chapter 5 describes a method for realizing such feature sensitive operations, through *alignment* of the deformation. We describe a method that allows a deformation to align with a particular line or curve. We also describe a more effective method for aligning the deformation with a particular *feature* of interest.

Chapter 6 provides an evaluation of volumetric deformation. This evaluation is two-fold. On one hand, we evaluate the rendering quality of our deformations via a quantitative analysis of the properties of the displacements and the rendering process. On the other hand, we evaluate the performance cost of our system. This evaluation helps us validate the interactivity and applicability of our approach. Further, it serves as a benchmark for future volume deformation methods.

Chapter 7 marks the beginning of Part II, where we extend our mechanism to the deformation of surface-based objects. We describe a method for deforming surface meshes without the need for remeshing. This is obtained by representing the regions undergoing deformation as a sampled *layered representation*, such as depthmaps or distance fields. We extend our rendering method to find accurate intersections with the surface mesh, and to integrate seamlessly with the mesh representation of the parts that are not deformed.

Chapter 8 provides an evaluation of our approach as applied to surface deformation. We compare our approach to mesh deformation and measure the performance across several dimensions.

Chapter 9 describes the applications of our approach and validates our method through a number of examples. Our method has applications in medical and biological illustration, surgical planning, volume clipping and as a focus+context visualization tool.

Chapter 10 presents some conclusions and directions for future work.

Chapter 2
Related Work

2.1 Introduction

Deformation refers to the change in time of the position and orientation properties of graphical elements. There has been a considerable amount of research for the deformation of surface meshes. Deformation is obtained by directly transforming the position of the vertices of the mesh. In some cases, re-meshing is needed to account for large deformations or cuts, which may change the explicit connectivity of the mesh. Volumetric datasets, in contrast, are represented using voxels, three-dimensional points that include appearance properties, such as opacity and color. Unlike surface meshes, connectivity information is not explicit. One of the most common way of representing them is by defining a regular grid in three-dimensional space. Rendering of such a grid is performed by integrating the color or attenuation of a ray traversing the volume. The use of a regular grid has been exploited by contemporary GPUs, as their 3D texture capabilities are better suited for regularly placed samples. However, this condition also complicates the efficient deformation of volumetric datasets. Unlike surface meshes, simply transforming the voxels makes it impractical for current ray-casting or texture-based algorithms, as the regular grid is deformed into an irregular grid. This chapter describes previous work in the field of volume and mesh deformation. The use of volumetric models also enables a number of techniques that have not been possible with surface meshes, such as cuts. Since the volumetric model contains information of the inside of the object, it directly benefits from discontinuous deformations. Surface meshes, on the other hand, do not contain information of the interior, so it must be synthesized artificially. This chapter also discusses the state-of-the-art in cutting and other discontinuous deformations.

Volumetric objects can be considered Sampled Object Representations (SOR), which define graphical models using data obtained by a sampling process, which takes a collection

of samples at discrete positions in space in order to capture certain geometrical and physical properties of one or more objects of interest. In our review of the state-of-the-art, we begin introducing the notion of Sample Object Representations, then we describe the methods for deforming SORs, placing particular emphasis on those techniques that introduce deformation as a stage in the rendering process. Then, we describe techniques for volume cutting and other discontinuous deformations, and finally, we conclude with a review of the state-of-the-art in mesh deformation.

2.2 Sampled Object Representations[1]

The general notion of a *sampled object representation* (SOR) is a set of samples $V = \{(\mathbf{p}_i, v_i) | i = 1, 2, \ldots, n\}$, where v_i is a value of a specific data type (e.g., Boolean, scalar, vector or tensor), which represents some property at each sample location $\mathbf{p}_i$ in k-D Euclidean space E^k. Typically these samples are associated with a spatial domain D^k, which is normally continuous or consists of several disjoint sub-domains. An object specified by an SOR is thus a function $f(\mathbf{p})$ that defines the value at every $\mathbf{p} \in D^k$ [19].

Digitization is the primary technology for acquiring SORs of real-life objects and phenomena. This technology, which is based on measuring various physical properties, is available in a wide range of modalities as listed in Table 2.1. In most of these modalities, a sampling process may involve the processing of multi-channel or multi-dimensional signals, including convolution and deconvolution, quantization, and signal space conversion.

In some modalities, sampling positions are defined by a regular grid in the object space. For example, computed tomography (CT) scanning normally utilizes a 3D anisotropic grid, where the sampling interval in the z-direction differs from that in the x and y directions. In many other modalities, sampling positions are defined by a regular grid in the image space. A primary example of such modalities is photography, where sampling results are recorded on a 2D isotropic grid though individual samples may not correlate uniformly to signal sources in the object space.

[1] Part of this section was published in our paper: Deforming and Animating Discretely Sampled Object Representations, M. Chen, C. Correa, S. Islam, M.W. Jones, P.-Y. Shen, D. Silver, S. J. Watson, P.J. Willis, Eurographics 2005, State of the Art Reports, pp. 113–140

Example Sampling Modality (*physical property*)	Data Dimension	Number of Channels	Representation Scheme
Black-white photography (*light reflection*)	2	1	2D regular grid
Color photography (*light reflection*)	2	3	2D regular grid
Raw laser scans (*distance to a plane*)	2.5	1	2D regular grid
Circular full-body scans (*distance to an axis*)	2.5	1	2D curvilinear grid
Computed tomography (*X-ray attenuation*)	3	1	3D regular grid
Magnetic resonance imaging (*relaxation of magnetized nuclei*)	3	1	3D regular grid
Raw 3D Ultrasonography (*sonic reflection*)	2.5	1	unstructured 2D images
Processed 3D Ultrasonography (*sonic reflection*)	3	1	3D regular grid
Electron microscopy (*electron diffraction*)	3	1	3D regular grid
Spatial distance fields (*distance to a surface*)	3	1	3D regular grid
Spatial vector fields (*e.g., velocity*)	3	3	3D regular grid
3D photographic imaging (*light reflection*)	3	3	3D regular grid
Movies and videos (*time-varying light reflection*)	3	3	3D regular grid
Particle simulation results (*space-time position, etc.*)	4	1	time-series, 3D point set
Motion capture data (*space-time position*)	4	1	time-series, 3D point set
Seismic measurements (*space-time density, temperature, etc.*)	4	n	time-series, 2D point set

Table 2.1: Example data capture modalities, and their typical characteristics and representation schemes.

SORs can also be obtained by sampling continuous object representations. For example, a continuous surface representation can be approximated by an unstructured point dataset using a randomized discretization process or by a volume dataset using a voxelization process. In many science and engineering disciplines, such as finite element analysis and computational fluid dynamics, SORs are commonly used to approximate continuous spatial and temporal data representations derived from theoretic studies, scientific modeling and computer simulation.

SORs commonly exhibit a subset of the following characteristics, which collectively signify the differences between SORs and other schemes for representing graphical objects and scenes.

- *Limited geometrical information* — Most SORs do not contain any explicit geometrical description of the objects represented, while some contain partial geometric information (e.g., in a point set). It is common to translate sampled physical information (e.g., X-ray attenuation) to geometrical information (e.g., an isosurface of a tumor). In addition, SORs are particularly suited for modeling amorphous objects, such as fire, dust and smoke, for which a precise geometrical description is difficult to obtain.

- *Limited topological information* — The only topological information available in a SOR is the spatial or temporal order in which samples were captured. Such information does not imply a definite topological relationship between any two data points in the object space, although it is often used to derive, analytically or statistically, more meaningful topological information, such as the possible connectivity between two sampling points

in the context of 3D model acquisition and the association of a set of voxels to the same object in the context of segmentation.

- *Little semantic information* — Although a SOR, such as a photographic image and a computed tomography scan, may capture a collection of objects in a scene, it does not normally contain any semantic information, about the objects of interest, such as object identification and object hierarchy.

- *Multiple data channels* — Many SORs capture data from a complex signal source (e.g., reflectance) or multiple signal sources (e.g., a combination of density, sonic, temperature and imagery logging in seismic measurements).

- *Multi-valued data channels* — Many SORs contain data sampled in an integer or floating-point real domain. In some situations, this facilitates a high level of accuracy (e.g., the texture of a piece of textile in an image), but in others, this brings about a degree of uncertainty (e.g., the boundary of a piece of textile in an image).

2.2.1 Sampled Layered Representations (SLR)

Many object representations are specified in an analytical manner, for instance, using a mathematical function to define the shape of an object. Such a representation is referred to as an *analytical object representation* (AOR). One of the most popular analytical objects are parametric curves. In general we can consider surface meshes as piecewise AORs, where the surface is usually approximated by a triangular patch. SORs are often used to represent analytical object representations (AORs) as well. The use of SORs to represent triangular meshes has been proposed in several occasions, usually by defining a grid and sampling the shortest distance to the surface along one, two or three dimensions. Such a representation is said to be an *implicit* representation of the surface. Because the sampling is usually done along *layers*, we call them *Layered Representations*. This name is also consistent with one of the first applications of this idea for the representation of image-based objects, called Layered Depth Images (LDI) [102].

In the simplest case, an SLR can be a depth map or relief map [87], where each sampled position stores the closest distance to the surface along a given direction. One of the most common uses of depth maps is displacement mapping, first introduced by Cook [22], where each

depth value refers to a displacement added to the meso-structure of an object to create realistic details. The use of displacement mapping as an object representation has been suggested by Xu et al. [131] and Kautz [57]. Another type of layered representation is a multi-layer depth map, which overcomes the limitations of depth maps for representing objects with concavities, by combining a number of depth maps into a single structure. Policarpo and Oliveira used four depth maps to represent non-height-field surface details on objects [90]. This idea was further developed by Wang et al. [122, 121] in their generalized displacement maps. Instead of 4 depth maps, Wang et al. allow any number of layers, described as a 3D texture. Unlike the depth maps, this structure stores the distance to the closest point in the meso-structure surface.

A generalization structure is *signed distance fields*, a 3D grid where each point stores the distance to the closest point to the surface, or, in some cases, it also stores the coordinates of the closest point and the direction of the gradient. A number of methods for computing the signed distance field of polygonal objects have been proposed, e.g., [103, 110]. Distance fields may require considerable memory. As an alternative, we can use a collection of arbitrarily oriented depth maps. One example is a *depth map cube*, which uses depth maps along the faces of a cube to represent a complex object, such as in [57]. Note that for objects with concavities and hidden features, a simple cube may not be sufficient. It is possible to improve the representation of the object by subdividing the cube into a more complex geometry, closely bound to the desired surface. This leads to the displaced subdivision surfaces proposed by Lee et al. [66], and displacement volumes [10].

2.3 Deforming Sampled Object Representations

In this thesis, the term *deformation* refers to intended change of geometric shape of an object under the control of some external influence such as a force. To facilitate the computation of geometric changes, a *deformable model* normally has two primary components, a data representation and an algorithm based on a physical or mathematical concept. Applications of deformation techniques include computer animation, object modeling, computer-aided illustration, surgical simulation, and scientific visualization. Here, we distinguish two types of methods for deforming SORs, empirical deformable models and physically-based models. However, the

line between the two types of deformations is becoming blurred. This comes as the result of the introduction of constraints to otherwise non-physics-based deformation to account for desired properties found in "real-life" situations, such as volume conservation and prevention of self-intersection. In this thesis, we refer to non-physically-based deformation to those methods where deformation is not obtained explicitly as the product of applying forces to a mesh, volume or set of points. Furthermore, dynamic deformation effects, such as inertia and secondary forces, are of little importance.

2.3.1 Empirical Deformable Models

Empirical deformable models are non-physically-based deformable models which are designed to imitate physical behaviors of deformable objects with little or very limited physics in their computation algorithms.

Some of the methods that fall into this category are:

- *Global and local deformation* [6, 5] to volume datasets through ray deflectors [64] or spatial transfer functions [20].
- *Free-form deformation* [100] to volume datasets through volume bounding boxes [18]
- Skeleton-based volume deformation such as volume wires [120] or the approach in [105].
- Pre-defined procedural deformation specifications to segmented volume datasets in interactive data exploration [76];
- *Implicit models* as a parametric control for deforming the volume dataset [52];
- Splitting operations to volume datasets and hypertexture in a combinational manner using spatial transfer functions [54].
- *Chain-mail algorithms*, which uses the grid topology in a volume dataset to propagate displacements [42].

2.3.2 Physically-Based Models

Although empirical models can be implemented in real-time for very large datasets, accurate deformations cannot always be realized, especially in emulating physical responses to an input force. For this reason, there have been many physically-based models proposed for deformation.

Almost all physically-based models are associated with a mesh data representation, typically with triangular or rectangular elements for surfaces and tetrahedral or hexahedral elements for solids or volumes. In most applications involving SORs, such data representation can be extracted or reconstructed using a number of techniques, such as marching cubes [74], Delaunay tetrahedralization, among many others.

Typical physically-based models include *continuum mechanics*, *mass-spring systems*, *particle systems*, *smoothed particle hydrodynamics* and *fluid dynamics*. In these models, a deformable object is essentially a function of the forces acting on the material properties of the object. Deformation is computed is by finding a solution to the equilibrium state of energy functionals. *Finite difference*, *finite element* and *finite volume* methods are commonly used to obtain approximate solutions of mesh-based partial differential equations found in the Lagrangian formulation of motion in continuum mechanics [82].

- *Mass-spring Models* — In these models, an object is approximated as a finite mesh of points. The mechanics of deformation is defined as coupled ordinary differential equations, which specifies equilibrium at the mesh points. Vertices are adopted as nodes in a mass-spring model, which are connected via springs to their neighbors. An initial condition can be assigned to each vertex and the internal force acting on a vertex is calculated based upon its local neighbors. This force is then used to calculate vertex motion using Newton's law of motion.

- *Finite Element Methods* (FEM) — Unlike mass-spring models, where the equilibrium equation is discretized and solved at finite mass points, the FEM system is discretized by representing the desired function within each element (e.g., line, triangular, quadrilateral, tetrahedral and hexahedral elements), as a finite sum of element-specific interpolation functions. FEM is used extensively in computer graphics for deformation (e.g., [98,

11]). In computational science, FEM is normally used in conjunction with a non-linear elasticity model, while it is common in computer graphics to employ a reduced linear model as discussed below.

- *Low Degree-of-freedom Models* — For many applications, such as surgical simulations and interactive modeling, real-time solutions are necessary. This class of models are designed to reduce the computational costs of above mentioned physically-based models. For example, one may use a system of equations that are linearly independent [98], have a restricted class of deformation functions [128], use iterative solutions for the first-order differential equation of deformation [7], preprocessing non-linearity in high-order differential models [25], and linear elasticity theory [79, 58].

Point-based data representations are becoming a popular alternative to mesh-based representations in computer graphics. As SORs, they lack in the topological connectivity necessary for the application of most physically-based deformation models. One can either superimpose a mesh structure over a point set, or define neighborhood using an approximation, such as moving least squares [81, 80]. The latter is referred to as meshless or mesh-free deformation.

Deformable Models in Surgical Simulation

The role of deformable models in surgery simulation and training is diverse, since they are required for collision detection, rendering and haptics simulation. When the user interacts through a virtual tool, forces applied to the model produce a deformation, described as a set of displacements of the underlying geometry, and they generate internal forces and vibrations which are fed back to the user as haptic stimuli.

Although non-physical models, such as 3D ChainMail [42] and free form deformation [100], are computationally inexpensive, physically based models are the dominant paradigm because of their accuracy. These include mass-spring models [21, 77] and finite element methods (FEM). Of these two, the latter is the most common, because it is more accurate and can accommodate different material properties through a small number of parameters. Furthermore, the focus of most surgical simulation systems is a simulation on localized regions, which FEM can handle properly (i.e., no need to simulate large displacements).

FEM, however, is computationally expensive for real-time simulation, since it requires solving large partial differential equations (PDEs). Techniques for achieving real-time finite element simulation can be classified into two categories: those that simplify the mathematics, and those that speed-up the solution algorithms. In the former category we find approaches that simplify the modeling of elastic tissue using linear models [133, 11]. Linear elasticity is often preferred because it reduces the problem to a linear equation that can be solved quickly by pre-computing the inverse of the stiffness matrix. However, linear elasticity only is accurate for small deformations. Large deformations, such as global rotations, usually result in an unrealistic volume growth of the model. For this reason, different techniques have been proposed to handle large deformations, such as warping of the stiffness matrix [78] and quasi non-linear deformation [25]. Zhuang and Canny propose real-time deformation using non-linear elasticity [134]. In surgery simulation, the problem is often described as a dynamic problem. Alternatively, the problem can be reduced to a static problem, which ignores body forces, inertia and energy dissipation. BroNielsen proposed this simplification for surgery simulation for obtaining real-time response [11]. However, loss of dynamics may affect the realism of the simulation and static systems are mostly used in surgery planning, where the desired solution is the equilibrium state of the deformable model after being subjected to forces, with no interest in the intermediate states.

The second category for real-time deformation includes techniques for speeding up the solution of the resulting equations. BroNielsen and Cotin proposed a technique based on condensation [11], which reduces the size of the PDE to be solved by ignoring the internal finite elements in the computation. They also proposed explicit integration over implicit integration for its reduced computation time and memory requirements. However, explicit integration leads to instability for large time steps. Another possibility for speed-up is the use of multiresolution techniques, as suggested by Debunne et al. [28] and Wu [129]. Wu and Tendick propose a multigrid integration scheme [130] to solve non-linear deformations in real-time. Real-time deformation has also been possible with increased computation capabilities, such as parallel processing and specialized hardware [112, 39].

A challenge in surgical simulation that prevents extensive use of precomputed quantities in FEM is real-time cutting. Pre-computation of the stiffness matrix and applied forces forbids

topology changes in the mesh, required for simulating cutting. Cotin et al. [25] propose a hybrid approach for real-time cutting that uses a static model in regions that do not require topology changes, and a dynamic model for a limited region where cutting and tearing is needed. For a complete survey in deformation for surgical simulation, refer to [72].

Because of the high computational costs of physically-based methods, they are not suitable for illustration and visualization. In this thesis, we consider an empirical deformation method instead.

2.4 Rendering Deformation

Traditionally, deformation is performed at the modeling stage, which results in an explicit deformed object to be forwarded to the rendering stage. Earlier approaches to volume deformation followed this paradigm, such as [53, 48, 67, 37], aimed towards morphing of volumes, or the methods in [40], aimed to volume animation. It is often desirable to couple the modeling and rendering of deformation together to facilitate interactive deformation or reduce the needs for generating explicit deformed objects at each time step. This approach is particularly effective when deformation rendering is accelerated by using modern graphics cards. Another reason for coupling the modeling and rendering of deformation is that the generation of deformed objects is limited by the sampling frequency of the deformed object. In the case of large deformations, such as a large pull or a twist, this sampling may be much larger than the sampling frequency of the original undeformed object.

In order to describe the space of volume deformation approaches, let us define a deformation as a mapping function $T_F : \mathbb{R}^3 \mapsto \mathbb{R}^3$, such that, for a given point $\mathbf{p}$, we can obtain a new position $\mathbf{p}'$, as depicted in Figure 2.1.

$$\mathbf{p}' = T_F(\mathbf{p}) \tag{2.1}$$

T_F is a continuous deformation if $T_F \in C^0$. Here, C^0 denotes the set of all continuous functions in 3D space. A useful, but not necessary, property is that T_F is differentiable at every point $\mathbf{p}$.

We denote T_F as a *forward transformation*. Let P_V be the set of all points $\mathbf{p}$ in the volume representation V of an object. Hence, we obtain a new set $P'_V = \{\mathbf{p}' | \forall \mathbf{p} \in P_V, \mathbf{p}' = T_F(\mathbf{p})\}$. Let

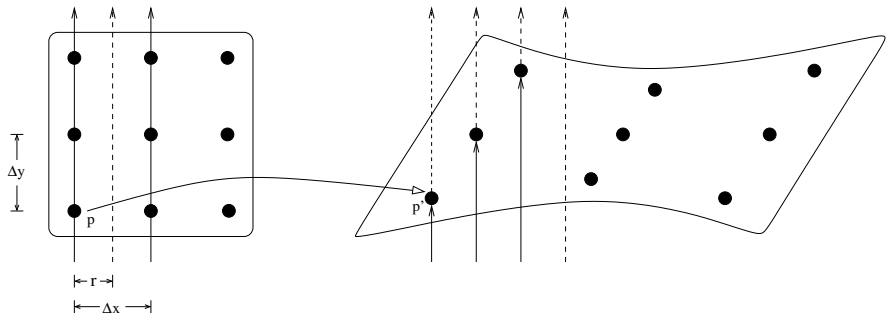


Figure 2.1: Direct transformation

V' be the new axis-aligned bounding volume for all the points in P'_V. We call this new volume V' the *deformed volume*.

Rendering of the deformed volume is obtained by sampling the points in V'. Let f' be the scalar function defined for the deformed volume. Since we assume continuity of the scalar values,

$$f'(\mathbf{p}') = f(\mathbf{p}) \text{ , for } \mathbf{p}' = T_F(\mathbf{p}) \tag{2.2}$$

where $f(\mathbf{p}$ is the scalar function that represents the SOR defined by volume V. Forward transformation is of limited use in the deformation of volumetric objects. This approach was used by McGuffin et al.[76], where rendering and deformation are coupled by treating each voxel as a rendering primitive. This approach, however, suffers from undersampling problems as the space between voxels are not rendered or deformed in any way. That is, it treats the volume as a disconnected set of points. However, most volumetric objects are obtained by sampling a continuous object.

For this reason, volume deformation techniques are commonly in the category of *space warping* techniques. We distinguish two different types of methods: those that perform *indirect space warping*, also referred to as *proxy-based space warping*, and those that perform *direct space warping*. At the core of these two is the idea that volume rendering is obtained by sampling the deformed volume V'.

2.4.1 Indirect Space Warping

Indirect or proxy-based space warping is obtained by defining a set of control points Q which are deformed using a forward transformation. Let us define Q' as the deformed set of control

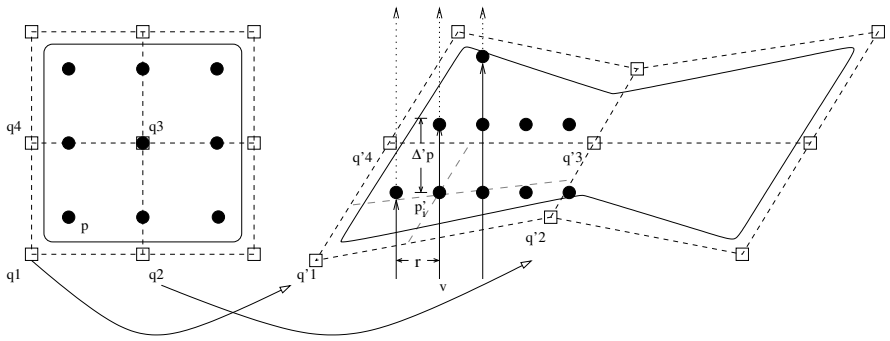


Figure 2.2: Indirect space warping

points, i.e.,

$$Q' = \{\mathbf{q}' | \forall \mathbf{q} \in Q, \mathbf{q}' = T_F(\mathbf{q})\}. \tag{2.3}$$

In order to render this deformed volume, it suffices to sample the space at regular intervals $\Delta\mathbf{p}$, along a ray direction $\overrightarrow{v}$. The sampling points are:

$$\mathbf{p}'_i = i\Delta\mathbf{p}\,\overrightarrow{v} \tag{2.4}$$

and the sampled value of the scalar field f' is obtained through interpolation. This is depicted in Figure 2.2.

A number of approaches have been proposed which differ in the type of proxy geometry and interpolation function. Westermann and Rezk-Salama [126] define Q as a free-form lattice. Deformation is obtained by directly transforming the points in Q and sampling is obtained by slicing the deformed lattice into view-aligned slices. Since the vertices are parameterized in the undeformed space, also known as the *texture space* of the volume, the sampled value is computed using the Bernstein polynomial as described by Sederberg and Parry [100]:

$$f'(\mathbf{p}') = f'(s,t,u) = \sum_{i,j,k}^{3} f_{ijk} B_i(s) B_j(t) B_k(u) \tag{2.5}$$

where $B_i = B_{i,3}$ are the Bernstein polynomials of degree 3, $f_{ijk} = f(\mathbf{q}_{ijk})$ are the scalar values evaluated at the control points, and (s,t,u) is the parametrization of the sample point $\mathbf{p}'$.

A simpler interpolation mechanism was proposed by Rezk-Salama et al., using trilinearly interpolated patches instead of a free-form deformation lattice. In their work, Q defines a grid

of linear patches, usually of a size much smaller than the original grid P_V. The interpolation is obtained with Eq.(2.5), except that $B_i = B_{i,1}$ are the Bernstein polynomials of degree 1 [56]. This approach proves to be very effective since trilinear interpolation is readily available in most contemporary graphics processors. One of the disadvantages of this implementation is that automatic parameterization of the sample positions into texture space is ambiguous, due to irregular deformation of the linear patches. Depending on the viewpoint, the slicing mechanism and (automatic) rasterization of the proxy slices generates different parameters (s,t,u). This is observed also when rendering a textured irregular quadrilateral. Depending on the triangulation of the quadrilateral, a different parameterization is obtained. This problem can be solved with a tessellation of the proxy slices, either on a finer grid or by introducing a new vertex in the barycenter of the slice polygon. This works since the parameterization of a triangle is unique.

Another group of methods that can be considered in this category as skeleton-based deformation techniques, such as the approach by Gagvani et al.[40] by Singh et al. [105] and Volume Wires [120]. Instead of having a grid of axis-oriented patches, the proxy points Q are defined along a curve-skeleton, centered within certain features of interest. Each segment in this skeleton is used to define a *cuboid*, which is an elongated cube along the principal direction of the feature. The corners of all the cuboids define the control set of points Q. By transforming the points via rotations and translations, it is possible to animate and manipulated articulated volumetric objects. Similar problems appear in the joints of two segments due to ambiguous trilinear interpolation. This was solved by adding additional points in the barycenter of the slices that resulted from sampling an irregular cuboid.

Physics-based volume deformation approaches fall also into this category, and Q is usually defined as a mass-spring model or a finite element mesh. In the former, control points are points with mass connected with elastic links. Examples of both mass-spring models and FEM are found in [21, 133].

Recent approaches to model deformations consider the actual voxels as control points. Unlike the above, where a mesh is explicitly defined between the control points, these approaches do not require an explicit mesh topology. These methods fall in the general category of meshless deformation. Although the literature is abundant for its application in surface meshes and image warping, a few methods have been proposed for volume deformation [44]. In general,

the set of control points is a subset of the set of sample points of the undeformed volume, i.e.,

$$Q \subseteq P_V \tag{2.6}$$

Since no connectivity is required, the interpolation function must operate over scattered data [97]. Several methods are proposed for this, of which the most representatives are the Shepard's interpolant, radial basis functions and moving least squares. Shepard's interpolation is in general a weighted average of the scalar value of the control points, i.e.

$$f'(\mathbf{p}') = \sum_{i=1}^{n} w_i(\mathbf{p}') f(\mathbf{q}_i) \tag{2.7}$$

where n is the number of control points, $w_i : \mathbb{R}^3 \mapsto \mathbb{R}$ a weighting function such that satisfy the following conditions:

$$w_i(\mathbf{q}_i) = 1 \ , \ \sum_{i=1}^{n} w_i(\mathbf{p}) = 1 \ , \ w_i(\mathbf{p}) \geq 0$$

$w_i(\mathbf{p}')$ is often defined as a function of the distance between $\mathbf{p}'$ and control point $\mathbf{q}_i$.

Radial basis functions construct the interpolation as a linear combination of basis functions. One of the properties is that it can be defined to be infinitely differentiable, which provides a smooth and continuous deformation. However, as we shall see in the next sections, this poses a problem when creating discontinuous deformation, that may arise when simulating cuts and breaks. Solutions can be found to obtain such discontinuous deformation through modifications of the radial basis function, or by employing moving least squares.

Indirect space warping has become very popular because of its implementation feasibility in the vast majority of contemporary GPUs, by exploiting hardware accelerated tri-linear interpolation. However, the reliance on this capability requires a proper handling of the proxy geometry, which needs to be properly tessellated.

2.4.2 Direct Space Warping

To avoid dealing with complex meshes, deformation can be defined as a point-wise warping of the volume. For each point $\mathbf{p}' \in P'_V$,

$$\mathbf{p} = T_F^{-1}(\mathbf{p}') \tag{2.8}$$

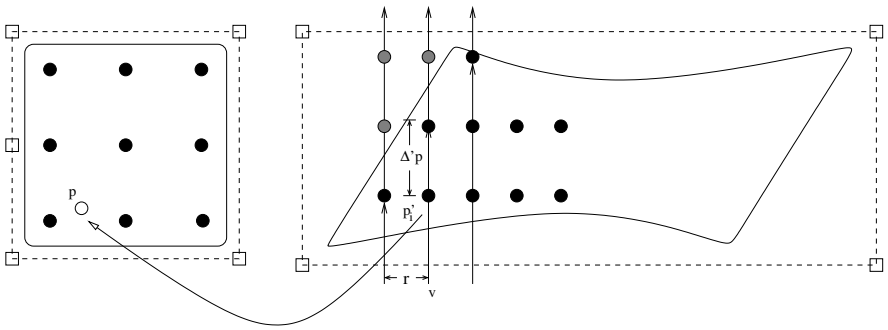


Figure 2.3: Direct Space Warping

where T_F^{-1} is the corresponding *inverse transformation* of T_F. The scalar value $f'(\mathbf{p}')$ can be found by sampling the original scalar field, i.e., $f(\mathbf{p})$, as seen in Figure 2.3

This approach was first introduced in ray tracing systems. With the advent of programmable shaders, it has been recently extended to 3D texture based volume rendering and GPU-based ray casting. In ray casting, space warping can be implemented by deforming or *deflecting* the rays into curves, giving the illusion of rendering a deformed object. This approach has been proposed by Kurzion and Yagel [64]. In their work, a procedural definition of a *deflector* is used to model different kinds of deformations. A deflector changes the sampling positions been traversed by a ray during the rendering process, by assigning a displacements towards a user-defined *sink* position. Although this can be extended to model a number of deformations, it is in general a non-intuitive approach to deformation. Kurzion and Yagel also proposed an extension for hardware rendering. Since deflecting rays were difficult to implement with the existing GPU capabilities, they proposed to tessellate the slices and then applying the deformation on the 3D vertices [64]. Unlike their original ray deflection work, their hardware extension is really an indirect space warping method.

Another approach that uses ray-casting is the concept of spatial transfer functions (STF) [20]. Unlike the above, rather than deforming the ray, the position of each sampled point is transformed via a procedural definition, represented as a spatial transfer function. The spatial transfer function is then a realization of the inverse warping function T_F^{-1}. One of the advantages of STFs is the ability to create complex deformations by combining algebraically simpler ones.

Another recent approach has been proposed by Chen et al. [18], using the GPU to perform

interactive ray casting. Unlike the ray deflectors or spatial transfer functions, they embed the rays into a free-form deformation lattice, providing the user a range of deformations from the manipulation of a few control points. It is similar to the approach by Westermann and Rezk-Salama [126], where a FFD lattices is also used, except that Chen et al. do not require a deformable proxy geometry to render the volumetric object.

With the programmability of current GPUs, and in particular, of fragment shaders, it has been possible to perform direct space warping in texture-based rendering and GPU-based ray-casting of volumes. Rather than applying a deformation to control vertices in a proxy mesh, it is applied directly on the 3D points obtained through slicing or ray casting, each of them corresponding to a single pixel in the final image. The inverse deformation T_F^{-1} is evaluated for each fragment (intermediate pixel) in the image. This has certain advantages over the previous approaches. The most notable one is that tessellation is not needed and therefore more complex deformations can be obtained without extra computation. In addition, this approach achieves smooth rendering of non-linear deformation. With indirect space warping, they are approximated by tri-linear interpolation. However, indirect space warping is in general much faster, since the number of control points (3D vertices) are far less than those in direct space warping (pixels for each slice).

The approach proposed in this thesis falls into this category. A recent approach by Brunet et al. [15] also uses GPU capabilities to perform inverse warping. They exploit the fragment shader to sample an inverse deformation function, which can be represented procedurally on the fragment shader or pre-computed and stored as a 3D texture.

2.4.3 Discontinuous Deformation

The majority of previous work on volume deformation considers the transformation function as continuous. Particularly for surgery and medical illustration, deformations often contain discontinuities. Therefore T_F and T_F^{-1} are not necessarily continuous.

There are a number of ways to define discontinuous deformation. The most intuitive is by using forward deformation. This approach was used by McGuffin et al [76]. By using points as primitives, it was possible to define discontinuities since no intermediate space is needed between the points. However, as pointed out before, this method proves insufficient for volume

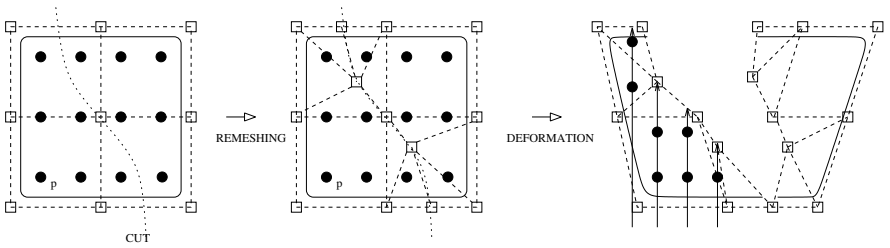


Figure 2.4: Introducing Cuts for Indirect Space Warping

rendering.

Introducing discontinuities in space warping approaches is more challenging. In the case of indirect space warping, this challenge arises because the interpolating functions required for the sampling of the proxy mesh are inherently continuous. Therefore, discontinuous volume deformation requires, similarly to surface-based deformation, a re-tessellation of the proxy geometry. A discontinuous deformation can be defined as a collection of disconnected continuous deformations. Therefore, the proxy geometry is split and re-meshed along the cuts. Sampling occurs at those points inside the proxy mesh and not in the empty space generated by the cut. This process is shown in Figure 2.4. Note how the discontinuity appears as no sampling rays are cast into the empty space. This approach was used by Kurzion and Yagel in their hardware implementation of ray deflectors [64]. In a recent work, Bruckner and Gröller [13] use re-tessellation to split volumetric objects and generate exploded views of volume data on the fly. In their system, a volume object can be split with planar cuts defined by the user, and each generated sub-volume is transformed linearly via rigid transformations.

One of the shortcomings of this approach is the requirement of re-tessellation on the fly, which may be computationally expensive for complex non-linear cuts and deformations, and the inability to represent arbitrarily smooth cuts due to the linear interpolation imposed by the proxy mesh.

A mechanism to overcome the need for re-meshing is to use point-based deformation. In this case, Q is a subset of the original sample points P_V. Deformation is obtained by forward transformation. Unlike cuberille rendering, the sampling of the space is done by interpolating the scalar value in a local neighborhood centered at each deformed point $\mathbf{p}'$. Because no mesh is required for Q, introducing cuts is easier. For more information on point-based and meshless deformation, refer to [82].

Another mechanism is to model discontinuities via direct space warping. Unlike the previous approaches, direct space warping requires the sampling of all the points in the deformed space, including the *empty space* introduced by a discontinuity. The deformation equation in 2.8 is no longer valid, since the transformation is undefined for the points in the empty space. Ideally, for any forward transformation, T_F from P_V to P'_V, we should always have its corresponding T_F^{-1}, such that, for each point $\mathbf{p}' \in P'_V$, there exists $\mathbf{p} \in P_V$ and $\mathbf{p}' = T_F(\mathbf{p})$. With a discontinuous deformation, this condition is no longer met. This can be solved by allowing the definition of a *modified* inverse transformation T_B, as follows.

Let $P'_{V'}$ be a collection of all points in V'. Since P'_V is a set of all points located in V' with a pre-image in V, the empty space in V' is thus defined by a set of points $P'_{empty} = P'_{V'} - P'_V$. We thereby replace the T_F^{-1} in Eq.(2.8) with a modified *backward transformation* T_B as:

$$\mathbf{p} = T_B(\mathbf{p}') = \begin{cases} T_F^{-1}(\mathbf{p}') & \mathbf{p}' \in P'_V \\ \varnothing & \mathbf{p}' \in P'_{empty} \end{cases} \tag{2.9}$$

where $\varnothing$ denotes a null position, indicating a point that does not have an origin prior to the manipulation. In general, such points are considered empty, or completely transparent. We thereby assume that, for purposes of rendering:

$$f'(\varnothing) = 0 \tag{2.10}$$

This method is depicted in Figure 2.5. Note how the rays are also cast into the empty space. Points in that region are depicted as white dots and they are inversely mapped to the null space $\varnothing$.

This method has been implemented as Spatial Transfer Functions in [20] and Spatial and Temporal Splitting [54], using raycasting, where discontinuities are handled by maintaining a special value for a null position. Although this has produced good results with ray-casting, it requires a good sampling of the surface of the cut to obtain anti-aliased images. Further, this approach does not extend easily to interactive volume rendering. Discontinuous deformation can also be realized with ray deflectors, in [64]. One problem with ray deflectors is that, in order to define a discontinuity, rays would have to be deformed away from each other at some

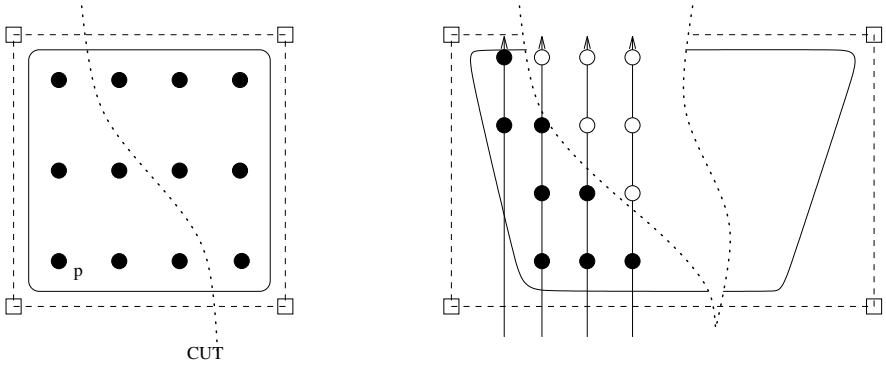


Figure 2.5: Introducing Cuts for Direct Space Warping

point, and they would never "hit" the surface of the cut. As an alternative, Kurzion and Yagel define discontinuous deflectors by allowing intersecting rays. Since a sample point cannot be mapped to two different positions in the original volume V, they only allow points on one side of the intersection to sample the volume in that side of the intersection. As the sampling is done smoothly along the deflected ray, this approach produces rendering of cuts with little aliasing in comparison with Spatial Transfer Functions. However the specification of such ray deflectors is not straight forward, and may be difficult for complex discontinuous deformations.

2.5 Other Volume Manipulation

There are other methods for transforming volumetric objects that differ slightly from deformation. In these methods, a manipulation of the volume is achieved by either *carving*, *clipping* or *ghosting* parts of the volume. Volume clipping is a popular tool to remove parts of an object in order to gain visibility of occluded parts. Clipping is usually performed by applying a difference boolean operation of the volume with another clip volume, usually defined procedurally. Planar cuts are the most common, and methods for rendering clipped surfaces have been proposed [47]. Weiskopf et al. proposed the use of general shape clipping volumes for effective visualization [124]. A similar idea is the process of volume carving [30], where parts of the volume are interactively removed as a modeling process.

2.6 Mesh Deformation and Cuts

Triangular meshes, as opposed to volumes, are not sampled object representations. However, many deformation techniques draw similarities from these approaches. Deformation of meshes is essentially a direct mapping problem, as described above. In empirical models, this mapping is obtained via a number of methods, namely:

- Direct transformation of the vertices of the mesh, as it is common in consumer 3D editing software, such as 3D Studio [31] and Maya [4].
- Global or local deformations [6]. Deformations such as twists and bends are usually defined as procedures. As an alternative to mesh deformation, Barr proposes inverse transformation for the ray tracing of deformed surfaces [5].
- Transformation of an embedding lattice, of which the most common one is free-form deformation [100, 75], where the lattice is a tri-cubic grid. Other examples include skeleton-driven deformations [16]. Pose-space deformation techniques [69], which are at the core of current character animation techniques, also achieve deformation by the transformation of an embedding structure, also known as *envelopes*.
- Sketching [59, 83], where the deformation is obtained by a mapping from a 2D curve, usually sketched by the user, into a 3D transformation.

Recent efforts have been made to enforce certain constraints in mesh deformation, in the hopes of obtaining realistic results at interactive rates without the need for complex physics. Some of the properties often enforced in mesh deformation are:

- Smoothness of the deformation [135, 132]. This property leads to the prevention of folding artifacts or breaks in the mesh.
- Volume conservation [50, 92, 10, 118], where a volumetric functional is minimized in order to achieve realistic deformations of non-compressible solids.
- Prevention of self-intersection [41, 118, 38].
- Preservation of surface details [83, 10].

Another aspect of deformation is the modeling of cuts and breaks. Simulation of fractures was introduced by Terzopoulos and Fleischer [113]. Several models have been proposed for discontinuous deformation: cutting, as the surface interacts with a cutting tool, e.g., for surgical simulation [11, 29, 8, 101], or illustration [88], fracturing of brittle [85] or ductile materials [84] and splitting of objects [109].

One of the problems with mesh deformation is the requirement for an explicit mesh. In the case of large continuous deformations, such as a twist, the resulting mesh may contain large degenerate triangles, which may intersect. Most mesh-based deformations handle the problem by an adaptive re-meshing of the problematic region. In the case of cuts and breaks, a mesh is required to propagate the break or cut in the desired direction. In addition, the modeling of the interior requires a solid mesh representation, usually a tetrahedral mesh. A re-meshing is also required in this case, as the cut is seldom introduced along the actual edges of the mesh. Unlike the re-meshing for continuous deformation, cuts imply topology changes in the mesh.

2.7 Summary

This chapter has described the state-of-the-art in deformation of volumetric and surface-based objects. Volumetric objects can be described as sampled object representations, obtained by sampling a continuous object in a regular grid. Because of this sampling, deformation of sampled object representations requires an inverse *warping* mechanism. This space warping handles the sampling problems that occur when attempting to produce deformation via forward point transformation. The most common method has been indirect space warping, where a control proxy mesh is deformed first and the volume deformation is obtained through interpolation. Current GPUs enable interactive volume deformation through direct space warping, where an inverse transformation is used to directly warp the sample points in the deformed volume. However, the modeling of cuts and breaks and high quality rendering has been more problematic, due to the topological changes that may occur. Since volumetric objects do not have explicit topology, the introduction of cuts seems straight forward. However, accurate rendering of volumetric cuts has been difficult, due to sampling problems and the lack of a unifying mechanism for encoding discontinuous deformations.

The following chapter describes a method for deforming volumes at interactive rates, based on a generalization of displacement maps, which also allows the modeling and high quality rendering of large deformations and discontinuities.

Chapter 3
Illustrative Deformation

We define Illustrative Deformation as a non-physics-based deformation method, where the shape of an object is modified via global and local deformations, to mimic the type of deformations commonly found in scientific illustrations. Unlike physics-based approaches, the deformation is applied as the result of user intervention via a set of transformation templates, rather than as the result of a set of virtual forces being applied to certain control points on the object. There are several properties of scientific illustrations which artists and scientists use to provide a better understanding of objects and phenomena. First, deformation often depict cuts and breaks, so that important parts that are being occluded are made visible, while preserving contextual information of the object. Second, these cuts and deformations are ubiquitous, in the sense that they can be applied anywhere in the object, and at several scales. Finally, they do not adhere necessarily to reality, as shown in certain anatomical illustrations, e.g., 1.2(c), where the placement and deformation of certain layers such as muscle may not describe a physically feasible deformation, but it provides the most comprehensive view of the different anatomical layers.

We describe this method as a top-down approach. Traditional deformation approaches model reality in a bottom-up approach: first, a series of equations are specified, which are designed to model certain physical properties of the movement of bodies. The equations are integrated in time to obtain displacements for the graphical elements, which in turn produce a 3D image. This process is refined by modifying the equations, setting parameters or adding constraints in order to obtain the desired result.

In our approach, we begin with an abstract description of the deformation. This description defines a global deformation of an object, such as a peel, a twist, a bend, etc. We represent these generic deformations as templates, which are described in the following section. We then

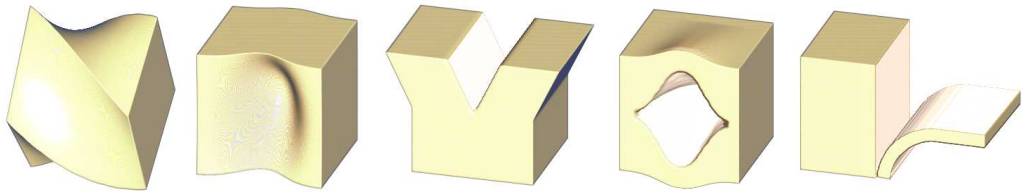


Figure 3.1: Iconic representations of deformation templates

apply the template to the object, in a similar fashion to the way textures and displacements are applied to surface meshes. Our rendering process then produces the resulting image. We then can refine our results by either adding new deformation templates, or combining simple deformations to form a complex one.

3.1 Deformation Templates

At the core of our approach is the idea of *deformation templates*. Unlike physically-based methods, where deformation is obtained via a physical simulation, here we define global and local deformations as displacement maps. These in turn are defined procedurally, as in [6], via interpolation of point-wise transformations, as the tool described in [54], or as a combination of simpler deformations. Displacement maps are sampled structures where each position contains a displacement vector, which is used to transform the position of a point in 3D space. This idea was introduced as a means to add geometric details to a base surface [22], in a similar fashion to the way textures are used to change the appearance of an object. During the reminder of the thesis, we use the terms displacement maps and deformation templates/metaphors interchangeably. Deformation templates can be continuous or discontinuous. Examples of continuous deformations are twists, bends, pulls and pokes. Examples of discontinuous deformations are cuts, peels and slices. When these deformations are applied to a volumetric cube, we obtain iconic representations of our deformation templates, as shown in Figure 3.1.

An important aspect of deformation templates is that they are generic, and therefore can be applied to any dataset. This means that deformation is decoupled from the object representation.

3.2 Process Overview

In our approach, deformation is part of the rendering process. The deformation is specified as a displacement map, available from a pool of deformations. Since volumetric objects are sampled representations, deformation cannot be obtained by directly transforming the sample points. Instead, inverse warping is required. That is, rendering of a deformed volume is obtained by sampling the deformed space and warping each point into the original sampled representation in order to find the necessary color and opacity values. Because of this inverse warping, deformation templates store *inverse* displacements. The overall rendering process is then as follows:

1. The user selects a deformation and *applies* it to the model. This application is defined in a similar fashion to traditional texture mapping. We call this the *mapping* stage of our algorithm.

2. The deformation mapping specified by the user defines a volume in 3D space, where part of the deformed volume needs to be rendered. This deformed space is traversed at regular intervals along the view direction (*slicing* stage)

3. Each sampled generated by the slicing process is inversely warped according to the deformation template. We call this the *warping* stage.

4. Each warped coordinate is used to retrieve opacity and color information from the original volumetric object via sampling. This is the *sampling* stage. In addition, for each sample we obtain normal estimation for lighting models. This is the *lighting* stage.

5. In the final stage, the *compositing* stage, color and opacity values are composited together to produce the final image.

