SIMPLIFICATION, APPROXIMATION AND DEFORMATION OF LARGE MODELS

TERESA PARADINAS SALSÓN

Dipòsit legal: GI-1413-2011
http://hdl.handle.net/10803/51293

Universitat de Girona

PhD Thesis

Simplification, Approximation and Deformation of Large Models

Teresa Paradinas Salsón

2011

Doctorat del Programa Oficial de Postgrau en Tecnologia

PhD advisor

Dr. Narcís Coll Arnau

Memòria presentada per a optar al títol de Doctor per la Universitat de Girona

Universitat de Girona

Dr. Narcís Coll Arnau, Titular d'Universitat del Departament d'Informàtica i Matemàtica Aplicada de la Universitat de Girona,

CERTIFICA:

Que aquest treball titulat "Simplification, Approximation and Deformation of Large Models", que presenta Teresa Paradinas Salsón per a l'obtenció del títol de Doctor, ha estat realitzat sota la meva direcció.

Signatura

Dr. Narcís Coll Arnau

Girona, 5 de Juliol de 2011

a la mama, al papa, al tato i a la tata,

a en Marc, el meu titi.

Abstract

Computer graphic applications are demanding more challenging requirements. The complexity of geometric models used in interactive applications is constantly increasing due to the advances in the data acquisition systems and the need for a convincing level of realism. In this context, geometry processing techniques have become essential for the efficient management of such models by allowing a trade-off between complexity and performance. The central focus of this thesis is the *simplification, approximation and deformation of large models*. We propose new robust and efficient techniques that represent a step forward with respect to the state of the art.

First, we present an edge-collapse-based simplification method that provides an accurate low-resolution approximation from a multi-chart textured model that guarantees geometric fidelity and correct preservation of the appearance attributes. Then, we introduce a new mesh structure called *Compact Model* (CM) that approximates dense triangular meshes of arbitrary topology. Sharp features are well preserved, adaptive reconstructions are possible, textured models are supported and the whole approximation process can be completely parallelized. The requirement of a flexible framework for easier mesh editing leads us to design a new space deformation technique based on a multi-level system of cages enclosing the model. The proposed deformation scheme, called $*Cages$, preserves the smoothness of the mesh between neighbouring cages and is extremely versatile, allowing the use of heterogeneous sets of coordinates and different levels of deformation thus obtaining fast evaluations and a reduced memory footprint. Finally, a hybrid approach for the deformation of large meshes that takes advantage of the previously developed methods. We provide a CM with the ability to be deformable, making it possible to apply any of the existing deformation techniques on large models. This powerful and valuable system leads to high quality approximations of deformed models with a reduced memory footprint and a high performance.

Resum

Les aplicacions de gràfics per computador tenen requisits cada vegada més exigents. La complexitat dels models geomètrics utilitzats en aplicacions interactives és cada vegada més gran degut als avenços en els sistemes d'adquisició de dades i la necessitat de tenir un nivell de realisme convincent. En aquest context, les tècniques de processat de geometria són essencials per a la gestió eficient d'aquests models, permetent aconseguir un equilibri entre complexitat i rendiment. L'enfoc central d'aquesta tesis és la simplificació, l'aproximació i la deformació de models de grans dimensions. Proposem noves tècniques robustes i eficients que representin un pas endavant respecte l'estat de l'art.

En primer lloc, presentem un mètode de simplificació basat en la contracció d'arestes que proporciona una aproximació precisa de baixa resolució d'un model amb un atles de textures que garanteix fidelitat geomètrica i una correcta preservació de la seva aparença. A continuació, introduïm una nova estructura de dades per malles anomenada *Compact Model* (CM) que permet aproximar malles triangulars denses de tipologia arbitraria. El mètode desenvolupat preserva correctament els trets més distintius del model, permet reconstruccions adaptatives, suporta models texturats i, a més a més, tot el procés d'aproximació pot ser paral·lelitzable per complet. L'exigència d'un entorn flexible que faciliti l'edició de malles ens va portar a dissenyar una nova tècnica de deformació de l'espai basada en un sistema de caixes multi-nivell que engloba al model. L'esquema de deformació proposat, denominat $*Cages$, conserva la suavitat de la malla entre caixes veïnes i és extremadament versàtil, ja que permet l'ús de conjunts heterogenis de coordenades i diferents nivells de deformació, obtenint, per tant, temps d'execució baixos i un consum de memòria reduït. Per últim, proposem un mètode híbrid per la deformació de models complexos a partir dels mètodes desenvolupats anteriorment. Dotem a un CM de la capacitat de ser deformable permetent que es pugui aplicar qualsevol de les tècniques de deformació existents. Aquesta poderosa i valuosa eina permet obtenir aproximacions d'alta qualitat dels models deformats amb una memòria reduïda i un alt rendiment.