In this thesis, we explore the different aspects of this rendering process in order to obtain meaningful deformations that can be used for medical illustration, surgical simulation or as a visualization tool in general. One difficulty with inverse warping is the simulation of cuts and breaks, since they imply that the warping function is not invertible. We solve this by encoding discontinuity information along with the displacement in such a way that C^1 continuity is

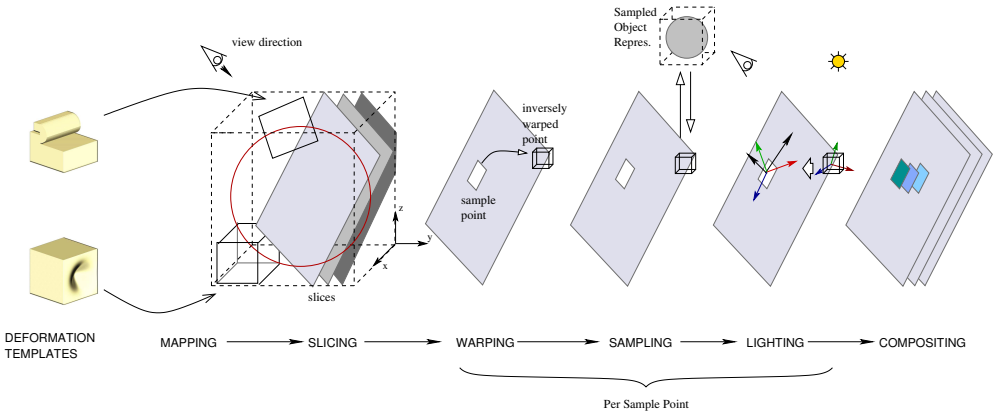


Figure 3.2: Overview of Illustrative Deformation of volumes

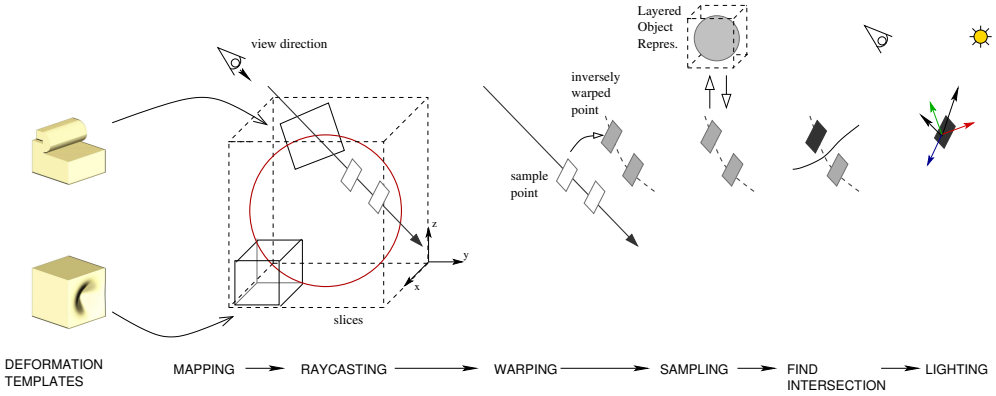


Figure 3.3: Overview of surface-based deformation

preserved. This method is described in Chapter 4. Another challenge is the introduction of feature-sensitive operations. For instance, some deformations require the preservation of feature of interest, such as internal structures or rigid parts. In our approach, this can be done in the mapping stage, as a transformation of the displacement mapping, or in the sampling stage, as a modification of opacity information. This method is described in Chapter 5. The final stage, compositing, makes sense for volumetric objects where each voxel contributes to the final image. However, as described in the previous chapter, surface meshes can also be defined using sampled layered representations. Rather than doing slicing along the view, we trace rays in the view direction along each pixel. As we inversely warp the points along the ray and sample the layered representation of the object, we test for intersections. Once we find an intersection, we render the point according to the lighting parameters. The overall process is depicted in Figures 3.2 and 3.3, and described in detail in Chapter 7.

Part I

Volume Deformation

Chapter 4
Discontinuous Displacement Mapping [1]

4.1 Introduction

In Chapter 2, we described volume deformation as a point-wise mapping that transforms the position of all points within a volume to new positions. We also described that this point-wise mapping can be thought of as either a forward mapping, such as in traditional mesh deformation, or inverse mapping or space warping, more appropriate for discretely sampled volumes, where an inverse transformation is applied to the points in the deformed space to find the corresponding point in the undeformed object. However, this method implies challenges when incorporating cuts and discontinuities, since the mapping may not be defined for some points in the deformed space.

This chapter describes a novel and general method for performing volume deformation. As described in Chapter 3, we define deformation in a top-down approach, starting from a rough specification of the deformation and, by means of combination of refinement, reaching to the desired deformation state. The process of adding deformation to the volume can be thought of as a process of "adding" details, and therefore are similar to the idea of *displacement maps*.

Displacement maps are discretely sampled objects, typically defined as 1D, 2D or 3D textures, where each element defines a spatial displacement rather than a color attribute. Displacement maps are commonly used to add visual details to a base surface by perturbing points on the surface for a small distance along the corresponding surface normals. For this reason, traditional displacement maps are generally (i) applied along the surface normal and (ii) assumed to be continuous. These conditions make it difficult to simulate large and complex deformations such as cuts and flips.

[1] Portions of this chapter were published in our paper: Discontinuous Displacement Mapping for Volume Graphics, by C. Correa, D. Silver and M.Chen, Eurographics and VGTC Workshop on Volume Graphics, 2006, pp.9–16

It is our objective to extend the idea of displacement maps to model large deformations and discontinuities in volumetric objects by relaxing these two conditions. The first condition has been relaxed in some recent work. Wang et al. [122] employed volumetric displacement functions in order to simulate non-orthogonal displacements on surface objects. Similar ideas are found in [121] and [91]. These approaches, nonetheless, consider only displacements within small volumetric regions along the surface. The removal of the second condition is critical to rendering large cuts and breaks. Surface meshes do not contain adequate volumetric information, such as surface thickness and interior structure, to allow the creation of correct visual effects. Although this can be handled using a tetrahedral description of the interior, re-tessellation of such meshes is a time-consuming task, which limits the quality, smoothness and thickness of cuts and breaks.

Here, we introduce a generalized notion of a displacement map, which allows for unconventional features such as unorthogonal and discontinuous displacements. Figure 4.1 illustrates the difference between the traditional and generalized displacement mapping. We discuss the major technical difficulties associated with this generalization, and outline our solutions to these problems. In particular, we consider a GPU-based volumetric approach without involving any mesh structure and the intensive computation associated. By employing inverse displacement maps in 3D vector space, we are able to apply complex displacements to volumes. Our rendering approach involves the use of a proxy geometry for sampling of the inverse displacement map, which is then mapped into the original object space. Correct calculation of surface normals becomes a particular issue since conventional normal estimation would lead to noticeable lighting artifacts on surfaces displaced in varying directions, and at breaking points in a discontinuous displacement map.

4.2 Related Work

Displacement Mapping. Displacement mapping was introduced by Cook [22] as a type of texture mapping technique for modifying the geometry of a surface, resulting in correct shadows and silhouettes (in contrast to bump mapping [9]). Typical approaches to the realization of displacement mapping include *explicit surface subdivision* (e.g., [23]), *direct ray tracing* (e.g.,

[73, 89]), and *image space warping* (e.g., [99]).

In surface subdivision [23], geometric primitives are subdivided into micro-polygons, resulting in an explicit representation of the displaced surface. The method has been made available through commercial software such as RenderMan^{TM}and Maya^{TM}. Hardware solutions were also developed [43, 32]. Discontinuities are introduced with costly re-meshing of the object. This approach cannot be extended easily for volume graphics, since no surface model is available.

In ray tracing, the conventional approach is to pre-compute an inverse displacement map, and perform the intersection calculation in the displaced space of a surface [73, 89], similar to Barr's suggestion for rendering deformed objects [5]. To alleviate the cost of ray-tracing an entire scene, recent approaches to displacement mapping propose to traverse rays through extruded triangles in the texture space [122, 121, 91]. In image space warping, the visual effect of a displacement is achieved in the image space rather than the object space. The method was first introduced by Schaufler and Priglinger [99], focusing on warping the image of the base surface according to the projected displacement, and later extended by Oliveira *et al.* [87]. The extension of ray tracing to the modeling of cuts is difficult, since it requires determining a ray intersection with the new surface produced by the cut. Further, current approaches for displacement mapping based on extruded triangles cannot model large or discontinuous displacements. In our approach, we exploit the GPU capabilities of contemporary graphics boards to solve the limitations of the inverse displacement maps and image-space warping approaches when applied to discontinuous displacement on volumetric objects.

Displacement as a means of deformation. Most deformation techniques have considered deformation as a mapping problem. For surface mesh deformation, this mapping is usually explicit by finding the displacement of each element. In a recent development, Von Funck et al. used a vector-field representation in order to model deformation on surface meshes [118]. Their structure is actually a velocity map, which can be defined as an infinitesimal displacement map. The actual displacement is found for each point through time integration.

In volume deformation, the use of a discretely sampled structure to store the displacements was hinted at by Westermann and Rezk-Salama in [126] and later in [94]. However, these approaches did not handle the case of discontinuities, and many details necessary for a GPU

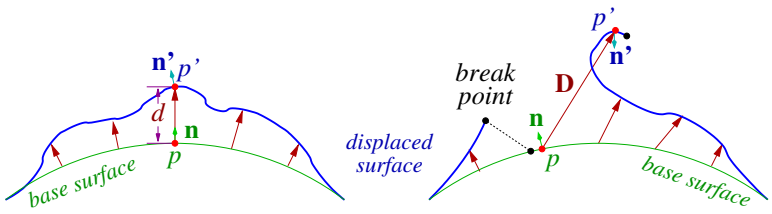


Figure 4.1: A cross section illustration of the traditional displacement mapping (left) and the generalized displacement mapping allowing for unorthogonal and discontinuous displacement (right).

implementation were not explored. Brunet et al. [15] recently explored the use of GPU programmability to pre-compute deformations in a 3D texture. Unlike a displacement map, their approach stores the result of the transformation. Although the end result is, in principle, equivalent for both approaches, displacement maps have the advantage of being algebraically operatable, which provides means for combination and composition of deformations in a flexible way.

4.3 Displacement Mapping

Displacement mapping is traditionally considered as a variation of 2D texture mapping, and it is used to alter the base surface geometrically. A volumetric displacement mapping can be considered as a variation of 3D texture mapping, where 3D displacements are used to perturb the volume, enabling the simulation of large deformations and cuts. Since texture mapping in 2D or 3D often involves textures defined in a higher dimensional parametric space, we consider a general notion of space Λ without explicitly distinguishing between geometry and texture spaces.

A Generalized Notation. Let Λ be a reference coordinate system used by an object position function P and Λ_D a reference coordinate system used by an object displacement function $\overrightarrow{D}^{\triangleright}$. Let $W : \Lambda_D \mapsto \Lambda$ be a coordinate transformation between the two systems, and W^{-1} its inverse. We consider the following generalized mapping from a point on the object $P(\lambda)$ to a new point $P'(\lambda)$:

$$P'(\lambda) = W((W^{-1} \circ P)(\lambda) + (\overrightarrow{D}^{\triangleright} \circ W^{-1} \circ P)(\lambda) \tag{4.1}$$

where $\lambda \in \Lambda$ and $F \circ G$ is the composition of transformations F and G. P defines a coordinate mapping from Λ to a point in $\mathbb{R}^3$. $D^{\triangleright}$ is a vector function that can be specified procedurally or by using a discretized representation such as a texture. In Eq.(4.1), it is no longer a must that the surface normal determines the direction of displacement, and function $\overrightarrow{D}^{\triangleright}$ is no longer assumed to be continuous. As illustrated in Figure 4.1, in comparison with the traditional notion on the left, this generalized notion allows for unorthogonal and discontinuous displacement.

Λ can be a 2D parametric space as in the traditional notion, the 3D Euclidean space as in [61], or any reference system appropriate to an application. In this work, we use $\Lambda = \Lambda_D = \mathbb{R}^3$ as the reference coordinate systems. In the following discussions, we use mainly the inverse form of Eq. (1), that is,

$$P(\lambda) = W((W^{-1} \circ P')(\lambda) + (\overrightarrow{D}^{\triangleleft} \circ W^{-1} \circ P')(\lambda) \tag{4.2}$$

where $\overrightarrow{D}^{\triangleleft}$ is the inverse of $\overrightarrow{D}^{\triangleright}$. Since $\Lambda = \mathbb{R}^3$, we can make $P(\lambda) = \mathbf{p}$ and $P'(\lambda) = \mathbf{p}'$ where $\mathbf{p}, \mathbf{p}' \in \mathbb{R}^3$. By decomposing $\overrightarrow{D}^{\triangleleft}$ as $D(P'(\lambda)) = D(\mathbf{p}')$, and following our notation for volume deformation, Eq. 4.2 can thus be rewritten as:

$$\mathbf{p} = W(W^{-1}\mathbf{p}' + D(W^{-1}\mathbf{p}') \tag{4.3}$$

In the following discussions, we shall assume that the displacement function and the position function are defined in a common reference coordinate system Λ and that the coordinate system mapping W is the identity transformation. We will revisit this later on in Section 4.7.1 to define affine transformations on displacement maps and in Chapter 5 to define feature-aligned deformations. Based on this assumption, the displacement mapping function is then a direct inverse warping transformation, defined as:

$$\mathbf{p} = T_F^{-1}(\mathbf{p}') = \mathbf{p}' + D(\mathbf{p}') \tag{4.4}$$

For simplicity, we consider the backward displacement mapping function in the form of D in the following discussions. There is no constraint as to the displacement values of D. The displacement can be of any direction and magnitude.

4.4 Modeling discontinuities via Displacements

As described in Chapter 2, previous approaches such as Spatial Transfer Functions consider the discontinuity by introducing a null position $\varnothing$, and the deformation becomes:

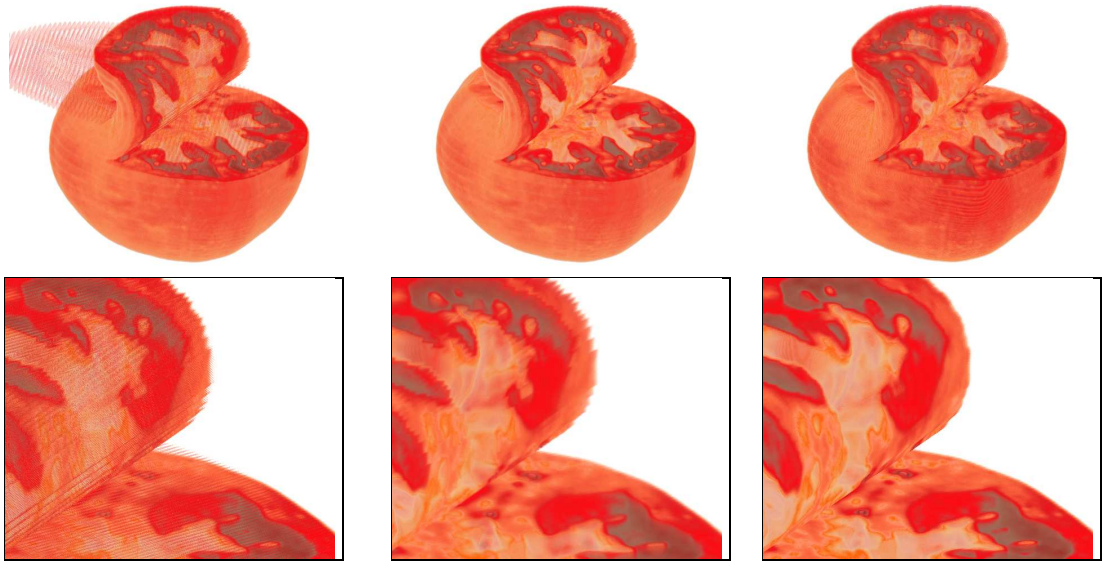
$$\mathbf{p} = T_B(\mathbf{p}') = \begin{cases} T_F^{-1}(\mathbf{p}') & \mathbf{p}' \in P'_V \\ \varnothing & \mathbf{p}' \in P'_{empty} \end{cases} \tag{4.5}$$

where P'_V is the set of points in the deformed volume that have a pre-image in the original set of points P_V, and P'_{empty} the set of points with no pre-image. Rendering of the volume is performed by sampling a scalar function f', defined as:

$$f'(\mathbf{p}') = \begin{cases} f(\mathbf{p}) & \mathbf{p} \neq \varnothing \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The problem with this method is the difficulty of expressing a null value in the displacement map. A zero vector would indicate a zero displacement, which is not the same as $\varnothing$. A vector of ones would indeed map a sample point to a position outside the original volume, which, in principle, could be assumed to be defined as $\varnothing$, since no valid displacement can be that large. However, this introduces problems for hardware implementation (and even for traditional ray-casting), due to tri-linear interpolation used to sample the displacement map. We refer to such method of encoding discontinuities as *out-of-bounds* encoding. With an arbitrarily large value for the definition of $\varnothing$, points near cuts would have a displacement interpolated between this value and the displacement of the closest non-cut point. This value, however, may be a valid displacement, but in a direction and with a magnitude unintended by the deformation. This causes ragged edges and aliasing artifacts in cuts as shown in Figure 4.2(a).

For this reason, we consider the displacement as a pseudo-inverse D^+ of the original forward displacement $\overrightarrow{D}^{\triangleleft}$, which is defined as the inverse displacement for those points in the co-domain of $\overrightarrow{D}^{\triangleright}$, and an interpolated value D^* for those points in the region of discontinuity, such that D^+ maintains C^0 continuity. In other words, D^+ is valid for all points in the displacement map, including those where there should be a discontinuity. For the rest of this chapter,



(a) Out-of-bounds displacement (b) Binary alpha mask (c) Smooth alpha mask

Figure 4.2: Modeling of discontinuities using discretely sampled displacements. Top Row: peeling of a tomato dataset. Bottom Row: Zoomed view. (a) Using out-of-bounds displacement results in jagged lines and unintentional displacement (b) Binary alpha masks results in aliasing (c) Smooth alpha masks.

we refer to such complete displacement simply as D.

In order to model the actual discontinuity, we use an alpha mask with the same size and resolution of the displacement map. This alpha mask is used to tag certain points as parts of the cut. One alternative is to define the alpha mask as a binary map, where a value of 1 indicates a valid displacement and a value of 0 defines a region of discontinuity. The problem with this binary map is the presence of aliasing in the surface of the cut, as shown in Figure 4.2(b).

A better approach is to define the alpha mask as a smooth scalar field, such that a value greater than or equal to 0 is considered as a valid displacement, and a value less than 0 as a region of discontinuity. This smooth scalar field can be obtained by smoothing the binary definition of the cut, or by computing its signed distance field. In fact, the alpha mask is an implicit representation of the cut surface, where the actual surface is the zero-set $\{\mathbf{x}|A(\mathbf{x}) = 0\}$ This approach results in a better and smoother surface in cuts and breaks, as shown in Figure 4.2(c).

Therefore, T_F becomes invertible and its inverse transformation $T_B = T_F^{-1}$ is:

$$\mathbf{p} = T_F^{-1}(\mathbf{p}') = \mathbf{p}' + D(\mathbf{p}') \tag{4.7}$$

And the sampling of the scalar function can be defined as:

$$f'(\mathbf{p}') = \begin{cases} f(\mathbf{p}) & A(\mathbf{p}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where $A(\mathbf{p})$ is a scalar field defining the continuity mask of $D(\mathbf{p})$.

4.5 Rendering Pipeline

Our approach uses texture-based volume rendering with view-aligned slices. We use slicing as a rendering mechanism for discrete sampling of the displaced object space, rather than for storing a view-dependent representation of the displaced object. Since this space is unknown we define a bounding box object as a *proxy scene geometry*.

Let O be the function of an object and O' that of the displaced object. The proxy scene geometry in effect defines a bounded spatial domain of O'. If the proxy scene geometry contains a volume data set, we can use the texture-based volume rendering method which samples the proxy scene geometry with slices parallel to the view plane. Each pixel in a slice is then mapped back to the original object space where O is defined, via an inverse displacement map. Function O can be any object representation such that given a position $\mathbf{p}$ in the original object space, it returns appropriate luminance attributes at $\mathbf{p}$. $O(\mathbf{p})$ can easily be an implicit surface function, a level set surface, a distance field or a color volume texture. In our case, we use a scalar volume dataset, which we denote as f, and store it as a 3D texture in GPU memory.

4.5.1 Displacement Setup

To create a displacement texture, D, of size $w \times h \times d$, we first specify $\overrightarrow{D}^{\triangleright}$ procedurally, and then sample its inverse transformation $D = \overrightarrow{D}^{\triangleleft}$ at discrete positions $\{x,y,z | x = 0,1,\ldots,w; y = 0,1,\ldots,h; z = 0,1,\ldots,d\}$. As described above, the inverse may not be defined for all points in

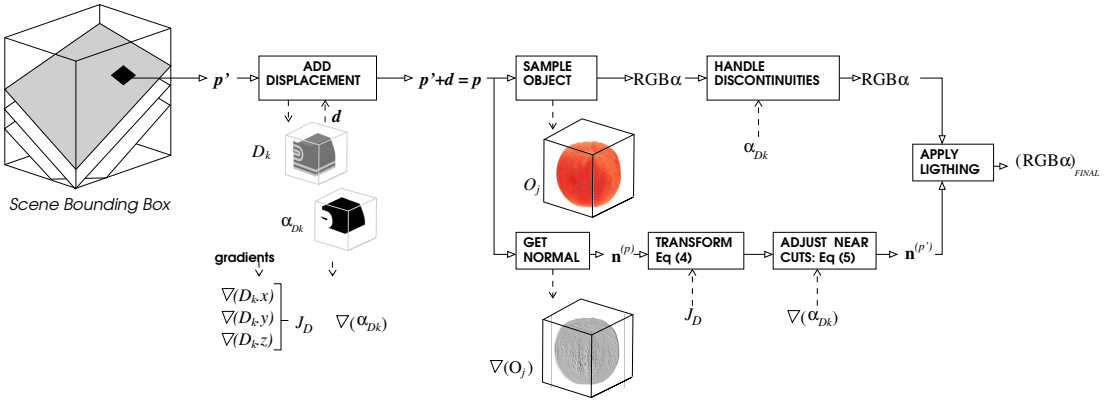


Figure 4.3: System diagram for discontinuous displacement mapping. As we sample the bounding box of the scene, each fragment $\mathbf{p}'$ is displaced by a distance d obtained by sampling the displacement texture D. The resulting position $\mathbf{p} = \mathbf{p}' + d$ is used to sample the object texture f and gradient texture $\nabla(f)$, to obtain color and normal information. Color, normal and opacity (obtained from alpha mask A) are used to compute the final color of the fragment.

the domain of the deformation, so D is extended such that at least C^0 continuity is obtained. C^1 continuity is desired, as it will imply that the discontinuity is differentiable at all points. The displacement values must be normalized in the interval $[-1,1]$, then scaled and biased to fit in the range of valid values of GPU textures ($[0,1]$). The alpha mask can be created as a binary mask, so that it is 1 if $\mathbf{p}'$ has a pre-image in the co-domain of $\overrightarrow{D}^{\triangleright}$ and 0 otherwise, and discretize it in a 3D texture. As described above, aliasing-free rendering is possible as long as the alpha mask is smooth. Therefore, we define the final alpha mask A as an implicit representation of such binary map, obtained through smoothing or via a distance field of the binary map. Similarly to D, alpha mask values must be biased and scale to fit into the range of 3D textures. D and A are illustrated in Figure 4.3, where the striped pattern in the displacement map is used to show the stretching that occurs at the discontinuity. The actual break is depicted as a dark region in the alpha texture.

4.5.2 Displaced Object Points

In order to determine the displaced volume, we slice the proxy scene geometry into view-oriented slices, as shown in Figure 4.3. The bounding box of O' can easily be found by combining the bounding boxes of the object(s) and their displacements. The slices are rendered in back-to-front order and finally composited using alpha blending. For each point $\mathbf{p}'$ on the slice,

we must find the appropriate displacement, since there can be more than one displacement acting on the object (See Section 4.7 on composite maps). We use a fragment program to find the opacity and color values of a given pixel with texture coordinates $\mathbf{p}'$. This program computes $\mathbf{p} = \mathbf{p}' + D(\mathbf{p}')$, where $\mathbf{p}$ is the position in the original object O that corresponds to the texture coordinate $\mathbf{p}'$. It then samples the 3D texture f at the position $\mathbf{p}$ and retrieve the color components. Finally, in order to handle discontinuities, it samples the alpha mask A at the position $\mathbf{p}'$ and modulate the pixel's color components with the mask, according to Eq.(4.8). The process of the fragment shader is depicted in Figure 4.3.

4.5.3 Displaced Surface Normal

In order to properly shade the object, we need the normal information at each point. Since we store objects as volumes, normals can be obtained using finite differences or filters such as the Sobel operator. For interactive rendering, the gradient can be pre-computed and stored in a 3D texture. When processing fragments, a texture fetch of the gradient texture yields the normal of the voxel and its magnitude (This is shown as $\nabla(f)$ in Figure 4.3). There are three cases to be considered: the points that are not displaced, the displaced points, and those in the boundary of a discontinuity. For points not displaced, the pre-computed normals can be used directly. The other two cases are handled as follows.

Normals at Displaced Points

For a point undergoing displacement, we must obtain a new normal. Figure 4.4(b) shows the case where the pre-computed normal is used, which results in incorrect shading. In Figure 4.4(b), we peel the top of a piggy bank object (the light is above the piggy bank). We see the incorrect shading of the peeled surface. Since the surface was originally facing away from the light, it remains dark even though it is now facing the light after peeling. A more accurate shading of the surface can be seen in Figures 4.4(c) and 4.4(d) where the peeled surface is correctly shaded and toward the light.

Normals can be computed on-the-fly by sampling the neighboring voxels (after displacement) and applying finite differences. However, this method requires up to 6 (for central differences) additional displacement computations, which is computationally expensive. What is

needed is a way to transform the undeformed normals on the fly without additional sampling. This was proposed by Barr for forward deformation of surface meshes. The new normal at $\mathbf{p}'$ can be obtained by transforming the original one at $\mathbf{p}$ using the Jacobian of the deformation:

$$\overrightarrow{\mathbf{n}}^{(p')} = \det \mathsf{J}_F (\mathsf{J}_F^{-1})^\top \overrightarrow{\mathbf{n}}^{(p)} \tag{4.9}$$

where $\overrightarrow{\mathbf{n}}^{(p)}$ is the normal vector through the point $\mathbf{p}$, J_F is the 3×3 Jacobian of the *forward* transformation T_F and $\det \mathsf{J}_F$ its determinant .

Our approach, however, defines an inverse space warping, so it is necessary to obtain a similar expression based on the inverse transformation. Given $T_B = T_F^{-1}$ as the inverse transformation, and since $\mathbf{p} = T_B(\mathbf{p}')$, Eq.(4.9) leads to:

$$\overrightarrow{\mathbf{n}}^{(p)} \quad = \quad \det \mathsf{J}_B (\mathsf{J}_B^{-1})^\top \overrightarrow{\mathbf{n}}^{(p')}$$

or equivalently:

$$\frac{1}{\det \mathsf{J}_B} (\mathsf{J}_B)^\top \overrightarrow{\mathbf{n}}^{(p)} \quad = \quad \overrightarrow{\mathbf{n}}^{(p')} \tag{4.10}$$

where J_B is the Jacobian of the inverse transformation $T_B = T_F^{-1}$.

Since $T_B(\mathbf{p}') = T_F^{-1}(\mathbf{p}') = \mathbf{p}' + D(\mathbf{p}')$, we can defined it as a multi-variate mapping in 3D:

$$\mathbf{p} = T_B(x,y,z) = \begin{pmatrix} T_B x(x,y,z) \\ T_B y(x,y,z) \\ T_B z(x,y,z) \end{pmatrix} = \begin{pmatrix} x + D_x(x,y,z) \\ y + D_y(x,y,z) \\ z + D_z(x,y,z) \end{pmatrix}$$

and the Jacobian J_B is the derivative of T_B with respect to the spatial coordinates. Therefore:

$$\begin{aligned}
\mathsf{J}_B(x,y,z) &= \begin{bmatrix} \frac{\partial T_B x}{\partial x} & \frac{\partial T_B x}{\partial y} & \frac{\partial T_B x}{\partial z} \\ \frac{\partial T_B y}{\partial x} & \frac{\partial T_B y}{\partial y} & \frac{\partial T_B y}{\partial z} \\ \frac{\partial T_B z}{\partial x} & \frac{\partial T_B z}{\partial y} & \frac{\partial T_B z}{\partial z} \end{bmatrix} \\
&= \begin{bmatrix} 1+\frac{\partial D_x}{\partial x} & \frac{\partial D_x}{\partial y} & \frac{\partial D_x}{\partial z} \\ \frac{\partial D_y}{\partial x} & 1+\frac{\partial D_y}{\partial y} & \frac{\partial D_y}{\partial z} \\ \frac{\partial D_z}{\partial x} & \frac{\partial D_z}{\partial y} & 1+\frac{\partial D_z}{\partial z} \end{bmatrix} \\
&= \mathsf{I}+\mathsf{J}_D
\end{aligned}$$

where I is the identity matrix and J_D is the Jacobian of the displacement map. Then, Eq. (4.10) becomes

$$\overrightarrow{\mathbf{n}}^{(p')} = \frac{1}{\det(\mathsf{I}+\mathsf{J}_D)}(\mathsf{I}+\mathsf{J}_D)^\top \overrightarrow{\mathbf{n}}^{(p)}$$

Since normal vectors must be normalized, the $\frac{1}{\det(\mathsf{I}+\mathsf{J}_D)}$ factor can be omitted. This leads to our normal transformation equation:

$$\overrightarrow{\mathbf{n}}^{(p')} = (\mathsf{I}+\mathsf{J}_D^{(p')})^\top \overrightarrow{\mathbf{n}}^{(p)} \tag{4.11}$$

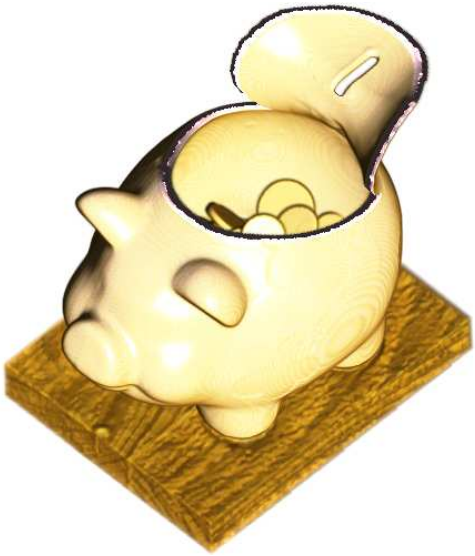
This last step avoids the division for zero that might occur when the Jacobian is singular. However, the Jacobian is only singular at regions of breaks. In our approach, the displacement map is well defined and continuous for all points, and the breaks are handled by a different mechanism in the pipeline (see Figure 4.3). In practice, the Jacobian of the displacement map is approximated as the gradient of the displacement texture, i.e., $\mathsf{J}_D \approx \left(\nabla_{D_x}, \nabla_{D_y}, \nabla_{D_z}\right)^\top$, using finite differencing. For speed up, one can pre-compute the matrix

$$\mathsf{B} = (\mathsf{I}+\mathsf{J}_D^{(p')})^\top \tag{4.12}$$

for each voxel in the displacement map, and store it as 3D textures.

(a)

(b)



(c)

(d)

Figure 4.4: Lighting computation for the piggy bank object. The light is above the piggy bank. (a) No lighting. (b) Using the pre-computed gradient results in incorrect lighting, notice how the underside of the cut surface is dark even though it is facing the light. (c) Correct lighting, but artifacts occur at discontinuities – rim of the cut area. (d) Correct lighting with proper handling of normals at discontinuities. Now the rim, which is facing the light, is lit properly.

Normals at Discontinuities

Unfortunately, even with correct normal transformation, we still see artifacts on the resulting objects at the boundaries of cuts. In this case, the normals of an object may change even when that part of the object does not undergo displacement. An example of this is depicted in Figure

4.4(c). Note that the rim of the piggy bank at the cut is dark. This is because those normals have not undergone transformations (like in the underside of the peeled surface) and are therefore incorrectly pointing away from the light source. This is especially noticeable on solid objects with a uniform interior. Even for the case of objects with a heterogeneous interior, as long as their normals are not directed orthogonal to the cut, the cut surface will be lit incorrectly.

To properly compute the normals at the discontinuities, we need a way to determine the new surface that has been created. Luckily, this information is stored in the alpha map A. We can compute the gradient of the alpha mask ∇_A, and use this value only at the boundary of a cut. However, applying only this gradient in the boundary, may generate artifacts near the boundary. To solve this, we gradually correct the normal in the vicinity of the cut to the desired normal, via blending:

$$\overrightarrow{\mathbf{n}}^{(p')} = \omega(\mathrm{I} + \mathrm{J}_D)^{\top} \overrightarrow{\mathbf{n}}^{(p)} + (1-\omega)\nabla_A^{(p')} \tag{4.13}$$

where $\omega \in [0,1]$ is a blending factor. Figure 4.4(d) shows the result of applying this method for the piggy bank object. Note that the pixels at the rim of the cut are now properly shaded. This blending mechanism is similar to the solution proposed by Weiskopf et al. [124] for volumetric cutaways. Although the alpha gradient can be computed on the fly using finite differencing, it can also be precomputed and stored in a 3D texture for speedup.

4.5.4 Compositing

Because our displacement method is integrated with rendering, the final image is obtained by compositing the samples obtained by slicing the deformed space. An important consideration is that the sampling frequency of the deformed space affects the result of the composition. To picture this problem, let us consider a deformation where a large compression is simulated. Because all the visible voxels are being compressed into a narrow part, it might happen that only a few samples are assigned to that compressed part, resulting in aliasing or undersampled space. For this reason, it is necessary to adjust the sampling frequency when rendering deformed volumes. However, this is not always feasible in texture-based rendering. Therefore, we can alternatively adjust the *transparency* or *emission* properties of the traversed voxels.

For the analysis of this aspects, let us review the volume integral equation. For a ray $\overrightarrow{v}$

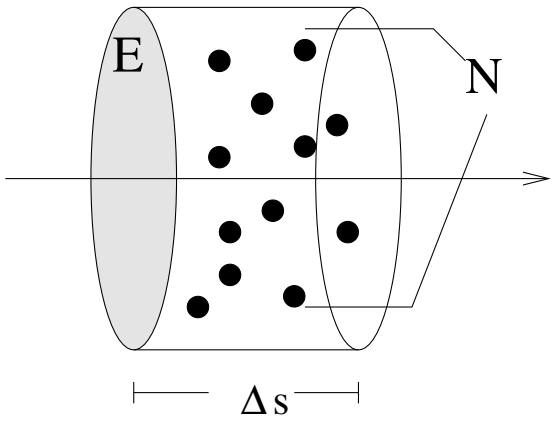


Figure 4.5: Cylindrical slab of area E and length Δs. Light attenuation is due to the interaction of light with a number of particles in this slab.

parameterized by t, we want to obtain the intensity at the eye position $\mathbf{p_e}$. This is obtained as follows:

$$I(t_e) = \int_0^{t_e} c(t) e^{-\int_t^E \tau(s) ds} dt \tag{4.14}$$

where $c(t)$ is the emission coefficient of the point $\mathbf{p} = \mathbf{p_B} + t\overrightarrow{v}$, for $\mathbf{p_B}$ a point at the back of the volume ($t = 0$), and $t = t_e$ at the eye ($\mathbf{p_e}$). The exponential is the absorption or extinction part of the integral. When discretized, the extinction part becomes:

$$\begin{aligned} e^{-\int_t^S \tau(s) ds} &\approx e^{-\sum_{i=0}^{n} \tau(i\Delta s)\Delta s} \\ &= \prod_{i=1}^{n} e^{\tau(i\Delta s)\Delta s} \\ &= \prod_{i=1}^{n} t_i \end{aligned}$$

where t_i is also referred to as the transparency of the voxel.

It is debatable as to what should be the behavior of light in a deformed volume. For most physical objects, deformation implies a compression or expansion of particles, which alter the way light is absorbed as it traverses the volume. Those cases apply when *mass conservation* is assumed. On the other hand, a volume may represent an abstract 3D object, where the notion of particles does not apply. In those cases, it is more important to maintain the intensity values constant across the deformation, for all the light rays. Here we consider the two cases.

Mass Conservation

The attenuation function $\tau(s)$ is derived as the effective area occluded by the particles in a cylinder slab of area E and length Δs, as seen in Figure 4.5. Let ρ be the local density of particles in that cylinder slab. The number of particles in the slab is $N = \rho E \Delta s$, and the attenuation coefficient is $\tau = A_P \rho$, where A_P is the cross sectional area of each particle. Now, for a deformed volume, which is the one being rendered, let us define $\tau' = A'_P \rho'$ as the attenuation coefficient, where $A'_P = A_P$. The slab in the deformed volume, with volume $V' = E' \Delta' s$, maps back to a volumetric figure of volume $V = |\det J_B| V'$, where J_B is the Jacobian of the backward transformation. Since the number of particles remains constant:

$$N = \rho' V' = \rho V \tag{4.15}$$

or equivalently:

$$\rho = \frac{\rho' V'}{V} = \frac{\rho'}{|\det J_B|} \tag{4.16}$$

Because the cross sectional area of particles remain constant, the attenuation coefficient in the undeformed volume is then:

$$\tau(s) = A_P \rho = A_P \frac{\rho'}{|\det J_B|} = \tau'(s) \frac{1}{|\det J_B|} \tag{4.17}$$

and therefore:

$$\tau'(s) = |\det J_B| \tau(s) \tag{4.18}$$

Now let us consider only the extinction part of a voxel. In the deformed space, which is the one to be rendered, this is defined as:

$$T(t) = e^{-\int_0^t \tau'(s) ds} \tag{4.19}$$

$$= e^{-\int_0^t |\det J_B| \tau(s) ds} \tag{4.20}$$

Discretizing the attenuation coefficient along the view direction Δs:

$$T(t) \approx \prod_{i=1}^{n} e^{\tau(i\Delta s)|\det \mathrm{J}_B|\Delta s} \tag{4.21}$$

$$= \prod t_i^{|\det \mathrm{J}_B|} \tag{4.22}$$

where t_i is the transparency of the *original* voxel i along the ray.

In most renderers, it is the opacity $\alpha_i = 1 - t_i$, rather than the transparency, the quantity used to represent extinction. Let α_i' be the opacity of a voxel i along a view ray when sampling the deformed volume. The opacity adjustment can be found as follows:

$$\alpha_i' = 1 - (1 - \alpha_i)^{|\det \mathrm{J}_B|} \tag{4.23}$$

This is the approach taken by Chen et al. in [18].

Intensity Constancy

Now let us consider a ray $\overrightarrow{v}$ traversing the *deformed* space, parameterized such that a sample point is defined as $\mathbf{p}' = \mathbf{p_e}' + t\overrightarrow{v}$. In the undeformed space, this ray maps into a curve u, parameterized by s.

If we assume intensity constancy, the attenuation due to particles along the ray and along the curve u should be the same. This can be achieved by considering the integration along the ray in the deformed space as a line integral along a curvilinear coordinate transformation, defined by the inverse transformation T_B. Let $T(X)$ be the extinction coefficient in the undeformed space along a curve u, and $T'(X)$ the extinction coefficient in the deformed space.

$$T(X) = e^{-\int_0^X \tau(s)ds} \tag{4.24}$$

$$= e^{-\int_0^X \tau(s(t))\frac{ds}{dt}dt} \tag{4.25}$$

$$= e^{-\int_0^X \tau'(t)dt} \tag{4.26}$$

$$= T'(X) \tag{4.27}$$

Because $s = s(\mathbf{p}) = s(T_B(\mathbf{p}'))$, then $\frac{ds}{dt}$ is a multivariate derivative and is defined as :

$$\frac{ds}{dt} = |J_B \overrightarrow{v}| \tag{4.28}$$

where $\mathsf{J}_B = \mathsf{I} + \mathsf{J}_D$ is the Jacobian of the inverse displacement as described in the previous section and $|\overrightarrow{v}|$ is the length of vector $\overrightarrow{v}$. And then we have:

$$\tau'(t) = |J_B \overrightarrow{v}|\, \tau(t) \tag{4.29}$$

Following a similar analysis to the one derived for the mass conservation assumption, we determine that the opacity adjustment for a particular sample in the deformed space is obtained as:

$$\alpha_i' = 1 - (1 - \alpha_i)^{|J_B \overrightarrow{v}|} \tag{4.30}$$

The main difference with the result for the mass conservation assumption is that this method neglects the change in the extinction parameters in the directions orthogonal to the deformation. Therefore, the intensity values at the eye points remain constant across deformations, from any viewpoint. This result is useful for the rendering of abstract or non-physical volumetric objects. However, it is also useful for the depiction of deformed isosurfaces. Maintaining the intensity constant for an isosurface provides a cue for continuity. The results of the composition mechanism for a deformed volume are explored in Chapter 6.

4.6 Construction of Displacement Maps

An important part of the process of realizing illustrative deformation is the construction of displacement maps. Here, we describe three methods. The first two allow the creation of primitive displacement maps, while the third, described in the following section, uses algebraic operations on primitive maps to create more complex displacements. The two methods for primitive displacement map creation are through procedural description and through interpolated nodal displacements using Radial Basis Functions. More information is given in Appendix C.

4.6.1 Procedural Description

The first mechanism for creating displacement maps is through a procedural description. As an example, consider the poke operator, as shown in Figure 4.10. This displacement uses a 3D Gaussian function to simulate a pull in the Z direction. The displacement can be constructed as:

$$D(x,y,z) = \begin{pmatrix} 0, & 0, & -ze^{\frac{(x-0.5)^2+(y-0.5)^2}{2\sigma^2}} \end{pmatrix}^\top \tag{4.31}$$

for (x,y,z) in a unit cube, and σ chosen so that displacement becomes 0 at the XY boundaries of the unit cube. This displacement is discretized and stored in a 3D texture of size $64 \times 64 \times 64$. Note that the z value is used to modulate the amplitude of the Gaussian pull, and that the Z component of the displacement is negative, since D stores the *inverse* displacement. For instance, the point $(32,32,32)$ in the 3D texture, corresponding to the normalized point $(0.5,0.5,0.5)$ in the unit cube, contains the displacement vector $(0,0,-0.5)$.

When considering cuts, we follow a similar process. First, we compute the alpha channel of the cut procedurally, so that $A(x,y,z)$ is 0 whenever there is a cut, and 1 elsewhere, and discretize it in a texture volume. In addition, we apply a smoothing operator over the alpha mask in order to obtain a smooth region around the boundaries needed for the blending of the normals, as described in Section 4.5.3. The result is stored in the alpha component of the displacement texture.

It is important to note that procedural displacements must be derived in the inverse space, rather than as a direct transformation. For twists, bends and tapering, we use the derivations by Barr [6]. For all others, we derived a series of mathematical expressions, summarized in Appendix C.

4.6.2 Inverse Weighted Interpolation

Some displacement maps might be difficult to derive analytically. In addition, manual displacement of all grid points may become difficult to manage. For this reason, many deformation techniques use nodal displacements to drive the deformation of a set of given control points and scattered data interpolation to determine the displacements at grid points. Because we have inverse displacements, the interpolation methods are often referred to as *inverse distance*

weighted interpolation methods, where a given function is interpolated within a 3D region, based on the distance to user-specified control points [97]. The most common interpolation methods are Shepard's interpolant, based on Radial Basis Functions (RBFs) and Moving Least Squares. In this work, we use Radial Basis Functions for the interpolation of nodal displacements. In this method, the interpolation function is constructed as a linear combination of radially symmetric basis functions, centered at the control points. A function g at a point $\mathbf{p}$ can be interpolated from a set of control points $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ as follows:

$$g(\mathbf{p}) = \sum_{i=1}^{n} \alpha_i g_i(d(\mathbf{p}, \mathbf{p_i})) + p_m(\mathbf{p}) \tag{4.32}$$

where $d(\mathbf{p}, \mathbf{p}_i)$ is the distance from point $\mathbf{p}$ to control point $\mathbf{p}_i$, g_i is a basis function, p_m is a polynomial of degree m, and α_i are coefficients. These coefficients are found by putting the data points into the RBF equation and solving the system of linear equations. A number of radial basis functions have been proposed, of which we use Hardy's inverse multiquadrics:

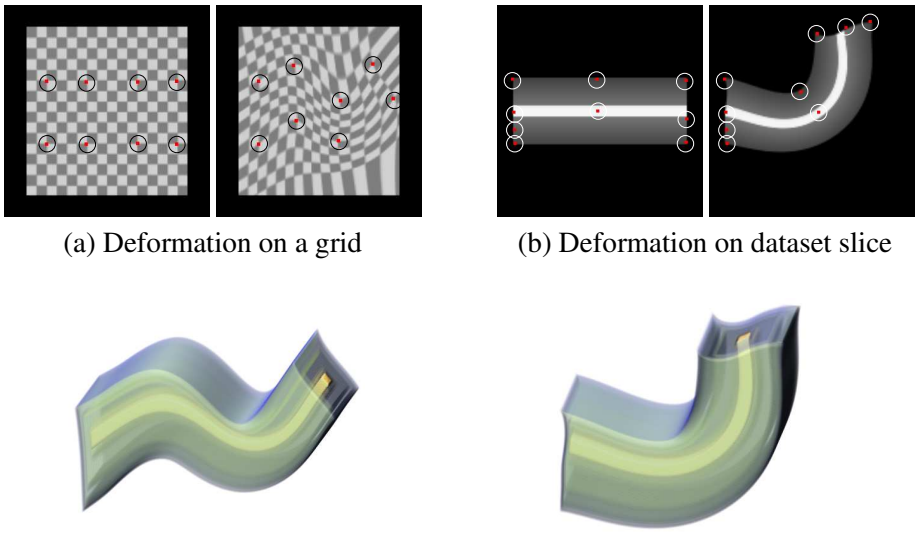
$$g_i(d) = (d^2 + r_i^2)^{\mu} \tag{4.33}$$

Depending on the value of μ, the radial basis function can be differentiable to a certain degree. For $\mu = -0.5$, the interpolation if C^{∞}. Because of the ability to predict the differentiability of radial basis functions, we have used this method for the creation of displacement maps. In addition, we use a polynomial of degree 1:

$$p_1(\mathbf{p}) = 1 + x + y + z \tag{4.34}$$

for a point $\mathbf{p} = (x, y, z)^{\top}$.

We built a 2D warping editor that allows us to deform a 2D image using radial basis functions. In this case, the interpolated function g is a 2D displacement. A 3D displacement can be created by extending the 2D displacement along the Z axis. The use of 2D instead of 3D enables easier control of the deformation and can be used to find deformations along a particular axis. For instance, Figure 4.6 shows the process of creating 3D displacements from a 2D deformation. In Figure 4.6(a), we show the deformation of a 2D grid using 8 control points



(a) Deformation on a grid

(b) Deformation on dataset slice

(c) Resulting 3D Deformation

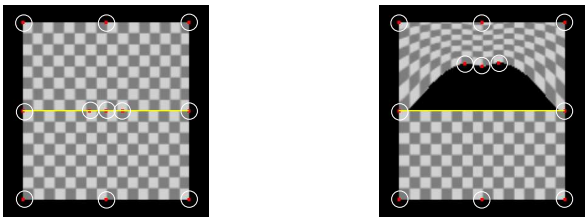
(d) Resulting 3D Deformation

Figure 4.6: Displacement map creation using Radial Basis Functions. (a),(b) Nodal deformation on 2D slices (c),(d) Resulting deformation on a 3D volumetric dataset.