Resumen

Las aplicaciones de gráficos por computador tienen requisitos cada vez más exigentes. La complejidad de los modelos geométricos utilizados en aplicaciones interactivas es cada vez mayor debido a los avances en los sistemas de adquisición de datos y la necesidad de un nivel de realismo convincente. En este contexto, las técnicas de procesado de geometría son esenciales para la gestión eficiente de dichos modelos, permitiendo conseguir un equilibrio entre complejidad y rendimiento. El enfoque central de esta tesis es la simplificación, la aproximación y la deformación de modelos de gran tamaño. Proponemos nuevas técnicas robustas y eficientes que representan un paso adelante respecto al estado del arte.

En primer lugar, presentamos un método de simplificación basado en la contracción de aristas que proporciona una aproximación precisa de baja resolución de un modelo con atlas de texturas que garantiza fidelidad geométrica y una correcta preservación de su apariencia. A continuación, introducimos una nueva estructura de datos para mallas llamada *Compact Model* (CM) que permite aproximar mallas triangulares densas de topología arbitraria. El método desarrollado conserva correctamente los rasgos más distintivos del modelo, permite reconstrucciones adaptativas, soporta modelos texturados y, además, todo el proceso de aproximación puede ser paralelizado por completo. La exigencia de un entorno flexible que facilite la edición de mallas nos llevó a diseñar una nueva técnica de deformación del espacio basada en un sistema de cajas multi-nivel que engloba al modelo. El esquema de deformación propuesto, denominado $*Cages$, conserva la suavidad de la malla entre cajas vecinas y es extremadamente versátil, ya que permite el uso de conjuntos heterogéneos de coordenadas y diferentes niveles de deformación, obteniendo, por tanto, tiempos de ejecución bajos y un consumo de memoria reducido. Por último, proponemos un método híbrido para la deformación de modelos complejos a partir de los métodos desarrollados anteriormente. Dotamos a un CM la capacidad de ser deformable permitiendo que se pueda aplicar cualquiera de las técnicas de deformación existentes. Esta poderosa y valiosa herramienta permite obtener aproximaciones de alta calidad de los modelos deformados con una memoria reducida y un alto rendimiento.

Acknowledgements

First of all, I would like to apologize for express my sincere appreciation to all the people who have made this thesis possible in my mother language.

Finalment, després de moltes de dies de dedicació i esforç, s'ha pogut obtenir com a resultat la tesi que es presenta en aquest document. Tot això no hagués estat realment possible sense el suport de tot un conjunt de persones a les quals, és poc menys que impossible, donar-les-hi les gràcies i dedicar-les-hi, sincerament, tota la feina feta.

En primer lloc, voldria començar per en Narcís Coll, el meu director de tesi. Per la teva paciència, el teu temps de dedicació, els teus consells i, en general, tota la teva ajuda i suport que han permès tirar endavant aquest gran projecte. Un llarg camí és el que he recorregut al teu costat, des de professor a primer de carrera, passant per ser director de tots els meus projectes universitaris (enginyeria tècnica, superior i master tesis). Durant tots aquests anys hem viscut moments de tot tipus que units han fet que acabéssim formant un gran equip. Gràcies, no només per haver dirigit, supervisat i aportat idees per assolir aquesta tesis, sinó per haver-me "aguantat" en els moments més durs, animat quan estava baixa d'ànims i confiat en la meva feina.

M'agradaria donar les gràcies especialment a en Toni Sellarès que, tot i que des de la distància, ha seguit l'evolució d'aquest llarg procés. Com a cap del nostre modest grup de geometria, fent carinyosament la funció de "papa de família", t'has interessat per l'estat del projecte aconsellant-me, sempre que ho he necessitat, amb el teu gran coneixement i experiència, oferint-me la teva ajuda en qualsevol situació i animant-me en tot moment, sempre mostrant-te molt proper i sincer.

Seria impossible no mencionar als que han estat els meus companys de despatx durant tot aquest llarg viatge, en Narcís, la Marité, la Marta, en Yago, en Nacho i les dues últimes incorporacions, en Fran i l'Isma. Amb vosaltres he compartit molts moments. Plegats hem viscut penes i alegries, bones i males notícies, *papers* rebutjats i *papers* acceptats, ... Voldria donar-vos les gràcies per l'ajut que m'heu donat, les llargues xerrades que hem

compartit i per mostrar-me el vostre suport sempre que us ho he demanat, tant a nivell de coneixements com a nivell personal.

A més a més, voldria agrair en especial el suport que m'han brindat en Fran i l'Isma, uns grans companys i amics. Després de tot el temps que portem junts us heu convertit en companys de "teràpia", en confidents. Vosaltres heu fet que continués endavant en moments complicats, fent-me un cop de mà sempre que ho he necessitat, ajudant-me a prendre decisions i oferint-me el vostre recolzament i consell.

A tots els diversos companys amb els que he compartit distretes xerrades durant els dinars, interessants converses de passadís i grans moments de diversió, m'agradaria donar-vos les gràcies per haver fet més amens i distesos els dies de treball. Com a "company" i membre del grup, no em voldria oblidar de mencionar a en Gus que, amb les seves xerrades representatives, el seu toc distintiu i la seva "hiperactivitat" característica, ha sigut una de les persones més presents en aquesta tesi, especialment amb la interessant col·laboració que finalment hem fet al llarg de la última etapa del procés.

No podria deixar de fer un agraïment especial a tota la meva família que ha viscut indirectament tota l'evolució d'aquesta tesi. Als meus pares, germans, cunyats i, en especial, als més petits de tots, els meus nebots, perquè gràcies a vosaltres he pogut assolir molts dels meus objectius a la vida. Moltes gràcies per recolzar-me en les meves decisions, per distreure'm dels meus maldecaps, per regalar-me moments molt especials, per preocupar-vos per mi i, sobretot, per tot el vostre suport incondicional.

Em seria impossible acabar sense dedicar aquesta tesi especialment a en Marc, la persona que més ha "patit" els efectes secundaris del procés. Perquè, encara que tots dos sabem que no ha estat gens fàcil, en aquest tram del camí que estem recorrent junts encapçalat per un gran *projecte comú*, m'has donat moltes forces per continuar endavant, per lluitar pels meus objectius i per aconseguir molts dels reptes que m'he proposat. Sense tu no hagués estat possible, ni molt menys, arribar fins on he arribat. Moltes gràcies per aguantar les meves dèries, els meus canvis d'humor, els meus "rotllos" sobre el mètode que per fi funciona, el *bug* que encara no he trobat, el *paper* que m'han rebutjat, ... Per estar sempre al meu costat, tant en els bons moments com en els que no ho són tant, per no deixar-me mai de recolzar i per acompanyar-me en aquest viatge ple d'il·lusions que entre tots dos estem construint. Espero poder-te aportar tant com tu ho fas amb mi.

Published Work

Accurate Simplification of Multi-Chart Textured Models

N. Coll and T. Paradinas

Computer Graphics Forum

September 2010

Volume 29, Issue 6, pages 1842-1853

DOI: 10.1111/j.1467-8659.2010.01652.x

Compact Models

N. Coll and T. Paradinas

Computer Graphics Forum

March 2011

Volume 30, Issue 1, pages 187-198

DOI: 10.1111/j.1467-8659.2010.01842.x

∗***Cages for mesh deformation***

T. Paradinas, F.Gonzalez, N. Coll and G. Patow

Submitted to SIGGRAPH Asia 2011

Contents

Chapter 1

Introduction

Geometry processing emerged to solve geometric modelling problems that arose during the manipulating of complex meshes. The efficient acquisition, representation, optimization, editing, and simulation of geometric objects are the main focus of this huge area of research.

For decades, computer graphic designers have created digital 3D models by using complex interactive tools to reproduce real-world objects or imaginary ones. Nowadays, digital representations of real surfaces can be obtained automatically with various acquisition devices such as 3D scanners. These new fast and accurate data sources increase the surface resolution by several orders of magnitude, providing higher precision to applications that require digital surfaces (see Figure 1.1). Fundamental advances in 3D modelling, simulation, and data capture technologies continually increase the complexity of geometric models used in interactive applications. Industrial CAD models of airplanes, ships, production plants, and buildings; geographic information systems; oil and gas exploration; medical imaging; virtual prototyping; scanned 3D models; games and movies; unorganized information spaces; and high-end scientific simulations are domains that create highly complex digital models which provide vast databases. Moreover, current applications with these complex geometrical models are demanding a number of challenging requirements, like real-time interaction, mainly with respect to geometry but increasingly in terms of appearance, illumination, visibility, and other features that create a convincing level of realism.

Unfortunately, the richness of the geometry produced by these media represents even more complex models and very densely sampled meshes containing hundreds of millions and, even, billions of 3D primitives that cannot be used directly in real-time visualization or complex numerical simulation. Moreover, the representation of such models results in files of substantial size, which are expensive to store and slow to transmit. In recent

Figure 1.1: Polygonal mesh obtained from the scanning of the Max Planck bust.

years, hardware devices and computer architectures have evolved enormously. However, the complexity of these highly detailed digital models grows faster than the ability of our graphics hardware to process, edit, and render them interactively. The capacity of the mesh rendering pipeline is a limited resource in many graphic applications, which has motivated researchers to find suitably compact mesh representations. A trade-off exists between the accuracy with which a surface is modeled and the amount of time required to process it. In general terms, there is tension between realism and speed, between fidelity and frame rate, between complexity and performance. Despite many advances in interactive modelling, the management of high quality geometric models is still a very time-consuming task that requires technical skills.

Therefore, all major computer graphic applications can benefit from geometry processing techniques, including: computer-aided design, shape editing, physical simulation, virtual reality, medical imaging, architecture, engineering, archaeological study, special effects, computer animation and video games. Since many problems arising in these highly diverse fields follow the same fundamental geometric principles, geometry processing plays an important role in a large variety of applications and it has become a fascinating research field with a high potential impact. Consequently, a lot of research in recent years has focused on the creation of new efficient data structures and algorithms for geometry processing.