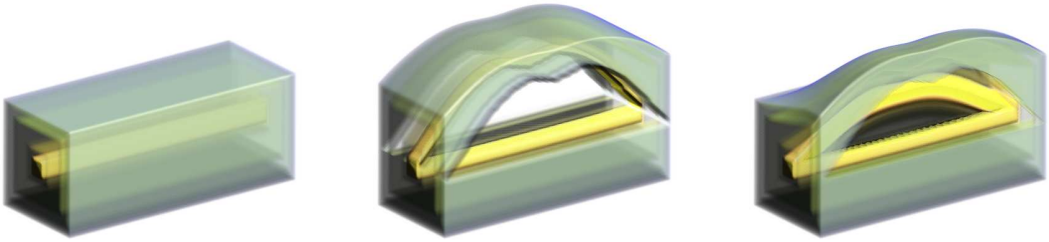
(marked in red). Figure 4.6(c) shows the result of applying the generated displacement to a bar dataset. Certain displacements can be generated with a particular dataset as a target, as the case depicted in Figure 4.6(b) and (d), where a 2D deformation of 9 control points is used to derive a bending transformation on a bar dataset. The use of a slice of the target dataset allows a better manipulation of the control points to achieve the desired result.

One of the advantages of RBFs is the ability to obtain displacements that are at least C^1 continuous. However, this poses a problem for defining discontinuous deformation. In the case of cuts, C^1 continuity is desired in the sub-regions that appear after the cut. For this reason, we devised what we call *Decoupled Radial Basis Functions* (DRBF). For simplicity, let us consider a DRBF in 2D with two continuity regions (i.e., there is a single cut). The two continuity regions divide the image (or volume for 3D) into the regions V_P and V_Q. In addition, we assign control points to a particular region, i.e., control points are divided into the sets $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ and $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$, respectively. The 2D displacement with cuts is then given by:

$$D(\mathbf{p}) = \begin{cases} \sum_{i=1}^{n} \mathbf{a}_i g_i(d(\mathbf{p}, \mathbf{p_i})) + \mathbf{p}_m(\mathbf{p}) & \mathbf{p} \text{ in } V_P \\ \sum_{j=1}^{m} \mathbf{b}_j h_j(d(\mathbf{p}, \mathbf{q_j})) + \mathbf{p}_m(\mathbf{p}) & \mathbf{p} \text{ in } V_Q \end{cases} \tag{4.35}$$



(a) Original Grid (b) Deformed grid (Decoupled RBF)



(c) Original Dataset (d) Applied Deformation (e) Smoother Deformation

Figure 4.7: Discontinuous displacement map creation using Decoupled Radial Basis Functions.

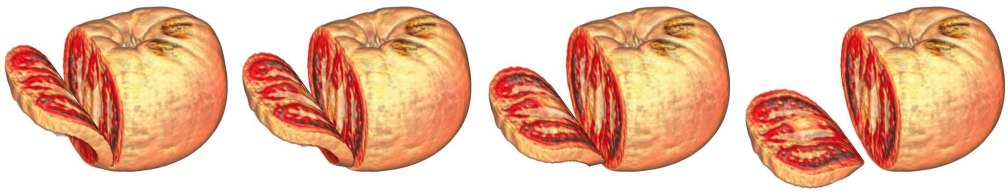
where g_i and h_j are radial basis functions, and $\mathbf{a}_i = (\alpha_{i1}, \alpha_{i2})^\top$ and $\mathbf{b}_j = (\beta_{j1}, \beta_{j2})$ are the 2D coefficients, which are found by putting the control points and their inverse displacements into the equation and solving the system of linear equations. The discontinuity map, A is then obtained as follows:

$$A(\mathbf{p}) = \begin{cases} 1 & (\mathbf{p} \text{ in } V_P \text{ and } \mathbf{p} + D(\mathbf{p}) \text{ in } V_P) \text{ or } (\mathbf{p} \text{ in } V_Q \text{ and } \mathbf{p} + D(\mathbf{p}) \text{ in } V_Q) \\ 0 & \text{otherwise} \end{cases} \tag{4.36}$$

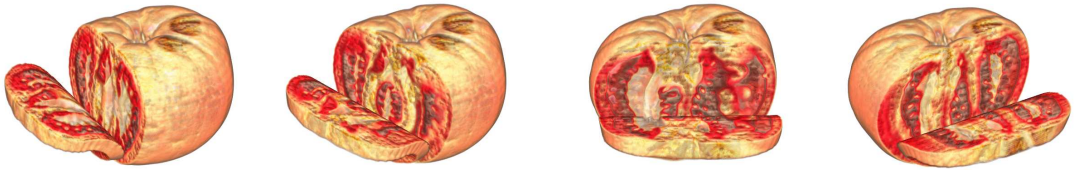
An example is shown in Figure 4.7, where a cut (shown as a yellow line) divides the grid guide in two regions. When applying a deformation, the grid is split as it crosses the boundary defined by the cut, effectively forming a discontinuous deformation. Figure 4.7(d) also shows the result of applying the displacement to the bar dataset. Note also that a complex deformation can be found by interpolating the displacement values down to zero, in order to create a smoother cut (Figure 4.7(e)). Other methods for creating displacement maps are described in the following section, as the result of algebraic operations.

4.7 Algebraic Operations on Displacement Maps

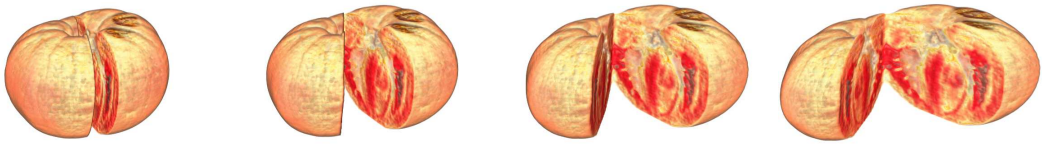
One of the advantages of displacement maps is the ability to operate on them algebraically. This enables the creation of complex deformations from the combination of simple primitive



(a) Translations



(b) Rotations



(c) Scalings

Figure 4.8: Affine transformation of displacement on the tomato dataset.

displacements. This section describes several mechanisms for operating displacement maps, such as affine transformations, addition and composition.

4.7.1 Affine Transformations

One of the operations defined on the displacement map is the translation of the displacement map by a constant vector $\mathbf{u}$, i.e., the displacement equation becomes:

$$\mathbf{p} = \mathbf{p}' + D(\mathbf{p}' - \mathbf{u}) \tag{4.37}$$

Similarly, scaling of the displacement map can be defined:

$$\mathbf{p} = \mathbf{p}' + \sigma D(\sigma^{-1}\mathbf{p}') \tag{4.38}$$

where $\sigma \neq 0$ is a real value.

We can generalize this notion by unifying this operation, together with rotations and shears, as an affine transformation of the displacement. An affine transformation on a vector $\mathbf{p}$ can be

defined as $\mathsf{M}\mathbf{p}+u$, where M is a 3×3 affine matrix and u is a global translation. Alternatively, we can use homogeneous coordinates of a point $\mathbf{p}$. Let use define $\hat{\mathbf{p}}$ as the representation of p in homogeneous coordinates, i.e.

$$\hat{\mathbf{p}} = \left\{ \begin{array}{c} \mathbf{p} \\ 1 \end{array} \right\} \tag{4.39}$$

Using homogeneous coordinates, we can generalize the displacement by defining an affine coordinate transformation between the displacement space and the object space. That is, related to Eq.(4.1), $W(\mathbf{p}) = \mathsf{M}\mathbf{p}$, where M is a 4×4 affine transformation. Then, the displacement equation becomes:

$$\begin{aligned} \hat{\mathbf{p}} &= \hat{\mathbf{p}}' + \mathsf{M}\times D(\mathsf{M}^{-1}\hat{\mathbf{p}}') \\ &= \mathsf{M}\times\left(\mathsf{M}^{-1}\hat{\mathbf{p}}' + D(\mathsf{M}^{-1}\hat{\mathbf{p}}')\right) \end{aligned} \tag{4.40}$$

The last expression is useful, since it explains the way the affine transformation is applied. First, coordinate frame Λ is transformed via the inverse transformation into the displacement coordinate frame Λ_D. Since the displacements are given with respect to Λ_D, the result is finally transformed in the coordinate frame defined by M.

The normal transformation when undergoing this type of coordinate transformation is given by the concatenation of the transpose of the Jacobians of the coordinate transformation. This can be derived as follows. The inverse transformation is defined as $T_B(\mathbf{p}') = \mathbf{p}' + (W \circ D \circ W^{-1})(\mathbf{p}')$. The normal transformation $\mathsf{J}_B^{\top}$ is:

$$\begin{aligned} \mathsf{J}_B^{\top} &= (\mathsf{I} + \mathsf{J}_W \mathsf{J}_D \mathsf{J}_W^{-1})^{\top} \\ &= \mathsf{I} + \mathsf{J}_W^{-\top}\mathsf{J}_D^{\top}\mathsf{J}_W^{\top} \\ &= \mathsf{J}_W^{-\top}\left(\mathsf{I} + \mathsf{J}_D^{\top}\right)\mathsf{J}_W^{\top} \end{aligned}$$

where J_W is the Jacobian of the transformation. Therefore, the normal transformation equation becomes:

$$\overrightarrow{\mathbf{n}}^{(p')} = \left[\mathsf{J}_W^{-\top}\left(\mathsf{I} + \mathsf{J}_D\right)^{\top}\mathsf{J}_W^{\top}\right]\overrightarrow{\mathbf{n}}^{(p)} \tag{4.41}$$

For an affine matrix, J_W is the 3×3 upper sub-matrix of M. In this work, transformations are

represented as a rotation followed by a scaling and a translation, i.e., $\mathsf{M} = \mathsf{T} \times \mathsf{S} \times \mathsf{R}$. Therefore, $\mathsf{J}_W = (\mathsf{S} \times \mathsf{R})$ and Eq. (4.41) becomes:

$$\overrightarrow{\mathbf{n}}^{(p')} = \left[(\mathsf{S}^{-1} \times \mathsf{R}) (\mathsf{I} + \mathsf{J}_D)^{\top} (\mathsf{R}^{\top} \times \mathsf{S}) \right] \overrightarrow{\mathbf{n}}^{(p)} \tag{4.42}$$

This transformation via affine matrix is an efficient mechanism for controlling the shape, position and orientation of a displacement map. Examples are shown in Figure 4.8, where translation, rotation and scaling of a slicing displacement produce interesting deformations on a tomato dataset. Since this method only involves multiplication of constant matrices, this is an efficient alternative for controlling the displacement in an interactive application.

4.7.2 Addition

Displacement maps offer the flexibility of creating new displacements via addition. Let D_1 and D_2 be two displacement maps. Then, a new displacement map $D_S = D_1 + D_2$ can be defined, such that:

$$\mathbf{p} = \mathbf{p}' + D_1(\mathbf{p}') + D_2(\mathbf{p}') \tag{4.43}$$

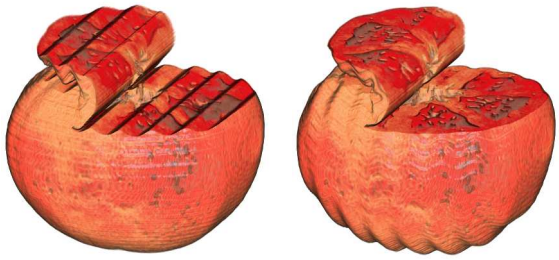
Further, since the displacements contain discontinuities, the new alpha mask is defined as:

$$A_{1+2} = \rho min(|A_1|, |A_2|) \tag{4.44}$$

$$\rho = \begin{cases} -1 & A_1 < 0 \wedge A_2 < 0 \\ 1 & \text{otherwise} \end{cases} \tag{4.45}$$

Addition can be performed on a pre-processing stage to create a new displacement and then apply it to a volumetric object. Alternatively, they can be added on the fly. In that case, the normal transformation can be obtained by simply adding the Jacobians of the displacements:

$$\overrightarrow{\mathbf{n}}^{(p')} = (\mathsf{I} + \mathsf{J}_{D1}^{(p')} + \mathsf{J}_{D2}^{(p')})^{\top} \overrightarrow{\mathbf{n}}^{(p)} \tag{4.46}$$



(a) $\mathbf{p} = \mathbf{p}' + D1(\mathbf{p}' + D2(\mathbf{p}'))$ (b) $\mathbf{p} = \mathbf{p}' + D2(\mathbf{p}' + D1(\mathbf{p}'))$

Figure 4.9: Composition of two displacement maps $D1$ (wave) and $D2$ (peel) in different order.

where J_{D1} is the Jacobian of D_1 and J_{D2} is the Jacobian of D_2. Since we pre-compute the matrices $\mathsf{B}_1 = (\mathsf{I} + \mathsf{J}_{D1})^\top$ and $\mathsf{B}_2 = (\mathsf{I} + \mathsf{J}_{D2})^\top$. Eq.(4.47) can be re-written as follows:

$$\overrightarrow{\mathbf{n}}^{(p')} = (\mathsf{B}_1 + \mathsf{B}_2 - \mathsf{I})^\top \overrightarrow{\mathbf{n}}^{(p)} \tag{4.47}$$

4.7.3 Composition

In general, two transformations can be combined as follows:

$$\mathbf{p} \;=\; G_1(G_2(\mathbf{p}')) \tag{4.48}$$

where $G_1(\mathbf{u}) = \mathbf{u} + D_1(\mathbf{u})$ and $G_2(\mathbf{v}) = \mathbf{v} + D_2(\mathbf{v})$ are displacement mappings.

For computing the normal, we must simply concatenate the Jacobians of the two mappings:

$$\overrightarrow{\mathbf{n}}^{(p')} = (\mathsf{B}_1 \times \mathsf{B}_2)\, \overrightarrow{\mathbf{n}}^{(p)} \tag{4.49}$$

where $\mathsf{B}_1 = (\mathsf{I} + \mathsf{J}_{D_1})^\top$ and $\mathsf{B}_2 = (\mathsf{I} + \mathsf{J}_{D_2})^\top$ are the pre-computed normal transformation matrices of the displacement mappings, as defined in Eq.(4.11). An example is shown in Figure 4.9, where two different displacements are combined in different order and applied to the tomato dataset. Since each displacement changes the frame of reference, composition is, unlike addition, not commutative in general.

Pre-computed combination of displacement maps results in another displacement map and

does not require any changes in the GPU rendering process. It is a useful mechanism to create complex displacements from simple ones. On the other hand, on-the-fly combination enables the interactive creation and manipulation of independent displacement maps, though it requires a modified GPU implementation. Composite maps can be realized in the GPU program in a single pass by iterating the displacement procedure on the voxel positions for each displacement map, before sampling the original object. This approach, however, cannot be generalized easily to many compositions. Research on a general multi-pass rendering process is being undertaken.

4.8 Results

We have implemented the displacement mapping approach within an interactive program which allows the user to rotate and scale the object and to move or change the displacement map. The displacement map can be changed interactively via linear transformations (see Section 4.7). Figure 4.4 shows a peeling of the side of a piggy bank dataset, revealing the coins in the inside. In order to validate our approach, we created a number of displacement textures and applied them to various datasets. Figure 4.10 shows four displacements applied to the box, bar, engine and tomato datasets. The first two are synthetic objects created procedurally. The strips in the box dataset help us track the continuity and smoothness of the displacement. A similar effect is obtained by tracking the inner bar (in yellow) of the bar dataset. More information on the datasets can be found in Appendix A.

One of the difficulties of the use of displacement maps is the definition of large or global deformation, since the region of influence is bounded. Scaling and affine transformations help realize arbitrarily large deformations in a scalable manner. For global deformations, such as kinematic movement, a displacement would require a size comparable to that of the dataset. Further, out-of-bounds deformations are difficult to model. For instance, consider the slicing deformation shown in Figure 4.11. The sliced portions of the tomato are part of the deformation definition, but soon fall out of the bounds of the displacement map, and we require a explicit definition of how the displacement is extended beyond the texture boundaries. This can be modeled using texture clamping capabilities. For instance, it is possible to specify that deformation values are clamped to 0 outside the map bounds, that they are clamped to the values on

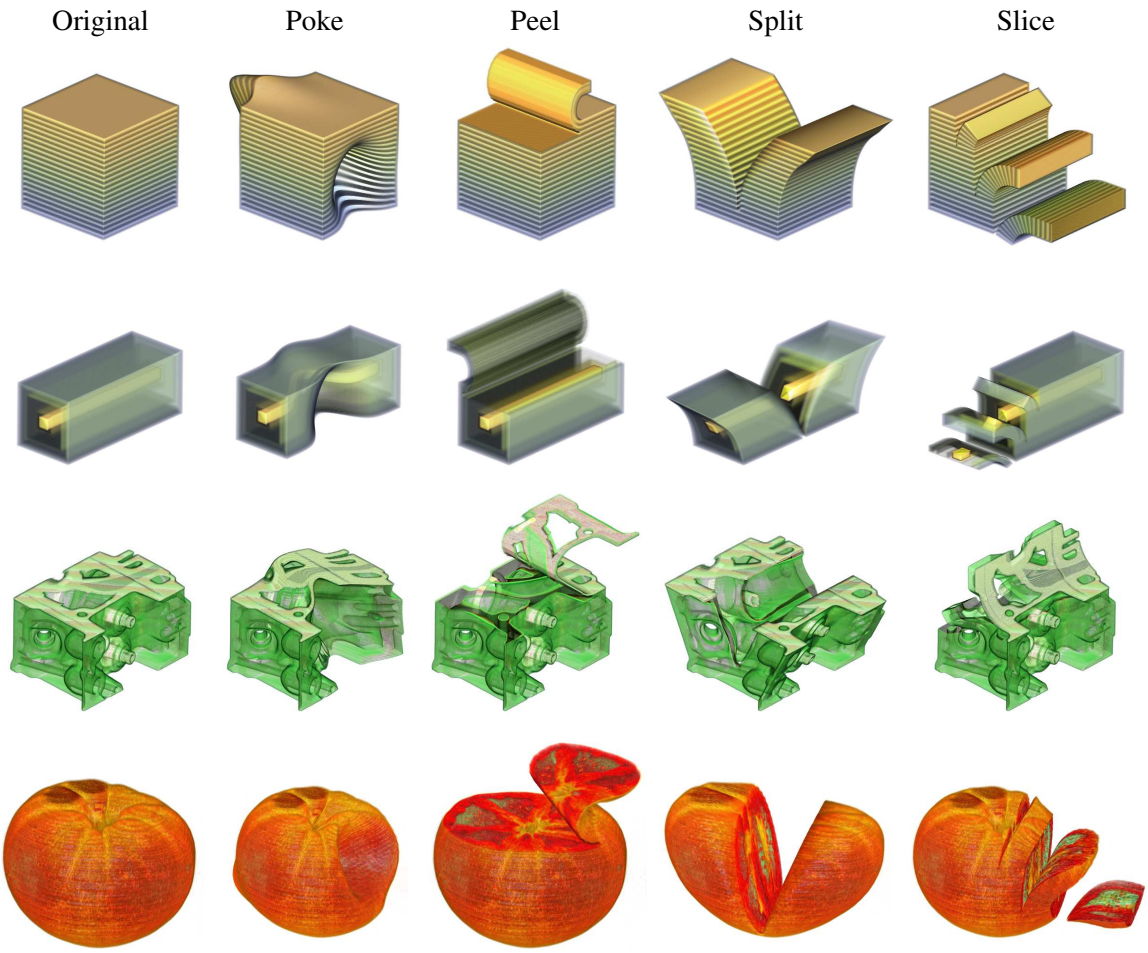


Figure 4.10: Example deformations on volumetric objects

the edge, or that the deformation repeats. However, this method has limited expressiveness and cannot capture all possibilities. For this reason, we had to explicitly define the out-of-bounds interpolation of displacement values for the slicing deformation, as part of the warping process. A more general and scalable solution is yet to be found. In Figure 4.11, several time steps are rendered which show a progressive slicing as the user moves the displacement vertically. This displacement can be obtained from a simple slicing displacement and repeating it periodically along the main diagonal of the displacement map.

4.9 Chapter Summary

In this chapter, we have demonstrated how discontinuous displacement maps can simulate many different types of volume effects such as fracturing, slicing, deforming and cutting of graphical

Figure 4.11: Slicing of the tomato

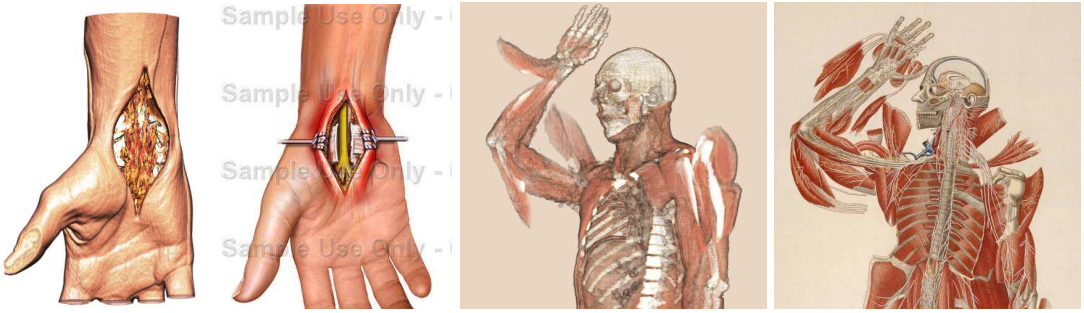
objects. We have employed inverse displacement maps in 3D vector space to solve for large and discontinuous displacements. We have also devised a collection of techniques, including computing surface normals changed due to unorthogonal displacement, correcting lighting artifacts at fractures, and creating composite maps from primitive maps on the fly. The displacement map can easily be represented as a 3D texture and the entire rendering process can be coded into a fragment program yielding interactive results. We have shown their effect on a number of different models, and their ability to be combined in multiple ways, demonstrating the generality, interactivity, and usability of this approach.

Chapter 5
Feature Aligned Deformation[1]

5.1 Introduction

In the previous chapters, we described a method for modeling cuts and deformations in volumetric objects. One of the applications of this method is in scientific illustration, where deformations are used to depict the stages and outcome of a procedure, to uncover hidden features, or to reveal the spatial relationship between different components of an object. However, scientific illustrations often recurse to *feature-sensitive operations* to be effective. Feature-sensitive operations are only applied to a semantic component of an object, such as the skin in Figure 5.1(b) and the muscles in Figure 5.1(d), without affecting other parts of the object.

[1] Portions of this chapter were published in our paper: Feature Aligned Volume Manipulation for Illustration and Visualization, by C. Correa, D. Silver and M. Chen, IEEE Transactions on Visualization and Computer Graphics, 2006, pp. 1069–1076



(a) (b) *Illustration* (c) (d) *Illustration*

Figure 5.1: Feature-aligned volume manipulation (a) A feature-aligned retraction applied to a human hand data set, showing bones (left) and vessels (right) (b) Surgical illustration of a hand (Copyright ©2006 Nucleus Medical Art. All rights reserved. www.nucleusinc.com) (c) Multiple peel and cutting away operators applied to the visible human data set (d) Illustration of human anatomy with dissected "flaps", by Antonio Scrantoni and Paolo Mascagni, 1833 (U.S. National Library of Medicine), similar to exhibitions such as BodyWorlds [119], and Bodies, the Exhibition [1].

We can see from these illustrations that, when specifying a cut or peel, one important consideration is "alignment". Here, *aligned manipulation* refers to those cuts or peels that are applied to certain layers while other features of interest are preserved. The simplest is *axis-aligned*, which aligns the operator with the axis-planes. It is feature-insensitive and is applied to all points within the volume bounds. Deformations and cuts depicted in the previous chapter can be all considered as axis-aligned. This however is not always satisfactory as the "object" within the volume is not necessarily cubic. In this chapter, we introduce a number of approaches for obtaining feature-aligned deformation, which can be classified as either *transformation-based* or *mask-based* methods. Transformation-based methods use a coordinate frame transformation to align an axis-aligned deformation to an arbitrary line, curve or surface. Mask-based methods use a volumetric mask along a surface or segment of interest. We extend our rendering algorithm to accurately estimate the normals along the surface of cuts and dissections while maintaining continuous normals, which allows us to obtain correct shading of the object being manipulated, opaque or translucent.

5.2 Related Work

As described in Chapter 2, most volume deformation techniques treat the volume as an homogeneous collection of voxels. The notion of feature sensitivity has been explored in the context of multi-dimensional transfer functions [60, 45] and illustrative techniques [117, 14] , where the distinction of features of interest is made in the rendering process.

In the context of volume deformation, feature sensitivity has been largely unexplored. One of the first to suggest the use of features when deforming volumes was McGuffin *et al.* [76], who proposed to define semantic layers of the data where primitive points are extracted for rendering. Because of the rendering process, which treats voxels as their own disjoint primitives, meaningful deformation could be obtained by transforming the voxels associated with a particular layer. In this way, they were able to achieve deformations where a layer such as skin is moved or peeled away to gain visibility of the muscle and bone layers. However, as described before, this method does not yield smooth surfaces or volumes.

Other volume rendering techniques, such as [125, 93, 108, 18] do not include any type of

feature sensitivity and consider the volume as an homogeneous collection of points. Most of these, however, were intended for continuous deformation, where the feature preservation may be of limited use, since there is no natural way to explore the interior of the object. In contrast, discontinuous deformation of volumes exploit the idea of feature alignment to be more effective. Since there is a mechanism to gain visibility of internal structures by cutting, splitting or exploding, it becomes necessary to specify a way of keeping features of interest untransformed. Approaches that enable such operations are Spatial Transfer Functions [20], volume splitting [54] and exploded views of volumetric objects [13]. In all these cases, the feature and "background" volume (i.e., the difference volume between the original volume and the feature of interest), are obtained through a pre-processing stage where two volumes are utilized. Therefore, the problem of feature alignment is reduced to the rendering of a multi-object scene: one object, corresponding to the feature of interest, is rendered using traditional volume rendering techniques, while the other, the remaining volume, is deformed according to each methodology. One of the problems with these methods is that explicit segmentation of volumes may introduce aliasing at the boundaries, and pre-processing of the datasets is required. It also decouples the rendering of the objects, so there is no guarantee on the smoothness and continuity of the deformation in the regions where the feature of interest and the background volume meet. In this chapter, we describe a method where feature alignment is part of the deformation and rendering process, so that it can be achieved in a single rendering pass.

Another important aspect of feature-sensitive deformation is the correct lighting of the surfaces of the cut. Because cuts are feature-aligned, the "underside" must represent the surface left by the removal of the feature of interest. Correct representation of this surface via shading is essential for providing cues about the shape of the object. In previous approaches, this has been relatively unexplored, and surfaces of cuts often present artifacts or use a simpler shading model to hide them. A mechanism for adjusting the normals near the surface of cuts for volumetric clipping was proposed by Weiskopf et al. [124], which allow correct rendering of cut surfaces. We extend this idea for the rendering of deformed feature-aligned volumes, in a similar way as we define the normals in Chapter 4.

5.3 Transformation-Based Alignment

In many cases, deformation operators need to be aligned with a particular line, curve or surface. For instance, a peel on the cranium needs to be aligned with the surface of the skull, which in turn can be approximated by a curve in the surface of a sphere. In such cases, the alignment can be represented via a transformation. In Chapter 4, we defined generalized displacements as:

$$\mathbf{p} = W(W^{-1}(\mathbf{p}') + D(W^{-1}(\mathbf{p}')) \tag{5.1}$$

where W is a mapping function from the displacement space to the object space and W^{-1} its

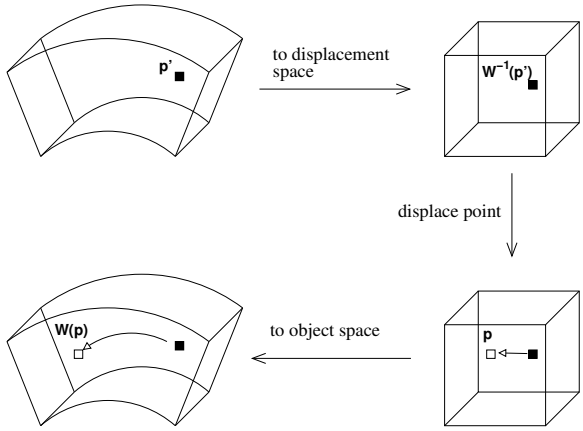


Figure 5.2: Transformation-based alignment.

inverse. This process is depicted in Figure 5.2. The mapping function W defines an embedding of the displacement space into object space. The simplest one, as described in chapter 4 is via linear transformations. This process embeds the cubic space of the displacement into a cuboid of arbitrary orientation and scale. This is achieved by defining W as an affine matrix A:

$$\begin{aligned} \hat{\mathbf{p}} &= \mathsf{A} \times \left(\mathsf{A}^{-1}\hat{\mathbf{p}}' + D(\mathsf{A}^{-1}\hat{\mathbf{p}}')\right) \\ &= \hat{\mathbf{p}}' + \mathsf{A} \times D(\mathsf{A}^{-1}\hat{\mathbf{p}}') \end{aligned} \tag{5.2}$$

where A is a 4×4 affine matrix and $\hat{\mathbf{p}}'$ is the sampling point expressed in homogeneous coordinates.

Another possibility is to align the deformation with an arbitrary curve in space. This is achieved by first parameterizing the space along the length of the curve and then defining an interpolation function in the enclosed volume. One of the simplest methods is piecewise

bilinear interpolation. Here, we define two parameters, s and t along a connected sequence of line segments and the mapping function $W : (s,t) \mapsto (x.y)$ as a coordinate transformation function. The third spatial parameter r is assumed to coincide with the object spatial coordinate z.

Since we can safely ignore the third coordinate, the mapping function W can be defined at each line segment as a quadrilateral interpolation function. Given $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ and $\mathbf{p}_4$ the corners of a quadrilateral, and $s,t \in [0,1]$ two parameters:

$$\mathbf{p}'(x,y) = W(\mathbf{p}(s,t)) = (1-s)(1-t)\mathbf{p}_1 + s(1-t)\mathbf{p}_2 + (1-s)t\mathbf{p}_3 + st\mathbf{p}_4 \tag{5.3}$$

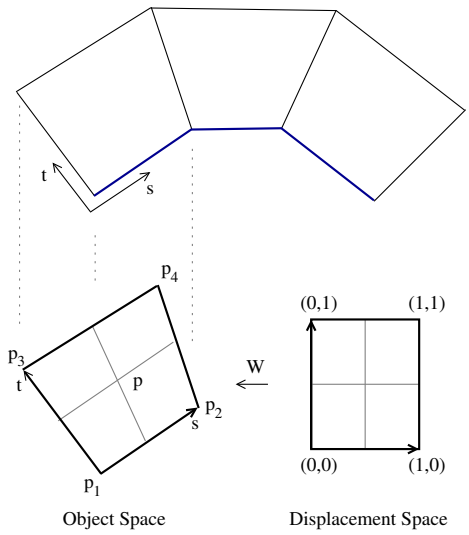


Figure 5.3: Piecewise linear alignment with a curve of three segments. Each line segment is used to define a quadrilateral by extending its normals. Bilinear interpolation is used as the displacement-object space mapping

Other finite elements can be used to accommodate different types of alignment shapes (such as tetrahedra or hexahedra) and various types of interpolation (tri-linear, tri-cubic, etc.). In this work, we use bi-linear hexahedra as shown above. A similar analysis can be performed for other shape functions, to align the deformation to a surface.

5.3.1 Normal Estimation

Following the analysis introduced in the previous chapter, it is also necessary to estimate the normals of deformed points in order to obtain proper lighting of deformed volumes. In Chapter 4 we derived an expression for normal estimation of inversely displaced points, using the transpose of the Jacobian of the inverse transformation. According to vector calculus, given two differentiable transformations F and G, the Jacobian of the composite transformation $F \circ G$ evaluated at a point $\mathbf{p}$ is:

$$\mathsf{J}_{F\circ G}(\mathbf{p}) = \mathsf{J}_F(G(\mathbf{p}))\mathsf{J}_G(\mathbf{p}) \tag{5.4}$$

Also, as a corollary of this equation, the Jacobian of the inverse function F^{-1} is:

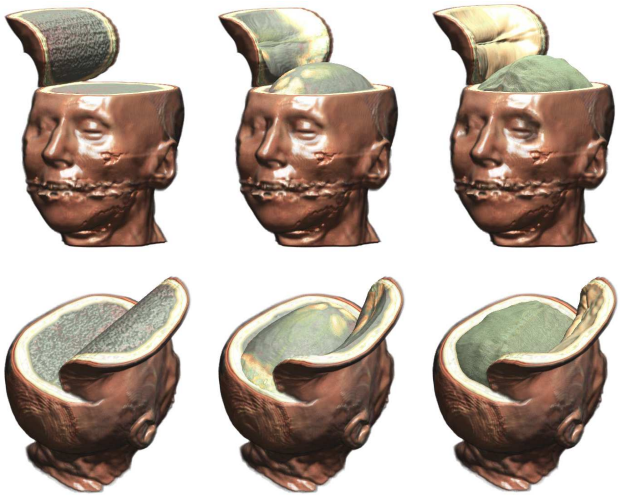
$$\mathsf{J}_{F^{-1}}(\mathbf{p}) = \left(\mathsf{J}_F(F^{-1}(\mathbf{p})\right)^{-1} \tag{5.5}$$

For a transformation-based deformation, the new normal can be estimated by concatenating the transpose Jacobians of each transformation mapping, as follows:

$$\begin{aligned} \overrightarrow{\mathbf{n}}^{(p')} &= \left(\mathsf{J}_W(W^{-1}\mathbf{p}' + D(W^{-1}\mathbf{p}'))(\mathsf{I} + \mathsf{J}_D(W^{-1}\mathbf{p}))\mathsf{J}_{W^{-1}}(\mathbf{p})\right)^{\top} \overrightarrow{\mathbf{n}}^{(p)} && (5.6)\\ &= \left(\mathsf{J}_W(W^{-1}\mathbf{p}' + D(W^{-1}\mathbf{p}'))(\mathsf{I} + \mathsf{J}_D(W^{-1}\mathbf{p}))\mathsf{J}_W^{-1}(W^{-1}(\mathbf{p}))\right)^{\top} \overrightarrow{\mathbf{n}}^{(p)} && (5.7) \end{aligned}$$

5.3.2 Implementation details

The implementation of this into our rendering pipeline requires a mechanism for computing the mapping transformation W and its inverse. W can be computed on the fly using the expression in Eq.(5.3). The inverse transformation W^{-1}, however, is more complicated to compute. As an alternative, we use the slicing mechanism to generate the s and t parameters directly. In the case of three-dimensional elements, an additional parameter r can be also computed via slicing. Trilinear interpolation on hardware of these parameters yield the correct parameterization of each point in object space. For each sample point, now we have two texture coordinates, one corresponding to the coordinates in object space (x,y,z) and the other in displacement space (s,t,r). In addition, higher order transformations (e.g., quadratic or cubic) can also be accommodated, by adding additional shape functions to the computation of W. However, to avoid extra computation, a generic mechanism can be devised based on texture lookups. W can be precomputed in a 2D texture or a 3D texture, indexed by the parameters (s,t) and (s,t,r) respectively.



(a) Axis Alignment (b) Surface Alignment (c) Segment Alignment

Figure 5.4: An example of different types of alignment. (a) Axis aligned peel. Note how the peeled layer is thick and flat, since it is aligned with an orthogonal axis. (b) Surface aligned peel, aligned with a computed distance field. Notice how it approximates a surface. (c) Segment aligned peel, based on segmentation, which is more accurate. Note that in the feature based alignment (b) and (c) the peel is thin and rounded.

5.4 Mask-based Alignment

The second category of alignment deformations is mask-based alignment. In this case, rather than matching the "shape" of the deformation with the feature of interest, we "mask-out" those points that should not be deformed with a volumetric mask. This has proven to be a better and faster alternative to transformation-based alignment. An example of this type of alignment can be seen in Figure 5.4(c), where the operator is said to be aligned with the brain tissue, since the peeler appears to follow its contour. In this chapter, we propose two *mask-based* techniques for allowing deformations that are sensitive to the specific features on a volume model. We consider surface features that are defined by an iso-surface, and volume features that are defined by a segment, resulting in the notions of *surface-aligned* and *segment-aligned* deformation respectively. The merits of such alignments can be seen Figures 5.4(b) and 5.4(c).

Consider the skin of the head is the *context*, and the brain is the *focus* of these visualizations. From Figure 5.4(a) where a peel is performed using axis-alignment, we can observe that the part being peeled is wedge shaped and the operator cuts through the brain. The visualization conveys limited information about the part supposing to be in focus. Feature alignment corrects

this problem. Virtual surfaces are created without segmentation (except for background), using a distance field computation. The distance field is computed based on a boundary, from which virtual shells of different thicknesses can be defined. For example, in 5.4(b), a surface-aligned peel is applied to the top of the head with a uniform depth from the surface of the skin. Such alignment can be used to investigate and illustrate layered structures without the pre-knowledge of segmentation. However, while this is a good approximation , the operator still cuts through the brain. If we have the specification of volume features, typically obtained from segmentation or defined by a range of iso-values, we can create a more effective focus+context visualization. In 5.4(c), with the segmentation knowledge of external layers including the skin and skull, we apply a segment-aligned peel to the head. The brain, that is, the focus, is not only highlighted but also visualized with correct geometry. In addition, the correct shading at the back of the peel provides further meaningful context to the visualization.

5.4.1 Mask-based Volume Deformation

In order to define a mask-based deformation function, we introduce a *masking function* M, which defines the feature-sensitivity of points in the original volume V, and is typically represented by a volume dataset. When $M(\mathbf{p}) \geq 0.5$, $\mathbf{p}$ is part of the feature to be preserved, and cannot be transformed. An example of this is shown in Figure 5.4(c) where the peel is applied to the skin and not to the brains (which is masked as a feature of interest).

Using the notation introduced in Chapter 4, let P_V be the set of points of the original volume and P'_V the deformed set of points. Let P_M be the subset of P_V, such that $P_M = \{\mathbf{p} | \mathbf{p} \in P_V, M(\mathbf{p}) \geq 0.5\}$, and V_M is an axis-aligned bounding volume of P_M. Any point not in P_M is operatable. In addition, the backward transformation function T_B, is also accompanied by a *bounding volume*, V_T, such that the manipulation is only performed over those points residing in V_T.

Unlike our original definition of deformation, in feature-aligned deformation, an inversely transformed point, $\mathbf{p}'$, may have been masked as non-operatable by M, which results in empty space. To handle the complexity of this inverse mapping, we introduce an initial "probe" $\mathbf{p}^0$,

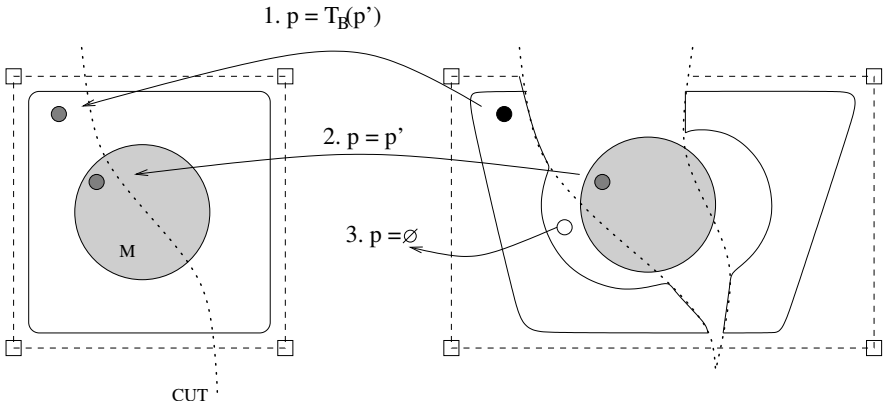


Figure 5.5: Inverse warping cases for mask-based feature-aligned deformation

for each point $p' \in P'_{V'}$ as follows:

$$\mathbf{p}^0 = \begin{cases} T_B(\mathbf{p}') & \mathbf{p}' \text{ resides in } V_T \\ \mathbf{p}' & \text{otherwise} \end{cases} \tag{5.8}$$

We then obtain $\mathbf{p}$ by taking the feature mask into account as:

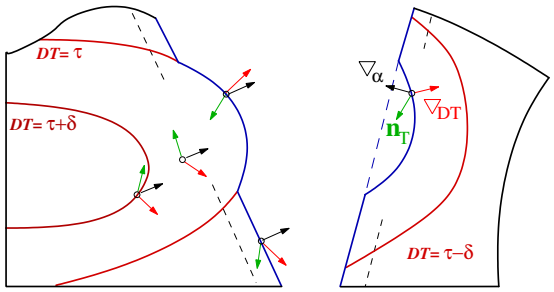
$$\mathbf{p} = \begin{cases} \mathbf{p}^0 & \mathbf{p}^0 \in P_V \wedge (M(\mathbf{p}') < 0.5 \wedge M(\mathbf{p}^0) < 0.5) \\ \mathbf{p}' & \mathbf{p}' \in P_V \wedge M(\mathbf{p}') \geq 0.5 \\ \varnothing & \text{otherwise} \end{cases} \tag{5.9}$$

These three cases are shown in Figure 5.5, namely: (1) the point is transformed, (2) the point is masked and therefore untransformed, and (3) the point is empty due to the feature-aligned cut.

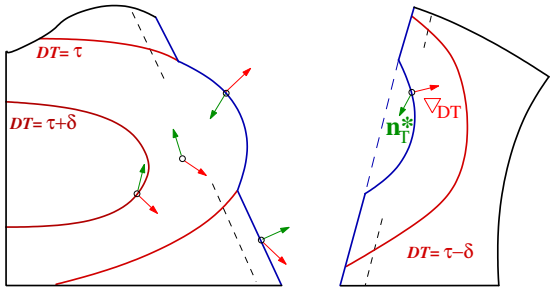
Following the same analysis as other deformation functions, the normal at a point p' is defined as:

$$\overrightarrow{\mathbf{n}}_{\mathsf{T}} = \begin{cases} \overrightarrow{\mathbf{n}}_{\mathsf{J}} & \mathbf{p} = T_F^{-1}(\mathbf{p}') \\ \overrightarrow{\mathbf{n}} & \mathbf{p} = \mathbf{p}' \\ \mathbf{0} & \mathbf{p} = \varnothing \end{cases} \tag{5.10}$$

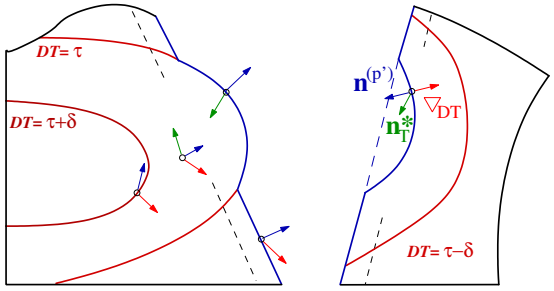
where $\overrightarrow{\mathbf{n}}_{\mathsf{T}}$ denotes the resulting normal at the *inversely transformed* point, $\overrightarrow{\mathbf{n}}$ denotes the original normal, $\overrightarrow{\mathbf{n}}_{\mathsf{J}}$ denotes the transformed normal at sampling point $\mathbf{p}'$, and $\mathbf{0}$ is the zero vector. $\overrightarrow{\mathbf{n}}_{\mathsf{J}}$ is obtained using the transpose of the Jacobian of the inverse transformation, as derived in the previous chapter, and therefore denoted here with a subscript J.



(a) Original Normals



(b) Normals after axis-aligned cut adjustment



(c) Normals after feature-aligned cut adjustment

Figure 5.6: Normal blending for surface alignment. Black arrows indicate the normal of the axis-aligned part of the cut, green arrows indicate the inversely transformed normal, red arrows are the normals orthogonal to the surface of the cut, and the blue arrows indicate the corrected normal after blending.

5.4.2 Surface Alignment

As stated previously, for certain deformations axis alignment is not sufficient. Peeling is one such deformation, since generally the peeling operation is applied to a specific surface layer of an object. In this case, feature-based alignment is desired. As an approximation, a surface-aligned feature can be obtained by defining a mask based on distance from the surface. This can be obtained with the distance field of the volume, after a background segmentation. Let us define DT as the distance field stored as a volume. Then, the mask M can be defined such that

$M(\mathbf{p}) \geq 0.5$ for $DT(\mathbf{p}) \geq \tau$, and $M(\mathbf{p}) < 0.5$, for $DT(\mathbf{p}) < \tau$. One such function is:

$$M(\mathbf{p}) = 0.5\frac{DT(\mathbf{p}) - \tau}{\delta} + 0.5 \tag{5.11}$$

where $\delta \neq 0$ is a thickness parameter that describes how large is the falloff region at both sides of the cut, i.e., the thickness of a slab aligned with the isosurface $M(\mathbf{p}) = 0.5$.

Here $\tau > 0$ is a parameter that specifies the desired distance from surface. For instance, τ can be thought of as the "depth" of the peeled surface, which is to be transformed in order to reveal the feature underneath.

The normal at a point is defined by the influence of different normal information. For a point in the interior of the volume, the normal is defined by the transformed normal $\overrightarrow{\mathbf{n}}_J$. For a point in the vicinity of the cut, it is defined by either the gradient of the distance field ∇_{DT}, which defines the surface of the feature of interest, or the gradient of the alpha map ∇_α, which defines the surface of the cut.

To blend these three normals, we first require special handling of the boundary around the cut surface defined by $DT(\mathbf{p}) = \tau$. For a feature point (i.e., $M(\mathbf{p}) \geq 0.5$), the gradient of the distance field ∇_{DT} points outwards from the interior to the surface. However, for a non-feature point on the boundary (i.e., $M(\mathbf{p}) < 0.5$), the gradient ∇_{DT} points incorrectly from the surface to the interior. We solve this by using a signed weighting function β, which takes values from -1 to 1. First, the normal, as obtained using Eq.(5.10) is adjusted for the axis-aligned cut, as described in chapter 4:

$$\overrightarrow{\mathbf{n}}^*_T = (1-\omega)\overrightarrow{\mathbf{n}}_T + \omega\nabla_\alpha \tag{5.12}$$

where $\omega \in [0,1]$ is a weighting function that decreases with the distance to the discontinuity, i.e., for a point in the boundary, $\omega = 1$, and for a point at a pre-defined distance D from the boundary, $\omega = 0$. This parameter ω controls the gradient smoothness of the cut surface, and it is similar to the "impregnation" region described by Weiskopf et al. in [124] for performing volumetric clipping.

Then, the final normal at a point is a combination of this adjusted normal and the normal of

the surface of interest ∇_{DT}.

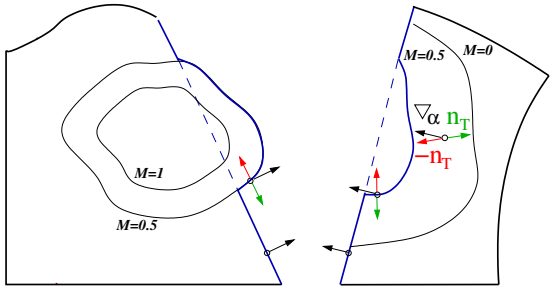
$$\begin{aligned} \overrightarrow{\mathbf{n}}^{(p')} &= (1-|\beta|)\,\overrightarrow{\mathbf{n}}^{*}_{\mathsf{T}} + \beta \nabla_{DT} \\ \beta &= \begin{cases} \frac{\tau - DT(p)}{\delta} - 1 & \tau - \delta < DT(p) < \tau \\ \frac{\tau - DT(p)}{\delta} + 1 & \tau \leq DT(p) < \tau + \delta \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{5.13}$$

β is used to gradually blend the normal of the distance field with the transformed normal. Note that it is asymmetric with respect to τ. That is, for a point with DT in the interval $[\tau, \tau + \delta)$, β ranges from 1 down to 0, but for a point with DT in the interval $(\tau - \delta, \tau)$, β takes values from 0 down to -1. This negative weighting blends the transformed normal $\overrightarrow{\mathbf{n}}_{\mathsf{J}}$ and the normal between the normal of the distance field in the *opposite* direction. This results in correct normals at both sides of the break. This is illustrated in Figure 5.6, and an example of surface-aligned peeling is shown in 5.4(b).

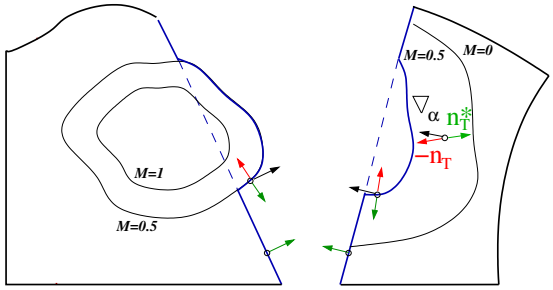
5.4.3 Segment Alignment

Segment alignment is obtained by defining $M(\mathbf{p})$ based on a volume feature, that typically is determined through segmentation. It can be seen quite easily that the above technique for surface alignment can be extended to handle an arbitrary volumetric mask by replacing $DT(\mathbf{p})$ directly with $M(\mathbf{p})$. Normal adjustment is handled as follows:

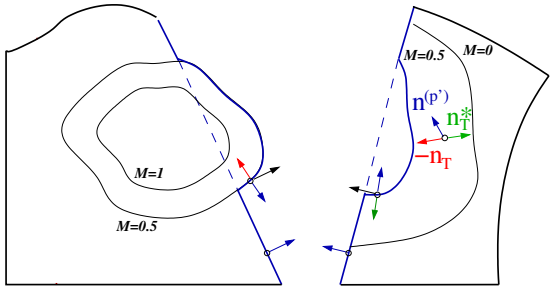
First, we assume that the gradient of the original volume is computed with the aid of the segmentation information, stored as a volume texture. This correctly estimates the surface gradient of the features of interest. Further, for the boundary between two features, the normals on either side point to the opposite direction of those of the other side. This leads to a problem in texture-based volume rendering, since trilinear interpolation of these opposite normals would yield an incorrect zero gradient at the surface of the cut. To overcome this problem, we estimate the gradients for segmented data so that they always point outwards from the feature of interest on both sides of the boundary. When computing the gradient volume texture using finite differences, we consider the values of the neighbors of a voxel as 0, if they correspond to a different segment or an empty voxel, or as 1 if they correspond to the segment of interest. This is equivalent to computing the gradient of the distance field of the segmented features rather



(a) Original Normals



(b) Normals after axis-aligned cut adjustment



(c) Normals after feature-aligned cut adjustment

Figure 5.7: Normal blending for segment alignment. Black arrows indicate the normal of the axis-aligned part of the cut, green arrows indicate the transformed normal, red arrows are the normals orthogonal to the outer surface of the segment, and the blue arrows indicate the corrected normal after blending.

than on the binary volume.

Finally, we invert the normals in the non-feature side of the cut following the blending mechanism in the previous section. To define a thick area where this blending can be possible, we assume that the mask $M(\mathbf{p})$ defines a smooth scalar field. The region where we need to invert the directions of the normals is defined by the isosurfaces $M(\mathbf{p}) = 0.5$ and $M(\mathbf{p}) = 0$, as shown in Figure 5.7. After adjustment for the axis-aligned part of the cut using Eq.(5.12) , we

can obtain the normal as:

$$\begin{aligned} \overrightarrow{\mathbf{n}}^{(p')} &= (1-\gamma)\,\overrightarrow{\mathbf{n}}^{*}_{\mathsf{T}} \\ \gamma &= \begin{cases} 2M(p)+1 & 0 < M(p) < 0.5 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{5.14}$$

Previous approaches for achieving the same results of segment alignment often use pre-segmented datasets and two-pass rendering, where the segment of interest to be preserved is rendered first, and then the non-feature part of the volume is manipulated and rendered afterwards. However, this approach results in aliased boundaries, because of pre-segmentation. Although this can be addressed with pre-smoothing and pre-processing of colormaps, our approach works on a single pass and requires no pre-processing of the dataset or the transfer function.

5.5 GPU Implementation

In order to implement this, we extended the implementation described in the previous chapter, to include queries about the feature mask. The mask is stored as a 3D texture. To properly model feature alignment, the fragment shader must sample the volume at both the original and warped positions, which decreases the performance. In addition, determining which case to apply when computing the final warped position, as defined in Eq.(5.9), requires conditional statements which are known for being slow in current GPU architectures. With the introduction of new GPU architectures, with dynamic branching, our approach can be greatly accelerated. To obtain the correct color attributes, the normals must be determined, by evaluating Eq. (4.11). The normal of each fragment requires at most three gradient texture samples: the normal obtained from the transformation (∇_T), as discussed in Section 4.1, the normal of the cut (∇_α), and the normal of the feature mask, depending on the alignment. These are blended together as shown in Eqs. (5.13) and (5.14).

The GPU memory requirements for this process are predominantly determined by the resolution needed to store the volumetric dataset (e.g., the head dataset requires 256^3 bytes and its gradient requires 3×256^3 bytes). An additional requirement is imposed by the pre-computation of the manipulation operators. However, these are in general very small compared to the 3D

volume data. When resources are scarce, normals can be computed on the fly using finite differences, not only for the original dataset, but also for the alignment mask and the operator itself.

5.6 Results

One of the applications of feature-aligned volume manipulation is medical and biological illustrations. In Figure 5.1(b), an illustration from hand surgery is shown. Figure 5.1(a) demonstrates a retraction operator on a CT hand mimicking the same type of cut. Figure 5.1(d) is an image from the illustration of human anatomy by Antonio Scrantoni and Paolo Mascagni, dated 1833 (NLM). Interestingly, this image is very similar to contemporary exhibitions such as BodyWorlds [119] and Bodies, The Exhibition [1] which portray dissections of actual bodies. Figure 5.1(c) shows a similar type of operation applied to the Visible Man dataset. The dataset was first posed using [40] to position the arm upright. Five peel operators were then applied to both arms.

Figure 5.8 shows a comparative table of applying mask-based deformations to various datasets. In the first row, we see a peeler deformation applied to the CT head dataset, simulating the kind of illustrations used for craniotomies. Axis-aligned manipulation inevitably deforms bone and brain tissue and results in a flat cut. Surface alignment is a good approximation of the shape of the skull, and finally, segment alignment provides the best illustration of bone tissue. In the second row, we see a cut on a forefoot CT dataset. A similar cut is applied to a hand dataset in the third row. Feature alignment is used to reveal bone tissue. Surface alignment, as seen in the foot deformation, provides a fast mechanism to explore and visualize tissue difficult to segment, such as small veins and vascular structures. Finally, the fourth row shows a simulation of dissection of a frog dataset. Axis alignment does not allow to visualize the internal organs of the frog. While surface alignment gives a hint of the internal organs, the best results are obtained with segment-alignment.

In addition to illustration-like effects, the manipulation operators also improve on clipping and slicing and generate focus+context visualizations. Now slices can be arbitrary geometries, and there is a focus+context mechanism (peeling) for keeping the sliced portion in view. In

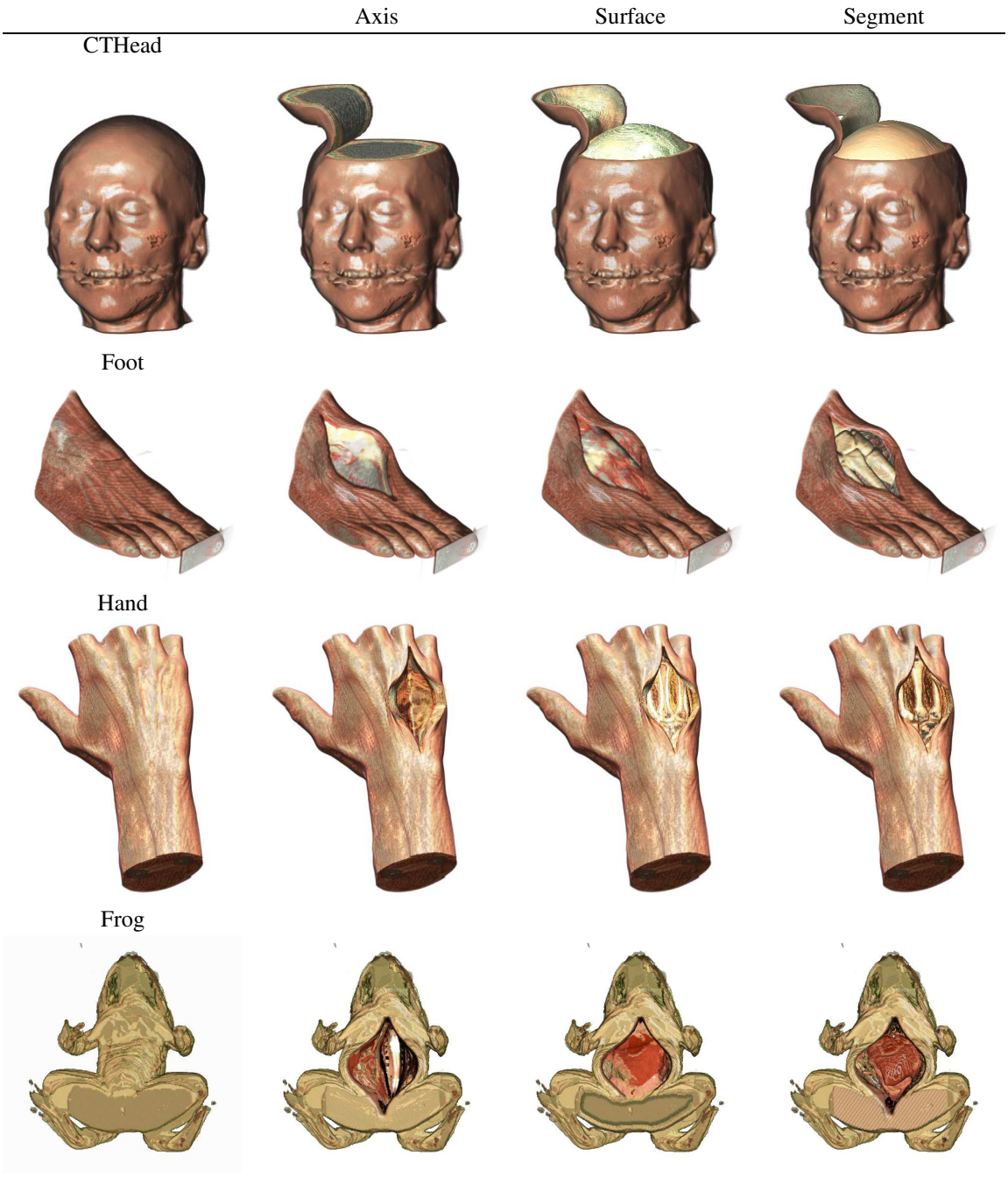


Figure 5.8: Comparative Results

Figure 5.4(c) one gets to see the underside of the peel or skin.

5.7 Constrained Deformation

Another important part of feature preservation is the ability to constrain the deformation smoothly. So far, our solution to feature preservation is to “mask” out the points that are not deformable.

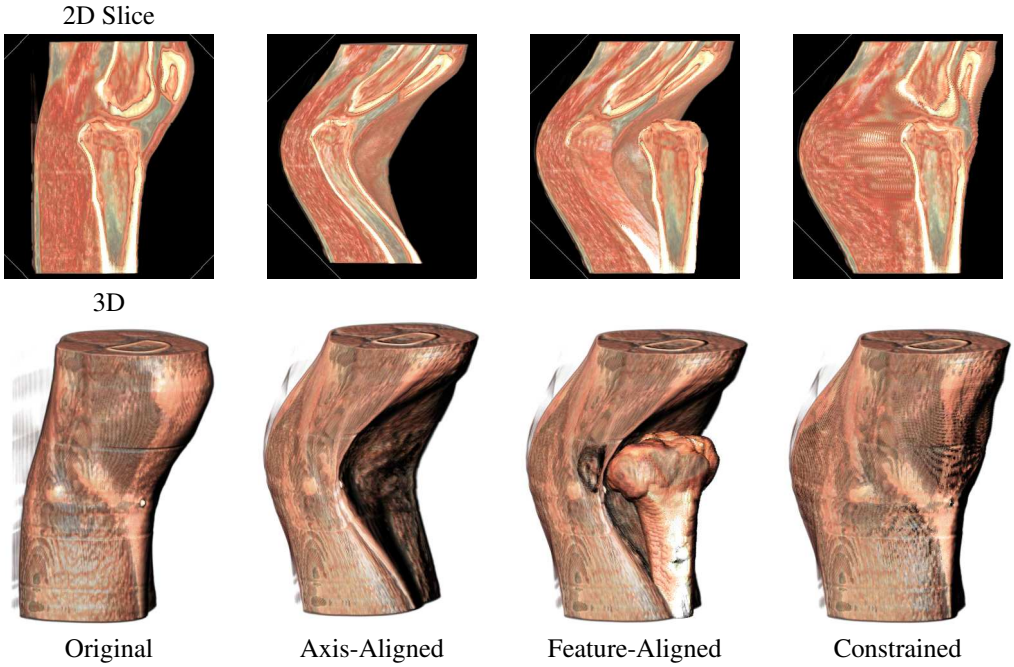


Figure 5.9: Deformation of a knee CT Scan

This method has proven to be very useful and fast. However, there are some shortcomings. One is the possibility of self-intersecting deformation due to the masking operation. To see this problem, let us consider a simple deformation applied to a CT scan of a knee (Figure 5.9). Axis-aligned deformation provides a continuous transformation, but does not preserve the rigidity of the bone structure. Feature-aligned, which can be used to model the bone tissue as rigid, does not solve the problem either, as the other non-bone tissue is deformed through the bone, giving an unrealistic transformation. A more plausible deformation *constraints* the movement of points so that self-intersection is prevented. Here, we further exploit the mask volume to modulate the displacement, as can be seen in Figure 5.9. In our method, we guarantee certain constraints by modulating the magnitude (and the direction) of the displacement with a certain scalar value. Since prevention of self-intersection is desired, this scalar value is usually defined as a distance transform to the feature of interest.

Depending on how we perform the modulation, we derive two different methods, as explained in the following sections.

Figure 5.10: Constrained Deformation of CT Head Dataset

5.7.1 Modulated Displacement

This method represents a constraint via a scalar field $C : \mathbb{R}^3 \rightarrow \mathbb{R}$. This scalar field is used to confine the displacement of a given point p within a sphere of radius $C(p)$, and we use this radius to modulate the displacement as follows:

$$\mathbf{p} = \mathbf{p}' + C(\mathbf{p}')D(\mathbf{p}') \tag{5.15}$$

In many occasions, we want to prevent self-intersection with a feature of interest, and therefore, $C(\mathbf{p}')$ is defined as a function of the distance transform: $C(\mathbf{p}') = \frac{1}{max(D(\mathbf{p}'))}DT(\mathbf{p}')$, where $DT(\mathbf{p})$ is a scalar field representing the distance field to a feature of interest. This definition has two properties:

1. Elements within the feature of interest are not deformed. This occurs since the one-sided distance field has value $C(\mathbf{p}) = 0$, and therefore the displacement is the zero vector.

2. Elements outside the feature of interest cannot be deformed through the region of interest. This happens because a point $\mathbf{p}'$ in the sampled space can only be moved within a sphere of radius r, defined as:

$$\begin{aligned} r &= \mathbf{p} - \mathbf{p}' \\ &= C(\mathbf{p}')D(\mathbf{p}') \\ &= \frac{1}{max(D(\mathbf{p}'))}DT(\mathbf{p}')D(\mathbf{p}') \\ &\leq DT(\mathbf{p}') \end{aligned}$$

We used this method to represent collision-free deformation on the knee (Figure 5.9). Figure 5.10 shows this approach applied to the deformation of the CT head. We define a constraint so that the bone tissue, defined by the skull and jaw bones, is undeformed. Note how the bone remains still while the skin, cartilage and muscle tissue deforms.

For a constrained deformation, normals can be obtained by evaluating the Jacobian of the constrained displacement. According to vector calculus [27], for a vector field G, defined as the product of vector field F and a scalar field f, $G = fF$, we can define the jacobian J_G of G using the product of derivatives as follows:

$$\mathsf{J}_G = f\mathsf{J}_F + F\nabla_f{}^\top \tag{5.16}$$

where J_F is the Jacobian of the vector field F and ∇_f is the gradient of scalar field f. Therefore, the normal estimation for constrained deformation can be defined as:

$$\overrightarrow{\mathbf{n}}^{(p')} = \left(\mathsf{I} + C\mathsf{J}_D + D\nabla_C^\top\right)^\top_{(p')} \overrightarrow{\mathbf{n}}^{(p)} \tag{5.17}$$

This can be rewritten as:

$$\begin{aligned}\overrightarrow{\mathbf{n}}^{(p')} &= (\mathsf{I} + C\mathsf{J}_D^\top + \nabla_C D^\top)\,\overrightarrow{\mathbf{n}}^{(p)} \\ &= ((1-C)\mathsf{I} + C\mathsf{I} + C\mathsf{J}_D^\top + \nabla_C D^\top)\,\overrightarrow{\mathbf{n}}^{(p)} \\ &= ((1-C)\mathsf{I} + C\mathsf{B} + \mathsf{D})\,\overrightarrow{\mathbf{n}}^{(p)}\end{aligned} \tag{5.18}$$

where $\mathsf{B} = (\mathsf{I} + \mathsf{J}_\mathsf{D})^\top$ is the precomputed matrix in Eqn.(4.12), and $\mathsf{D} = \nabla_c D^\top$.

One of the limitations with this approach is the assumption of linearity of the displacements implied by the modulation process. This may cause unrealistic deformations for nonlinear displacements, such as twisting, where decreasing the magnitude of a large twist angle does not represent a less pronounced twisting. For this reason, we propose a different modulation mechanism, which modulates the sample coordinates.

5.7.2 Coordinate Modulation

Some deformations, such as twisting and poking, have an interesting property: the deformation of a given region is equivalent to the deformation of another region with a different magnitude. For example, a twisting of 120° is equivalent to perform 12 times a twisting of 10°. In the twisting displacement, both twists are present, but in different regions of the 3D volume. Examples of these types of displacements are twists, bends, pokes, peels and retractors, among others. One of the advantages of this approach is that it overcomes the problems of linear constraints on non-linear deformations, as shown in the previous section. Here, the magnitude of a constrained displacement is not necessarily a scaling of the unconstrained displacement, and the direction may be different.

The constrained displacement is then defined as:

$$\mathbf{p} = \mathbf{p}' + D(T(\mathbf{p})) \tag{5.19}$$

for $T(\mathbf{p})$ a coordinate transformation.

To understand the nature of T, let us consider the twist displacement. A twist around the Z axis can be defined as follows:

$$\begin{aligned} D(x,y,z) &= \begin{pmatrix} (x-0.5)\cos\alpha(z) + (y-0.5)\sin\alpha(z) - (x-0.5) \\ -(x-0.5)\sin\alpha(z) + (y-0.5)\cos\alpha(z) - (y-0.5) \\ 0 \end{pmatrix} \\ \alpha(z) &= 2\pi z \end{aligned} \tag{5.20}$$

A twisting of $\sigma\alpha$ radians is equivalent to sampling the displacement map at $(x,y,\sigma z)$. Therefore, we can define the constraint transformation as:

$$T(x,y,z) = (x,y,DT(x,y,z)z) \tag{5.21}$$

where $DT(x,y,z)$ is the same scalar function as before, which represents the distance field to a feature of interest.

Normal estimation is also simpler, as it can be done by concatenating the transpose of the Jacobians:

$$J = I + J_D(T(\mathbf{p}))J_T(\mathbf{p}) \tag{5.22}$$

For the case of the twisting constraint in Eq.(5.21), the Jacobian is defined as:

$$J_T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{\partial DT}{\partial x}z & \frac{\partial DT}{\partial y}z & DT(x,y,z) + \frac{\partial DT}{\partial z}z \end{bmatrix} \tag{5.23}$$

One of the advantages of this approach is that it works for cuts as well, as there is no need for specifying additional modulation for the alpha mask. It also enables the introduction of more complex constraints such as rigid movement of features. Let us consider the twisting example, where the constraint is defined along the z-axis. Let us define the modulation function DT as follows: for $z \leq 0.5$, it should be defined as the distance along the z-axis. For $z > 0.5$ it should behave as a rigid component, so that the displacement should be the same for all points

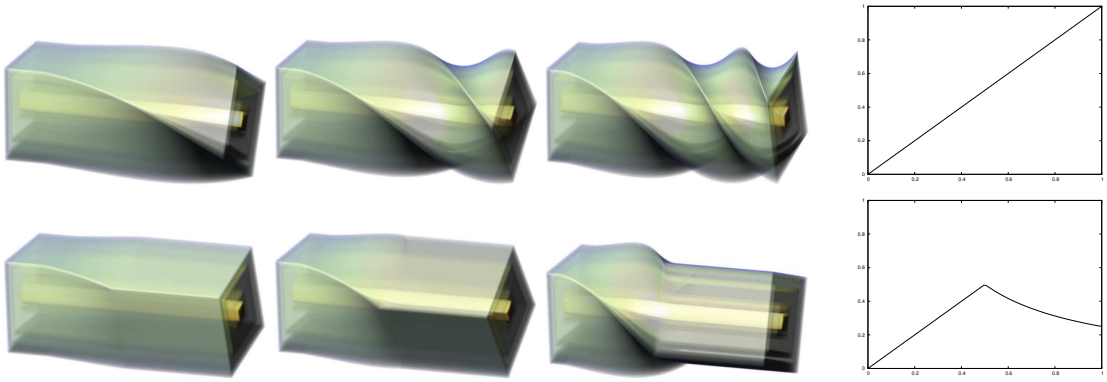


Figure 5.11: Constrained Deformation of Bar Dataset

in that region. For a point **p** in that region, we want $D(T(x,y,z)) = D(T(x,y,0.5))$. Therefore:

$$\begin{aligned} D(x,y,DT(x,y,z)z) &= D(x,y,DT(x,y,0.5)0.5) \\ DT(x,y,z)z &= 0.5^2 \\ DT(x,y,z) &= \frac{0.5^2}{z} \end{aligned}$$

Figure 5.11 shows a number of twisting operators applied to the bar dataset. For the first row, the modulation is defined as the distance along the z direction, which is equivalent to defining an *anchor* at one end of the bar. For the second row, we define an entire region as rigid, as described above.

By considering more complex modulation functions, it is possible to express complex rigidity constraints, such as kinematic chains of bones. This can be done by modulating several distance transforms for the different rigid structures, in a similar way as it was proposed by Little et al. for 2D images [71].

5.8 Chapter Summary

One of the characteristics of illustrative deformation is the preservation of certain features of interest. In those cases, the deformation is aligned with a certain feature so that both the deformed part and the undeformed part provide useful information of the concept or object being represented. In this chapter, we presented a number of mechanisms for achieving deformation alignment, classified into two categories. The first one, called *transformation-based* methods, achieve alignment by transforming the displacement space into object space so that it follows

a particular line, curve or surface. Examples of these methods are affine transformations, bilinear quadrilateral mapping and, in general, any other alignment via finite element interpolation. We presented a generic mechanism for specifying the transformation and its inverse based on the rendering pipeline presented in the previous chapter. One of the shortcomings of these approaches is that feature alignment is obtained through an approximation of the feature by a line, curve or surface. In most cases, perfect alignment would require a large number of elements. As an alternative, we devised another group of methods, collectively referred to as *mask-based* methods. These methods use a volumetric mask to specify the feature to be preserved. The deformation applied to the volume is then modified so that points in the mask are not transformed. Depending on how the mask is defined, we derived two mask-based methods. One, called *surface* alignment, is obtained by specifying the mask as a function of distance to a surface. This has proven to be a good approximation of certain features when no segmentation information is available. The second, called *segment* alignment is obtained by specifying the mask as a feature obtained through segmentation. We presented several methods for estimating accurately the normals in the deformed volume space and for adjusting the normals in the vicinity of cuts. We provided a GPU-based implementation that renders feature-aligned deformation in real-time with a quality comparable with that obtained using non-interactive raycasting methods. We also showed how feature alignment may not be applicable for certain types of deformation, where it is more important that tissues transform in a plausible way, without collisions or self-intersection. We defined a method for constraining the displacement using modulation, whereby the magnitude of the displacement is controlled in such a way that points are guaranteed to never cross the boundary of features of interest. Through a number of examples, we have shown the interactivity and operatability of these methods.

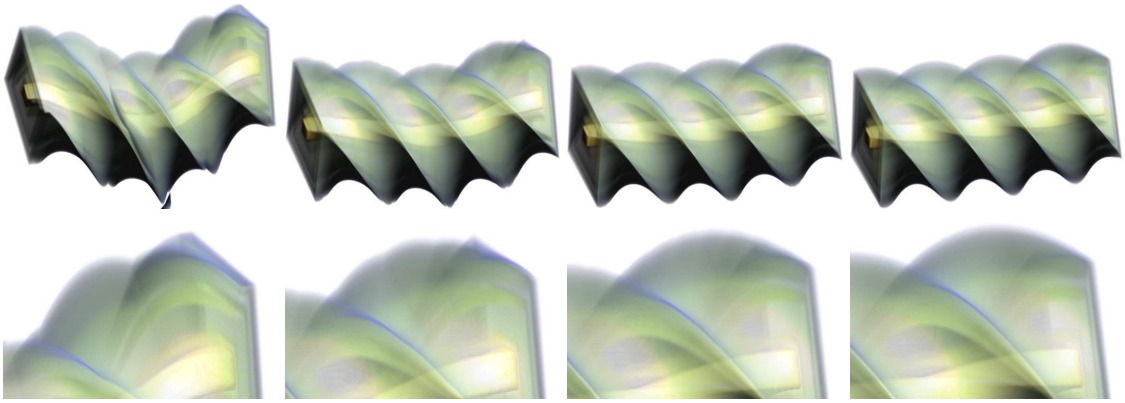
Chapter 6
Evaluation

6.1 Introduction

The previous chapters presented a method for implementing volume deformation via a generalized concept of 3D displacement maps. We showed how it can be used to represent complex deformations and cuts. This chapter evaluates the rendering quality and performance of discontinuous displacement maps in volumetric objects. Although rendering quality might be thought of as a subjective measure, it is possible to characterize it in terms of desired properties, such as *smoothness*, *local continuity* and correct shape cues provided by lighting. In turn, this can be reduced to the analysis of several properties of the displacement map, such as resolution and precision, and of the rendering method, such as lighting and compositing. In order to provide high-quality rendering of images, we must increase the resolution of the displacement and Jacobian maps for proper lighting, which in turn carries a performance overhead. The second part of this chapter deals with the study of the rendering time for the different factors that influence quality and performance.

6.2 Quantitative Evaluation of Rendering Quality

This section describes a quantitative evaluation of the rendering quality of deformed volumes. Quality can be described in terms of the smoothness of the deformation and the smoothness of cuts. Since the deformation is sampled as displacements, a number of factors influence the smoothness of the rendered image, namely: (1) The displacement *resolution*, i.e., the spatial discretization of the displacement volume; (2) the displacement *precision*, i.e., the number of bits used for representing a displacement value; and (3) the precision of the Jacobian matrix, which is also stored as a 3D texture.



8^3 16^3 32^3 64^3

Figure 6.1: Twisting operators at different displacement resolutions

6.2.1 Displacement Resolution

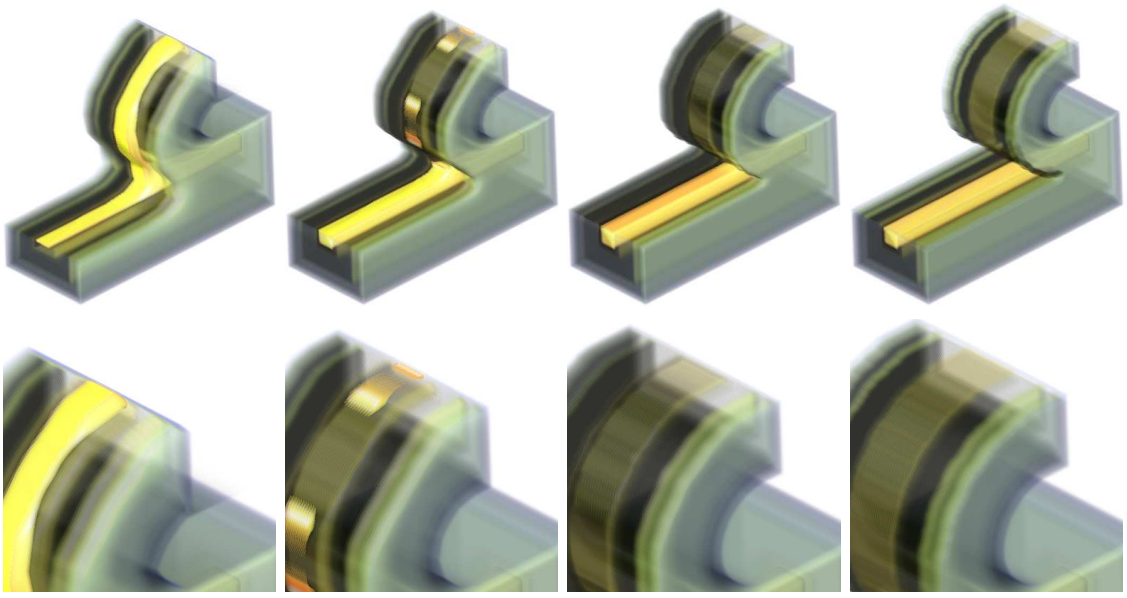
Since displacements are pre-computed and stored in 3D textures, there is a discretization error that must be taken into account. In the first case, numerical errors may be due to the spatial discretization of the displacement volume. We refer to this as the *resolution* of the displacement map, and is usually expressed in terms of the size in voxels of the displacement map. The larger the displacement map, the better the rendering quality of the deformation.

Figure 6.1 shows the result of applying a twisting operator at different resolutions, from 8^3 to 64^3, with a zoomed-in version for easy comparison. Note the dramatic difference between the first resolution and the last two. At resolutions of 64^3 or higher, the difference is not noticeable.

Figure 6.2 shows the result of varying the resolution for a discontinuous peel operator. In this case, resolutions higher than $256^2 \times 4$ are required to obtain a smooth surface of the peel. Note that the issue of resolution is more problematic in the case of cuts and breaks, since the discontinuity information, i.e., the alpha map, is sampled as well. Depending on this sampling, the thickness region necessary for proper lighting, as described in Section 3, changes, resulting in different luminance properties in the surface of the cut.

6.2.2 Displacement Precision

Errors can also be due to the roundoff error of GPU texture storage. Contemporary GPUs have a certain amount of bits for the representation of 3D textures. We refer to this as the *precision* of the displacement map. For instance, current GPUs usually store texture values as



$32^2 \times 4$ $64^2 \times 4$ $128^2 \times 4$ $256^2 \times 4$

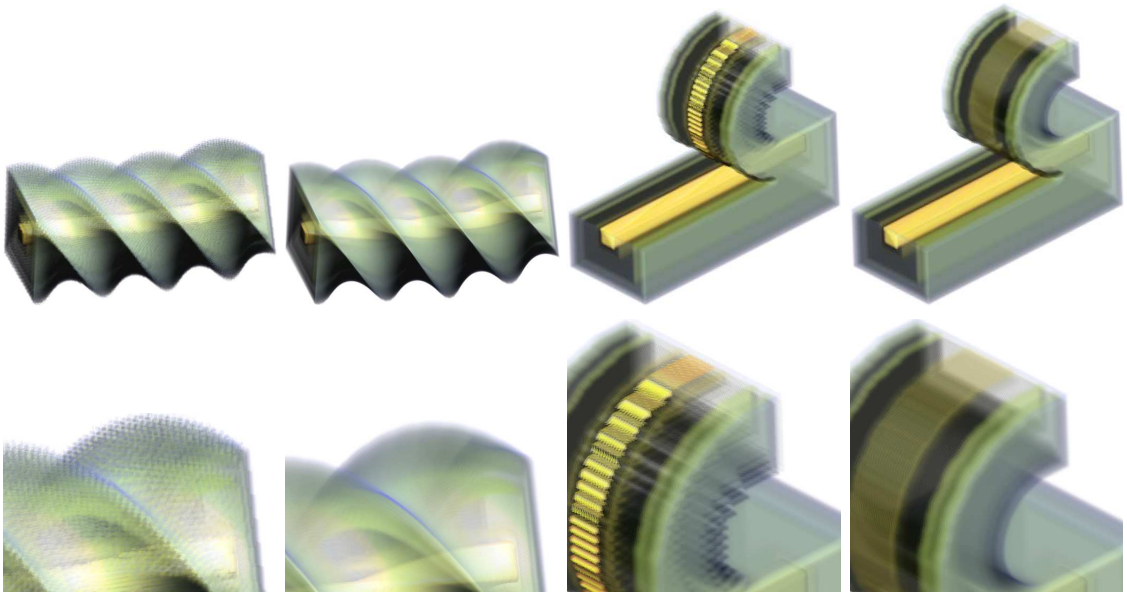
Figure 6.2: Peel operator at different displacement resolutions

32 bit quantities. If we store each component of the displacement map in an RGBA component, the resulting precision is of 8 bits per component. Alternatively, the displacement map can be stored in two separate textures, one for the XY components and another for the ZA components of the displacement, for a precision of 16 bits. A higher precision of 32 bits can be achieved by storing each component in its own texture.

Figure 6.3 shows the result of applying a twisting operator with 8-bit precision vs. 16-bit precision. Note that 8-bit precision results in a jagged deformation. In general, artifacts such as these are more distracting than those produced by low resolution but with higher precision, as the ones shown in Figure 6.1. Figure 6.3 also shows the result of applying a peel operator at 8-bit and 16-bit precision.

6.2.3 Precision of the Jacobian Matrix

As described in the previous chapters, normal information needed for lighting of volumes is obtained using the transpose of the Jacobian of the deformation. The Jacobian is a 3×3 matrix, which can be stored in 3 separate textures in the RGB channels. The maximum precision here is 10 bits per component, which can be realized using the R10G10B10A2 texture encoding. However, very small values and very large values in the Jacobian matrix are problematic. For



8*bit* (twist) 16*bit* (twist) 8*bit* (peel) 16*bit* (peel)

Figure 6.3: Continuous and discontinuous operators with different displacement precision

large values, they may not be represented properly within the encoding precision and they must be normalized. Given the Jacobian matrix B at a given point in the displacement space, the new normalized matrix B^* is:

$$\mathsf{B}^* = \frac{1}{\max_{i,j} |B_{ij}|} \mathsf{B} \tag{6.1}$$

For small values, this normalization may also improve the ability to properly encode the Jacobian. In order to evaluate the impact of the Jacobian precision, we applied a series of deformations to our test bar dataset and compared the normals estimated by this method vs. the normals estimated using 32-bit precision Jacobians. This 32-bit precision Jacobian was obtained on the fly with the maximum precision available in the GPU, instead of being encoded in a 3D texture. The deformations we applied where chosen as to represent the spectrum of Jacobian values that we might encounter in real applications. In general, the determinant of the Jacobian at a single point in the object represents the local volume change at that point. For *inverse* warping, the following conditions hold for the Jacobian matrix B:

- if $\det B = 1$, there is no local volume change. To represent this deformation we created a series of *twist* deformations, with total twisting angle increasing from 90 to 720 degrees.

- if $\det B > 1$, there is local contraction (Note that in traditional *forward* deformation, this

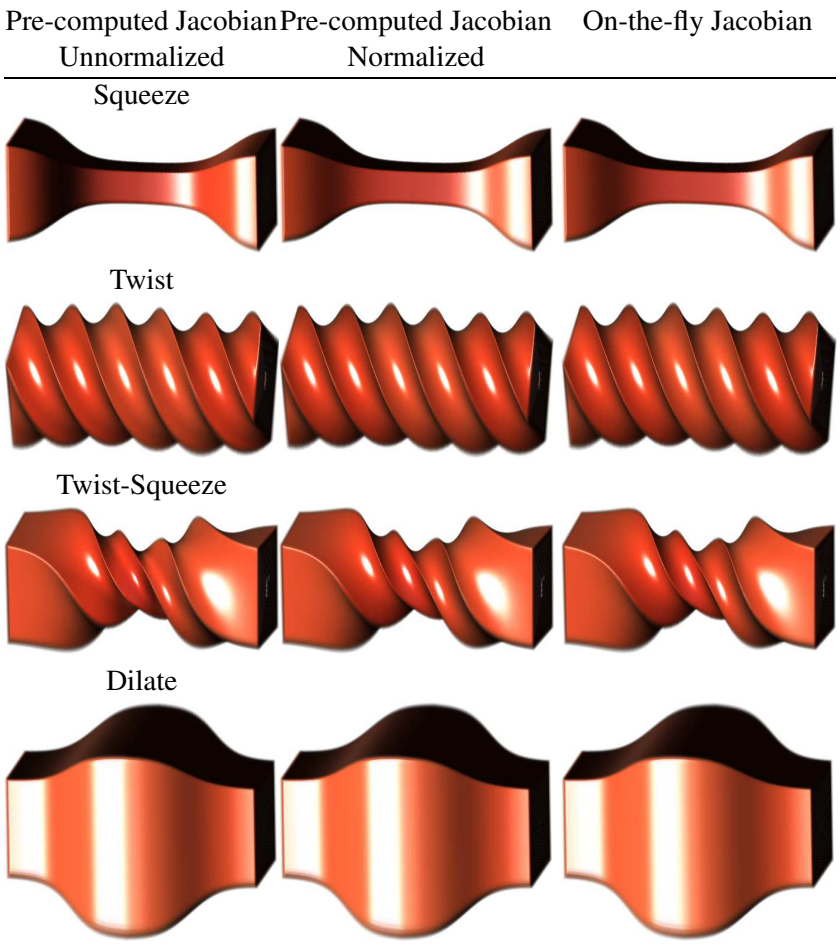


Figure 6.4: Test Volumes for analysis of the Jacobian Precision

case corresponds to local expansion, as the determinant of the inverse Jacobian is the reciprocal of the determinant of the forward Jacobian). For this case, we selected a *squeeze* deformation, with increasing contraction strength. We also selected a combination of the last two, a series of *twist and squeeze* deformations, with increasing contraction strength.

- if $\det B < 1$, there is local expansion. We selected a series of *dilate* deformations.

These test deformations are shown in Figure 6.4. Figure 6.4 also compares the resulting images obtained before and after normalization. Figure 6.5 shows the histogram of the Jacobian determinant as a reference. Incorrect Jacobian encoding (prior to normalization) results in incorrect normals. This can be appreciated in the changes in the shading of the objects, especially in the specular reflections. To measure the precision error, we compared the difference between the angles of the normals estimated with Jacobians stored as 3D textures and the normals

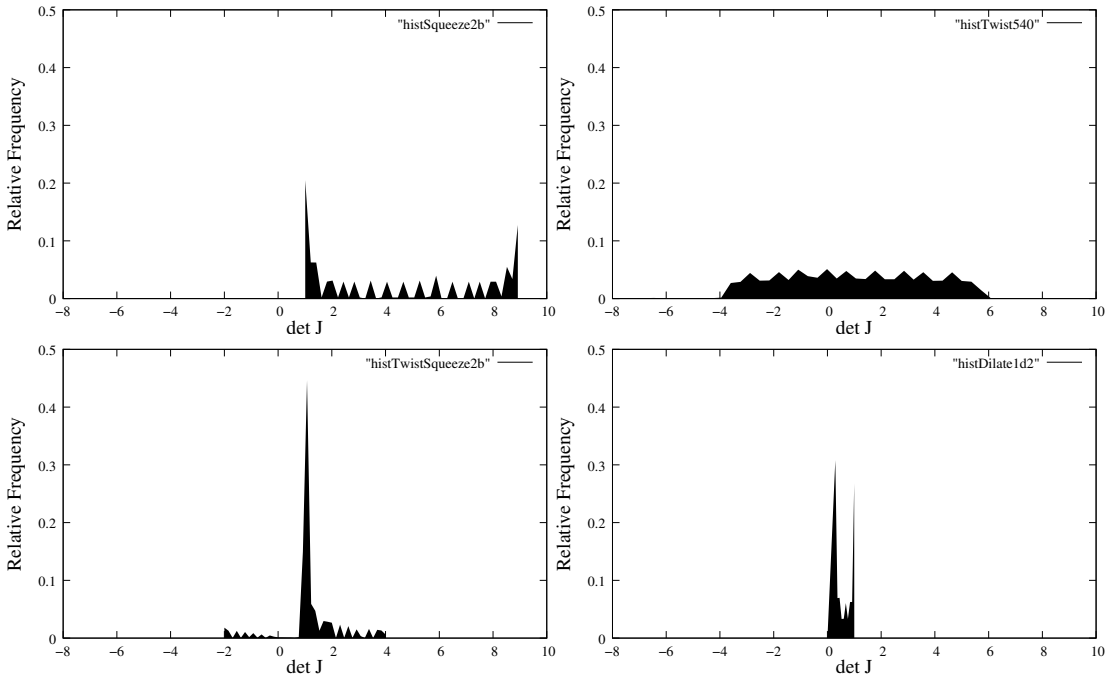


Figure 6.5: Jacobian Determinant Histogram for Test Volumes

estimated with Jacobians computed on the fly (32-bit precision). We obtained a single measure as the mean of the difference. To understand the effect on the different types of deformations, we plotted the normal angle error against the average of the determinants of the Jacobian for that deformation. This is depicted in Figure 6.6. For example, *squeeze* deformations appear spanning values in the determinant from 1 to approximately 6. In contrast, *dilate* deformations appear with determinant values less than 1. As we can see from Figure 6.6, deformations with local contraction are the most prone to errors due to the presence of large numbers in the Jacobian. After normalization, these errors drop considerably. For deformations such as twists, however, we also find large values in the Jacobian, although the average determinant is approximately 1. For expanding deformations, however, normalization does not seem to improve the error. Note, however, that the initial error is not large, due to the absence of large numbers in the Jacobian. To understand this effect, note the plot in Figure 6.7, where the normal angle error is plotted against the range of the Jacobian determinant $(max_{ij}|\det B_{ij}| - min_{ij}|\det B_{ij}|)$, instead of the average. Note how the error increases with the range for a particular deformation. Interestingly, the effect in *twist-squeeze* deformations increases slower than those for deformations with contractions.

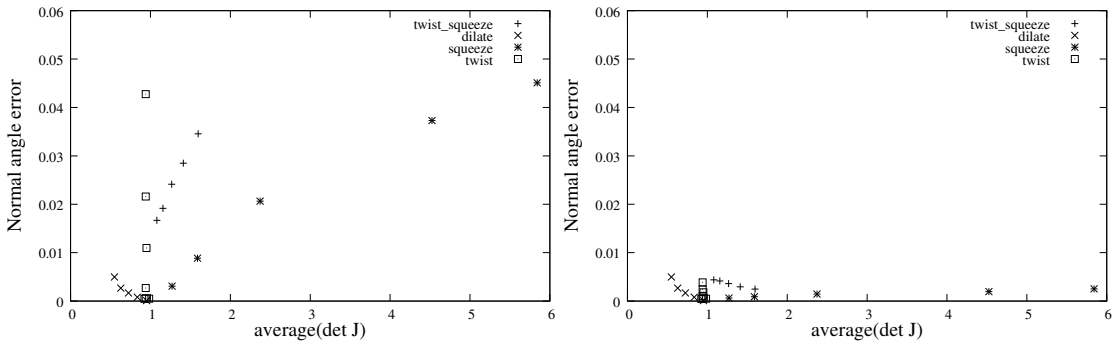


Figure 6.6: Normal Angle Error vs. average Jacobian Determinant (a) Unnormalized Jacobian Matrix using 10-bit precision (b) Normalized Jacobian Matrix using 10-bit precision.

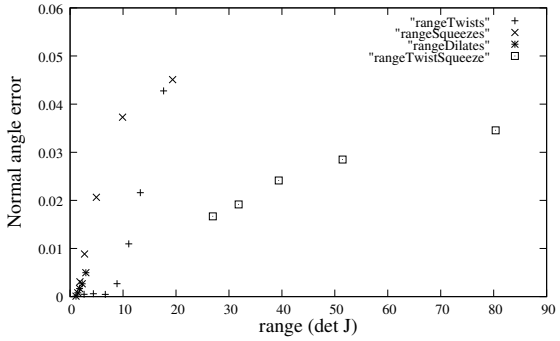
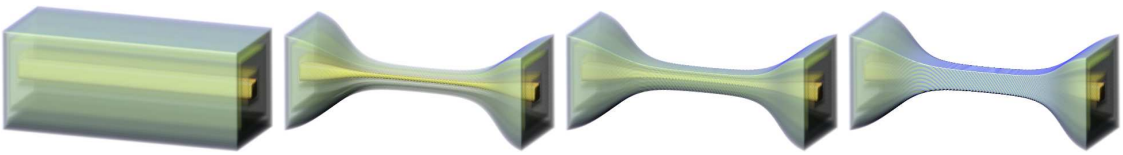


Figure 6.7: Normal Angle Error vs. Range of Jacobian Determinant

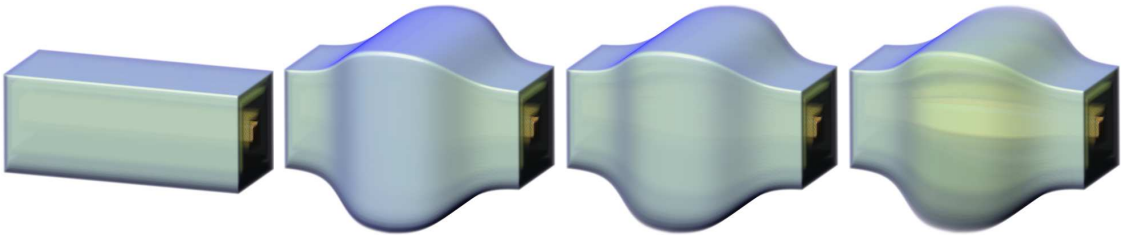
6.2.4 Transparency Adjustment

As described in Chapter 4, deformed volumes require a different sampling rate due to possible expansion or contraction of the volume. For slice-based volume renderers, where adaptive sampling is not possible or may be computationally expensive, an alternative is to adjust the transparency of the traversed voxels. Figure 6.8 show the effect of this adjustment in for a squeeze deformation. Note how the unadjusted renderer results in overly transparent voxels due to a reduced sampling. For the intensity constancy assumption, the volume is deformed so that the intensity remains essentially the same for all pixels. Compare to the original volume. For the mass conservation assumption, the region in the middle appears more opaque, since particles are assumed to be contracted and therefore there is a larger attenuation of light. In contrast, Figure 6.9 shows the effect for an expanding deformation. The opposite effects occur. No adjustment results in overly opaque regions, whereas a more physically-accurate deformation would make them more transparent, as particles are further away from each other.



Original No adjustment Intensity Constancy Mass Conservation

Figure 6.8: Transparency Adjustment for Squeeze deformation



Original No adjustment Intensity Constancy Mass Conservation

Figure 6.9: Transparency Adjustment for Dilate deformation

6.2.5 Gradient Modulated Rendering Images

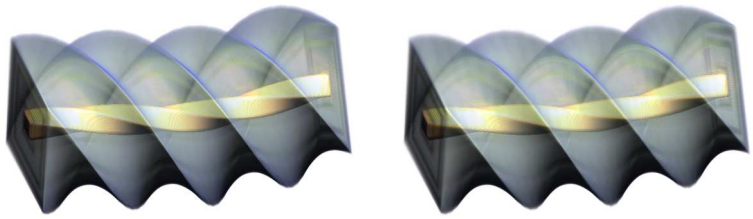
In volume rendering, it is common to modulate the opacity of a voxel by the magnitude of its gradient. When deforming a volumetric object, the magnitude of the gradient g' at a sample point $\mathbf{p}'$ is given by:

$$g' = \left| \mathsf{B}\,\overrightarrow{\mathbf{n}}^{(p)} \right| g \tag{6.2}$$

where g is the gradient magnitude in the original dataset prior to normalization, and $\overrightarrow{\mathbf{n}}^{(p)}$ is the undeformed normal and it is assumed to be of norm 1. Figure 6.10 shows the result of rendering the twisted bar using gradient modulation. Compare with the image obtained by computing the gradient on the fly.