Although many advances have been made in the geometry processing field, the interactive visualization and handling of such large models is still a challenge in computer graphics and visualization. Geometric models, often acquired using 3D scanning techniques, have to undergo post-processing and shape optimization techniques before being used in production. Scalable algorithms for geometry processing have been explored to solve the involved

non-trivial numerical problems in an efficient manner. In some cases, GPU architectures are used for their efficient implementation. An extensive survey of geometry modelling and processing techniques can be found in [BPR+06]. In the next paragraphs we summarize some of the main geometry processing techniques.

Surface Representation. The design of suitable data structures is needed for the efficient processing of different kinds of geometric objects. The data to be processed are geometric shapes, so each specific problem requires the use of the best shape representation to enable efficient access to the most relevant information. Surface representations can be classified in two main classes: *explicit* surface representations and *implicit* surface representations. Both representations have their own strengths and weaknesses, such that for each geometric problem the better suited one should be chosen.

Model Repair. Model repair techniques consist of removing artefacts from a geometric model to produce an output model suitable for further processing by subsequent applications which require certain quality requirements as an input. A single algorithm cannot be applicable since depending on the problem addressed, the model, the artefact and suitable concepts have to be redefined. Most model repair algorithms can roughly be classified as being either *surface oriented* or *volumetric*. On one hand, surface-oriented algorithms operate directly on the input data and try to explicitly identify and resolve artefacts on the surface. Removing gaps, closing holes and locating and resolving intersections are some surface repair operations that allow meshes to be *clean*. On the other hand, volumetric algorithms convert the input model into an intermediate volumetric representation from which the output model is then extracted. In general terms, model repair is needed in a wide range of applications.

Mesh Smoothing. Mesh smoothing is a key tool in geometry processing with many applications. Smoothing methods distinguish between two different goals: *denoising* and *fairing*. The first one consists of smoothing out the high frequency noise usually present in scanned models, such as small perturbations of the vertex positions, in such a way that global shapes, or the low frequency components, are preserved. Additionally, certain surface features like sharp edges and corners have to be preserved without "blurring". The second one corresponds to the design of high-quality fair surfaces. The fairing process results in surfaces that must satisfy certain aesthetic requirements. In general terms, smoothing techniques are used to reduce noise in scanned surfaces. After they are applied more regular triangulations are usually obtained.

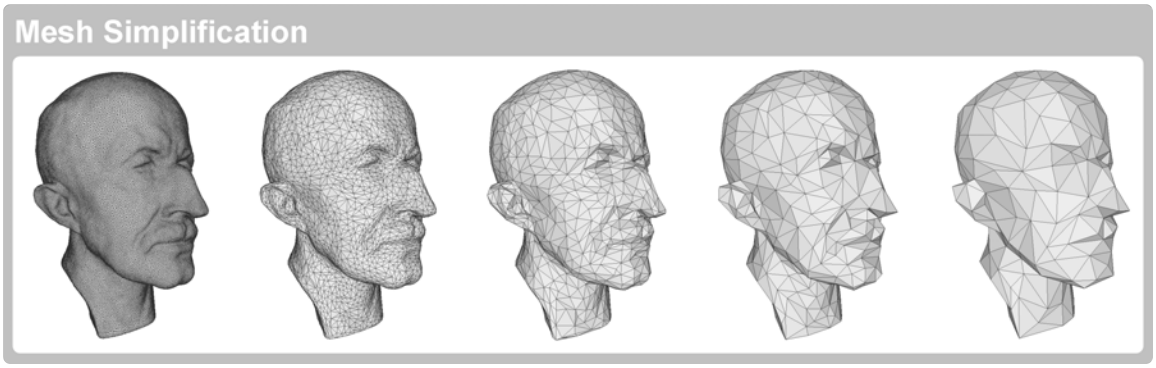

Figure 1.2: Different levels of detail of the Max Planck model obtained after applying a mesh simplification method.

Parameterization. The main goal of parameterization techniques consists in establishing bijective mappings between surfaces and parametric domains. Parameterization is behind a large number of applications in computer graphics and geometry processing such as texturing, compression, scattered data approximation, and remeshing. The main focus of parameterization methods is the reduction of parametric distortion, a property used to classify them. Angles, areas, and distances are measures used to prove the quality of the resulting parametrizations.

Mesh Simplification. Mesh simplification also known as mesh decimation, is a popular, much researched topic in geometry processing. It describes a set of algorithms that, given a polygonal mesh, reduces the number of geometric elements used to represent it, while still retaining important geometric and topological characteristics (see Figure 1.2). The vertex positions of the simplified mesh can be obtained as a subset of the original set of vertex positions, as a set of weighted averages of original vertex positions, or by resampling the original piecewise linear surface. The various approaches in the literature can be classified into vertex clustering, incremental and resampling decimation algorithms, respectively. The preservation of specific properties of the original model is usually controlled by some user-defined quality criteria, typically geometric distance or visual appearance. There are many applications for decimation algorithms. Due to the enormous complexity of meshes acquired by 3D scanning, mesh decimation techniques can be used to adjust the complexity of a geometric dataset with a controlled error. In this way, complex models can be used on computers with varying computing performance.