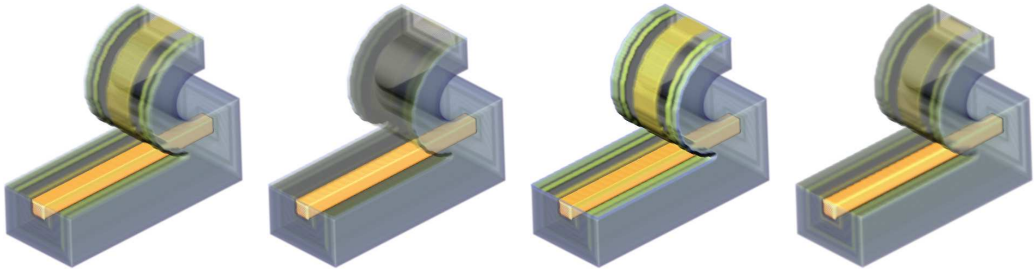
Further, when introducing cuts, a new surface appears, which changes the gradient magnitude of the points in the proximity of cuts. Similarly to the normal blending equation, we blend the gradient magnitudes near cuts:

$$g^* = \omega g' + (1-\omega) f(\mathbf{p}') \tag{6.3}$$



(a) Jacobian (b) Sobel on the fly (SotF)

Figure 6.10: Rendering a twisted bar using gradient modulation



(a) Correct weighting (b) No weighting (c) Incorrect weighting (d) SotF

Figure 6.11: Rendering a peeled bar using gradient modulation

where $\omega \in [0,1]$ is a blending factor and $f(\mathbf{p}')$ is the density value at point $\mathbf{p}'$. If no blending is performed, the resulting image may not appear to have a surface cut, since the gradient magnitude might be very small or zero. Figure 6.11 shows a gradient-modulated rendering of a peeled volumetric bar with and without blending. Note that no surface appears on the case of no blending. Compare this to the case of obtaining the gradient magnitude on the fly. In order to model holes or regions of low density, the blending equation requires the density value at the sample point $\mathbf{p}'$. Figure 6.11 also shows the case where the gradient is performed without regards of the density value, which treats the cut surface as a surface of homogeneous opacity. This may introduce occluding surfaces where there should be a hole.

6.3 Normal Estimation Validation

An important aspect introduced in this thesis is a method for estimating the normal in a deformed volume. One problem for validation is that no ground truth is available, as obtaining volumetric datasets is still a costly operation. Therefore, we lack a reference model for the deformed state of volumetric models. Our validation procedure is then obtained by two properties of accurate normal estimation. First, if the surface is smooth (as is the case for the test dataset),

and the deformation is smooth, the rendering of the deformed dataset should be smooth as well. Second, normal information, and lighting in general, is a rendering mechanism for providing hints of the shape of an object. Therefore, if the normals are estimated correctly, it should be possible to estimate the shape of the object.

For the first procedure, we compared our method with others. The methods are:

Jacobian. This is our method, and follows Barr's idea of transforming the original normal via the inverse transpose of the Jacobian of the deformation. In the previous chapter, we extended this idea to obtain similar transformations for inverse mapping, as shown in Eq.(4.11).

Central Differences/Sobel on the fly. This is the prevalent method in previous volume deformation approaches. This method computes the normal on the fly, by sampling the neighbors of a point and approximating the gradient via central differences. This method is known to produce staircasing effects due to sampling. In addition, to reduce noise, it is common to apply a smoothing operator prior to differentiation. Since this may be prohibitive for computation on the fly, this can be done by applying central differences on a pre-smoothed dataset. The first method without smoothing is here referred to as *central differences on the fly (CotF)*, and the second method as *Sobel on the fly (SotF)*. Although being prevalent, it is very costly for volume deformation, as there is need to perform up to 6 extra warpings.

For a deformed volume, this method performs finite differencing on the deformed samples:

$$
\begin{aligned}
\overrightarrow{\mathbf{n}}(\mathbf{p}') &\approx \nabla f' \\
&= \begin{pmatrix} \frac{\partial f'}{\partial x} \\ \frac{\partial f'}{\partial y} \\ \frac{\partial f'}{\partial z} \end{pmatrix} \\
&\approx \begin{pmatrix} \frac{f'(\mathbf{p}+\delta_\mathbf{x})-f'(\mathbf{p}-\delta_\mathbf{x})}{2|\delta_\mathbf{x}|} \\ \frac{f'(\mathbf{p}+\delta_\mathbf{y})-f'(\mathbf{p}-\delta_\mathbf{y})}{2|\delta_\mathbf{y}|} \\ \frac{f'(\mathbf{p}+\delta_\mathbf{z})-f'(\mathbf{p}-\delta_\mathbf{z})}{2|\delta_\mathbf{z}|} \end{pmatrix} \\
&= \begin{pmatrix} \frac{f(T_B(\mathbf{p}+\delta_\mathbf{x}))-f(T_B(\mathbf{p}-\delta_\mathbf{x}))}{2|\delta_\mathbf{x}|} \\ \frac{f(T_B(\mathbf{p}+\delta_\mathbf{y}))-f(T_B(\mathbf{p}-\delta_\mathbf{y}))}{2|\delta_\mathbf{y}|} \\ \frac{f(T_B(\mathbf{p}+\delta_\mathbf{z}))-f(T_B(\mathbf{p}-\delta_\mathbf{z}))}{2|\delta_\mathbf{z}|} \end{pmatrix}
\end{aligned}
$$

where T_B is the inverse displacement.

DotProduct. A third method is to approximate directly the diffuse and specular components of a point by approximating the directional derivative in the direction of the light and the view vector, respectively. This is the approach used by Westermann et al. [126], which computes the diffuse intensity of a point as:

$$I_d = \overrightarrow{l} \cdot \nabla f \tag{6.4}$$

$$= \frac{df}{ds} \tag{6.5}$$

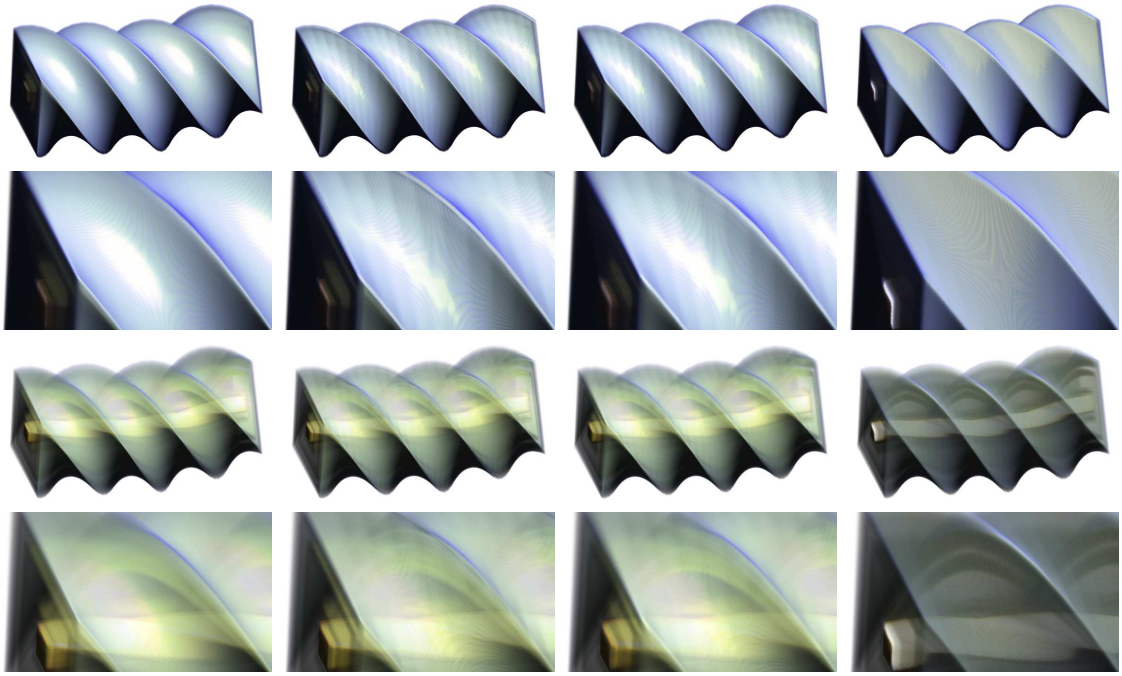
where s is a parameterization of rays in the light direction $\overrightarrow{l}$, such that a point $\mathbf{p}$ is defined as $\mathbf{p} = \mathbf{p_e} + s\overrightarrow{l}$. The directional derivative of f can be approximated using central differences on the deformed volume, which requires up to 2 extra warpings.

Figure 6.12 compares these methods for a twisting deformation on our test. It can be seen from the figure that our method provides the smoothest results. This is especially noticeable in the first two rows, which shows an isosurface of interest (as opposed to transparent rendering in the third and fourth rows). Note the highlight due to specular reflection. For central differences, it presents staircasing effects.

6.3.1 Shape Estimation

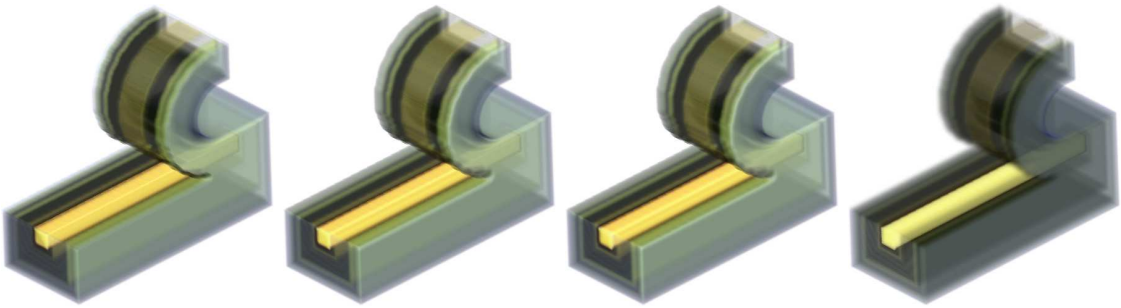
One of the difficulties of validating our approach is the lack of ground truth models. Ideally, we would have a volumetric object, say a physical bar as depicted throughout this Chapter, made of an elastic or plastic material, and have it deformed in several ways and scanned in into a volumetric representation. Because of the difficulty, cost and time constraints of MRI and CT scanning, this is not a viable solution. As an alternative, we focus into one aspect of the rendering of volumetric object, which is proper lighting. When lighting is properly simulated, shading gives shape cues that may not be available otherwise.

For this reason, we validate the normal estimation through shape estimation. Normals can be used to estimate the shape of an object. Therefore, accuracy in the normal estimation can be obtained by measuring the accuracy of the shape estimation. Estimation of depth from normal information has been used for shape-from-shading. In shape-from-shading problems, the



Jacobian CDiff on the fly Sobel on the fly DotProduct

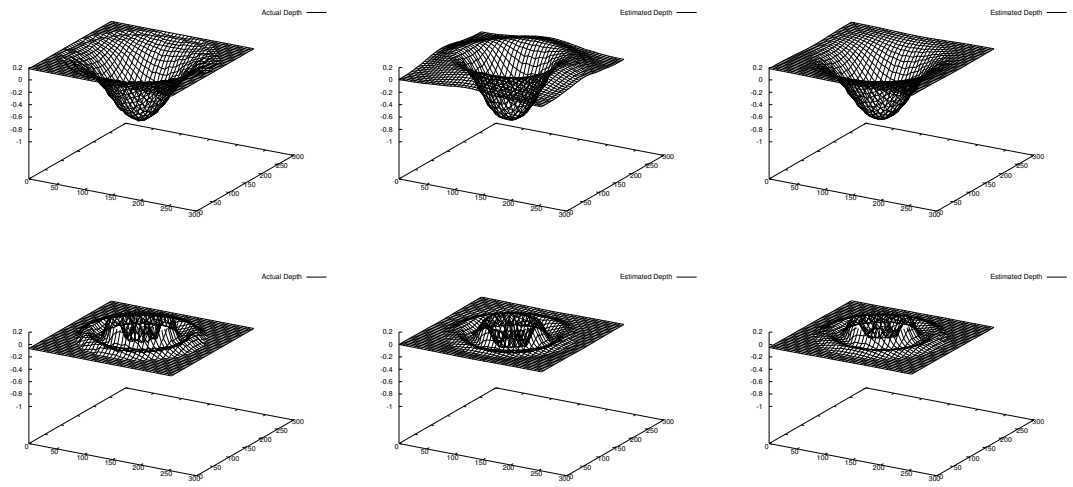
Figure 6.12: Comparison of lighting techniques for a continuous deformation



Jacobian CDiff on the fly Sobel on the fly DotProduct

Figure 6.13: Comparison of lighting techniques for discontinuous deformation

normal information is obtained first using images of the same object under different lighting conditions. The shape of the object, expressed as depth values, is obtained by integrating the normal information. In our case, since we already have the normal information, it suffices to use Horn's gradient integration to obtain the estimated depth values [51]. Although gradient estimation has been recognized as an ill-posed problem, this method is suitable for our experimentation since we deal with synthetically estimated normals rather than noisy data, and the smoothness of the displacement operators is known. For our validation, we used a volumetric cube and all our displacements are assumed to be C^1 continuous.



Actual Depth Estimated (1000 iters) Estimated (100000 iters)

Figure 6.14: Normal validation using Gradient Integration for Poke and Wave displacements

The validation procedure is then as follows:

1. Once a deformed volume is rendered, we store the estimated normals in an image I_N.

2. At the same time, we store the depth values of the deformed volume in image I_D. This image can be obtained by reading the depth buffer.

3. We perform gradient integration on image I_N. The result of this process is a depth image $\hat{I}_D$.

4. The error of the normal estimation is quantified as the error in the depth estimation, i.e., the difference $\left|\hat{I}_D - I_D\right|$.

Figure 6.14 shows the estimated depth from our computed normals for a poke and a wave deformation. Note that the depth converges to the actual depth of the deformed object, although the rate is faster for the poke operator. This occurs as the wave operator has a higher spatial frequency which demands for a larger resolution image.

Figure 6.15 shows the accuracy error of our normal estimation method, relative to the number of integration steps, for a number of displacements. As expected, the error decreases with number of integration steps, as the shape obtained from gradient integration converges to the actual shape of the deformed object.

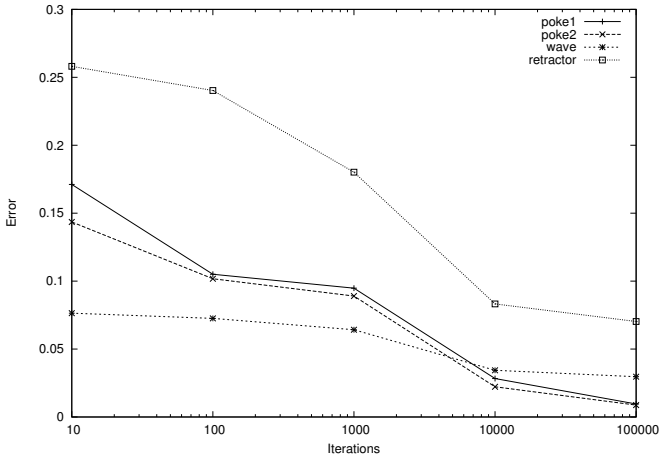
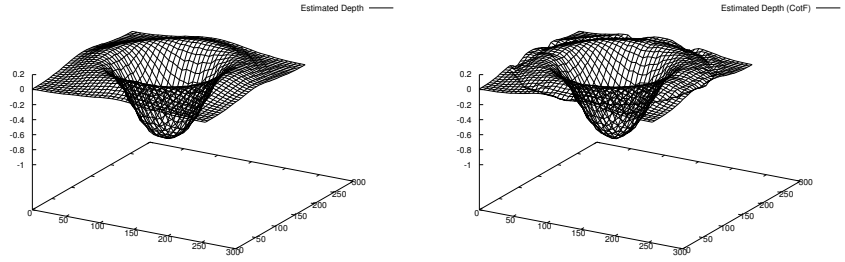


Figure 6.15: Error of shape estimation relative to the actual shape obtained from a depth map



Jacobian Central Differences

Figure 6.16: Estimated shape for two different normal estimation methods for a poke operator

This method also lets us compare the smoothness of the rendered images for different normal estimation algorithms, independently of the lighting and compositing conditions. For instance, estimating the normals via central differences results in staircasing effects, which are clearly visible from the estimated shape depicted in Figure 6.16. This comparison avoids the need for re-computing a deformed volume with density and normal information via sampling, which can be very large depending on the sampling requirements. However, it must be noted that the estimated shape via gradient integration is an approximation of the real shape and error is expected. Further, staircasing effects from central differences are expected to be smoothed out as more integration steps are performed. This condition also can be used for validation, as our method and central differences are expected to converge to the same surface. Experimentation showed that normal estimation via the Jacobian provides smoother surfaces than central differences, and it provides accurate normals for shape estimation.

Displacement	Resolution	Size in KB
Peel	$128 \times 128 \times 1$	320
Slicing	$128 \times 128 \times 1$	320
Extrusion	$32 \times 32 \times 32$	640
Split	$64 \times 64 \times 64$	5120

Table 6.1: Size in voxels of the displacement textures and texture memory requirement in total

6.4 Performance Evaluation

6.4.1 Memory Requirements

One aspect that affects the performance is the texture memory size and bandwidth. Volumetric displacement mapping requires the storage of the x, y, z components of the displacement. Although they can be stored in a single texture using 8 bits for each component, this precision is usually low for a smooth deformation, and results in visible jagged lines. In these cases, a 16-bit displacement can be used which would require at least two 3D textures. The first texture `DISPXY` stores the x and y components of the displacement as the luminance and alpha components, while the second texture `DISPZA` stores the z component and the α value (for discontinuities). This requirement does not pose a scalability problem in practice, since the use of general displacement maps allows the creation of complex cuts and deformations with relatively small 3D textures. Table 6.1 shows the size of the displacement textures used in this thesis, and the total amount of texture memory required, which includes the storage of pre-computed Jacobians. Note that they are well within the limits of current GPU technology.

6.4.2 Rendering Speed

A number of factors affect the rendering speed of our approach. Our rendering speed can be quantified depending on the number of texture lookups that must be performed and the relative complexity of the procedures that must be performed per pixel during the volume rendering stage. We obtained the results on a Pentium XEON 2.8 Ghz PC with 4096 MB RAM, equipped with a Quadro FX 4400 (512MB), with a viewport of size 512×512. We used a sampling distance of 1 for the rendering of the volumes, meaning that at least one sample per voxel was used for rendering.

Relative Size of Image. Volume rendering performance is primarily pixel-bound, the

relative size of the effective image with respect to the screen image is a considerable factor of performance. Although this is applicable for all rendering systems, it gives insight on how performance is degraded for a complex pixel shaders as ours, where deformation is computed. We use the *bar* dataset of size 128^3 voxels. To obtained the results, we disabled the lighting of the volume. Figure 6.17 plots the zoom level of the image vs. rendering time in milliseconds for two different displacements. Note how the performance improves exponentially with the zoom level.

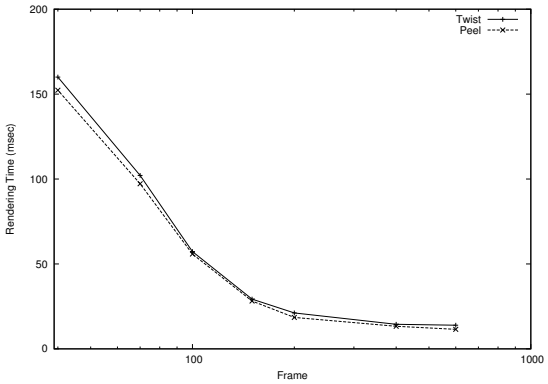
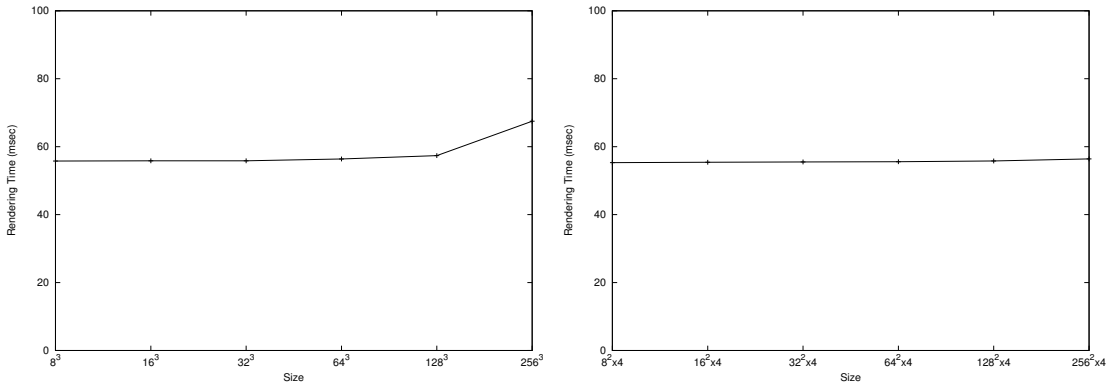


Figure 6.17: Plot of rendering performance (msec.) for relative size of image, in terms of zoom level (the larger the zoom level, the smaller the image)

Resolution of the displacement map. Since the resolution does not change the number of texture lookups, this usually does not affect the rendering speed. However, as the size increases, texture memory begins to fill quickly and a penalty due to texture memory swap may occur in some cases. A displacement map needs to be stored using two 32-bit textures when using 16-precision, as described earlier. This requires a total memory of $MEM_{displacement} = 8n$ Bytes, where n is the size of the displacement map in voxels. In addition, storing the jacobian of the displacement requires a total size of $MEM_{jacobian} = 16n$ Bytes. In total, the requires texture memory is $MEM_{total} = 24n$. This can be appreciated in Figure 6.18. Note how the rendering time is constant for most of the cases, except for the case of a 256^3 displacement, which requires a total memory of $384MB$.

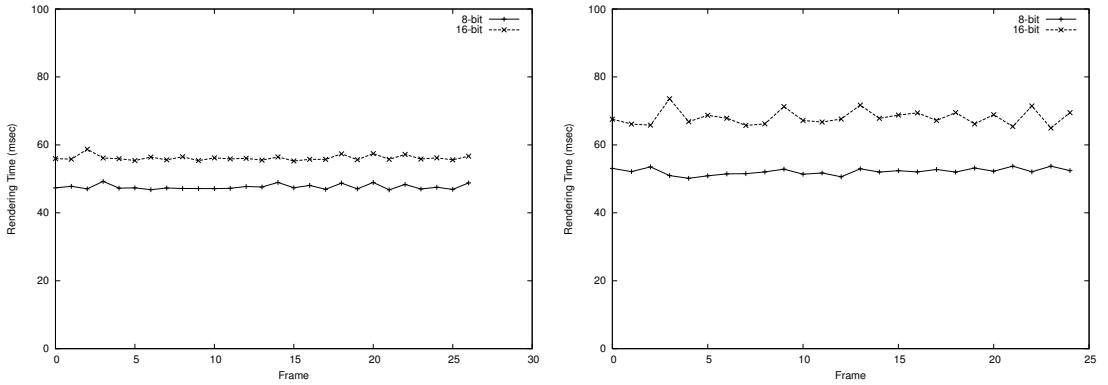
Precision of the displacement map. For 8-bit displacements, deformation can be achieved with a single texture lookup, while 16-bit displacements require two. This introduces a performance penalty, as can be seen in Figure 6.19. Due to texture capabilities of new graphics cards, the overhead time is becoming smaller. This is essential, as 8-bit displacements were shown to



Twist Peel

Figure 6.18: Resolution vs. Rendering Time for two displacements. Resolution is given in terms of size of displacement maps in voxels

be unacceptable for rendering of volume deformations. We can also see the effect of increased texture memory for the case of a twist of size 256^3. Because 8-bit rendering requires half the texture memory, the penalty of increasing the precision to 16-bit is noticeable.



Twist (64^3) Twist (256^3)

Figure 6.19: Precision vs. Rendering Time for two displacements.

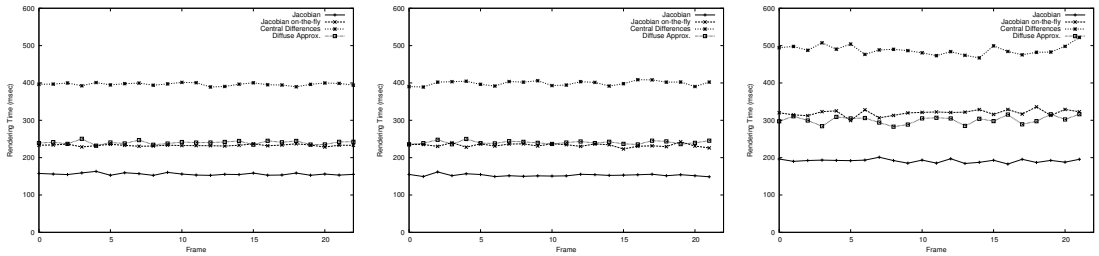
Lighting Computation. Lighting computation is an essential part of our framework. As described before, several methods have been proposed earlier, most of which require the on-the-fly computation of normals using finite differences. Since this computation requires the deformation of each point's neighbors, this can be computationally expensive, as it requires more texture lookups. The number of texture lookups can be reduced by approximating directly the diffuse component, as proposed by Westermann and Rezk-Salama [126]. In addition, it may be beneficial to compute the Jacobian of the displacement on the fly. The number of texture lookups required for each case are described in table 6.2, assuming 16-bit displacements. The

relative penalty for the extra texture lookups is basically confirmed by the results shown in Figure 6.20. It can be seen from table 6.2 that approximating the diffuse component is a viable

Method	Texture Lookups	Num. Texture Lookups	Overhead Factor
Unlit (base case)	1 (value) + 2 (deformation)	3	1
Jacobian	3 (base) + 4 (jacobian) + 1 (gradient)	8	2.67
Jacobian on the fly	3 (base) + 6 (neighbors) + 1 (gradient)	10	3.34
Central Differences	3 (base) + 6 $\times$ 3 (neighbors, each value and displacement)	21	7
Diffuse Component	3 (base) + 2 $\times$ 3 (neighbors, each value and displacement)	9	3

Table 6.2: Number of texture lookups for different lighting methods

alternative for lighting, in terms of performance. However, this approach does not yield the same results in terms of quality, as can be appreciated in Figure 6.12.

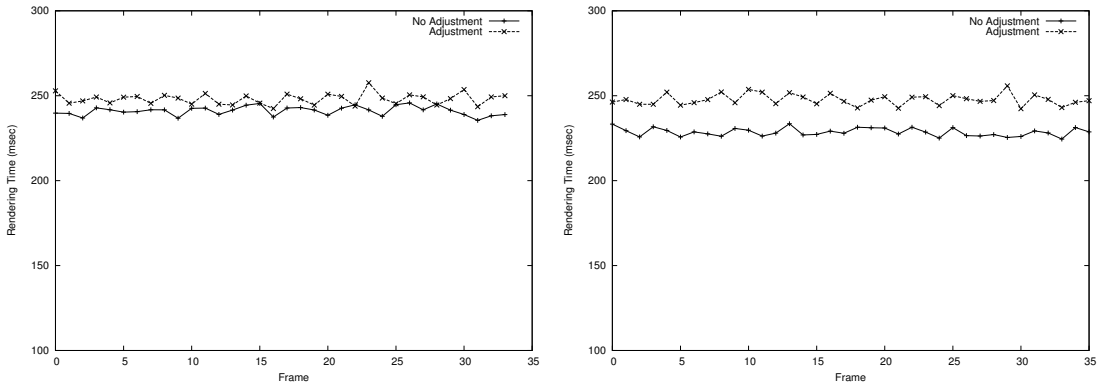


Twist (64^3) Peel ($64^2 \times 2$) Twist (256^3)

Figure 6.20: Lighting Method vs. Rendering Time for three displacements.

Transparency Adjustment. As described in the previous Chapter, deformation of volumes implies a change in the sampling rate, which can be appreciated in the transparency of voxels when using semi-transparent volumes. In order to account for homogeneity in the sampling rate, we must adjust the transparency of voxels. This in turn, implies a change using the Jacobian matrix. Figure 6.21 shows the performance cost of adding transparency adjustment. It can be seen that performance is not considerably degraded, which can be explained due to the nature of this process, which does not require extra texture lookups, but merely a matrix-vector multiplication and a reciprocal square root computation. In order to measure this factor, we enabled lighting of the volume, using the Jacobian method, since this requires the Jacobian matrix.

Feature Alignment. We simulated a number of deformations applied to different datasets for the different types of alignments described in Chapter 5. Table 8.1 shows the results obtained with our test datasets for the different alignment cases.



Squeeze Dilate

Figure 6.21: Transparency Adjustment vs. Rendering Time for two displacements.

Alignment	Dataset	Resolution	fps
Axis	foot	$143 \times 256 \times 183$	20.44
	stag beetle	$208 \times 208 \times 123$	9.24
	visible human 2mm	$256 \times 189 \times 436$	6.27
	cthead	$256 \times 256 \times 256$	5.06
Surface	foot	$143 \times 256 \times 183$	18.31
	stag beetle	$208 \times 208 \times 123$	7.32
	orange	$256 \times 256 \times 256$	6.11
	cthead	$256 \times 256 \times 256$	3.93
Segment	foot	$143 \times 256 \times 183$	17.95
	hand	$255 \times 250 \times 155$	5.54
	frog	$250 \times 235 \times 68$	8.08

Table 6.3: Performance results for different volume datasets, with $d = 1.0$ for the distance between view aligned slices of the volume

6.5 Chapter Summary

In this chapter, we have evaluated our deformation approach from a quantitative perspective. The evaluation is two-fold. On one hand, we studied the rendering quality in terms of different properties of the displacement and Jacobian maps, such as encoding precision and resolution, and of the rendering process itself, such as transparency adjustment, gradient modulation and the different lighting estimation methods. These properties in turn, reflect higher level properties of the deformation, such as smoothness and local continuity. On the other hand, we studied the performance cost of interactive deformation, in terms of the different factors. One of the conclusions we have obtained from this study is that the use of the Jacobian provides the highest rendering quality of the other methods. In order to avoid resolution problems, the Jacobian can be computed on the fly without a considerable degradation of performance, at least compared

to the prevalent alternative, which uses finite differences for estimating the normals. Displacement maps need at least a precision of 16-bit to provide smooth deformations. Although 8-bit precision is inaccurate, contemporary cards include certain texture modes that emulate 32-bit precision from lower resolution textures. This, in effect, removes the *ragged* edges that appear when using 8-bit. It does not improve resolution, though. The use of deformation also implies certain adjustments when dealing with semi-transparent volumes. One of them is the transparency adjustment due to the irregular sampling imposed by deformation. Another is the incorporation of gradient modulation. For the case of cuts and breaks, gradient modulation is intended to highlight the most salient isosurfaces of a volume. Because of the presence of cuts, new isosurfaces appear and therefore must be adjusted. This chapter provides a benchmark for future volume deformation methods.

Part II

Surface Deformation

Chapter 7
Complex Deformations and Cuts without Re-meshing

7.1 Introduction

In the previous chapters, we showed how complex deformations and cuts can be achieved for volumetric objects. Volumetric objects are commonly obtained from computed tomography (CT) or magnetic resonance imaging (MRI). In other cases, volumetric objects are the result of static or time-variant simulations. In all these cases, the object representation is a sampled volume. For the purposes of illustration, however, it has been very common to use polygonal surfaces rather than volumes to obtain high-quality renderings. In some cases, surfaces are preferred because of low memory consumption and ease of implementation. Surface rendering are not prone to aliasing effects inherent in the rendering of sampled representations. One of the shortcomings of surface representations is that a mesh topology needs to be defined explicitly. When adding large deformations or cuts, this requires a re-tessellation or re-meshing to create a finer resolution mesh [3]. Re-tessellation can be an expensive process and as the meshes grow, interactivity may be affected. Furthermore, cuts and other types of discontinuous deformation (where the mesh must be "broken") must always be re-tessellated as the cut is rarely along existing edges in the mesh. In this chapter, we extend our approach for illustrative deformation of volumetric objects to polygonal surfaces, which will enable us to produce high-quality deformations including cuts and twists without the need to re-tessellate a mesh. Example deformations are shown in Figure 7.10. The methodology works even for low resolution polygonal models. It can also produce complex deformations in real-time so that deformations can be interactively explored.

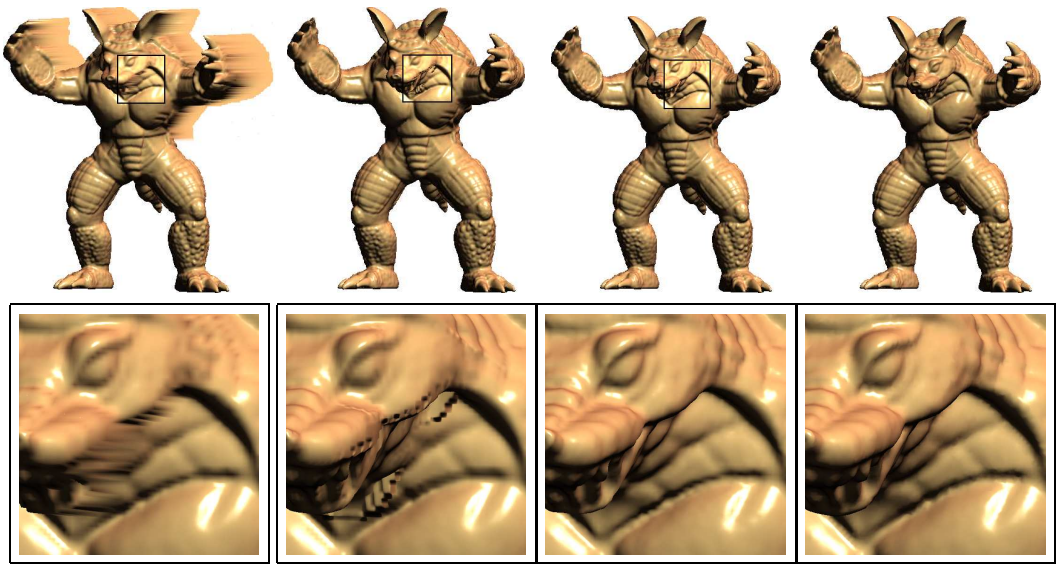
The key to our approach is to treat the areas of the object undergoing deformation as volumetric objects. Several representations have been proposed for the sampling of polygonal

surfaces into sampled structures, which we collectively called *layered representations*. Layered representations of objects have been used in the graphics community for various purposes, such as image-based rendering, displacement mapping, and multi-resolution rendering. e.g.,[102, 57, 10, 131]. A layered representation of an object is an implicit representation that describes a geometry as a set of numeric values along one or several layers, which are usually axis-oriented, such as depth maps, displacement maps and 3D signed distance fields. Here we make the tacit assumption that these are *sampled* representations, as opposed to the so-called multi-layer representations, which refer to the use of multiple nested surfaces to represent logical layers of an object, such as skin, muscle and bone. Complex deformations have not been fully explored with these representations. Some of the issues that arise include memory requirements, remeshing, aliasing and interactivity. Because surface models are explicit, rather than sampled representations, our approach needs to seamlessly integrate both representations. We achieved this with a hybrid rendering algorithm. At the core of this algorithm is a ray-casting rendering process that finds intersections of view rays with an object. This is different from the rendering algorithm used in the previous chapters for volume rendering, where a 3D dataset is sampled and *composited* at regular intervals. Deformation is applied in a similar manner through an inverse rendering process. As described in the previous chapters, interactivity is obtained by decoupling deformation from the layered representation.

This chapter further extends the idea of *Illustrative Deformation*, with (1) a unified method for handling continuous and discontinuous deformations, including complex deformations without the need for re-meshing, (2) a novel method for representing an object using a composite layered representation and a hybrid rendering algorithm which seamlessly integrates the various representations, and (3) a novel method for encoding cuts in a displacement vector field which preserves C^1 continuity of the deformation field and which allows rendering of sharp cuts free of aliasing artifacts.

7.2 Related Work

Sample Layered Representations (SLR) were introduced in Chapter 2 as implicit representations of surfaces, obtained by sampling the closest distance to the object in a 2D or 3D grid.



(a) Depth Map (b) Depth Map Cube (c) 3D Distance Field (d)Surface Mesh

Figure 7.1: Comparison of Sampled Layered Representations of the Armadillo model.

2D SLRs can be depth or height maps, which sample the closest distance to the object along a particular direction. One of the limitations of this representation is the inability to represent portions of a surface that are not visible from a point of view, as seen in Figure 7.1(a). A combination of depthmaps along the faces of a cube can be used to model the other main orientations of a surface model, as shown in Figure 7.1(b). However, concavities which are not visible from any of the faces of the cube cannot be represented. The most complete representation would be a 3D distance field, which samples the distance along a 3D grid. An example is shown in Figure 7.1(c). Compare with the rendering obtained with an explicit representation of the surface model, in Figure 7.1(d).

Deformation of layered structures. Traditional displacement mapping can be thought of as a local deformation of a base surface, which is usually defined as an inverse mapping problem from an object space, or "shell" space [91] to the displacement space. This mapping is done by sampling the space obtained by extruding prisms from each surface triangle. Rendering of complex large deformations poses additional challenges. Traditional displacement maps define displacement as an independent geometry, but, for general deformations, displacements are used to represent a change in the surface. Extrusion of triangles may not be sufficient, as a given surface primitive may undergo large deformations beyond the space defined by an extruded prism. Instead, it is easier to define arbitrary regions in 3D space where a layered

representation is sampled. Deformation of layered structures has been suggested before. Botsch and Kobbelt [10] suggest the use of "displacement volumes" to deform complex surfaces by first deforming a smooth version and then adding the detail as a normal displacement of the base surface. Smith et al. propose the use of displacement for animation of 3D models [106]. Elber [35] extends the idea of depth maps to model geometric deformations. Chen et al. propose an inverse approach for deforming light fields [17], in a similar way to the deformation of displacement-based representations.

The above, however, are designed to handle small continuous deformations. Our approach handles both large continuous and discontinuous deformation. We achieve this by encoding deformation as a displacement vector field and using an extra dimension for encoding continuity information. The idea of using vector fields for displacement was used by von Funck et al. [118] for the modeling of continuous deformation. One of the strengths of vector fields as deformation metaphors, as opposed to procedurally defined models or sketch based deformation, is the ability to store them as generic templates which can be combined algebraically to obtain more complex deformations and applied to different models. Furthermore, this can be efficiently implemented in current GPUs as textures. To the best of our knowledge, this is the first interactive method that combines cuts and breaks with other continuous deformations such as twists and bends using a single methodology and without remeshing.

7.3 Overview

Our approach works by integrating deformation into the rendering pipeline. First, a composite representation of an object is created. Let us define a 2D manifold S as the surface of a model and S_D a subset of this manifold where deformation is to take place. S_D is represented using a layered representation. Let S'_D be the desired deformed surface and $B(S'_D)$ be the bounding box of this new surface . A picture of a composite representation of an apple model is shown in Figure 7.2. A first stage of the rendering process is to render the surface $S - S_D$. This is done by discarding points of S that are within $B(S'_D)$, which can be implemented in fragment shaders using depth interval culling.

After rendering $S - S_D$, we need to render the surface S'_D. For a continuous deformation

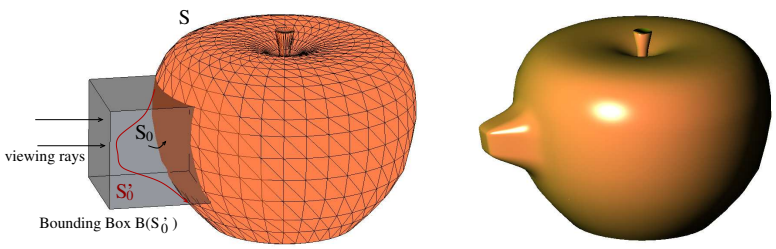


Figure 7.2: Composite Representation of an apple model. S, the original surface is rendered as a mesh. The bounding box $B(S'_D)$ is rendered as a layered representation. The resulting rendering appears to the right.

(without cuts), we do not explicitly compute the deformed surface, but we model it as a displacement vector field of the undeformed surface S_D. Rendering of this new surface is done by propagating rays through $B(S'_D)$ and deforming them by the vector field. The deformed ray is used to find the intersection point in the undeformed surface (Figure 7.3). This is an inverse problem similar to image warping, deformation of light fields [17] and ray deflectors [63, 20]. This process is summarized as follows:

1. A deformation field $D(\mathbf{p})$ is defined on the fly or re-used from a collection of deformations.

2. A composite representation of a portion S_D of the surface is created ($L(S_D)$). It could be a single depth map, a collection of oriented depth maps or a signed distance field.

3. The mesh $S - S_D$ is rendered (using a traditional renderer).

4. The deformation region is rendered, via ray casting on the geometry $B(S'_D)$. Each ray is "warped" into the layered representation to find surface intersections and their normals.

Section 4 describes in detail the process of rendering of a deformed layered representation, that is, finding the intersection with a deformed surface and estimating the normals. One important contribution of our work is a method for the rendering of cuts and breaks of layered structures. Unlike previous displacement approaches, where holes are pre-computed into the definition of a meso-structure, we apply the cuts interactively to an otherwise continuous representation of the object.

7.4 Rendering and Deformation of Layered Representations

A layered representation is an implicit representation of an object. Rendering of such a representation is performed by tracing rays along the layered region and finding its zero-crossings. Since the 3D texture representation of a signed distance field is the one with the most complete information about an object from our spectrum of representations, we will assume that we have such a representation and operations and queries, such as ray traversal and estimation of normals, are based on such a definition. Section 7.6.2 will describe how these operations can be applied to smaller representations.

Let L be a layered representation defined as a signed distance field of a surface, S_D, in R^3 sampled in a 3D grid. Each point in this grid contains the distance to the closest point on S_D. In addition, the sign of $L(\mathbf{p})$ is used to denote whether a point is in the interior or the exterior of the object. As a convention, we define $L(\mathbf{p}) < 0$ for a point outside the object and $L(\mathbf{p}) > 0$ for a point inside. L can be created in real-time for small sized regions using the algorithm by [103], where a banded distance field is used. Once the representation is obtained, finding the surface is equivalent to finding the zero-set $L(\mathbf{p}) = 0$.

Let us define a continuous deformation as a bijective transformation $T_F : R^3 \mapsto R^3$, such that a point $\mathbf{p}$ is deformed into $\mathbf{p}' = T_f(\mathbf{p})$, and its inverse transformation T_F^{-1}, such that $\mathbf{p} = T_F^{-1}(\mathbf{p}')$. Here, we assume that the inverse transformation is defined as a displacement: $\mathbf{p} = \mathbf{p}' + D(\mathbf{p}')$. The relation between $\mathbf{p}$ and $\mathbf{p}'$ is shown in Figure 7.3. The use of displacements has been explored widely and has a number of advantages which are exploited in our approach, namely: (1) are simpler to understand and operate than other deformations, (2) the inverse transformation can be obtained from forward transformations as a negative displacement, and (3) most physically-based methods have displacements as outputs.

Let us define L' as the signed distance field of the deformed surface S'_D in R^3. The deformed surface corresponds to those points $\mathbf{p}'$ in the zero-set $L'(\mathbf{p}') = 0$, which can be defined as the zero-set of the undeformed surface in a transformed coordinate system defined by T_F^{-1}. That is, the deformed surface is the zero-set:

$$L'(\mathbf{p}') = L\left(T_F^{-1}(\mathbf{p}')\right) = L(\mathbf{p}' + D(\mathbf{p}')) = 0 \tag{7.1}$$

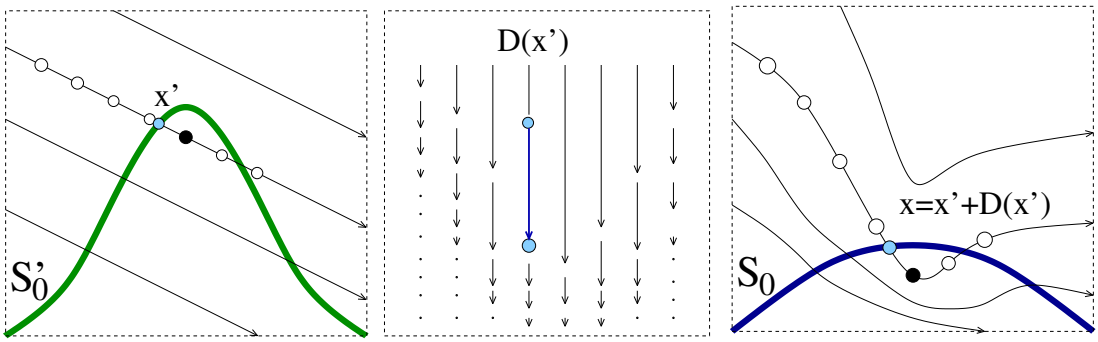


Figure 7.3: Finding an intersection of a deformed layered representation. A ray in the deformed space S'_D is transformed by a displacement field $D(\mathbf{p})$. The transformed ray maps into a curve in the undeformed space S_D and yields an intersection.

For clarity, in the reminder of this chapter, we use $\mathbf{p}$ in place of $\mathbf{p}'$.

7.4.1 Finding the Deformed Surface

The deformed surface is found by traversing rays along the region being deformed. Given a ray direction $\overrightarrow{v}$ and an initial position p, a given point along the ray is touching the surface if: $L(t) = L(\mathbf{p} + t\overrightarrow{v} + D(\mathbf{p} + t\overrightarrow{v})) = 0$ for a given parameter t along the ray. The general mechanism is depicted in Figure 7.3. The simplest way of finding an intersection is by sampling the parameter t linearly, and finding the point where a sign change in $L'(t)$ occurs. However, this has been shown to be prone to aliasing effects. As an alternative, most displacement methods use a combination of linear and binary search to narrow down to the correct intersection. Although this does not solve the problem in the general case, there are alternatives which produce close to exact solutions based on distance fields [33] or image pyramids [86]. These solutions may not suffice for extreme deformations. We present a general solution to this problem in Section 7.5.2.

7.4.2 Estimation of Normals

After finding the intersection point, it is necessary to find the normal to the surface at that point. Since it is a layered representation, the normal to the surface is determined by the gradient of L. For speed up, this can be obtained before deformation and stored as a texture. This has been widely used for depth maps, and can be extended to any of the representations in our spectrum. For a deformed surface, the new normal can be found by multiplying the original normal by the

inverse transpose of the Jacobian of the *forward* transformation function T_F [6]. From vector calculus, we have that for an inverse transformation, $J_T^{-\top}(\mathbf{p}) = J_{T^{-1}}^{\top}(T(\mathbf{p}))$ [27]. Therefore, the new normal can be found using the *transpose* of the Jacobian of the inverse transformation. For an inverse displacement, the Jacobian is defined as $J_{T^{-1}}(\mathbf{p}) = I + J_D(\mathbf{p})$, where $J_D(\mathbf{p})$ is the Jacobian of the displacement and I is the identity matrix. Therefore, normals can be obtained as:

$$\overrightarrow{n}'(\mathbf{p}) = normalize((I + J_D(\mathbf{p}))^{\top} \overrightarrow{n}(\mathbf{p} + D(\mathbf{p}))) \tag{7.2}$$

where $\overrightarrow{n}'(\mathbf{p})$ is the normal to the deformed surface at point $\mathbf{p}$ and $\overrightarrow{n}(\mathbf{p})$ is the normal to the undeformed surface at point $\mathbf{p}$. This applies only for continuous deformation. For a discontinuous deformation, such as a cut, a new surface may be created and normals are not a simple transformation of the surface before the cut (Section 4.4.2).