Remeshing. The shape of the geometry elements present in a model is an important aspect in different applications. Remeshing is a key technique for mesh quality preserva-

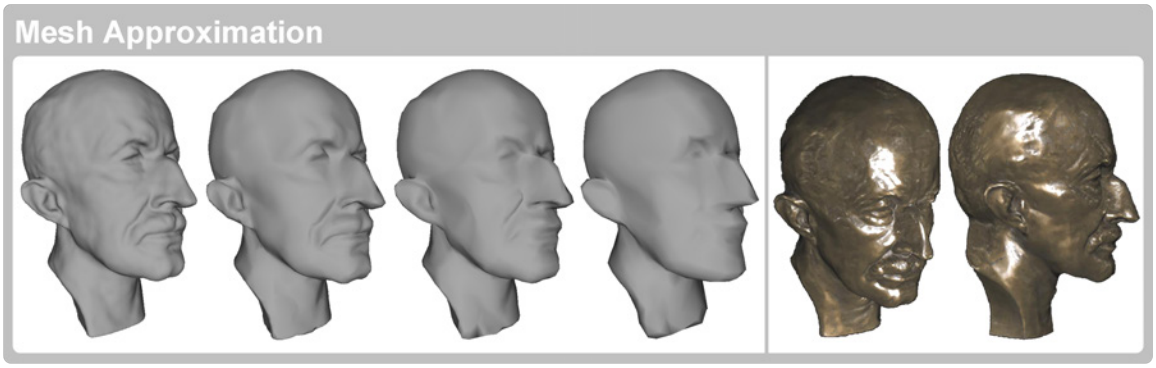

Figure 1.3: Approximations of the Max Planck at different resolutions. Adding appearance attributes to the mesh allows more realistic approximations to be obtained.

tion in many geometric modelling algorithms, for instance animation, shape editing, morphing and numerical simulation. Given a mesh, remeshing consists of computing another mesh whose elements satisfy some quality requirements. The goal of such techniques is to reduce the complexity of an input mesh subject to certain quality criteria and improve the quality of a model according to the downstream application. Depending on the application, different quality criteria and requirements can be used.

Mesh Approximation. Mesh approximation techniques are useful to obtain different versions of a given model (see Figure 1.3). According to the applications, they can be used to obtain a given geometric complexity with the best geometric approximation, to bound the maximum geometric deviation or to preserve the local or global volume.

Shape Deformation. Mesh editing is an active research field in computer graphics. It often requires that the global shape deform in a user-specified way while geometric details are well preserved (see Figure 1.4). Intuitive and interactive shape deformation is a useful tool in a variety of applications in computer modelling and animation. Several deformation schemes have been proposed, from global to local. Existing mesh deformation methods include: free-form deformation, multiresolution, RBF-based mesh deformation, curve-based deformation, skeleton and physical simulation. Freeform deformations allow a given surface to be deformed smoothly. Concerning global deformations, fine surface details are not deformed in a natural manner. Multiresolution hierarchies provide intuitive detail handling, which can enhance any freeform deformation technique. Interactive shape editing is achievable for large models since the reduced model and the harmonic fields can be computed previously and reused during an editing session. In short, mesh deformation is a valuable tool for geometry mod-

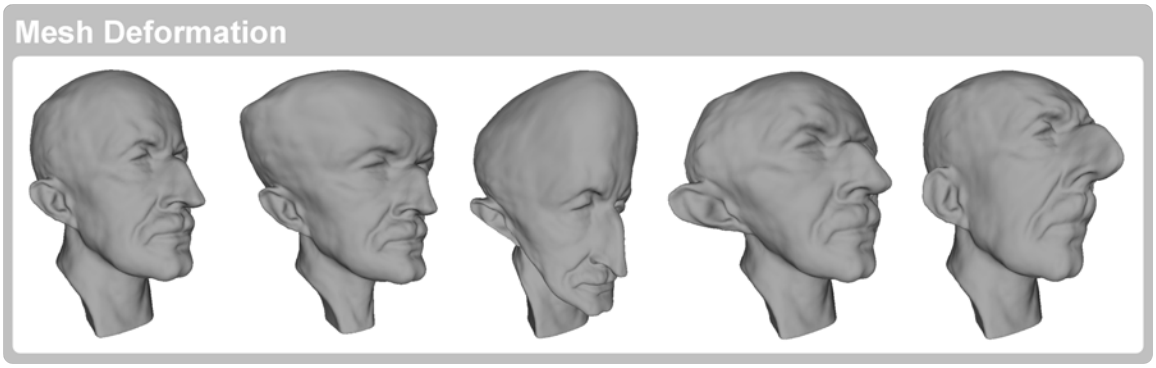

Figure 1.4: Different deformed versions of the Max Planck model obtained after applying a mesh deformation method.

elling and computer animation, since it provides a convenient way to edit the original mesh to meet various design requirements.

In this thesis, we propose new algorithms for the geometry processing of large models, designed to obtain a trade-off between accuracy, versatility, and time and memory efficiency.