7.4.3 Definition of Cuts

Deformation is performed in the inverse space, therefore, cuts may be difficult to model, since they imply that the transformation T is not invertible. Modeling discontinuities as special displacement values is problematic. The cut would have to be defined as a displacement value which maps to nothing, but it would lead to singularities in the displacement field. These singularities would be seen as artifacts near the edges of cuts (due to tri-linear interpolation of displacement values). Difficulties arise when computing the normals, since the displacement field would not be differentiable. As an alternative, we add a new dimension to the displacement field. We define a cut implicitly as a signed distance function, in a similar fashion to the definition of the layered structure of the surface. Because this signed distance function is defined for all points in a given sample space, we must define the displacement in those points as well, so that the displacement function D has at least C_1 continuity.

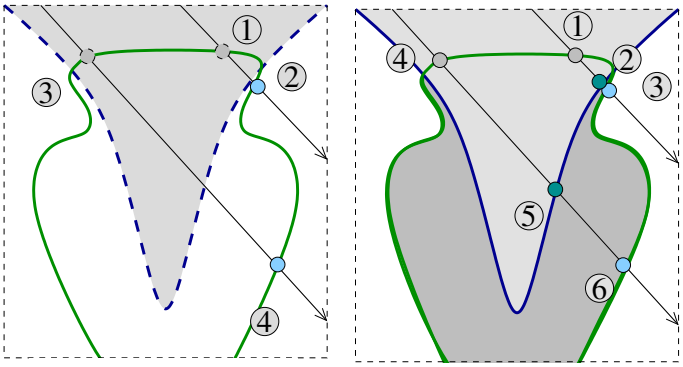
Let us define A as the implicit representation of the cut. A given point $\mathbf{p}$ in space is said to be "cut" if $A(\mathbf{p}) > 0$. The border of the empty space is a set of surfaces defined as $\{\mathbf{p}|A(\mathbf{p}) = 0\}$. Depending on whether the object is modeled as a solid or a thin surface, these surfaces may or may be not visible. In practice, this field A can be represented along with the displacement into a single structure. For a continuous deformation, A is not necessary.

7.4.4 Rendering of Cuts

Cuts on a polygonal object generally imply that the interior of the object will be visible. The interior could be part of S_D or part of $S - S_D$. $S - S_D$ will be rendered in step 2 of the process (Section 3). Note that with $B(S'_D)$ removed, the back faces of the object are rendered. Then, the layered representation is rendered creating the front cut surface. If the layered representation does not extend to the back of the object, the object would be seen as hollow, as shown in Figure 7.5(a). In this case, no intersection with the ray is found, and an empty region results. There is another case, where the layered representation encompasses a portion of the interior of the object (that is visible). For a hollow object, this can be obtained by ray intersection with the layered representation, as described in Section 4.4.1. However, we may want to model other types of objects, such as solids or partial solids which contain a thick shell. In these cases parts of the cut become the interior surface. Below we explain how to handle these different types of surfaces.

Rendering of Hollow Surfaces

For hollow surfaces, rays along the deformed region only intersect the original object, as long as they are outside of the region defined by the cut, i.e., the deformed surfaces are defined by the set of points $\{\mathbf{p}|L(\mathbf{p}+D(\mathbf{p}))=0 \wedge A(\mathbf{p})<0\}$. A naive approach to obtaining the intersections in the presence of a cut would be to compute the combined implicit representation of the deformed space as $L'(\mathbf{p})-A(\mathbf{p})=L(\mathbf{p}+D(\mathbf{p}))-A(\mathbf{p})$, and find the zero-crossings of this new representation. However, cuts are generally sharp, and chances of skipping the intersection increase considerably, which results in jagged boundaries. As an alternative, we exploit the fact that both surfaces are represented in disjoint structures and we compute intersections on each of them separately. The ray casting algorithm traverses the space looking for zero crossings of the continuous representation $L'(\mathbf{p})$. Once an intersection $\mathbf{p}$ is found, it is checked whether it belongs to the interior of the cut. If $A(\mathbf{p})>0$ then the point is discarded as a surface intersection and the search proceeds. This is shown in Figure 7.4(a). The intersections 1 and 3 are discarded and only 2 and 4 are used for rendering. Note that these intersections are on the interior or underside of the model. We can use the normals to render the interior in a different



(a) Thin object (b) Solid object

Figure 7.4: Diagram for Ray intersection of cut surface. (a) For a hollow object, we compute intersections with the object representation $L(\mathbf{p})$, discarding those inside the cut region (points 1 and 3), and stopping whenever it is outside the cut region (points 2 and 4). (b) For a solid object, we keep track of intersections with the cut region and the object. In this case, points 2 and 5 are selected and rendered.

color.

Rendering of solid objects

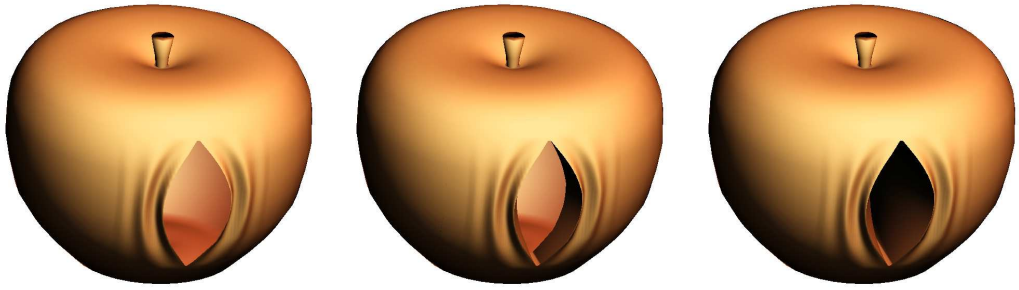
For rendering solid objects, we must compute the intersection with both the object and the cut geometry. This is equivalent to finding the intersection with the combined representation $L'(\mathbf{p}) - A(\mathbf{p})$. As described above, directly evaluating this expression may lead to artifacts. A similar algorithm is used, except that when an intersection with the object is discarded (because it is within the volume of the cut), it may be possible that it still intersects the cut geometry, in which case this needs to be computed. For this, we classify the intersection as INOUT if the ray went from the interior of an object to the exterior, and as OUTIN otherwise. This classification is obtained depending on the slope in the change of $L(\mathbf{p})$ or $A(\mathbf{p})$. The ray traversal is modified to compute two intersections: $\mathbf{p}_L$, the intersection with $L(\mathbf{p}) = 0$ and $\mathbf{p}_A$, the intersection with $A(\mathbf{p}) = 0$. If only one of these is an OUTIN intersection, the algorithm returns the corresponding point. If both intersections are OUTIN, the algorithm returns the closest of the two. If none of them are, the algorithm continues in the search of intersection. This is depicted in Figure 7.4(b). Only intersections 2 and 5 are rendered. Intersections with the cut surface have a different normal, which is obtained directly from the gradient of $A(\mathbf{p})$.

Rendering of thick objects

A combination of the above is the case where we have hollow objects with a "thick skin". In this case, there are regions where intersections with the cut are needed (the thick part of the outer shell of the object), and other regions where these are ignored . Remarkably, this can be obtained by modifying the surface representation and following the algorithm for solid objects. Let us define $\tau > 1$ as the thickness of the hollow object. Then, the new surface representation is:

$$\hat{L}(\mathbf{p}) = \begin{cases} \tau - L(\mathbf{p}) & L(\mathbf{p}) > \frac{\tau}{2} \\ L(\mathbf{p}) & \text{otherwise} \end{cases} \tag{7.3}$$

The different rendering of cuts can be seen in Figure 7.5.



(a) Hollow (b) Thick Hollow (c) Solid

Figure 7.5: Hollow, Thick and Solid Apple

7.4.5 Seamless Integration

The final step in our rendering process is to blend the composite representations in a seamless manner. In order to do this, we allow some overlapping in the boundaries of $S - S_D$, and the result from both rendering stages are alpha blended. As a condition, the deformation field $D(\mathbf{p})$ should also be blended to zero in this overlapping region to create a smooth border.

7.5 Defining Displacement Fields

Displacement fields can be defined procedurally, or they can be pre-defined as 2D or 3D displacement maps. Displacements here are obtained by sampling an inverse procedural definition

of a deformation. For example, twisting and bending can be computed by sampling the *inverse* transformation, as defined in [6]. Displacements can be collected and stored as textures. One of the advantages of the defining displacements this way is that the same "deformation metaphor" (i.e., texture) can be used for many different objects. The accompanying video shows a number of deformations applied to a number of objects. It also enables us to create complex deformations by combining simpler ones. Minimization can be used to obtain volume conserving deformations. By designing these generic templates, local self-intersection can be guaranteed as well, in a similar way as defined by Botsch and Kobbelt [10]. Global self-intersection, however, depends on the placement of the displacement fields within the object. Here, we do not address these problems, but rather describe a method for adapting the deformation to certain geometry.

7.5.1 Complex Cut Geometry

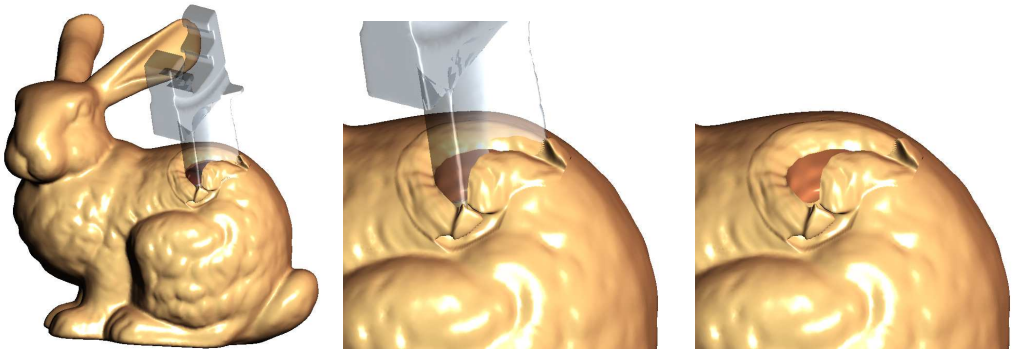
When computing cuts, the field A is usually specified along with the displacement field D. This is useful for creating simple generic cut tools which can be defined by a procedure, or sampled with very low resolutions. They can be defined procedurally, as the ripple pattern shown in Figure 7.6, where a pattern is added to the cut, as in Figure 7.6(a) or to the cut surface as in (b). One of the strengths of this method is that the implicit definition of cuts and deformations allows us to use other surfaces as the outline for the cut. They can be obtained in turn as a layered representation of another surface. The result would be a deformation or cut that follows the contour of a surface of arbitrary complexity. As an example, Figure 7.7 shows a Turbine Blade model used as a cutting tool applied to the Bunny model. To model a deformation while cutting, for instance, we can displace points along the gradient of the implicit representation of the cut object. This is done by first defining a displacement which maps points in the outer layers of an object, in an interval $[0, \omega_1]$ to another interval $[\omega_0, \omega_1]$. That is, the displacement is obtained by applying:

$$D(\mathbf{p}) = \left(\omega_0 + \frac{L_C(\mathbf{p})}{\omega_1} (\omega_1 - \omega_0) \right) \nabla_C(\mathbf{p}) \tag{7.4}$$

where $L_C(\mathbf{p})$ is the layered representation of the cut geometry, and $\nabla_C(\mathbf{p})$ its gradient. The cut information is defined as $A(\mathbf{p}) = -D_C(\mathbf{p})$. Cracks can be simulated in the model, as can be seen in Figure 7.7, by allowing discontinuities in the gradient field. Other materials can also be simulated by modifying the behavior of the gradient field.

(a) Cut with a complex geometry (b) Deformed surface with a complex geometry

Figure 7.6: Complex Cut Geometries. (a-b) Ripple cuts on the bunny model



(a) Transparent Turbine Blade (b) Zoom of the cut (c) Zoom of the view without Turbine Blade

Figure 7.7: Turbine Model (10,778 polygons) used as a cutting tool on the Bunny model (72,027 polygons)

7.5.2 Adaptive Sampling

One of the challenges with the rendering of implicit surfaces is accurately finding the first intersection with the surface. With linear search, regions of high spatial frequency might be

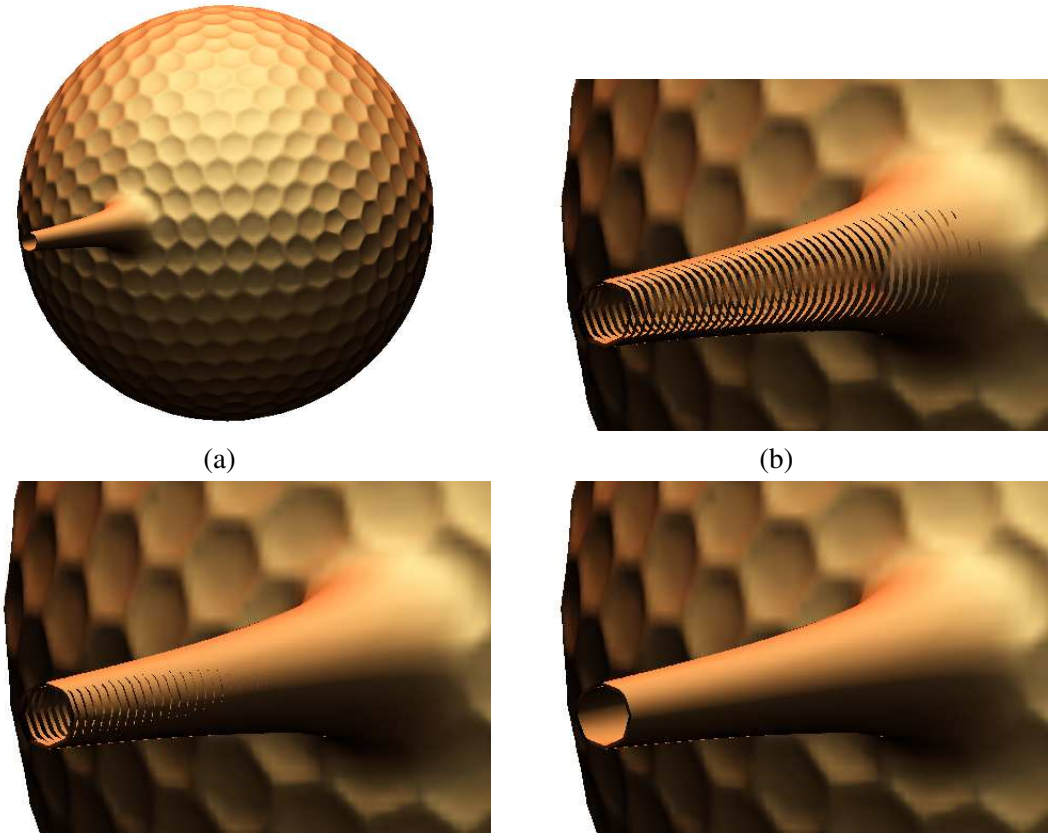
skipped, resulting in artifacts near edges. A number of solutions have been proposed. Previous approaches aiming toward ray tracing require "guaranteed" intersection-finding algorithms, based on Lipschitz constants. Hadwiger et al. use adaptive sampling based on a user-controlled threshold [46]. In our paper, we are interested in the cases where adaptive sampling is needed in the case of deformation. For this purpose, and without loss of generality, we assume that the original undeformed surface can be rendered robustly using one of the above techniques. With deformation, additional conditions must be met so that the deformed surface is rendered properly. For instance, a long narrow pull or a large twisting, may increase the required sampling of the deformed surface. Let us define a ray in deformed space $\mathbf{p}_0 + t\mathbf{v}$, where $\mathbf{v}$ is the view direction and $\mathbf{p}_0$ is the ray entry point. This ray corresponds to a 3D curve $\mathbf{R}(s)$ in undeformed space. We can consider $s = T^{-1}(t)$, for an inverse transformation T^{-1}, such that $\mathbf{R}(s) = T^{-1}(\mathbf{p}_0 + t\mathbf{v})$. To avoid missing intersections in the case of a sharp or large deformations, we ensure that samples in the deformed space correspond to uniform samples along the curve in the undeformed space. Taking the derivative of s with respect to t yields:

$$\frac{ds}{dt} = |J_{T^{-1}}\mathbf{v}|$$

where $J_{T^{-1}}$ is the Jacobian of the transformation T_F^{-1}. Since T_F^{-1} is a displacement, $J_{T^{-1}} = I + J_D$, where I is the identity matrix and J_D is the Jacobian of the displacement field. Then, assuming a constant sampling distance δs, an adaptive sampling (which we call Jacobian sampling) is obtained as:

$$\delta t_i = \frac{1}{|(\mathbf{I} + \mathbf{J}_D(t))\mathbf{v}|}\delta s \tag{7.5}$$

and points along the ray direction can be found as $\mathbf{p}_{i+1} = \mathbf{p}_i + \delta t_i \mathbf{v}$. Fig.(7.8) shows an example of a narrow pull on the golf ball model, with different sampling methods. Because adaptive sampling can be costly, we allow the programmer to define whether to use a threshold-based sampling as in [46], Jacobian sampling or no adaptive sampling at all.



(a) (b)

(c) (d)

Figure 7.8: (a) Narrow Pull over golf ball model (245K triangles). (b) Linear search with binary refinement results in missed intersections (c) Adaptive sampling based on threshold (thresh=0.04) cannot resolve all misses (d) Jacobian sampling finds the intersections properly.

7.6 Implementation Details

In our GPU-based implementation, we store layered representations as displacement maps, which can be 1D, 2D or 3D maps. Similarly, we define displacement fields as textures. We exploit the programmability of fragment shaders for our hybrid rendering pipeline. First, computing the surface $S - S_D$ can be done efficiently using render-to-texture capabilities. The entry and exit depth values of the geometry $B(S'_D)$ are obtained from the depth buffer, and used on the triangle mesh renderer to cull away fragments in that interval. Ray intersection is obtained by rendering $B(S'_D)$, assigning ray entry points as texture coordinates. Each fragment is used to trace rays along the bounding geometry and performing the necessary computations as described in the previous sections. For ray intersection, a linear search is first used to narrow the interval and followed by binary search. For cuts, this procedure needs to be repeated at possible

intersections along the interior. Because of current GPU iteration limitation (in terms of speed), A can be encoded explicitly, to skip the iterations in the interior of the cut.

7.6.1 Interactive Exploration of Deformation

We can explore the deformation interactively using two mechanisms. One mechanism is to translate or scale the displacement field within a given deformable region $B(S'_D)$. This has been used to interactively peel the face in Figure 7.9. For these deformations, it is not necessary to re-compute the displacement field, but instead, we apply a global transformation to it. This global transformation acts as a mapping from the object space to a translated, rotated or scaled displacement space. Given a space transformation $M : R^3 \mapsto R^3$ from the displacement space to the layered representation space, Eq.(7.1) can be generalized to $L'(\mathbf{p}) = L(\mathbf{p} + M(D(M^{-1}(\mathbf{p})))) = 0$. An example is shown in Figure 7.9. Another mechanism is to interactively move the deformation around the object. This method effectively transforms the bounding geometry along the deformation. Note that for this case, S_D changes, and we must obtain a new layered representation of $L(S_D)$.

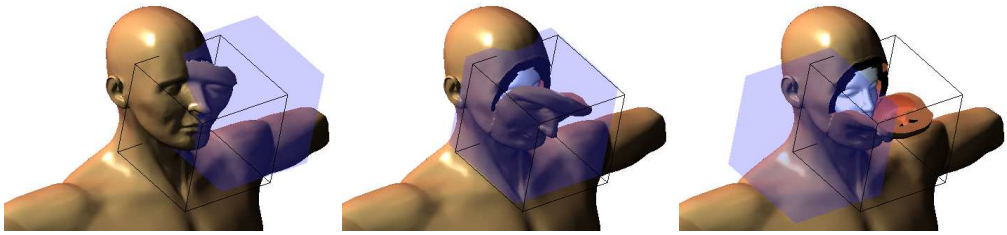


Figure 7.9: Interactive Exploration of a peel deformation. By translating the displacement space (blue box) in relation to the layered representation (wireframe box), an effect of peeling is obtained.

7.6.2 Memory Efficiency

As described above, there are certain representations which are more memory efficient than others. Furthermore, some representations are easier to obtain. For instance, depth maps can be obtained in real-time exploiting the ability of contemporary GPUs to render depth images into textures. In our case, only a small part of the object needs to be converted. It also has been shown that signed distance fields of complex objects can be obtained at interactive rates [103,

110]. Depending on the type of deformation and the region of an object where it is applied, one may choose different types of representations. Here we show how different representations can be defined in relation to a complete signed distance field.

A depth or height map, is a 2D mapping function $H : R^2 \mapsto R$, which contains depth values to the closest point in the surface along a given vector $\overrightarrow{n}$, normal to the plane where the depth map is defined. That plane is assumed to pass through a given point in space p_0. The implicit representation L can be defined as the depth of the projection of the point into the plane:

$$L(\mathbf{p}) = H(p_x) - d(\mathbf{p}) \tag{7.6}$$

where $d(\mathbf{p}) = |\overrightarrow{n} \cdot \overrightarrow{\mathbf{p}_0\mathbf{p}}|/|\overrightarrow{\mathbf{p}_0\mathbf{p}}|$ is the closest distance of $\mathbf{p}$ to the plane, and p_x is the projection of the point into the plane, defined in 2D coordinates local to that plane. It is usually represented as a 2×3 projection matrix, i.e., $p_x = P\mathbf{p}$.

For a *depth map cube*, the surface representation can be obtained as a combination of each of the representations obtained from each depth map, according to equation Eq.(7.6). Curless and Levoy [26] showed a mechanism for combining depth maps into a single representation. For the purpose of rendering, a simpler function would suffice. Given two representations L_1 and L_2, a new representation L can be found as:

$$L(\mathbf{p}) = \rho \cdot min(|L_1(\mathbf{p})|, |L_2(\mathbf{p})|) \tag{7.7}$$

where the sign ρ is 1 only for the cases where $L_1(\mathbf{p})$ and $L_2(\mathbf{p})$ are positive, i.e., when the point is in the inside of both representations, and -1, otherwise. For a depth map cube, the 3D representation is obtained by combining the depth maps from each face. A more general representation is displaced subdivision surfaces, where a number of depth maps are used to describe a complex object. The ability to compose layered representations for a single object enables us to overcome memory limitations imposed by the graphics hardware. As an example, a 3D layered representation of an object may need 8 MB of texture memory while a depth map cube of the same resolution requires only 384 KB. However, there is a visibility tradeoff, and the depth map cube can only be used for geometries which do not contain significant concavities. Using a subdivision criteria, this idea can be further extended to define the entire

object implicitly and allow complex global deformations.

7.6.3 Depth Estimation

Because the deformation region $B(S'_D)$ and the remaining geometry $S - S_D$ are disjoint, depth information within the deformation region does not need to be computed. This occurs because ray traversal is stopped whenever an intersection is found. This condition, however, cannot be met when rendering another non-deformable surface S_2, as depicted in Figures 7.10(a) and 7.7. In this case, the other surface may intersect the volume defined by $B(S'_D)$, and therefore depth information must be accurately estimated to provide correct inter-surface occlusion. One alternative is to include S_2 into the computation of the SLR of S_D. In this case, the remaining geometry is $S - S_D - S_2 \bigcap B(S'_D)$ and the layered representation of the deformation region is $L(S_D + S_2 \bigcap B(S'_D))$. However, since S_2 is not being deformed, this is usually unnecessary or may introduce additional sampling requirements. Instead, S_2 can be rendered independently. In such case, our rendering algorithm must be adapted to provide depth information of the ray intersections. The process is as follows: When rendering $B(S'_D)$, the entry and exit points, r_0 and r_1, in displacement space are passed as texture coordinates. At the same time, we obtain the z-buffer values from those points directly from the Z-buffer as d_0 and d_1. The pixel shader then computes the total traversal distance of a ray as $d_T = |d_1 - d_0|$. As the ray traverses the SLR, the traversed distance in displacement space is accumulated, to yield: $d_r = \sum \Delta t_i$ for the traversal iterations before intersection. Then, the depth of the intersection point can be computed as :

$$d = \frac{1}{(1 - \frac{d_r}{d_T})\frac{1}{d_0} + \frac{d_r}{d_T}\frac{1}{d_1}} \tag{7.8}$$

7.7 Results

Figure 7.10 shows a number of examples of our approach. In Figure 7.10(a) a deformation cut is applied to the face of the torso model, revealing the mask interior object, which was placed inside. Note how the underside of the cut can be seen. Figures 7.10(b) and (c) show a twisting transformation applied at different scales for the hand model (18,905 triangles) and to the elephant (39,290 triangles). In order to test the flexibility of our approach, we applied

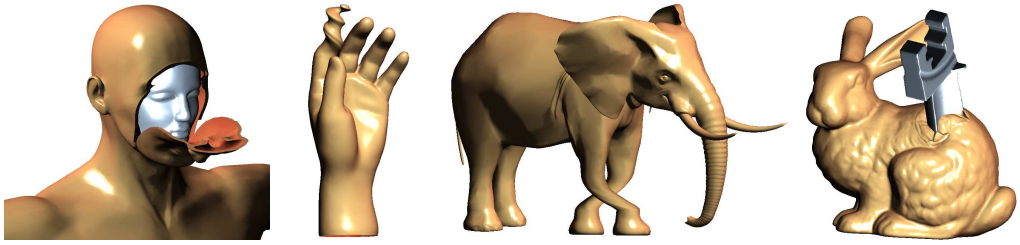


Figure 7.10: Example cuts and deformations on geometric models. From left to right: (a) Torso (25,528 triangles) with peeled skin and interior Mask model (10,213 triangles) (b) Hand with twisted finger (18,905 triangles) (c) Elephant with twisted legs (39,290 triangles) (d) Bunny (72,027 triangles) cut by Turbine Blade (10,778 triangles).

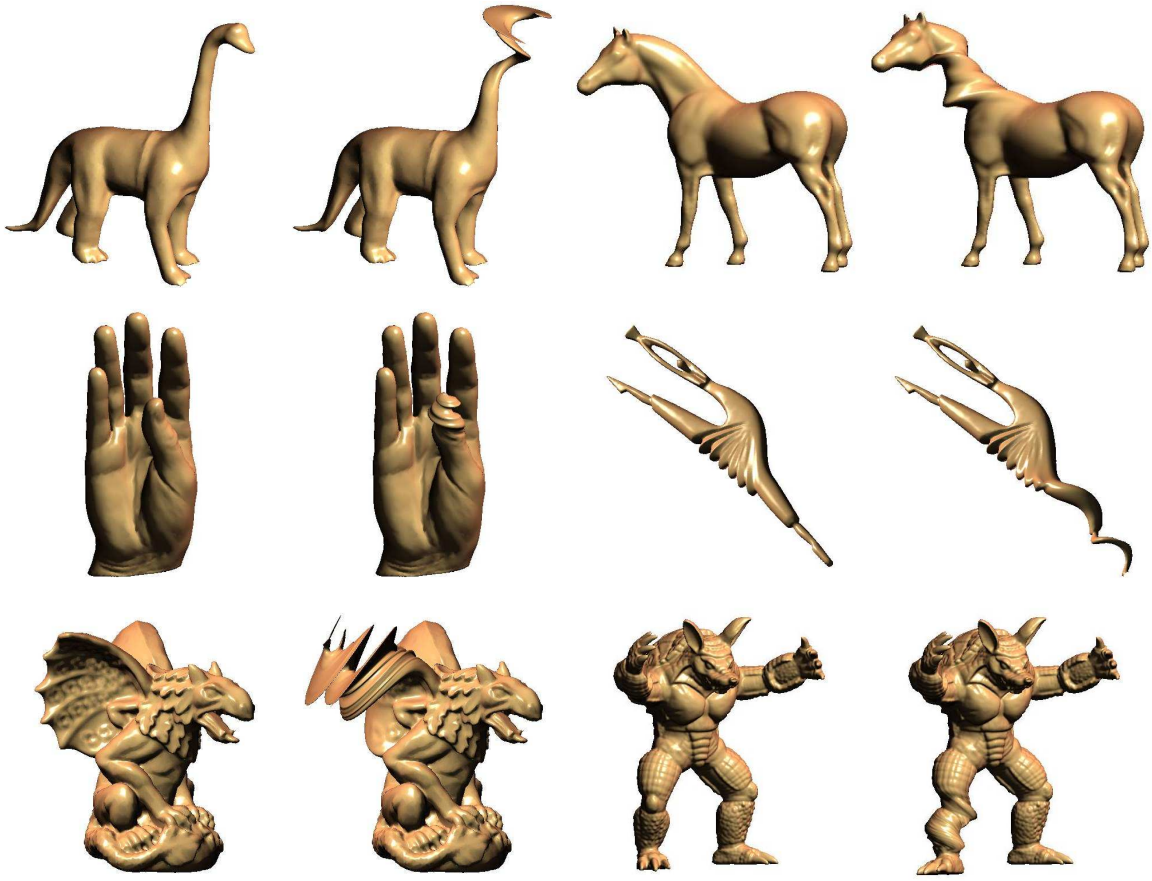


Figure 7.11: Twist deformation template applied to various polygonal models

the same deformation template to a number of polygonal models, of varying size. Figure 7.11 shows the result of a twist deformation for 6 objects. Similarly, Figure 7.12 shows the result of a peeler deformation for 6 different objects.

One of the possibilities of our approach is the inclusion of multiple deformations. Deformations which overlap (such as a cut in an area that is undergoing a twist) can also be handled

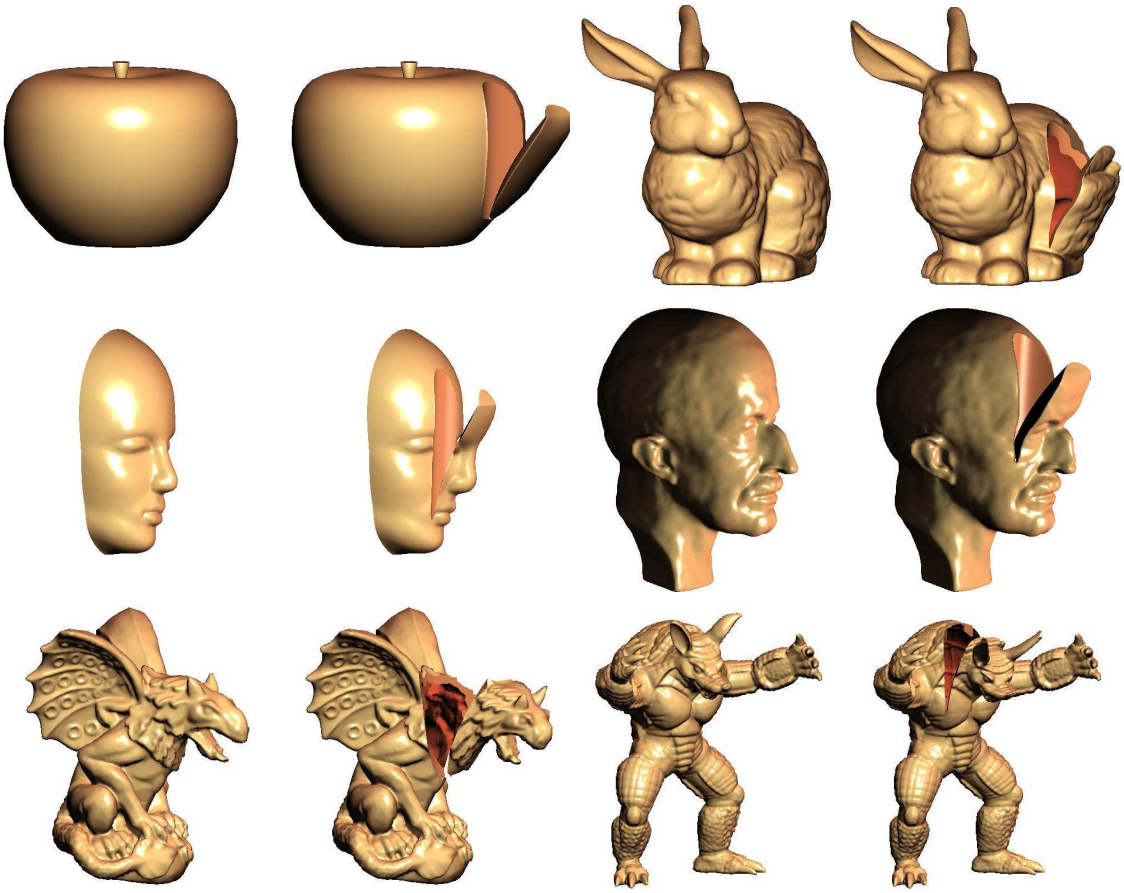


Figure 7.12: Peeler cut template applied to various polygonal models

by the framework. In this case, a second pass through the algorithm is performed, so that the layered representation is obtained progressively, one before deformation and the next after one of the deformations have been applied. Because this can be done directly while raycasting, there is no need for re-sampling of the space. Our approach can be also extended to models represented as points. In these cases, the deformation is handled for the layered renderer with our mechanism and the rest of the object with an appropriate rendering method. With our approach, it would be possible to create a library of deformations, in a similar way of libraries of models and textures.

7.8 Chapter Summary

This chapter presented a novel way of performing deformations on surface-based objects. We extended our volumetric method to accomodate sampled layered representations, which are

implicit representations of objects that can be defined in a 3D grid similar to volumes. Because we are deforming a volumetric representation, no remeshing is needed. Remeshing is often coupled with deformation to avoid collapsing of vertices and intersection of edges when performing large displacements or rotations, such as a twist, or to change the topology of the mesh in the case of cuts. In our approach, we use a layered representation of the region of an object undergoing deformation to obtain high quality deformations without remeshing. We defined a spectrum of layered representations, all of which are supported by our methodology. We showed how our approach can be efficiently implemented in the GPU, allowing complex deformations to be realized at interactive rates. We also described a hybrid rendering algorithn which seamlessly integrates the different representations. This makes it possible to create complex rendering merging surface-based and volumetric objects. Some of the applications is medical illustration. Because in medical illustration a depiction of internal organs and tissues is important, a volumetric object is desired. However, since the skin can be simulated with a surface, it may be beneficial to apply our method to the tissues of interest, which provides a smoother rendering of the deformed surface.

Chapter 8
Evaluation of Surface-based Deformation

8.1 Introduction

In this chapter, we evaluate our approach of surface-based deformation. Similar to volume deformation, we performed a series of quantitative experiments to validate the rendering quality of our approach, in regards to the smoothness of deformation without the need for remeshing. We also performed a series of experiments to measure the rendering performance and detect bottlenecks in our approach. We also propose mechanisms for improving the performance, such as *empty space skipping*.

8.2 Rendering Quality

Similar to our volume deformation algorithm, quality of the rendering depends on the smoothness and continuity of the resulting surface. Rather than replicating the experiments for volume deformation, which also apply for surface-based deformation, we focus on one of the problems of traditional surface-based deformation, which is the need for remeshing in order to obtain smooth results. We show that using our approach, smooth deformation is obtained for low and high resolution models without remeshing.

For the deformation of surfaces, traditional methods use explicit meshes, which may result in collapse of vertices and self-intersection, violating the smoothness principle of deformation. Figure 8.1 shows a comparison of our deformation approach with traditional mesh deformation using explicit twisting [6] on a model at different resolution. Since layered representations do not have explicit topology, artifacts due to the resolution of the mesh do not appear. We deformed a hand model of 18,905 triangles using a twisting motion. Note how this results in collapse of vertices and self-intersection of the surface, and is particularly problematic for a

low resolution model (2,874 triangles). Obtaining smooth results with a traditional deformation requires a re-meshing of the object into 52,463 triangles. In contrast, our approach provides smooth deformation results for both the low and high resolutions.

8.3 Rendering Performance

We performed a series of quantitative experiments on a Pentium XEON 2.8 Ghz PC with 4096 MB RAM, equipped with a Quadro FX 4400 with 512MB of video memory. We compared our rendering time for a continuous deformation and a discontinuous deformation. Since discontinuous deformations need to find more intersections, it is generally slower, since it requires the use of branching operations, which are known to be slow.

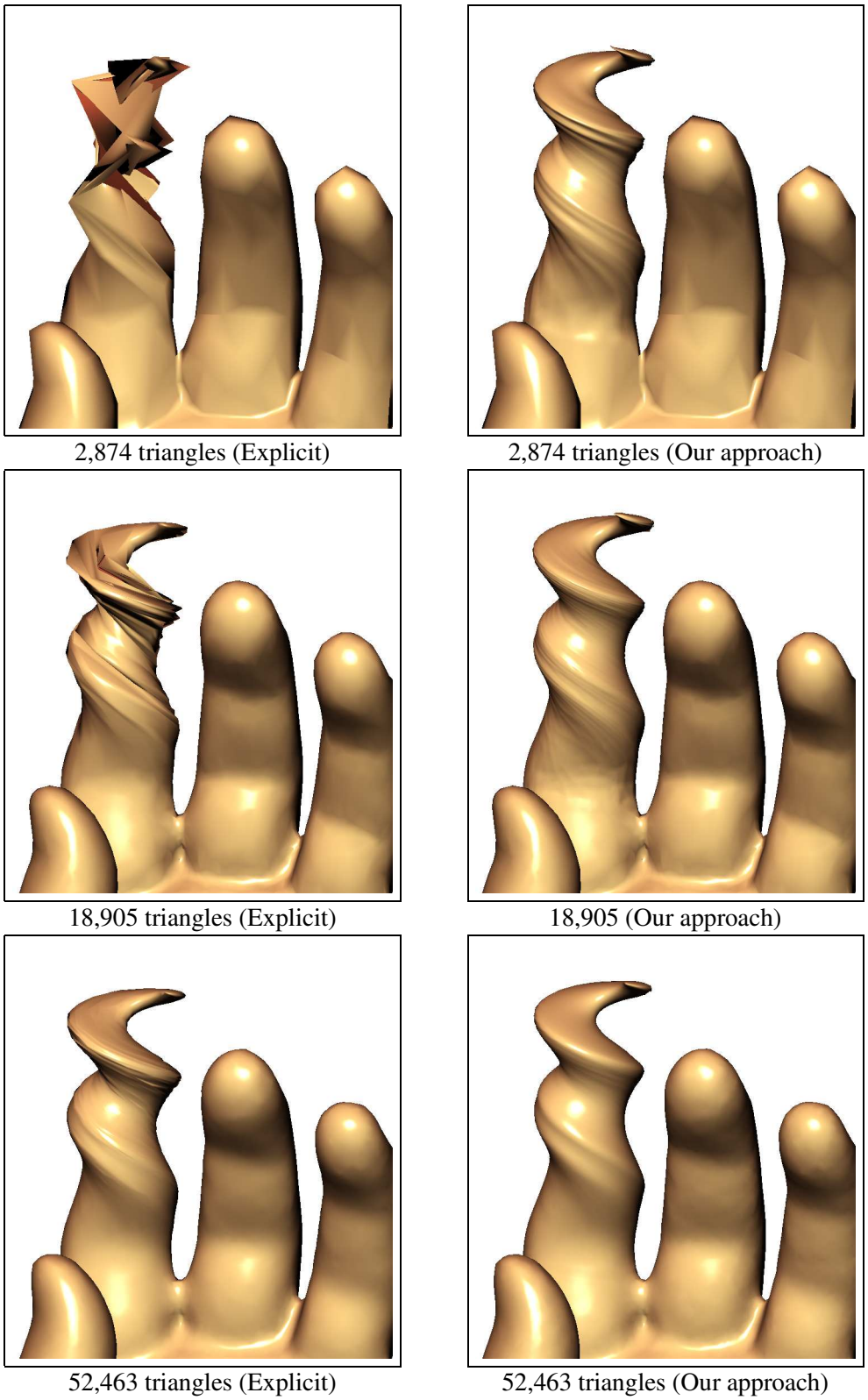
Our first experiment was designed to determine the impact of using different objects for deformation. Because deformation is applied only on a region of the object, performance depends both on the number of triangles as well as on the size of the sample layered representation used for the part undergoing deformation. However, since most complex operations are implemented in the deformed region, the performance is mostly determined by the rendering of the SLR. To test this, we rendered a deformation on objects of varying size and measured the rendering time. Because the rendering of the SLR depends on the number of pixels generated rather than the number of vertices, we measured the effective size of the deformation, as a percentage of the screen area (512×512). This is depicted in Figure 8.2. To summarize the rendering time for the two methods in a comparable way, we computed a weighted average on those results. The weighted average is computed as

$$\frac{\sum a_k t_k}{\sum a_k} \tag{8.1}$$

where $a_k \in [0,1]$ is the relative render area of the deformed space and t_k is the rendering time for a measure k. Table 8.1 shows the results of this averaging, where rendering time is given in milliseconds.

A one-way ANOVA showed no significant difference between the different objects ($F = 1.672, p > 0.05$) at a confidence level of 95%. This confirms that increasing the number of vertices does not affect significatively our approach.

For this reason, we are able to perform performance tests with little regards on the actual



2,874 triangles (Explicit) 2,874 triangles (Our approach)

18,905 triangles (Explicit) 18,905 (Our approach)

52,463 triangles (Explicit) 52,463 triangles (Our approach)

Figure 8.1: Comparison of explicit mesh deformation vs. our implicit approach. A hand model (original: 18,905 triangles) is twisted along a finger. Our deformation approach (right) provides smooth twisting across multiple resolutions of the object. Explicit deformation results in collapse of nodes and crossing of edges for the original model (middle row) and for lower resolutions (top row) (2,874 triangles). To obtain equivalent results, the mesh must be re-tessellated (52,463 triangles) as seen in the bottom row.

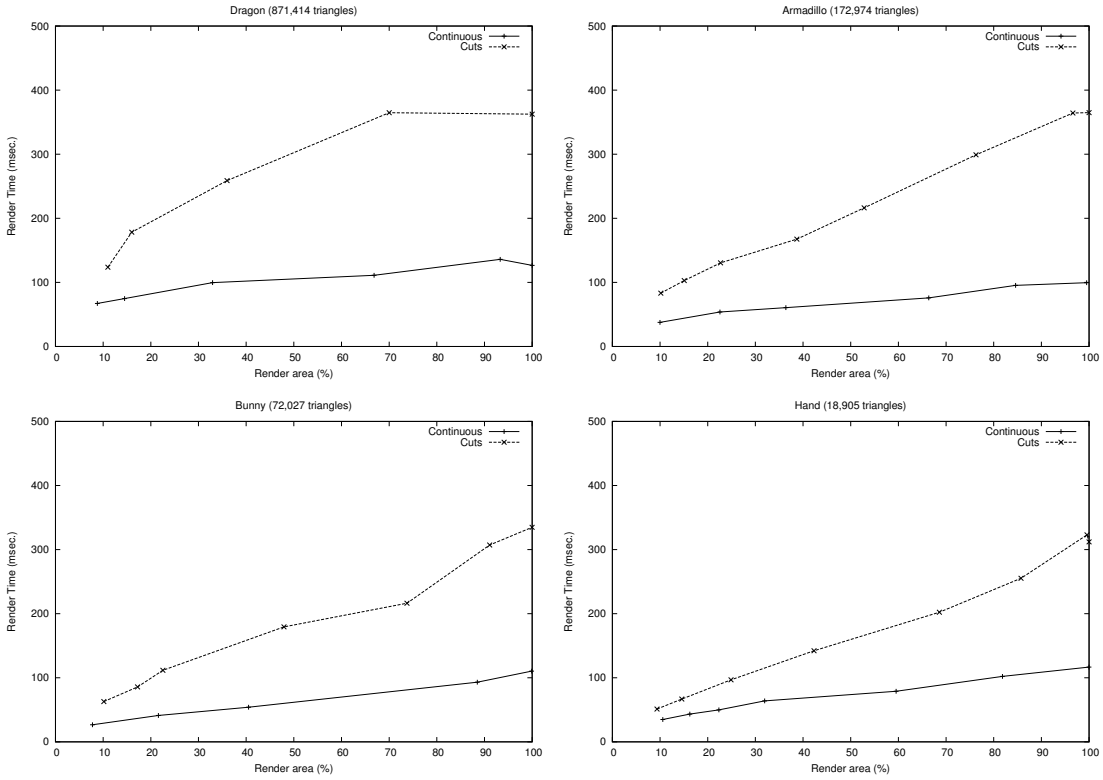


Figure 8.2: Rendering time for continuous deformation and solid discontinuous deformation, in relation to the relative size of the deformation bounding box, as a percentage of the screen area (512×512)

Model	Triangles	Continuous	Cuts
hand	18,905	89.79	225.46
bunny	72,027	87.32	249.92
armadillo	172,974	83.89	260.18
dragon	871,414	119.24	323.20
buddha	1,087,716	106.9	387.89

Table 8.1: Weighted average of rendering time for continuous and discontinuous deformation in milliseconds

mesh complexity of the surface, but rather on the complexity in terms of the properties of the sampled layered representation, and the rendering process itself.

8.3.1 Size

Here, we are interested in *size* in terms of number of voxels required to represent an SLR, or to represent a displacement. Both affect the amount of texture memory available in the GPU. Figure 8.3 shows the rendering time for an SLR of varying size, using a viewport of size 512×512 and an effective render area of approximately 0.9. Similarly, we also plotted the

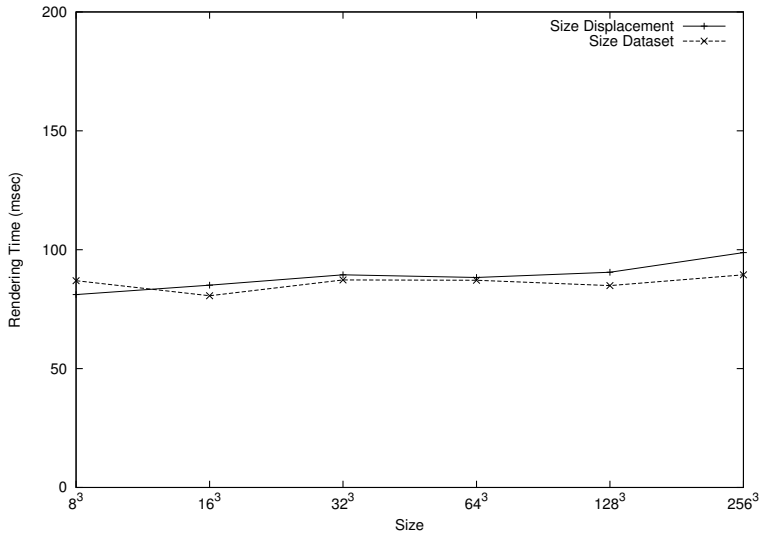


Figure 8.3: Effect of size in rendering performance

effect of varying the size of the displacement while maintaining the size of the dataset constant at 64^3 voxels. Since both the dataset and the displacement are stored as 3D textures, and the pixel shader does not make any difference between the roles of the textures, the performance is essentially the same. Note, however, that rendering time seems to grow for the case of the displacement compared to the size of the dataset.