1.1 Contributions

Throughout our research work, we treated and solved various problems related to geometry processing. Focusing on the simplification, approximation and deformation of large models, we present several original contributions in the fields of efficient processing, management and editing techniques. Next, we classify them according to the corresponding geometry processing field:

Simplification. The highly detailed surface meshes produced by scanning and acquisition methods need multi-chart parameterizations to reduce stretching and distortion. From these complex shape surfaces, high-quality approximations are automatically generated by using surface simplification techniques. Multi-chart textures hinder the quality of the simplification of these techniques for two reasons: either the chart boundaries cannot be simplified leading to a lack of geometric fidelity; or texture distortions and artefacts appear near the simplified boundaries. We present an edge-collapse based simplification method that provides an accurate, low-resolution approximation from a multi-chart textured model. For each collapse, the model is reparameterized by local bijective mappings to avoid texture distortions and chart

boundary artefacts on the simplified mesh due to the geometry changes. To better apply the appearance attributes and to guarantee geometric fidelity and thereby preserve the curved features of the model, we drive the simplification process with the quadric error metrics weighted by a local area distortion measure.

Approximation. Development of approximation techniques for highly detailed surfaces is one of the challenges faced today. We introduce a new mesh structure that allows dense triangular meshes of arbitrary topology to be approximated. The structure is constructed from the information gathered during a simplification process. Each vertex of the simplified model collects a neighbourhood of input vertices. Then, each neighbourhood is fitted by a set of local surfaces taking into account the sharp features detected. The simplified model plus the parameters of these local surfaces, conveniently stored in a file, is what we call a *Compact Model* (CM). The input model can be approximated from its CM by refining each triangle of the simplified model. The main feature of our approach is that each triangle is refined by blending the local surfaces at its vertices, which can be done independently of the others. Consequently, adaptive reconstructions are possible, textured models are supported and the whole approximation process can be completely parallelized.

Deformation. Cage-based deformation has been one of the main approaches for mesh deformation in recent years, with a lot of interesting and active research. The main advantages of cage-based deformation techniques are their simplicity, relative flexibility and speed. However, up to now there is no widely accepted solution that provides both user control at different levels of detail, and high quality deformations. We present $*$*Cages* (star-cages), a work obtained as a result of a very close collaboration with Francisco Gonzalez and Gustavo Patow, two fellow research group members. $*$Cages is a multi-level cage system which allows the use of multiple cages enclosing the model for easier manipulation while preserving the smoothness of the mesh in the transitions between them. It represents a significative step forward with respect to traditional coordinate systems. The proposed deformation scheme is extremely flexible and versatile, allowing the use of heterogeneous sets of coordinates and different levels of deformation, going from a whole-model deformation to a very localized one. This allows faster evaluation and a much reduced memory footprint.

Deformation of large models. Finally, we propose a hybrid approach that represents a very useful tool for the deformation of large meshes by taking advantage of the previously developed methods. We provide the CM with the ability to generate deformed approximations. The resulting representation combines the accuracy of the

simplification method, the versatility of the CM representation and the wide deformation knowledge obtained after developing ∗Cages. We make possible to apply any of the existing deformation techniques on large models. High-quality approximations of deformed models with a reduced memory footprint and a high performance can be obtained. Several applications can take advantage of this powerful and valuable representation.

All the methods developed in this thesis were integrated into a framework designed using object-oriented techniques.

1.2 Overview of the thesis

The rest of the thesis is organized as follows:

Chapter 2: **Background**

An introduction to the most relevant background related with the geometry processing techniques studied throughout the thesis is presented. It introduces context of simplification, approximation and deformation of polygonal models.

Chapter 3: **Simplification of Multi-chart Textured Models**

The accurate surface simplification approach developed for multi-chart textured models is presented. Previous work on simplification of surface meshes is reviewed. The performance of the proposed method is described in detail and their good behaviour is proved by providing different experimental results.

Chapter 4: **Compact Models**

Surface approximation of large models is reviewed by introducing related work. Next, we present our new mesh representation by describing both the generation and the reconstruction processes. An exhaustive analysis is performed by giving a wide range of results and a detailed quantitative analysis.

Chapter 5: ∗**Cages for Mesh Deformation**

In this chapter, we introduce ∗Cages, the work resulting from a close research collaboration. The multi-level cage-based approach developed for the manipulation of surface meshes is described here. The flexibility and versatility provided by the proposed deformation scheme are described and proved.

Chapter 6: **Deformable Compact Models**

A hybrid solution resulting of the combination from the previously developed methods culminates the dissertation. The versatile representation proposed allows large models to be manipulated by applying any of the existing deformation methods easily and efficiently.

Chapter 7: **Conclusions and Future work**

Finally, we conclude by providing the conclusions of the thesis, some final remarks and future work proposals.