Representation Size

Another factor related to size is the type of representation. As described in the previous chapter, some sampled layered representations are more memory efficient than others. For instance, depth maps only require 2D textures while distance fields require a 3D texture. Although the rendering algorithm is essentially the same, the distance to the surface is determined differently depending on the representation, as described in Section 7.6.2. Figure 8.4 shows the rendering time vs. the type of SLR. On the left, we see a comparison of representing the surface as a depth map vs. a 3D distance field. Note how the performance is essentially the same, given that both require the same number of texture samples (the only difference is the type of texture). On the right, we compare the rendering time for a depth map cube vs. a 3D distance field. In this case, we see how a depth map cube is much slower, as a result of requiring at most 6 texture lookups. This cost, however, is balanced by the benefit in texture requirement. For an SLR whose distance field is stored in a texture of size n^3 , a depth map cube requires only $6n^2$ bytes.

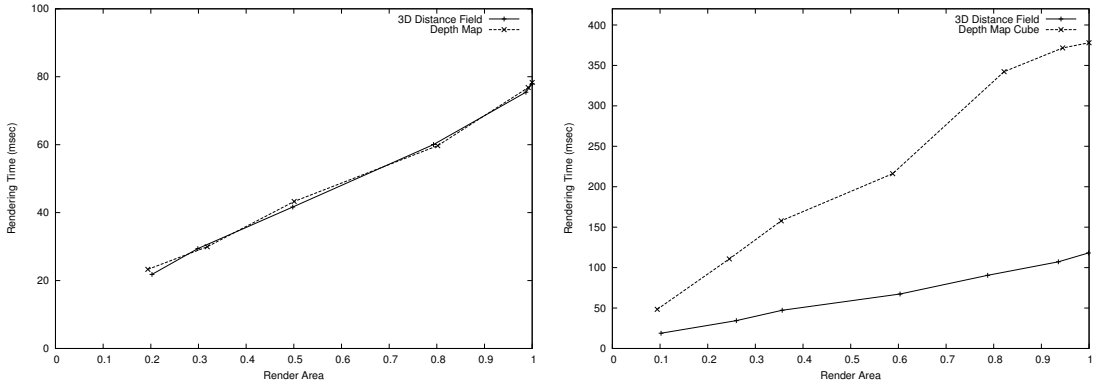


Figure 8.4: Effect of type of sampled representation in rendering time

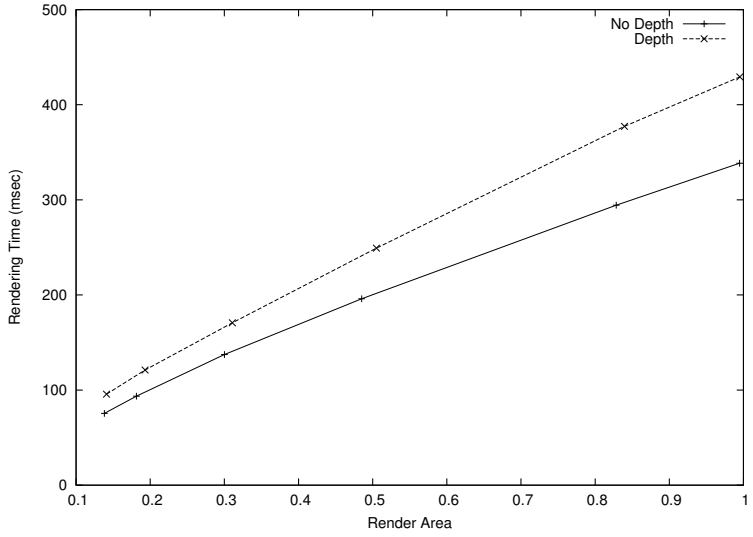


Figure 8.5: Performance overhead of depth estimation

8.3.2 Depth Estimation

In the previous chapter, we described how depth estimation is required whenever a surface S different from the one being deformed intersects the deformed region $B(S'_D)$. Depth estimation requires extra normalization factors and accumulation of traversed distances, which adds a computational cost to the rendering process. Figure 8.5 shows the performance penalty of adding depth estimation to the rendering algorithm, against the render area, for a deformation of size 64^3.

8.3.3 Empty Space Skipping

One of the important aspects of our approach is the accurate finding of intersections for smooth surface rendering. For a discontinuous deformation, we showed that smooth rendering requires the computation of more than one intersection. The rendering algorithm must compute intersections with the deformed surface and may discard them if they are in the region of cut. Depending on the complexity of the cut surfaces, the number of "false" intersections may increase, and the algorithm must then perform several iterations of the intersection finding process. Unfortunately, pixel shaders in current GPUs often limit the way loops are coded. For instance, the ones available to the moment of our experimentation do not allow variable sized loops. Although dynamic branching is enabled, i.e., the program control can be forced out of a loop by a break statement, there is a considerable penalty in performance due to branching. This can be appreciated in Figure 8.2, which showed a significant drop of performance when simulating discontinuous as compared to continuous deformation.

In another experiment, we measured the rendering time for a discontinuous deformation while varying the number of iterations of the intersection finding algorithm. The results are shown in Figure 8.6. We plotted the rendering time for two different cases, when the effective render area is 0.35 and 0.7. Note how the render time increases almost linearly with the number of iterations. What is worse is that in most of the test cases, the extra number of iterations are not needed as the correct intersection is the first intersection to be hit by a ray. In order to improve the rendering time, we use *Empty Space Skipping*, which uses information from the definition of the displacement to skip through the regions where there is no possibility of an intersection. As described in the previous chapter, a region within a cut, where there is nothing but empty space, is defined by $A(x) > 0$. Therefore, there is only need to traverse the space defined by $A(x) \leq 0$ in order to find the true intersections. This can be achieved by representing the cut surface A as a distance field, and using the distance to the closest point in the cut surface to determine the sampling distance of the next object. Whenever the surface is reached, a constant sampling distance is used.

Another mechanism is to represent A explicitly. In such case, A is defined by a surface. Instead of using $B(S'_D)$ as a proxy geometry to render the deformed region, we use the explicit

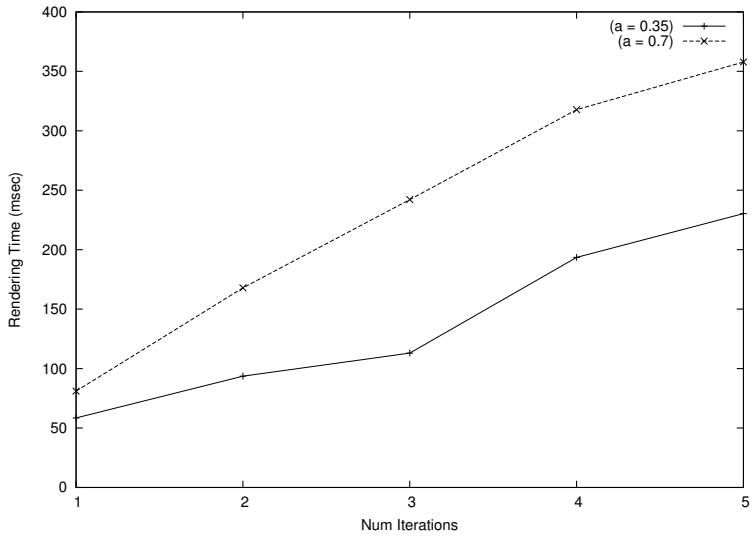


Figure 8.6: Number of iterations of the intersection finding process vs. rendering time (msec.)

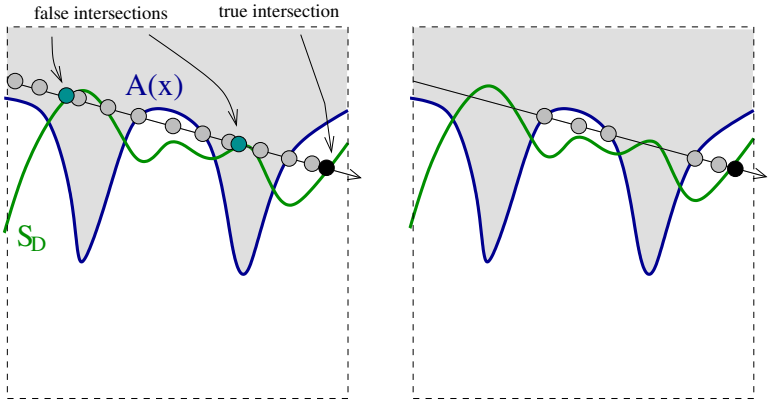


Figure 8.7: Empty Space Skipping. Left: With no empty space skipping, a number of unnecessary samples need to be taken and false intersections must be discarded. Right: Empty Space Skipping avoids the test for false intersections

representation $A + B(S'_D)$. Each generated ray is used to find the *first* intersection with the layered representation $L(x)$, without an the need for testing for false or true intersections. The explicit representation of A can be found as part of the process of the displacement creation, or by extracting the isosurface $A(x) = 0$ in the sampled representation of A. This process is depicted in Figure 8.7 Figure 8.8 shows the result of using empty space skipping, compared to the case described earlier in our results, where no optimization is performed. Note how the performance improvement is dramatic, and that, since no extra tests are need for the intersection process, the performance is essentially the same as the one for continuous deformation (compare with Figure 8.2) A t-test showed a significant difference between enabling and disabling

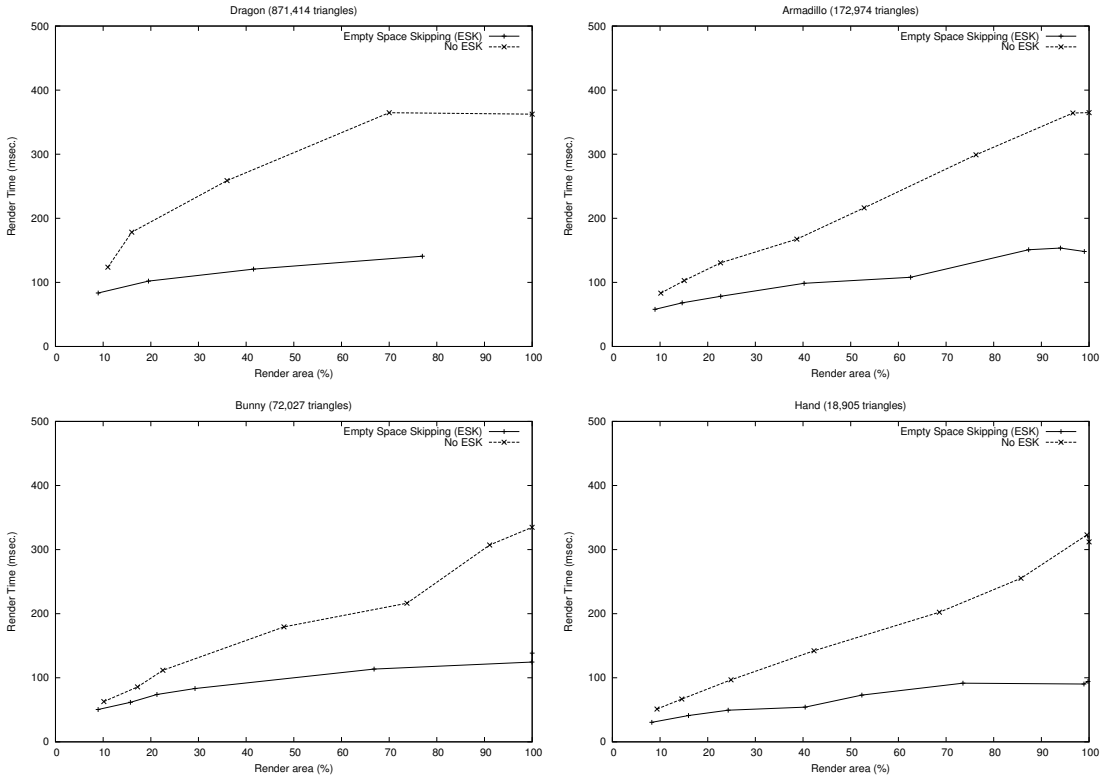


Figure 8.8: Speed up of empty space skipping for intersection finding algorithm vs. relative size of deformation bounding box (512×512)

Model	Triangles	Cuts (no ESK)	Cuts (with ESK)
hand	18,905	225.46	75.77
bunny	72,027	249.92	114.95
armadillo	172,974	260.18	126.00
dragon	871,414	323.20	126.5
buddha	1,087,716	387.89	120.8

Table 8.2: Weighted average of rendering time for continuous and discontinuous deformation in milliseconds

Empty Space Skipping ($t = 5.09$, $p < 0.0001$). These results are summarized in table 8.2, using the weighted average in Eq.(8.1).

8.4 Discussion

One of the results of our experiments is the validation of our approach as a mechanism for smooth deformation without the need for remeshing. Further, we also concluded that there is no significant impact of the number of vertices in the original mesh model and the rendering of a deformation. This is important for providing a constant rendering time for datasets of varying

size. However, this also implies that reducing the number of vertices does not have a significant impact on performance. We also detected a bottleneck in the rendering of cuts and other discontinuous deformations as a result of the need for estimating accurately multiple intersections along a given ray. Depending on the complexity of the cut geometry, the number of iterations of the intersection process varies. However, because of the lack of dynamic branching on contemporary GPUs, this is computationally expensive. As an alternative, the maximum number of iterations must be hardcoded into the deformation program. For cuts with concave geometry, for instance, there are at most two intersections, corresponding to the intersection with either the front or the back face of a surface. In order to improve performance, we devised an empty space skipping mechanism, which uses the distance to the surface of the cut to skip the regions where no intersection is valid. Further, performance is maximized when the cut is represented explicitly, and the deformation algorithm is simplified considerably, to the point where rendering performance is comparable to that of continuous deformation. Finally, our deformation algorithm proves to be a feasible mechanism for rendering complex cuts and deformations at interactive rates.

Chapter 9
Applications

9.1 Introduction

This chapter describes the applications of our approach in medical and biological illustration, as a rendering stage in surgical planning and simulation, and as a general tool for clipping and focus+context rendering in visualization.

9.2 Scientific Illustration

Illustrative deformation has been inspired by scientific illustration, in particular medical and surgical illustration. In turn, our framework can be used to generate on the fly illustrations of real scientific datasets. The high-quality rendering enable the illustrator to use the result as the final product, or as an intermediate stage, where the illustrator uses the deformed dataset as inspiration for a finished illustration.

Scientific illustration has been used for centuries for communicating ideas, abstracting complex structures and natural processes. Recently, hand-drawn illustrations have been the inspiration of many modern scientific visualization. For instance, the study of water by Leonardo has inspired many flow visualization software [55]. Hand-drawn illustration of internal organs has inspired modern cutaway and exploded views [30, 13]. In our work, the illustration of deformation and cuts has inspired our framework.

One of the most important milestones is Andreas Vesalius' *De Humanis Corporis Fabrica*, published in 1543, a compendium of human anatomy, of which stands out the detailed drawings of human dissections. The drawing "Vigesimaqvinta qvinti libri figvra" shows the internal organs of female anatomy through a detailed dissection of the abdomen. It was common of him and other contemporary illustrators to represent the retracted skin as it would result from an

actual dissection. The aspect of illustration we want to emphasize is the depiction of deformation, either to represent the range of movement of limbs and organs, or to represent a surgical procedure.

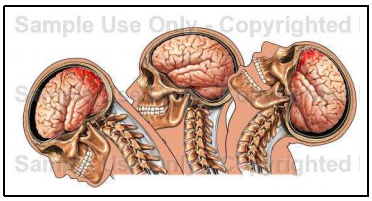
Contemporary illustrations are created through a variety of techniques, from traditional hand-drawings and airbrushed paintings, to computer assisted imagery. For the latter, however, the illustrator often uses a surface based model of an idealized anatomy. Within our framework, real patient data can be used to generate interactive illustrations. Throughout this chapter, we show a number of contemporary surgical illustrations that inspired some of our work and that we used to validate our approach.

9.2.1 Case Study Illustrations

In order to validate the applicability of our approach, we have selected a number of contemporary illustrations from an online repository (courtesy of Nucleus Inc. ©), and re-create similar illustrations on real medical datasets. We attempted to re-create the lighting and material characteristics depicted in the reference illustration.

Figure 9.1 shows an application of a continuous deformation. In this case, we simulate an illustration of a whiplash action. We applied a *bending* deformation on the CTHead dataset. The bending deformation is obtained from the inverse sampling of a forward bending, as defined by Barr in [6]. In order to gain visibility of the skull, we used gradient modulation. The brain tissue is not represented, as the CT dataset does not contain sufficiently distinct samples to reconstruct properly the surface of the brain (MRI scans are more appropriate for such goal). One important aspect to notice from our illustrative deformation is that lighting is computed properly as the head deforms, whereas the reference illustration seems to have been created by re-targeting a base illustration in three different poses, and lighting does not change accordingly.

Figure 9.2 shows an illustration of one stage of a craniotomy. One of the requirements for this illustration is the preservation of bone tissue. We applied this to the CTHead dataset, after an approximate segmentation of the skull. Note that for illustrations purposes, a complete and accurate segmentation is not required, as the system allows the user to explore interactively the region near cuts in order to adapt the resulting image to the desired state. This makes our

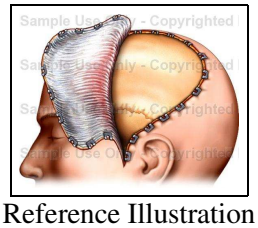


(a) Reference Illustration (Courtesy Nucleus Inc. ©)

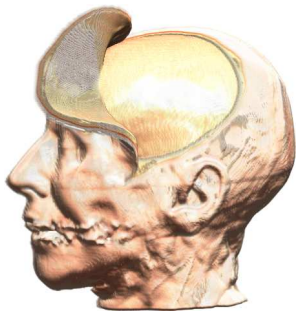
(b) Illustrative Deformation

Figure 9.1: Illustration of whiplash injury

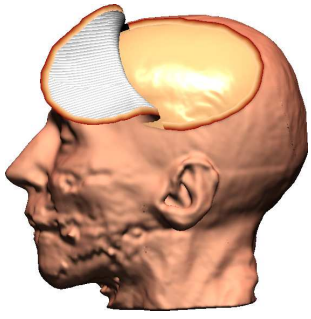
approach very attractive not only for illustration, but also as a visualization tool. To validate our surface deformation approach, we also applied the deformation to the isosurfaces corresponding to the skin and the skull. The surfaces were obtained using the marching cubes method [74], and decimated to about 50% the number of triangles. In addition, the resulting mesh was filtered for noise reduction. Deformation is applied to a layered representation of the isosurfaces, as described in Chapter 7.



Reference Illustration (Courtesy Nucleus Inc. ©)



Volume Deformation



Isosurface Deformation

Figure 9.2: Illustration of a craniotomy

Another important aspect of our approach is the flexibility offered by using generic deformation templates. For instance, we can easily transfer the deformation to another dataset.

Figure 9.3 shows the same deformation as applied to the head of the Visible Human dataset.

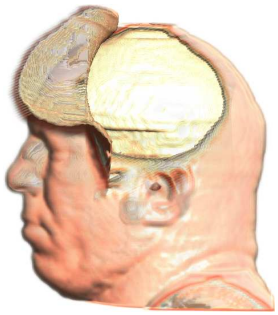
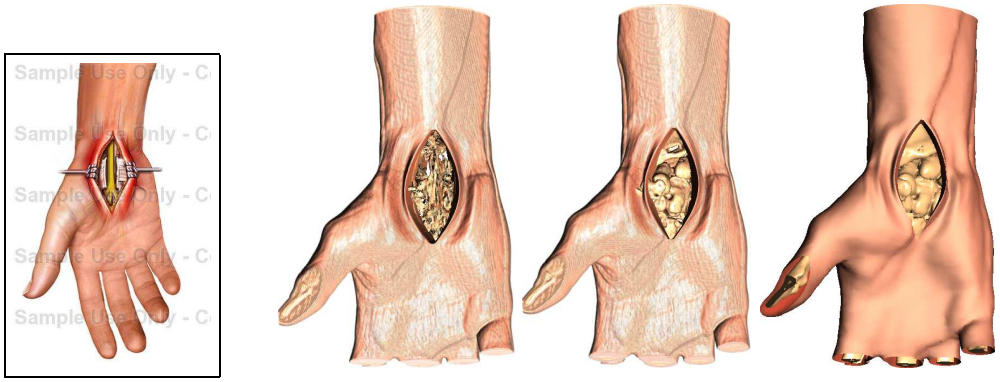


Figure 9.3: Transfer of deformation illustration to the Visible human dataset

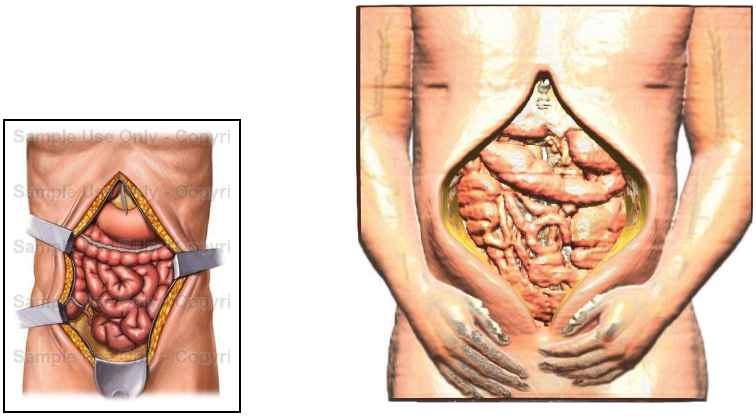
Another example of a surgical procedure is shown in Figure 9.4, where a carpal tunnel procedure is simulated on a CT hand dataset. First, an approximate segmentation of the veins and bones is obtained. Note that veins are difficult to segment from the CT dataset. For this reason, the two deformations on the right only show the bone tissue. Similar to the previous example, we applied surface deformation on the isosurfaces corresponding to skin and bone tissue.



Reference illustration (Courtesy Nucleus Inc. ©) | Volume (veins) | Volume (bones) | Isosurface (bones)

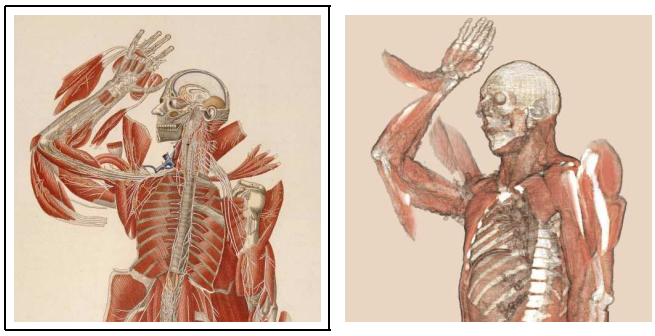
Figure 9.4: Illustration of carpal tunnel surgery

Figure 9.5 shows an illustration of an abdominal surgical procedure. For this case, we use a portion of the Visible Human dataset. Similarly to the previous illustrations, we used a feature mask to preserve the internal organs so that they do not undergo deformation. Volumetric deformation allows the visualization of the intermediate tissues between skin and organs, which gives depth and thickness to the deformed layer. This effect cannot be usually obtained with mesh deformation of the segmented isosurfaces.



Reference illustration (Courtesy Nucleus Inc. ©) Volume deformation

Figure 9.5: Illustration of abdominal procedure



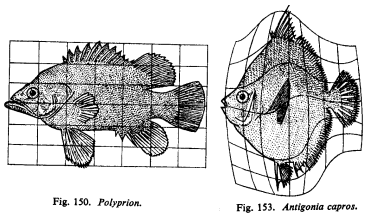
(a) Reference Illustration (b) Visible Human dataset

Figure 9.6: Anatomical Illustration with dissected flaps Dissected flaps

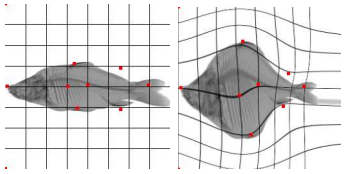
Another type of illustrations are of anatomical structures, where "unrealistic" flaps are used to separate muscle layers and allow the visibility of internal tissues and bone. These types of illustrations where prevalent during the early days of medical illustration, where it was common to represent the entire human being [95, 49], also similar to contemporary exhibits such as Bodies [1] and Bodyworlds [119].

9.2.2 Morphology Illustrations

Another use for deformation is the analytical exploration of morphology and the study of evolution, as pioneered by D'Arcy Thompson in his book "On Growth and Form" [114]. In his work, he uses the deformation of a 2D Cartesian grid to explain and illustrate the morphological differences between different animal species. In Figure 9.7, an example of his illustrations is depicted, where a rectangular grid is used to inscribe an illustration of the *Polyprion* species.



(a) Reference Illustration (From "On Growth and Form" [114])



(b) 2D Displacements using RBF

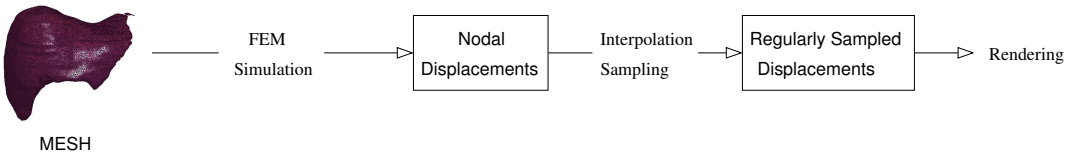
Figure 9.7: Application of our approach to Morphology Illustrations

Figure 9.8: Illustrative Deformation of Carp Dataset

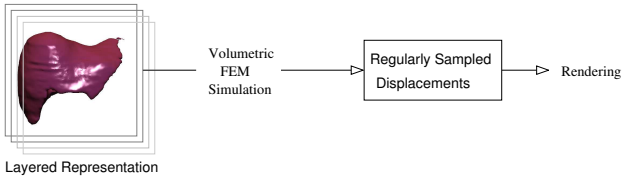
Another species, *Antogonia carpos* can be inscribed in a non-linear grid, obtained by deforming the rectangular grid. This method of deformation is essentially an interpolation mechanism given the displacements of control points, carefully placed in matching feature points. In our case, this can be accomplished with our displacement generation method using Radial Basis Functions, as described in Chapter 4. We used a projection image of the carp dataset, using maximum intensity projection, and inscribed it into a 2D grid. After deforming the image, we generated a 3D displacement that smoothly interpolates this 2D displacement along the Z direction. The results as applied to the 3D dataset are seen in Figure 9.8. Our approach can be extended to evolutionary morphing simulation directly on volumetric objects, extending the surface morphing approach in [127].

9.3 Surgery Planning and Simulation

Although this thesis is aimed towards *illustrative* deformation, our rendering algorithm can accommodate physically-based deformations. As described in Chapter 8, displacements can be obtained from layered representations of surface objects. This is useful for representing cutting and deforming tools required for surgical simulations, such as pliers, knifes and needles.



(a) Surgery Simulation with Nodal Displacements



(b) Surgery Simulation Process with Displacements in a Grid

Figure 9.9: Surgery simulation process for incorporation of our illustrative deformation approach

In addition, physical accurate displacements can be used. Instead of using generic deformation templates, displacement maps can be obtained from a physical simulation, either from a mass-spring or finite element simulation. However, our approach assumes that displacements are stored in a regular grid. In contrast, FEM or mass-spring methods have a explicit mesh usually composed of either triangles or tetrahedra. In this case, displacements are obtained at each node in the mesh. Recent alternatives, such as meshless deformation, define displacements at scattered points. In order to have a suitable displacement "format" for our approach, nodal displacements must be transformed into regularly sampled displacements. This requires a regularization and sampling process, which may be computationally costly. This process is depicted in Figure 9.9(a). Another alternative is to use regularly sampled displacements for the physical simulation itself. This is the approach used for physically-based deformation of volumes. One of the difficulties with this approach is the accurate simulation of boundary conditions, as sampled representation does not have explicit geometry. A possible solution for this problem is use the sampled layered representation as a mechanism for specifying boundary conditions. This process is depicted in Figure 9.9(b). Figure 9.10 shows an example of using generic template displacements for surgery simulation. In this case, we simulate an incision into an MRI neck dataset, and a subsequent pull operation on one the carotid arteries. Figure 9.11 shows an example of our approach applied to the simulation of a frog dissection. We used a segmented frog dataset and combined two deformations: a retractor deformation as depicted in Figures 9.11 (a) through (d), and a poke operator, simulating the deformation due to contact

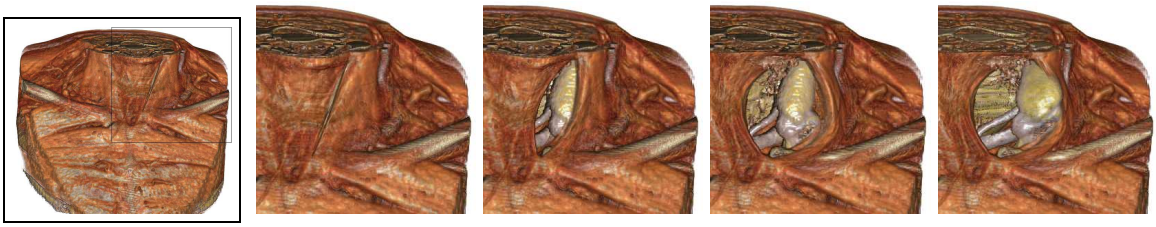
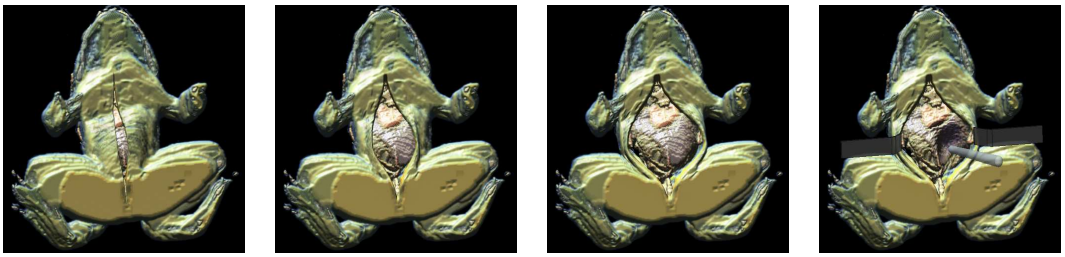


Figure 9.10: Neck surgery simulation



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

Figure 9.11: (a-c) Retractor operator used to simulate dissection of a segmented frog dataset. (d) A plier operator is applied to the internal organs, while simultaneously retracting the skin. Geometric models are embedded in the scene to show the placement of the operators.

with a surgical tool, as shown in Figure 9.11(d). Note also that surface meshes can be added where deformation occurs to represent the virtual tools. Many current surgical simulations are designed to work with a triangular or tetrahedral representation of an object, usually obtained via segmentation from a volumetric model. In our approach, we are able to deform surface object representations, by transforming them into sampled layered representations. Figure 9.12 shows our approach applied to a layered representation of the liver. We can simulate incisions without the need for re-meshing, as described in the previous Chapter.

9.4 Volume Clipping and Focus+Context Visualization

Another of the applications of our approach is as a visualization tool in general. Deformations have been used before as focus+context techniques, where a feature of interest is distorted

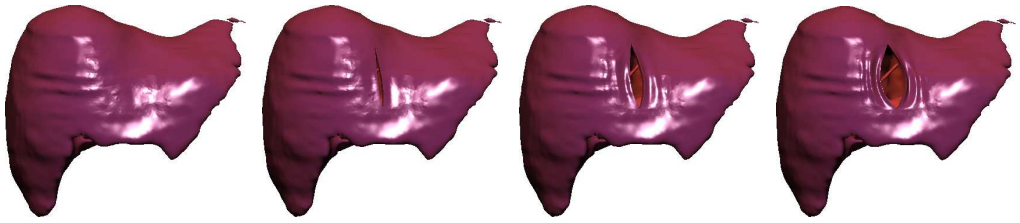
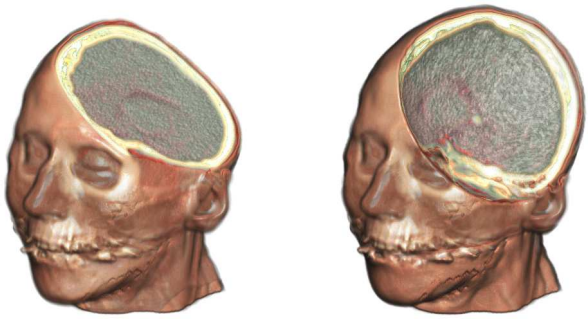


Figure 9.12: Liver surgery simulation



(a) Plane Cut (b) Spherical Cut

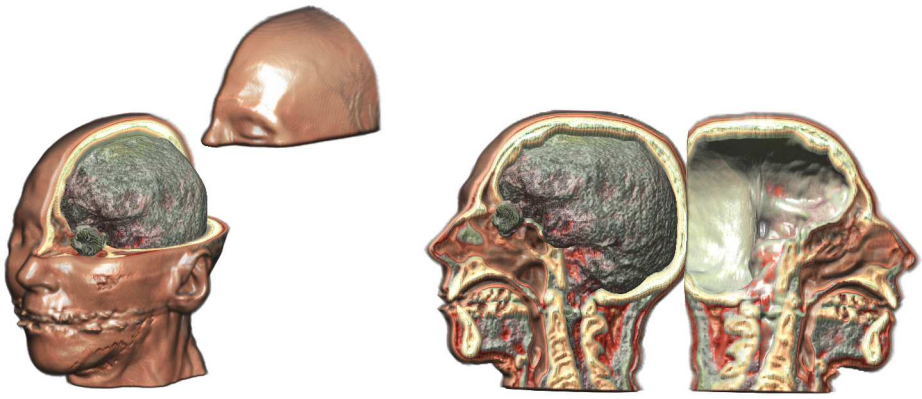
Figure 9.13: Clipping using discontinuous displacement mapping

so that it becomes highlighted, while still maintaining a view of the surrounding regions for context. Examples of these are magic volume lenses [123] and magnification lenses [65].

In our approach, focus+context visualizations can be obtained with both continuous and discontinuous deformations. In the case of a continuous deformations, a dilate deformation acts as a 3D distortion lens, which enhances the rendering of internal objects, while reducing the rendering regions of surrounding objects. In the case of discontinuous deformation, cuts and peels can be thought of as focus+context clipping mechanism. In the simplest case, when no deformation is specified, our approach works as a clipping mechanism with arbitrary clipping geometry. Examples are shown in Figure 9.13. When the clipped portion of the volume is retained along with the original volume, and a feature is preserved, it enables a focus+context visualization. Examples are shown in Figure 9.14. The first illustration shows a cutaway of the CT Head skin and skull to provide visibility of the brain. In the second, a peel deformation is used to visualize both the brain (region in focus) and the underside of the skull (as the context).

9.5 Chapter Summary

In this chapter, we have shown a series of examples to illustrate the range of applications of our approach. Inspired by biomedical illustration, our approach can be used for depicting surgical procedures, anatomical structures or natural phenomena with “real” volumetric datasets, much in the way illustrations are used. However, we can now rotate around our illustration. Further, our surface-based deformation approach enables the illustration of more abstract models, which



(a) Cutaway (b) Peel

Figure 9.14: Cutaway and Focus+Context Visualization of the brain in the CT Head dataset

may be modeled by an artist or extracted from MRI or CT data, which, although do not include information about the internal structure of an object, it provides smoother images. One of the advantages of our approach is the ability to explore the deformation space interactively, which is an aid for an illustrator in the task of depicting or visualizing complex procedures. Another application is in surgical planning and simulation. In this case, interactivity is crucial. We have shown two methods in which illustrative deformation can be incorporated into surgical simulation systems. First, as a rendering process, where nodal displacements, usually obtained from a Finite Element simulation, are interpolated into grid displacements. Second, as a modeling process, where the grid structure is incorporated into the simulation of forces in order to bypass the costly interpolation stage. Finally, we have shown how our approach can be used as a powerful visualization technique, that goes beyond clipping and transfer functions, by enabling deformation of occluding parts to achieve visibility of hidden objects in a natural way. Movies demonstrating our approach are available at http://www.caip.rutgers.edu/˜cdcorrea/feature/index.html and http://www.caip.rutgers.edu/˜cdcorrea/displacement/index.html.

Chapter 10
Conclusions

In this thesis, we have presented a unified framework for specifying and rendering complex continuous and discontinuous deformations of volumetric objects. We call this *Illustrative Deformation*. We have generalized the notion of 3D displacement maps, which have been previously used in computer graphics to add surface detail to mesh objects. Not only did we apply this notion to volume graphics, but we also extended it to allow large deformations and discontinuities such as cuts and breaks.

Chapter 4 described this notion, under what we have dubbed "discontinuous displacement mapping". We proposed a novel method for encoding discontinuity information on displacement maps which guarantees C^1 continuity in the displacement. This continuity is essential for the rendering of sharp cuts and breaks free of aliasing artifacts, and also, since the displacement can be differentiated at every point, it allows correct lighting of the volumetric object. One of the limitations of this general notion of displacement is the difficulty for specifying semantic constraints, which are necessary for surgical simulation and medical illustrations. One such constraint is the preservation of features of interest, which should not undergo a deformation. For example, simulating the illustration of a surgical procedure requires the deformation of skin and muscle tissues, but the preservation of bone structure. Chapter 5 described an efficient method for specifying feature sensitive operations. This method introduces feature *masks* into the displacement, which are used to modify the transformation so that elements within the mask are preserved. We also presented a novel method for adjusting the shading in the regions near cuts, which enables the accurate rendering of deformed surfaces in the vicinity of the cut.

In Chapter 6, we validated our approach through a series of quantitative tests, which are related to the properties of rendering quality, such as smoothness, continuity and preservation of detail. Some important outcomes of this evaluation is the validation of our approach as the one

with the best image quality compared to previous attempts for volume deformation. Not only does it provide the best quality, it was also shown that it is the most efficient in terms of performance, due to the ability of pre-computing quantities in 3D textures. This speed up, however, comes with a price in extra texture memory requirements. However, we have also shown that our alternative for reducing texture memory is as efficient as other previous alternatives, with a much higher rendering quality. Another advantage of our approach is the ability to model complex deformations with guaranteed cost. As shown in our evaluation chapter, performance cost depends on the resolution and precision of the displacement, and not to the complexity of the deformation. That is, two deformations sampled at the same frequency yield almost identical results. However, it is important to note that the optimal sampling frequency for a displacement map might differ and require different resolutions.

We also showed that our approach can be extended to the deformation of surface-based objects. Traditional approaches to mesh deformation transforms the vertices of the mesh. For large deformations or cuts, this usually implies a remeshing stage, which may be computationally costly and can limit the complexity of the deformation. In this thesis, we were able to simulate complex deformations on surface models without remeshing. This was possible with the sampling of the surface into a layered representation, an implicit representation of the surface which samples the closest distance to the surface at regularly spaced points. The layered representation can be 2D, as it is the case of depth or height maps, or 3D, such as distance fields. We validated our approach with different layered representation methods in order to explore the trade off between texture memory requirement, speed and image quality.

Our approach has applicability in a number of fields. In medical and biological illustration, our approach allows the illustration of anatomical structures and surgical procedures on real 3D datasets, obtained via computed tomography or magnetic resonance imaging. We showed that, due to the generality of displacement maps, deformations can be easily transferred to different datasets. We also showed that surface-based deformation can be applied by first obtaining an isosurface description of the features of interest. One future direction for our work is the combination of surface and volume rendering to obtain better illustrations. Surface deformation rendering would be critical for the simulation of smooth surfaces such as the skin, whereas volume rendering is necessary for the depiction of internal layers and features.

Another application of our work is surgical simulation and planning. Our approach can be introduced in simulation engines as a rendering process. However, most simulation approaches, such as mass-spring and finite elements, require nodal displacements obtained from an explicit mesh, instead of sampled displacements in a grid. One mechanism is to introduce an intermediate interpolation stage which converts nodal displacements into grid-sampled displacements, but this can be computationally expensive. Alternatively, it should be possible to modify the physical simulation engine to solve the equations of motion directly on the grid. Finally, our approach works as a general tool for visualization and graphics, where deformation is used as a focus+context mechanism, and discontinuous deformation can be used as a clipping tool.

10.1 Directions for future work

In this thesis, we have described a unified framework for deformation of volumetric objects. We have shown that surface-based models can also be accommodated within our approach, by describing them as a layered representation. Surface-based models have an advantage over volumetric objects in that they are generally smoother and require less information to be stored. Volumes, on the other hand, contain material information or both the exterior and interior of objects. A more comprehensive method for illustrative deformation would treat certain surfaces of interest, such as the skin, as surfaces, whereas other parts can be treated as volumes. Further, our deformation framework can also be complemented with *illustrative or non-photorealistic rendering* (NPR). As opposed to the so-called "photorealistic" rendering, illustrative rendering simplifies the lighting model and uses illustration-inspired drawing techniques to enhance or abstract certain parts of the object, for a better understanding of shape or function. Common NPR techniques use simplified shading, hatching and stippling to accentuate the shape of an object. When the shape is undergoing deformation, these hatch or stipple patterns can also be used to accentuate the deformation. Since deformations are displacement fields, a lot of information can be extracted by analyzing its properties. For instance, the rendering of field lines yields the direction of the deformation, while the Jacobian determinant can used to illustrate the "strength" of the deformation. These cues can be added to the rendering process to provide the user with a better understanding of the manipulation process. The idea of using illustrative

cues for visualizing movement has been explored by Joshi and Rheingans [55] and we believe it can be further explored for interactive deformation.

As graphic processors units become more powerful, and with the introduction of physics processor units (PPUs), it now becomes possible to accelerate physics-based-deformation of complex datasets. Our approach, rather than becoming obsolete, will greatly benefit from the increased processing power of upcoming hardware. Features such as hardware accelerated collision detection could be used to generate the displacements needed to simulate the interaction of volumetric objects with a virtual surgical tool. However, recent research efforts of using the GPUs as general purpose processors (GPGPUs) questions whether the PPUs would be of any use beyond their utility as yet another GPGPU [2].

The idea of using deformation to manipulate data can be further extended as a visualization tool for other types of objects, such as flow volumes, general vector fields, video data or discrete spatial data, such as graphs. For volumetric objects, our method extends easily, but special care must be taken depending on the domain. For instance, lighting computation does not apply for the visualization of video, as they are formed by images of an already lit scene. A better idea would be to focus on the tracking and preservation of features of interest, as described in Chapter 5. We believe that the use of deformation to manipulate data can become a key enabler for exploration techniques beyond slicing, rotations and transfer functions.

Appendix A
Index of Datasets

The following table shows the index of the datasets used throughout this thesis. For each dataset, we define its size in number of voxels, its source, which may be Computed Tomography (CT), Magnetic Resonance Imaging (MRI) or procedurally defined and the credits. Any special pre-processing is noted where it applies.

Name	Size	Source	Credits	Notes
Cube	128^3	Procedural	VIZLAB Rutgers	
Bar	128^3	Procedural	VIZLAB Rutgers	
Engine	$256 \times 256 \times 128$	CT	General Electric	
Tomato	$256 \times 256 \times 64$	MRI	Lawrence Berkeley Laboratory	
Piggy Bank	$256 \times 256 \times 179$	CT	Siemens	
Hand	$255 \times 240 \times 155$	CT	University of Iowa	
Visible Man	$255 \times 189 \times 436$	MRI	National Library of Medicine	Segmented
CT Head	$256 \times 256 \times 256$	CT	Stanford University	
Foot	$143 \times 256 \times 183$	CT	University of Iowa	
Frog	$250 \times 235 \times 68$	MRI	Whole Frog Project	Segmented
Knee	$256 \times 256 \times 256$	CT	University of Iowa	
Visman Head	$256 \times 256 \times 256$	MRI	National Library of Medicine	Segmented
Visman Abdomen	$256 \times 170 \times 256$	MRI	National Library of Medicine	Segmented
Neck	$256 \times 256 \times 256$	MRI	Robert Wood Johnson Hospital	
Carp	$200 \times 100 \times 512$	CT	University of Erlangen, Germany	

Appendix B
Index of Illustrative Deformations

B.1 Tomato Illustrations

This illustration (Figure 4.8) is obtained using the tomato dataset (CT) and the peel displacement. An ambient intensity of 0.5 is used to simulate the translucent nature of the material (as opposed to opaque). Figure 4.9 combines the peel displacement with a waving displacement. The order of composition alters the order. Figure 4.11 is produced by applying the slicer displacement and repeating it periodically. The repetition is obtained by wrapping around the texture coordinates as follows:

```
int iy = (int)((texcoord.y-1)/0.125)+1;
texcoord.x = texcoord.x + iy*0.25;
texcoord.y = texcoord.y - iy*0.125;
```

B.2 Discontinuous Displacement Showcase

This collection of illustrations are obtained by applying the poke, peel, split and slice displacements onto the cube, bar, engine, and tomato datasets.

B.3 CT Head Peel

Figure 5.4 shows a peel of the CT head dataset. Surface alignment is obtained with a distance field of the dataset after background segmentation. Segment alignment is obtained after segmentation of the brain tissue. Segmentation is produced by an intensity-guided space filling approach, after performing edge detection. Figure 5.8 shows a similar peel, with alignment on the skull. The skull is segmented using thresholding of the bone tissue density and space filling on the interior of the skull with randomly positioned seeds.

B.4 Hand Dissection

The illustration of hand dissection (Figure 5.1) was obtained using the retractor displacement over the hand dataset (CT). Feature alignment was obtained by an approximate segmentation of bone tissue. Intermediate tissue is approximated as well to provide visibility of vessels. The illustration in Figure 5.8 applies the retractor2 displacement to the same dataset. Surface alignment is obtained using a distance field of the hand dataset after background segmentation. Segment alignment is obtained after segmentation of the bone tissue and noise filtering. It can be seen that other tissue between the proximal phalanges are preserved in the feature mask.

B.5 Foot Surgery

This illustration (Figure 5.8) was obtained using the retractors displacement on the foot dataset (CT). Surface alignment was obtained with the distance field of the foot after background segmentation and setting the desired layer at $\tau = 0.74$. Segment alignment is obtained with a segmentation of the bone tissue, by density thresholding. The feature mask is smoothed out with a Gaussian filter.

B.6 Frog Dissection

This illustration (Figure 5.8) was obtained using the retractors displacement on the frog dataset. The dataset was obtained from the segmented frog dataset, after smoothing and merging the different segments. Surface alignment is obtained with the distance field after background segmentation. Segment alignment is obtained using the original segmentation masks of bones and organs. To produce a smooth mask, a Gaussian filter is applied.

The illustration in Figure 9.11 is obtained in the same manner, with a different transfer function. A poke operator is added to the feature of interest to show the use of multiple displacements. In addition, geometric models are added to depict the surgical tools.

B.7 Whiplash

The whiplash illustration (Figure 9.1 is obtained with the CT head dataset, using the bend displacement. Depending on the orientation of the displacement, a forward or backward bend is obtained. In order to show the skull, we applied gradient modulations with an exponent of $e_g = 0.5$ and slice opacity $\alpha = 1$.

B.8 Craniotomy

Figure 9.2 shows a craniotomy of the CT head dataset. Feature alignment is obtained via segmentation of the bone tissue, using density thresholding. The mask was smoothed out with a Gaussian filter. The surface-based deformation is obtained by deforming the skin isosurface. The isosurfaces of skin tissue and the skull are obtained using the marching cubes method. To obtain a smooth result, the isosurfaces are decimated at about 30% of the original triangle cut and smoothed out using a Laplacian operator. The backside of the cut is rendered with a procedural texture, to resemble the lines depicted in the reference illustration.

B.9 Carpal Tunnel Surgery

This illustration (Figure 9.4) was obtained using the `cutquad` displacement on the hand dataset (CT). Feature alignment is obtained with a segmentation of the bone tissue. The surface-based illustration is obtained using the isosurfaces from bone tissue and skin. Both isosurfaces were decimated and smoothed using a Laplacian operator. Prior to isosurface extraction, the dataset is thresholded into the appropriate density range and an *antialiasing* filter is applied.