Chapter 2

Background

This chapter provides an overview of some background material related to the mesh processing field that is used throughout the rest of the dissertation. First, an introduction to surface meshes is given by defining some basic concepts, notations and terminologies (Section 2.1). Then, the pipeline of the graphics hardware is described before presenting the main detailed mapping applications and the required mesh parametrization techniques (Section 2.2). After providing surface and graphics hardware definitions, the need to use level of detail techniques for the management of complex surface models is examined (Section 2.3). Mesh simplification and mesh refinement techniques are introduced as two of the bases for the obtention of multiresolution models. Finally, mesh deformation, a widely used mesh processing technique, is reviewed, focusing on cage-based approaches such as main space deformation methods (Section 2.4).

2.1 Surface Meshes

This section reviews the main definitions related to surface meshes. First, the basic concepts related to surfaces and meshes are given. Then, a data structure commonly used for the management of surface meshes is summarized. Finally, the sharp features that can be present in surface meshes are classified. These have to be taken into account when working with mesh processing techniques.

2.1.1 Surfaces

A *two-dimensional manifold* is a closed set of points S included in $\mathbb{R}^3$ satisfying that for each point $p \in S$ there exists an open ball B_p centred at p and a *homeomorphism* (continuous and bijective map) $\varphi_p : B_0 \longrightarrow B_p$ with $\varphi_p(D_0) = B_p \cap S$, where B_0 is the open ball centred at the origin of $\mathbb{R}^3$ and D_0 is the disk $B_0 \cap (\mathbb{R}^2 \times \{0\})$. Informally it is equivalent to say that all points of S have a neighbourhood which is topologically equivalent to a disk (see Figure 2.1). Otherwise, a *two-dimensional manifold with boundary*, also called *open two-dimensional manifold*, is a closed set included in $\mathbb{R}^3$ all of whose points have a neighbourhood which is topologically equivalent to either a disk or a half-disk. When a surface non-possesses boundary, it can be emphasized by calling it a *closed two-dimensional manifold*.

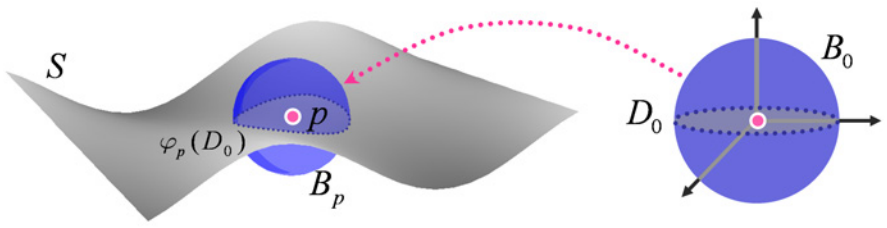

Figure 2.1: Two-dimensional manifold definition.

A homeomorphism φ_p assigns a sign to all points $q \in B_p - S$ in the following way. If $\varphi_p^{-1}(q) \in \mathbb{R}^2 \times \mathbb{R}^+$ the sign of q is $+$, otherwise it is $-$. Two homeomorphisms φ_p and $\varphi_{p'}$ are said to be compatible if they assign the same sign to all points in $B_p \cap B_{p'}$. A two-dimensional manifold S is said to be *orientable* if it is possible to find a set of compatible homeomorphisms φ_p. Most two-dimensional manifolds encountered in the physical world are orientable. Spheres, planes, and tori are some orientable examples. Otherwise, Möbius strips are non-orientable (see Figure 2.2).

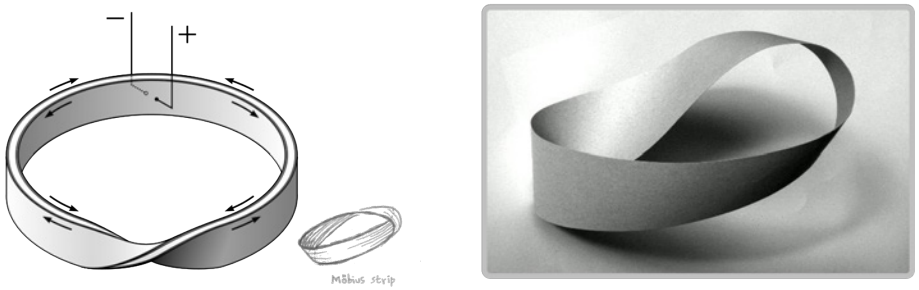

Figure 2.2: Möbius strip: a non-orientable two-dimensional manifold.

A *surface* is a fitted, connected and orientable two-dimensional manifold that can be closed or open depending on the manifold type [O'N66]. Complicated surfaces can be analysed by defining maps from the surface to $\mathbb{R}^2$ [ST67, Lef49]. Each of these homeomorphisms, called *chart*, takes an open disk of the surface down to an open disk in the plane. An *atlas* of a surface is a finite collection of charts. The collection of chart domains must completely cover the surface, i.e., every point on the surface must be in the domain of one or more charts. The inverse homeomorphism is called the *parameterization* of the chart domain, or simply the *chart parameterization*.