B.10 Abdominal Surgery

This illustration (Figure 9.5) is obtained using the retractor dataset on a portion of the segmented Visible Human dataset. The segmented dataset is smoothed using a Gaussian filter. The feature is obtained from the segments of the colon, organs and bone tissue which are merged together and smoothed using a Gaussian filter.

B.11 Anatomical Illustration

This illustration (Figure 9.6) is obtained using the segmented Visible Human and a series of flap displacements. The dataset used contains only the bone and muscle tissues. The placement of displacements is obtained via affine transformations for proper alignment. A cut displacement (using only the discontinuity mask) is used to "carve" out portions of the abdomen. The rendering was obtained using an ambient intensity of 0.7 and an opacity of $\alpha = 0.5$.

B.12 Carp Illustration

This illustration (Figure 9.8) was obtained using the carp dataset. The 3D displacement was obtained from a 2D displacement using radial basis functions over a number of control points. The 2D displacement was extruded along the z-direction (orthogonal to the 2D displacement) such that it gradually fades into zero. This was necessary as to provide a plausible deformation of the fish, where the mid-parts are more deformed than the surface of the sides.

B.13 Neck Surgery

This illustration (Figure 9.10) was obtained using the neck dataset (MRI) with the retractor displacement. A transfer function is applied to hide the skin tissue and provide visibility of the muscles. A feature mask is applied, containing vessels and bones. This was obtained using thresholding of the corresponding density values. Given that a contrast dye was used in the original scan, the vessels and the bones are obtained from the same range of densities. A second operator is applied to the feature segment, to simulate a surgical procedure on the carotid artery.

B.14 Liver Surgery

This illustration (Figure 9.12) were obtained from a surface extracted from the segmented Visible Human dataset. The surface was decimated and post-processed for smoothing.

Appendix C
Displacement Templates

This appendix shows the procedural definition of the displacements used in this thesis, as functions of the x,y, and z coordinates of a normalized volumetric cube, i.e., $x,y,z \in (0,1)$.

C.1 Poke

$$\begin{aligned} D(x,y,z) &= \begin{pmatrix} 0 \\ 0 \\ -ze^{\frac{(x-0.5)^2+(y-0.5)^2}{2\sigma^2}} \end{pmatrix} \\ A(x,y,z) &= 1 \end{aligned}$$

C.2 Twist

$$\begin{aligned} D(x,y,z) &= \begin{pmatrix} (x-0.5)\cos\theta(z) + (y-0.5)\sin\theta(z) - (x-0.5) \\ -(x-0.5)\sin\theta(z) + (y-0.5)\cos\theta(z) - (y-0.5) \\ 0 \end{pmatrix} \\ A(x,y,z) &= 1 \end{aligned}$$

where $\theta(z)$ is a twisting angle, as a function of the z coordinate (usually linear).

C.3 Wave

$$
\begin{aligned}
D(x,y,z) &= \begin{pmatrix} 0 \\ a\sin(\omega\sqrt{(x-0.5)^2+(z-0.5)^2}) \\ 0 \end{pmatrix} \\
A(x,y,z) &= 1
\end{aligned}
$$

where a is the amplitude of the wave and ω its frequency.

C.4 Bend

$$
\begin{aligned}
\hat{\theta} &= \tan^{-1}\frac{y-y_0}{z-z_0} \\
\theta &= clamp(\hat{\theta}, \theta_{min}, \theta_{max}) \\
\hat{y} &= \frac{\theta}{2\pi} + y_0 \\
y' &= \begin{cases} (y-y_0)\cos\theta - (z-z_0)\sin\theta + \hat{y} & \theta_{min} < \theta < \theta_{max} \\ \hat{y} & \text{otherwise} \end{cases} \\
z' &= \begin{cases} (y-y_0)\sin\theta + (z-z_0)\cos\theta + z_0 & \theta_{min} < \theta < \theta_{max} \\ z_0 + \sqrt{(y-y_0)^2+(z-z_0)^2} & \text{otherwise} \end{cases} \\
D(x,y,z) &= \begin{pmatrix} 0 \\ y'-y \\ z'-z \end{pmatrix} \\
A(x,y,z) &= 1
\end{aligned}
$$

where θ_{min} and θ_{max} define the minimum and maximum bending angles, and $(0, y_0, z_0)^\top$ is the center of the bend.

C.5 Squeeze

$$\begin{aligned} \sigma &= s\frac{1}{1+e^{5-20*|0.5-z|}} \\ D(x,y,z) &= \begin{pmatrix} \sigma(x-0.5) \\ \sigma(y-0.5) \\ 0 \end{pmatrix} \\ A(x,y,z) &= 1 \end{aligned}$$

where $s \in (0,1)$ is the strength of the squeeze deformation

C.6 Dilate

$$\begin{aligned} D(x,y,z) &= -D_{squeeze}(x,y,z) \\ A(x,y,z) &= 1 \end{aligned}$$

C.7 Peel

$$
\begin{aligned}
\theta &= \tan^{-1}\frac{x}{y} \\
r &= \sqrt{x^2+y^2} \\
y' &= \begin{cases} -y & -\pi < \theta \le \pi/2 \\ y & -\pi/2 < \theta \le 0 \\ r & \text{otherwise} \end{cases} \\
x' &= \begin{cases} 1-x & -\pi < \theta \le \pi/2 \\ x & -\pi/2 < \theta \le 0 \\ \frac{\theta}{\pi} & \text{otherwise} \end{cases} \\
D(x,y,z) &= \begin{pmatrix} x'-x \\ y'-y \\ 0 \end{pmatrix} \\
A(x,y,z) &= \begin{cases} 0 & y < -2R \text{ or } r > 2R \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
$$

C.8 Split

$$
\begin{aligned}
\sigma &= f(z) \\
D_x &= \begin{cases} \sigma & x < 0.5-\sigma \\ -\sigma & x > 0.5+\sigma \\ 0.5-x & \text{otherwise} \end{cases} \\
D_y &= 0 \\
D_z &= 0 \\
A(x,y,z) &= \begin{cases} 1 & x < 0.5-\sigma \vee x > 0.5+\sigma \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

where $\sigma = f(z)$ is a function describing the size of the split. For a linear split, $f(z) = 0.25z$.

C.9 Retractor

The retractor is obtained by combining two mirrored one-sided retractors. The one-sided retractor is defined as follow:

$$
\begin{aligned}
c_c &= 0.5 \\
c_s &= c_s + \sigma(z)p(x) \\
c_e &= c_c + 0.4 \\
D_x &= 0 \\
D_y &= \begin{cases} c_c - y & c_c < y < c_s \\ c_c + \frac{y-c_s}{c_e-c_s}(c_e - c_c) - y & c_s < y < c_e \\ 0 & \text{otherwise} \end{cases} \\
D_z &= 0 \\
A(x,y,z) &= \begin{cases} 0 & c_c < y < c_s \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
$$

where c_c is the center of the cut, c_s is the *start* and c_e the *end* of the retracted layer, $p(x)$ is the shape and $\sigma(z)$ is the strength of the retractor. For our illustrations, $\sigma(z) = kz$, and $p(x)$ is a Gaussian function:

$$p(x) = e^{5\left(\frac{0.5-x}{L}\right)^2} \tag{C.1}$$

where $L \in (0, 1.0)$ is the length of the cut.

C.10 CutQuad

Similar to the retractor, this is obtained as a combination of two mirrored one-sided cuts, which in turn are a combination of the retractor and a sinusoidal rippling effect.

$$
\begin{aligned}
c_x &= 0.5 \\
c_y &= 0.5 \\
c_s &= 0.5 + (1 - s(y))z^2 \\
c_e &= 0.9 \\
s(y) &= \left[\left(\frac{y - c_c}{L} \right)^2 \right]_0^1 \\
t(x) &= \frac{x - c_s}{c_e - c_s} \\
D_x &= \begin{cases} c_x + t(x)(c_e - c_x) - x & x < c_e \\ 0 & \text{otherwise} \end{cases} \\
D_y &= 0 \\
D_z &= (1 - t(x))(1 - s(y))z^2 \sin^2 2\pi t(x) \\
A(x,y,z) &= 0.5 + (x - c_s)
\end{aligned}
$$

where c_s and c_e are the start and end of the retracted layers, $s(y)$ is the strength of the retraction and L is the length of the cut, and $[x]_0^1$ is a clamping operator.

C.11 Peeler

$$
\begin{aligned}
z_0 &= 4\,(x-0.5)^2 \\
d(y) &= 0.5y^2 + z_0 \\
D(x,y,z) &= \begin{pmatrix} 0 \\ 0 \\ -min(d(y)-z_0, [z-z_0]_0^1) \end{pmatrix} \\
A(x,y,z) &= \begin{cases} z_0 - z & z < z_0 \\ z - d(y) & z > d \\ max(z-d, z_0-z) & \text{otherwise} \end{cases}
\end{aligned}
$$

Appendix D
Pixel Shaders

This appendix shows the pixel shaders for implementing illustrative deformation on commodity hardware. Our implementation uses Cg (nVidia) language for the description of pixel shaders, compatible with fragment profiles `fp20` through `fp40`.

D.1 Simple Deformation Renderer

This renderer is the basic algorithm, which implements our framework of discontinuous displacement mapping. Two implementations are shown. One, a simple volume renderer without any lighting, useful for fast deformation or previewing tool. The second, a volume renderer with lighting, where the normals must be computed and transformed.

D.1.1 Unlit volumes

```
float4 main(
     float3 texCoord : TEXCOORD0,
     uniform sampler3D dataTexture,
     uniform sampler1D transferFunction,
     uniform float sliceAlpha
):COLOR {
   // warp
   float4 warpedCoord = warp(texCoord);
   // sample
   float density = tex3D(dataTexture, warpedCoord.xyz);
   // classify
   float4 color = tex1D(transferFunction, density);
   // modulate discontinuity
   float opacity = (warpedCoord.w<0.5)? 0:1;
   color.w = color.w * opacity * sliceAlpha;
   return color;
}
```

D.1.2 Warp Procedure

```
uniform sampler3D disp_xy;
uniform sampler3D disp_za;
float4 warp(float3 texCoord) {
       float4 d1, d2;

       d1 = tex3D(disp_xy, texCoord);
       d2 = tex3D(disp_za, texCoord);

       float3 disp = 2*float3(d1.x, d1.w, d2.x) - 1;
```

```
        float4 warpedCoord;
        warpedCoord.xyz = vector.xyz + disp;
        warpedCoord.w = (2*d2.w-1);
        return warpedCoord;
}
```

D.1.3 Shaded Volumes

```
float4 main(
     float3 texCoord : TEXCOORD0,
     uniform sampler3D dataTexture,
     uniform sampler1D transferFunction,
     uniform float sliceAlpha
):COLOR {
   // warp
   float4 warpedCoord = warp(texCoord);
   // sample
   float density = tex3D(dataTexture, warpedCoord.xyz);
   // get normal
   float3 normal0 = 2*tex3d(gradTexture, warpedCoord.xyz).xyz - 1;
   float3 normal = deformNormal(normal0, texCoord, warpedCoord.w);
   // classify
   float4 color = getLighting(density, normal);
   // modulate discontinuity
   float opacity = (warpedCoord.w<0.5)? 0:1;
   color.w = color.w * opacity * sliceAlpha;
   return color;
}
```

D.1.4 Normal Estimation Procedure

```
float3 deformNormal(float3 normal0, float3 texCoord, float opacity) {
// 1. Estimate Jacobian
   float3 texCoordX = texCoord + float3(dx,0,0);
   float3 texCoordY = texCoord + float3(0,dx,0);
   float3 texCoordZ = texCoord + float3(0,0,dx);
   float3 texCoordX0 = texCoord - float3(dx,0,0);
   float3 texCoordY0 = texCoord - float3(0,dx,0);
   float3 texCoordZ0 = texCoord - float3(0,0,dx);

   float3 dispx;
   float3 dispy;
   float3 dispz;
   float3 dispx0;
   float3 dispy0;
   float3 dispz0;

   //X+
   dispx.xy = 2*tex3D(dispTex_, texCoordX).xw-1;
   dispx.z = 2*tex3D(dispTex2_, texCoordX).x-1;
   //Y+
   dispy.xy = 2*tex3D(dispTex_, texCoordY).xw-1;
   dispy.z = 2*tex3D(dispTex2_, texCoordY).x-1;
   //Z+
   dispz.xy = 2*tex3D(dispTex_, texCoordZ).xw-1;
   dispz.z = 2*tex3D(dispTex2_, texCoordZ).x-1;
   //X-
   dispx0.xy = 2*tex3D(dispTex_, texCoordX0).xw-1;
```

```
   dispx0.z = 2*tex3D(dispTex2_, texCoordX0).x-1;
   //Y-
   dispy0.xy = 2*tex3D(dispTex_, texCoordY0).xw-1;
   dispy0.z = 2*tex3D(dispTex2_, texCoordY0).x-1;
   //Z-
   dispz0.xy = 2*tex3D(dispTex_, texCoordZ0).xw-1;
   dispz0.z = 2*tex3D(dispTex2_, texCoordZ0).x-1;

   //Central diff
   float s = sizeDisp/2;
   float3 dPdx = s*(dispx - dispx0);
   float3 dPdy = s*(dispy - dispy0);
   float3 dPdz = s*(dispz - dispz0);

   float3x3 J;
   J[0].xyz = float3(1+dPdx.x,   dPdx.y,   dPdx.z);
   J[1].xyz = float3(  dPdy.x, 1+dPdy.y,   dPdy.z);
   J[2].xyz = float3(  dPdz.x,   dPdz.y, 1+dPdz.z);
// 2. Multiply Jacobian
   float3 normal = mul(J, normalize(preNormal));
// 3. Estimate Alpha Normal
   float3 alphaNormal;
   alphaNormal.x = s*2*(tex3D(dispTex2_, texCoordX)-tex3D(dispTex2_, texCoordX0));
   alphaNormal.y = s*2*(tex3D(dispTex2_, texCoordY)-tex3D(dispTex2_, texCoordY0));
   alphaNormal.z = s*2*(tex3D(dispTex2_, texCoordZ)-tex3D(dispTex2_, texCoordZ0));

   float w = 1 - saturate((opacity-0.5)/thickness);

   return normalize(lerp(normal, alphaNormal,w));
}
```

D.2 Feature-Aligned Renderer

This section shows the code for implementing feature-aligned deformation. This is obtained by querying the mask volume before obtaining density and normal samples from the volume. Similarly to our previous code, we show the rendering process for both unlit and shaded volumes.

D.2.1 Unlit Volumes

```
float4 main(
     float3 texCoord : TEXCOORD0,
     uniform sampler3D dataTexture,
     uniform sampler3D maxTex,
     uniform sampler1D transferFunction,
     uniform float sliceAlpha,
     uniform float layer
):COLOR {
   // warp
   float4 warpedCoord = warp(texCoord);

   // query feature mask
   float mask = tex3D(maskTex, texCoord.xyz).w;
   if(mask>=layer) {
        mask0 = tex3D(maskTex, warpedCoord.xyz).w;
        warpedCoord.w = (mask0<layer)? 0: warpedCoord.w;
   } else {
        warpedCoord.xyz = texCoord.xyz;
```

```
        warpedCoord.w = 1;
   }

   // sample
   float density = tex3D(dataTexture, warpedCoord.xyz);
   // classify
   float4 color = tex1D(transferFunction, density);
   // modulate discontinuity
   float opacity = (warpedCoord.w<0.5)? 0:1;
   color.w = color.w * opacity * sliceAlpha;
   return color;
}
```

D.2.2 Shaded Volumes

```
float4 main(
     float3 texCoord : TEXCOORD0,
     uniform sampler3D dataTexture,
     uniform sampler3D maxTex,
     uniform sampler1D transferFunction,
     uniform float sliceAlpha,
     uniform float layer
):COLOR {
   // warp
   float4 warpedCoord = warp(texCoord);
   // query feature mask

   float4 maskVector = tex3D(maskTex, texCoord.xyz);
   float3 normalMask = normalize(2*maskVector.xyz-1);
   float mask = maskVector.w;

   if(mask>=layer) {
        float4 mask0Vec = tex3D(maskTex, warpedCoord.xyz).w;
        normalMask = normalize(2*mask0Vec.xyz-1);
        mask = mask0Vec.w;
        warpedCoord.w = (mask0<layer)? 0: warpedCoord.w;
   } else {
        warpedCoord.xyz = texCoord.xyz;
        warpedCoord.w = 1;
   }

   // sample
   float density = tex3D(dataTexture, warpedCoord.xyz);
   // get normal
   float3 normal0 = 2*tex3d(gradTexture, warpedCoord.xyz).xyz - 1;
   float3 normalDefo = deformNormal(normal0, texCoord, warpedCoord.w);
   // adjust normal
   float3 normal = adjustNormal(normalDefo, normalMask, mask);

   // classify
   float4 color = getLighting(density, normal);
   // modulate discontinuity
   float opacity = (warpedCoord.w<0.5)? 0:1;
   color.w = color.w * opacity * sliceAlpha;
   return color;
}
```

D.2.3 Adjust Normal Procedure

```
float3 adjustNormal(float3 normal0, float3 normalMask, float mask) {
   float delta = mask - layer;
   float beta;
   if(delta<0) {
      beta = saturate(delta/featureThickness + 1);
   } else {
      beta = -saturate(-delta/featureThickness + 1);
   }
   float3 adjustedNormal = (1-abs(beta))*normal0 - beta*normalMask;
   return normalize(adjustedNormal);
}
```

D.3 Surface Deformation using Ray Casting

This section presents the code for surface deformation using ray casting. The core of the implementation is `findIntersection` which implements a combination of linear and binary search for finding zero crossings of the layered representation. We show the different implementations for continuous deformation, hollow and solid cuts.

D.3.1 Continuous Deformation

```
float4 main(v2f IN,
    uniform float3 viewDefo,
    uniform float3 lightVecDefo,
    uniform float delta,
    sampler2D image4,
    sampler3D dataTex,
    sampler3D disp_xy,
    sampler3D disp_za
    ):COLOR {
    float3 coord = IN.texcoord.xyz;
    float3 intersection;
    float3 warpedIntersection;
    bool intersect = findIntersection(coord, viewDefo, delta,
                                      intersection, warpedIntersection);
    if(!intersect) discard;

    // Get normal (the same way as obtained for volumes,
    // using the Jacobian of the displacement)
    float3 normal = normalize(getNormal(warpedIntersection));

    float3 halfVec = normalize(lightVec - viewDefo);
    float4 color = getLighting(normalVec, viewDefo, lightVec, halfVec);

    return color;
}
```

D.3.2 Hollow Cuts

```
float4 main(v2f IN,
    uniform float3 viewDefo,
    uniform float3 lightVecDefo,
    uniform float delta,
    sampler2D image4,
    sampler3D dataTex,
```

```
    sampler3D disp_xy,
    sampler3D disp_za
    ):COLOR {
    float3 coord = IN.texcoord.xyz;
    float3 intersection;
    float3 warpedIntersection;
    bool intersect;

    for(int i=0;i<num_iterations;i++) {
       intersect = findIntersection(coord, viewDefo, delta,
                                    intersection, warpedIntersection);
       if(intersect) break;
       coord = intersection + viewDefo*delta;
    }
    if(!intersect) discard;

    // Get normal (the same way as obtained for volumes,
    // using the Jacobian of the displacement)
    float3 normal = normalize(getNormal(warpedIntersection));

    float3 halfVec = normalize(lightVec - viewDefo);
    float4 color = getLighting(normalVec, viewDefo, lightVec, halfVec);

    return color;
}
```

D.3.3 Solid Cuts

```
float4 main(v2f IN,
    uniform float3 viewDefo,
    uniform float3 lightVecDefo,
    uniform float delta,
    sampler2D image4,
    sampler3D dataTex,
    sampler3D disp_xy,
    sampler3D disp_za
    ):COLOR {
    float3 coord = IN.texcoord.xyz;
    float3 intersection;
    float3 warpedIntersection;
    bool intersect;

    for(int i=0;i<num_iterations;i++) {
       intersectAlpha = findIntersectionAlpha(coord, viewDefo, delta,
                                              intersectionA, warpedIntersectionA);
       normal = getNormalAlpha(warpedIntersectionA);

       if(intersectAlpha) break;
       intersect = findIntersection(intersectionA, viewDefo, delta,
                                    intersection, warpedIntersection);
       normal = getNormal(warpedIntersection);
       if(intersect) break;
       coord = intersection + viewDefo*delta;
    }
    if(!intersect) discard;

    // Get normal (the same way as obtained for volumes,
    // using the Jacobian of the displacement)
    float3 halfVec = normalize(lightVec - viewDefo);
```

```
    float4 color = getLighting(normalVec, viewDefo, lightVec, halfVec);

    return color;
}
```

D.3.4 Find Intersection

```
bool findIntersection(float3 startCoord,
                      float3 ray,
                      float delta,
                      out float3 intersection,
                      out float3 warpedIntersection) {
   bool intersect;
   float3 coord = startCoord;
   float4 warpedCoord = warp(coord);

   float distance;
   float prevDistance = sampleImplicit(warpedCoord.xyz);

   // Linear search
   for(int i=0;i<num_linear_steps;i++) {
    warpedCoord = warp(coord);
    if(coord.x<0 || coord.y<0 || coord.z<0 || coord.x>1 || coord.y>1 || coord.z>1) {
      intersect = false;
      intersection = startCoord;
      break;
    }
    distance = sampleImplicit(warpedCoord);
    if(distance*prevDistance<0) {
      intersect = true;
      break;
    }

    intersection = coord;
    coord = coord + ray*delta0;
    prevDistance = distance;
  }

  // Binary Search
  float3 coord2;
  for(int i=0;i<num_binary_steps;i++) {
    delta = delta*0.5;
    coord2 = coord0+ray*delta;
    warpedCoord = warp(coord2);
    distance = sampleImplicit(warpedCoord);
    if (distance*prevDistance>0) {
      coord0 = coord2;
    }
  }

  // find alpha component of deformation
  float alpha = (warpedCoord.w<0.5)? -1:1;

  if(alpha<0) {
    // intersection within volume of cut
    intersect = false;
  }

  intersection = coord0;
```

```
  warpedIntersection = warpedCoord;

  return intersect;
}
```

D.3.5 Sample Implicit Representation

This method differs depending on the type of layered representation. For a 3D distance field represented as a 3D texture `dataTex` is:

```
float sampleImplicit0(float3 texCoord) {
  return (2*tex3D(tex_, texCoord).w-1);
}
```

For a depthmap, represented as a 2D texture `depthmap`.

```
float sampleImplicit(float3 texCoord) {
  return tex2D(depthmap_,texCoord.xy).x-texCoord.z;
}
```

Finally, for a depth map cube, represented as 6 textures `texZ1`, `texZ0`, `texX1`, `textX0`, `texY1` and `texY0`:

```
float sampleImplicit(float3 texCoord) {
  // samples are obtained at the center of voxels
  // therefore we need to apply an offset of 'deltasample'
  float deltasample =  0.5f/sizevol;
  float vZ1 = tex2D(texZ1_, texCoord.xy).x -deltasample;
  float dZ1 = vZ1-texCoord.z;

  float2 coordZ0 = float2(1-texCoord.x, texCoord.y);
  float vZ0 = tex2D(texZ0_, coordZ0.xy).x-deltasample;
  float dZ0 =  texCoord.z -(1-vZ0);

  float2 coordX0 = float2(1-texCoord.z,texCoord.y);
  float vX0 = tex2D(texX0_,coordX0).x - deltasample;
  float dX0 = vX0-texCoord.x;

  float2 coordX1 = texCoord.zy;
  float vX1 = tex2D(texX1_,coordX1).x - deltasample;
  float dX1 = texCoord.x -(1-vX1);

  float2 coordY0 = float2(texCoord.x,1-texCoord.z);
  float vY0 = tex2D(texY0_,coordY0).x - deltasample;
  float dY0 = vY0 - texCoord.y;

  float2 coordY1 = texCoord.xz;
  float vY1 = tex2D(texY1_,coordY1).x - deltasample;
  float dY1 = texCoord.y - (1-vY1);

  float depth = combine(dZ1,dZ0);
  depth = combine(depth, dX1);
  depth = combine(depth, dX0);
  depth = combine(depth, dY0);
  depth = combine(depth, dY1);
  return depth;
}
```

Here, the procedure `combine` is used to combine two depth fields, and it can be implemented as follows:

```
float combine(float depth1, float depth2) {
  float deltasample1 =  0; //strength_ * 0.5f/sizevol;
  float u = min(abs(depth1), abs(depth2));
  float signu = (depth1<0 && depth2<0)? -1:1;
  return signu*u;
}
```

References

[1] Bodies the exhibition, 2005.

[2] Ageia. Physics, gameplay and the physics processing unit. Technical report, 2005.

[3] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. Recent advances in remeshing of surfaces. In *The state-of-the-art report of the AIM@SHAPE EU network*, 2005.

[4] Maya: A 3d animation and visual effects tool, 2002.

[5] A. Barr. Ray tracing deformed surfaces. *Computer Graphics (Proc. SIGGRAPH 86)*, 20(4):287–296, 1986.

[6] Alan H. Barr. Global and local deformations of solid primitives. In *SIGGRAPH '84: Proc. of the 11th annual conference on Computer graphics and interactive techniques*, pages 21–30, New York, NY, USA, 1984. ACM Press.

[7] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*. SAIM, Philadelphia, PA, 2nd edition, 1994.

[8] Daniel Bielser, Pascal Glardon, Matthias Teschner, and Markus Gross. A state machine for real-time cutting of tetrahedral meshes. In *PG '03: Proc. of the 11th Pacific Conference on Computer Graphics and Applications*, page 377, Washington, DC, USA, 2003. IEEE Computer Society.

[9] J. F. Blinn. Simulation of wrinkled surfaces. *Computer Graphics (Proc. SIGGRAPH 78)*, 12(3):286–292, 1978.

[10] Mario Botsch and Leif Kobbelt. Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum*, 22(3):483–492, 2003.

[11] Morten Bro-Nielsen and Stephane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 15(3):57–66, 1996.

[12] Stefan Bruckner, Sören Grimm, Armin Kanitsar, and Meister Eduard Gröller. Illustrative context-preserving volume rendering. In *Proceedings of EuroVis 2005*, pages 69–76, May 2005.

[13] Stefan Bruckner and M. Eduard Groller. Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1077–1084, 2006.

[14] Stefan Bruckner and Meister Eduard Gröller. Volumeshop: An interactive system for direct volume illustration. In H. Rushmeier C. T. Silva, E. Gröller, editor, *Proceedings of IEEE Visualization 2005*, pages 671–678, October 2005.

[15] Tom Brunet, K.Evan Nowak, and Michael Gleicher. Integrating dynamic deformations into interactive volume visualization. In *Proceedings Eurographics/IEEE-VCTC Symposium on Visualization (EuroVis) 2006*, 2006.

[16] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 586–593, New York, NY, USA, 2002. ACM Press.

[17] Billy Chen, Eyal Ofek, Heung-Yeung Shum, and Marc Levoy. Interactive deformation of light fields. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 139–146, New York, NY, USA, 2005. ACM Press.

[18] H. Chen, J. Hesser, and R. Manner. Ray casting free-form deformed-volume objects. *The Journal of Visualization and Computer Animation*, 14:61–72(12), 2003.

[19] M. Chen, S. Islam, M. W. Jones, P.-Y. Shen, D. Silver, S. J. Walton, and P. J. Willis. Deforming and animating discretely sampled object representations. In *Eurographics State of the Art Reports (to appear)*, Dublin, Ireland, 2005.

[20] M. Chen, D. Silver, A. S. Winter, V. Singh, and N. Cornea. Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation. In *Proc. Volume Graphics '03*, pages 35–44. ACM Press, 2003.

[21] Y. Chen, Q. Zhu, and A. Kaufman. Physically-based animation of volumetric objects. In *Proc. IEEE Computer Animation '98*, pages 154–160, 1998.

[22] R. L. Cook. Shade trees. *Computer Graphics (Proc. SIGGRAPH 84)*, 18(3):223–231, 1984.

[23] R. L. Cook, L. Carpenter, and E. Catmull. The Reyes image rendering architecture. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4):95–102, 1987.

[24] Carlos D. Correa and Deborah Silver. Dataset traversal with motion-controlled transfer functions. In *IEEE Visualization 2005*, pages 359 – 366, October 2005.

[25] S. Cotin and H. Delingette. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):62–71, 1999.

[26] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, New York, NY, USA, 1996. ACM Press.

[27] Harry Davis. *Introduction to Vector Analysis*. Allyn and Bacon, 3rd edition, 1967.

[28] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 31–36, 2001.

[29] Hervé; Delingette, Stéphane Cotin, and Nicholas Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *CA '99: Proc. of the Computer Animation*, page 70, Washington, DC, USA, 1999. IEEE Computer Society.

[30] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive cutaway illustrations. In *Proceedings of Eurographics Conference '03*, 2003.

[31] Discreet. 3d studio max, (3dsmax), 2004.

[32] M. Doggett and J. Hirche. Adaptive view dependent tessellation of displacement maps. In *Proc. EG/SIGGRAPH Workshop on Graphics Hardware*, pages 59–66, Interlaken, Switzerland, 2000.

[33] W. Donnelly. Per-pixel displacement mapping with distance functions. In *GPU Gems 2*, pages 123–136., 2005.

[34] David Ebert and Penny Rheingans. Volume illustration: non-photorealistic rendering of volume models. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 195–202, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[35] Gershon Elber. Geometric deformation-displacement maps. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 156, Washington, DC, USA, 2002. IEEE Computer Society.

[36] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 9–16, New York, NY, USA, 2001. ACM Press.

[37] S. Fang, R. Raghavan, and J. T. Richtsmeier. Volume morphing methods for landmark-based 3D image deformation. In M. H. Loew and K. M. Hanson, editors, *Proc. SPIE Vol. 2710, p. 404-415, Medical Imaging 1996: Image Processing, Murray H. Loew; Kenneth M. Hanson; Eds.*, pages 404–415, April 1996.

[38] S. Fisher and M.C. Lin. Deformed distance fields for simulation of non-penetrating flexbile bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation*, pages 99–111, 2001.

[39] A. O. Frank, I. A. Twombly, T. J. Barth, and J. D. Smith. Finite element methods for real-time haptic feedback of soft-tissue models in virtual reality simulators. In *Proc. Virtual Reality*, pages 257–xxx, 2001.

[40] Nikhil Gagvani and Deborah Silver. Animating volumetric models. *Graph. Models*, 63(6):443–458, 2001.

[41] James E. Gain and Neil A. Dodgson. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):289–298, 2001.

[42] Sarah F. Gibson. 3d chainmail: a fast algorithm for deforming volumetric objects. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 149–ff., New York, NY, USA, 1997. ACM Press.

[43] S. Gumhold and T. Hüttner. Multiresolution rendering with displacement mapping. In *Proc. EG/SIGGRAPH Workshop on Graphics Hardware*, pages 55–66, 1999.

[44] Xiaohu Guo and Hong Qin. Real-time meshless deformation: Collision detection and deformable objects. *Comput. Animat. Virtual Worlds*, 16(3-4):189–200, 2005.

[45] Markus Hadwiger, Christoph Berger, and Helwig Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 40, Washington, DC, USA, 2003. IEEE Computer Society.

[46] Markus Hadwiger, Christian Sigg, Henning Scharsach, Katja Buhler, and Markus Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. In *Eurographics 2005*, 2005.

[47] Jan Hardenbergh and Yin Wu. Emissive clipping planes for volume rendering. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches & Applications*, pages 1–1, New York, NY, USA, 2003. ACM Press.

[48] Taosong He, Sidney Wang, and Arie Kaufman. Wavelet-based volume morphing. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 85–92, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[49] R. Herrlinger. *History of Medical Illustration from Antiquity to 1600*. Editions Medicina Rara, 1970.

[50] Gentaro Hirota, Renee Maheshwari, and Ming C. Lin. Fast volume-preserving free form deformation using multi-level optimization. In *SMA '99: Proc. of the fifth ACM symposium on Solid modeling and applications*, pages 234–245, New York, NY, USA, 1999. ACM Press.

[51] B.K.P. Horn. Obtaining shape from shading information. In *The Psychology of Computer Vision*, pages 115–155, 1975.

[52] Jing Hua and Hong Qin. Haptics-based dynamic implicit solid modeling. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):574–586, 2004.

[53] John F. Hughes. Scheduled fourier volume morphing. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 43–46, New York, NY, USA, 1992. ACM Press.

[54] Shoukat Islam, Deborah Silver, and Min Chen. Volume splitting and its applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):193–203, 2007.

[55] Alark Joshi and Penny Rheingans. Illustration-inspired techniques for visualizing time-varying data. In *IEEE Visualization*, page 86, 2005.

[56] Kenneth Joy. Bernstein polynomials. *On-line Geometryc Modeling Notes*, 2000.

[57] Jan Kautz and Hans-Peter Seidel. Hardware accelerated displacement mapping for image based rendering. In *GRIN'01: No description on Graphics interface 2001*, pages 61–70, Toronto, Ont., Canada, Canada, 2001. Canadian Information Processing Society.

[58] R. Keiser, M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Contact handling for deformable point-based objects. In *Proc. Vision, Modeling, Visualization (VMV)*, pages 315–322, November 2004.

[59] Youngihn Kho and Michael Garland. Sketching mesh deformations. In *SI3D '05:Proc. of the 2005 symposium on Interactive 3D graphics and games*, pages 147–154, New York, NY, USA, 2005. ACM Press.

[60] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 255–262, Washington, DC, USA, 2001. IEEE Computer Society.

[61] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *Computer Graphics (Proc. SIGGRAPH 96)*, pages 313–324, 1996.

[62] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 38, Washington, DC, USA, 2003. IEEE Computer Society.

[63] Y. Kurzion and R. Yagel. Continuous and discontinuous deformation using ray deflectors. In *Proceedings of GRAPHICON'96*, pages 102–110, July 1996.

[64] Yair Kurzion and Roni Yagel. Interactive space deformation with hardware-assisted rendering. *IEEE Comput. Graph. Appl.*, 17(5):66–77, 1997.

[65] E. LaMar, B. Hamann, and K. I. Joy. A magnification lens for interactive volume visualization. In *Proc. IEEE Pacific Conference on Computer Graphics and Applications*, pages 223–231, 2001.

[66] A. Lee, H. Moreton, and H. Hoppe. Displaced subdivision surfaces. In *Computer Graphics (Proc. SIGGRAPH 2000)*, pages 85–94. ACM Press, 2000.

[67] Apostolos Lerios, Chase D. Garfinkle, and Marc Levoy. Feature -based volume metamorphosis. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 449–456, New York, NY, USA, 1995. ACM Press.

[68] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, 1988.

[69] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[70] Wei Li, Klaus Mueller, and Arie Kaufman. Empty space skipping and occlusion clipping for texture-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 42, Washington, DC, USA, 2003. IEEE Computer Society.

[71] J. A. Little, D. L. G. Hill, and D. J. Hawkes. Deformations incorporating rigid structures. In *MMBIA '96: Proceedings of the 1996 Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA '96)*, page 104, Washington, DC, USA, 1996. IEEE Computer Society.

[72] A. Liu, F. Tendick, K. Cleary, and C. Kaufmann. A survey of surgical simulation: applications, technology and education. *Presence*, 12(6), December 2003.

[73] J. R. Logie and J. W. Patterson. Inverse displacement mapping in the general case. *Computer Graphics Forum*, 14(5):261–273, 1995.

[74] W. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, 1987.

[75] Ron MacCracken and Kenneth I. Joy. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH '96: Proc. of the 23rd annual conference on Computer graphics and interactive techniques*, pages 181–188, New York, NY, USA, 1996. ACM Press.

[76] Michael J. McGuffin, Liviu Tancau, and Ravin Balakrishnan. Using deformations for browsing volumetric data. In *Proceedings of IEEE Visualization (VIS) 2003*, pages 401–408, October 2003.

[77] Wouter Mollemans, Filip Schutyser, Johan Van Cleynenbreugel, and Paul Suetens. Tetrahedral mass spring model for fast soft tissue deformation. In *Proc. International Symposium on Surgery Simulation and Soft Tissue Modeling*, pages 145–154, 2003.

[78] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 49–54, San Antonio, Texas, 2002.

[79] M. Müller and M. Gross. Interactive virtual materials. In *Proc. Graphics Interface (GI 2004)*, pages 239–246, May 2004.

[80] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM Transactions on Graphics*, 24(3), 2005.

[81] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Cross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *Proc. ACM SIGGRAPH Symposium on Computer Animation*, pages 141–151, 2004.

[82] A. Nealen, M. Muller, R. Keiser, E. Boxerman, and M.Carlson. Physically based deformable models in computer graphics. In *Eurographics STAR Report*, 2005.

[83] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1142–1147, New York, NY, USA, 2005. ACM Press.

[84] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. In *SIGGRAPH '02: Proc. of the 29th annual conference on Computer graphics and interactive techniques*, pages 291–294, New York, NY, USA, 2002. ACM Press.

[85] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *SIGGRAPH '99: Proc. of the 26th annual conference on Computer graphics and interactive techniques*, pages 137–146, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[86] Kyoungsu Oh, Hyunwoo Ki, and Cheol-Hi Lee. Pyramidal displacement mapping: a gpu based artifacts-free ray tracing through an image pyramid. In *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 75–82, New York, NY, USA, 2006. ACM Press.

[87] M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *Computer Graphics (Proc. SIGGRAPH 2000)*, pages 359–368. 2000.

[88] Shigeru Owada, Frank Nielsen, Makoto Okabe, and Takeo Igarashi. Volumetric illustration: designing 3d models with internal textures. *ACM Trans. Graph.*, 23(3):322–328, 2004.

[89] M. Pharr and P. Hanrahan. Geometry caching for ray-tracing displacement maps. In *Proc. Eurographics Rendering Workshop*, pages 31–40, 1996.

[90] Fabio Policarpo and Manuel M. Oliveira. Relief mapping of non-height-field surface details. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 55–62, New York, NY, USA, 2006. ACM Press.

[91] Serban D. Porumbescu, Brian Budge, Louis Feng, and Kenneth I. Joy. Shell maps. *ACM Trans. Graph.*, 24(3):626–633, 2005.

[92] Ari Rappoport, Alla Sheffer, and Michel Bercovier. Volume-preserving free-form solid. In *SMA '95: Proc. of the third ACM symposium on Solid modeling and applications*, pages 361–372, New York, NY, USA, 1995. ACM Press.

[93] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-state rasterization. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 109–118, 2000.

[94] C. Rezk-Salama, M. Scheuering, G. Soza, and G. Greiner. Fast volumetric deformation on general purpose hardware. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 17–24, New York, NY, USA, 2001. ACM Press.

[95] Harry Robin. *The scientific image: From cave to computer.* H.N. Abrams, Inc., 1992.

[96] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[97] Detlef Ruprecht and Heinrich Müller. Image warping with scattered data interpolation. *IEEE Comput. Graph. Appl.*, 15(2):37–43, 1995.

[98] Stanley E. Scharoff, Alex Pentland, Irfan Essa, Martin Friedmann, and Bradley Horowitz. The thingworld modeling system: virtual sculpting by modal forces. *SIGGRAPH Computer Graphics*, 24(2):143–144, 1990.

[99] G. Schaufler and M. Priglinger. Efficient displacement mapping by image warping. In *Proc. Eurographics Rendering Workshop*, pages 175–186, 1999.

[100] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proc. of the 13th annual conference on Computer graphics and interactive techniques*, pages 151–160, New York, NY, USA, 1986. ACM Press.

[101] Guy Sela, Jacob Subag, Alex Lindblad, Dan Albocher, Sagi Schein, and Gershon Elber. Real-time haptic incision simulation using fem-based discontinuous free form deformation. In *SPM '06: Proc. of the 2006 ACM symposium on Solid and physical modeling*, pages 75–84, New York, NY, USA, 2006. ACM Press.

[102] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, New York, NY, USA, 1998. ACM Press.

[103] Christian Sigg, Ronald Peikert, and Markus Gross. Signed distance transform using graphics hardware. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 12, Washington, DC, USA, 2003. IEEE Computer Society.

[104] V. Singh and D. Silver. Interactive volume manipulation with selective rendering for improved visualization. In *Proc. IEEE Symposium on Volume Visualization and Graphics '04*, pages 95–102. IEEE Computer Society, 2004.

[105] V. Singh, D. Silver, and N. Cornea. Real-time volume manipulation. In *Proc. Volume Graphics '03*, pages 45–51. ACM Press, 2003.

[106] Raymond Smith, Wei Sun, Adrian Hilton, and John Illingworth. Layered animation using displacement maps. In *CA '00: Proceedings of the Computer Animation*, page 146, Washington, DC, USA, 2000. IEEE Computer Society.

[107] Mario Costa Sousa, David S. Ebert, Don Stredney, and Nikolai A. Svakhine. Illustrative Visualization for Medical Training. In László Neumann, Mateu Sbert Casasayas, Bruce Gooch, and Werner Purgathofer, editors, *Proceedings of the First Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging 2005 (May 18–20, 2005, Girona, Spain)*, pages 201–208, Aire-la-Ville, Switzerland, 2005. Eurographics Association.

[108] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware-based ray casting. In *Proc. Volume Graphics 2005*, pages 187–195, 2005.

[109] Denis Steinemann, Miguel A. Otaduy, and Markus Gross. Fast arbitrary splitting of deforming objects. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2006*, 2006.

[110] Avneesh Sud, Naga Govindaraju, Russell Gayle, and Dinesh Manocha. Interactive 3d distance field computation using linear factorization. In *SI3D '06: Proceedings of the*

2006 symposium on Interactive 3D graphics and games, pages 117–124, New York, NY, USA, 2006. ACM Press.

[111] Nikolai Svakhine, David S. Ebert, and Don Stredney. Illustration motifs for effective medical volume illustration. *IEEE Comput. Graph. Appl.*, 25(3):31–39, 2005.

[112] G. Szekely, Ch. Brechbuhler, R. Hutter, A. Rhomberg, N. Ironmonger, and P. Schmid. Modelling of soft tissue deformation for laparoscopic surgery simulation. *Medical Image Analysis*, 4(1):57–66, 2000.

[113] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *SIGGRAPH '88: Proc. of the 15th annual conference on Computer graphics and interactive techniques*, pages 269–278, New York, NY, USA, 1988. ACM Press.

[114] D'Arcy Wentworth Thompson. *On Growth and Form*. Cambridge University Press, 1961.

[115] Christian Tietjen, Tobias Isenberg, and Bernhard Preim. Combining silhouettes, surface, and volume rendering for surgery education and planning. In *Eurographics / IEEE VGTC Symposium on Visualization (EUROVIS 2005)*, pages 303–310, Leeds, UK, June 1-3 2005.

[116] S. M. F. Treavett and M. Chen. Pen-and-ink rendering in volume visualisation. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 203–210, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[117] Ivan Viola, Armin Kanitsar, and Meister Eduard Groller. Importance-driven volume rendering. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 139–146, Washington, DC, USA, 2004. IEEE Computer Society.

[118] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Vector field based shape deformations. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1118–1125, New York, NY, USA, 2006. ACM Press.

[119] Gunther von Hagens' Bodyworlds, 2005.

[120] S.J. Walton and M.W. Jones. Volume wires : A framework for empirical non-linear deformation of volumetric datasets. *Journal of WSCG 2006*, 14, 2006.

[121] L. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum. Generalized displacement maps. In *Proc. Eurographics Symposium on Rendering*, 2004.

[122] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum. View-dependent displacement mapping. *ACM Transactions on Graphics (Proc. SIGGRAPH 2003)*, 22(3):334–339, 2003.

[123] Lujin Wang, Ye Zhao, Klaus Mueller, and Arie E. Kaufman. The magic volume lens: An interactive focus+context technique for volume rendering. In *IEEE Visualization*, page 47, 2005.

[124] Daniel Weiskopf, Klaus Engel, and Thomas Ertl. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Trans. Vis. Comput. Graph.*, 9(3):298–312, 2003.

[125] Rüdiger Westermann and Thomas Ertl. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 169–177, New York, NY, USA, 1998. ACM Press.

[126] Rüdiger Westermann and Christof Rezk-Salama. Real-time volume deformations. *Comput. Graph. Forum*, 20(3), 2001.

[127] David F. Wiley, Nina Amenta, Dan A. Alcantara, Deboshmita Ghosh, Yong Joo Kil, Eric Delson, Will Harcourt-Smith, Katherine St. John, F. James Rohlf, and Bernd Hamann. Evolutionary morphing. In *IEEE Visualization*, page 55, 2005.

[128] A. Witkin and W. Welch. Fast animation and control of nonrigid structures. *Computer Graphics (Proc. SIGGRAPH 90)*, 24(4):243–252, 1990.

[129] Xunlei Wu, Michael S. Downes, Tolga Goktekin, and Frank Tendick. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. *Comput. Graph. Forum*, 20(3), 2001.

[130] Xunlei Wu and Frank Tendick. Multigrid integration for interactive deformable body simulation. In *Proc. ISMS*, pages 92–104, 2004.

[131] Yi Xu and Yee-Hong Yang. Object representation using 1d displacement mapping. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 33–40, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[132] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph laplacian. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 496–503, New York, NY, USA, 2005. ACM Press.

[133] Q. Zhu, Y. Chen, and A. Kaufman. Real-time biomechanically-based muscle volume deformation using FEM. In *Proc. Eurographics 1998*, pages 275–284, 1998.

[134] Y. Zhuang and J. F. Canny. Real-time global deformations. In *Proc. 4th International Workshop on Algorithmic Foundations of Robotics*, 2000.

[135] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH '97: Proc. of the 24th annual conference on Computer graphics and interactive techniques*, pages 259–268, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

Vita

Carlos D. Correa

2007	Ph. D. in Electrical and Computer Engineering, Rutgers University
2003	M. Sc. in Electrical and Computer Engineering, Rutgers University
1998	B. Sc. in Computer Engineering, EAFIT University, Colombia.

2006-2007	Teaching Assistant, Department of Electrical and Computer Engineering.
2005-2007	Research Assistant, Visualization Group
2001-2004	Research Assistant, Collaboration Group, CAIP Center, Rutgers U niversity
1998-2000	Research Assistant and Consultant. Virtual Reality Lab and Conexiones Project. EAFIT University

Relevant Publications and Presentations

• **Carlos D. Correa**, Deborah Silver and Min Chen. *Feature Aligned Volume Manipulation for Illustration and Visualization*. IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2006), vol. 12, no. 5, Sept.-Oct. 2006, pp. 1069-1076.

• M. Chen, **C.Correa**, S. Islam, M.W. Jones, P.Y.Shen, D.Silver, S.J.Watson and P.J. Willis. *Manipulating, Deforming and Animating Sampled Object Representations*. Accepted for publication in : Computer Graphics Forum, 2007.

• **Carlos D. Correa**, Deborah Silver and Min Chen. *Discontinuous Displacement Mapping for Volume Graphics*. Eurographics/IEEE VGTC Workshop on Volume Graphics 2006, VG'06, Boston, MA, 30-31 July, 2006, pp. 9-16.

• **Carlos D. Correa** and Deborah Silver. *Dataset Traversal with Motion-Controlled Transfer Functions*. Proceedings of IEEE Visualization 2005. Minneapolis, Min. 23-28 Oct. 2005, pp. 359-366.

• M. Chen, **C. Correa**, S. Islam, M.W. Jones, P.-Y. Shen, D. Silver, S. J. Watson, P.J. Willis (In Alphabetical Order). *Deforming and Animating Discretely Sampled Object Representations*. Eurographics 2005, State of the Art Reports (STAR), August 29 - September 2, 2005, pp. 113-140.

• **Carlos Correa**. *"Hands-in" Visualization: An Active Approach for Interactive Manipulation of Volumetric Objects*. IBM Graphics and Visualization Student Symposium (Invited Talk). Dec 6, 2005.