Given two surfaces S_1 and S_2 the connected sum $S_1 \# S_2$ is constructed by removing a disc from each one and then joining them along the boundaries of the holes. The classification theorem of closed surfaces, first proved by Dehn and Heergaard in 1907 [DH07], states that any (orientable) surface is homeomorphic to a sphere or the connected sum of g tori, for $g \geq 1$. The number g of tori involved is called the *genus* of the surface. The genus can be seen as the "number of handles" or the "number of holes" present in a surface (see Figure 2.3).

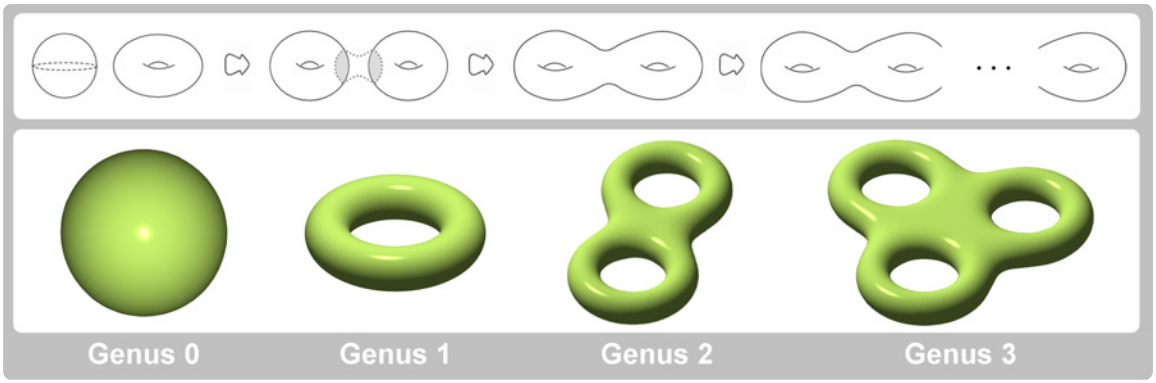

Figure 2.3: Genus of a surface with the generation scheme at the top.

2.1.2 Tangent plane and normal vector

Let $\phi : D \longrightarrow \phi(D) \subset S$ be a chart parameterization of surface S, where $\phi(u,v) = (x(u,v), y(u,v), z(u,v))$ and the parameters (u,v) vary within a certain domain $D \subset \mathbb{R}^2$. Let $p_0 = \phi(u_0, v_0)$, then a surface S is of class C^k at a point p_0 if a chart parameterization ϕ of class C^k at p_0 can be find, i.e., functions $x(u,v)$, $y(u,v)$ and $z(u,v)$ are of class C^k at (u_0, v_0). If $k > 0$, it is said that S is *differentiable* at p_0 or p_0 is a differentiable point of S.

Suppose that S is differentiable at p_0. The parameterization ϕ defines two curves passing

though p_0 as follows:

$$\phi(u, v_0) = (x(u, v_0), y(u, v_0), z(u, v_0), \qquad \phi(u_0, v) = (x(u_0, v), y(u_0, v), z(u_0, v)\,,$$

and their corresponding tangent vectors at p_0:

$$\phi_u(u_0, v_0) = \frac{\partial \phi}{\partial u}(u_0, v_0) = \left(\frac{\partial x}{\partial u}(u_0, v_0), \frac{\partial y}{\partial u}(u_0, v_0), \frac{\partial z}{\partial u}(u_0, v_0)\right)\,,$$

$$\phi_v(u_0, v_0) = \frac{\partial \phi}{\partial v}(u_0, v_0) = \left(\frac{\partial x}{\partial v}(u_0, v_0), \frac{\partial y}{\partial v}(u_0, v_0), \frac{\partial z}{\partial v}(u_0, v_0)\right)\,.$$

If the parameterization ϕ is regular at p_0, that is, if $\phi_u \neq 0$ and $\phi_v \neq 0$, the plane through p_0 and parallel to the vectors $\phi_u(u_0, v_0)$ and $\phi_v(u_0, v_0)$ is called the *tangent plane* of S at p_0 (T_{p_0}). Moreover, the vector defined over the tangent plane

$$n(u_0, v_0) = \phi_u(u_0, v_0) \times \phi_v(u_0, v_0) / \parallel \phi_u(u_0, v_0) \times \phi_v(u_0, v_0) \parallel$$

is called the *unit normal vector* of S at p_0 (see Figure 2.4).

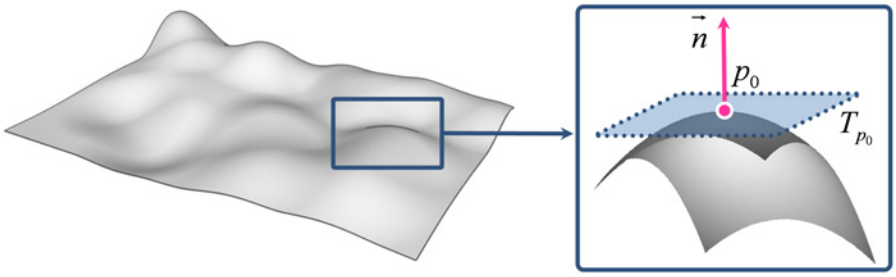

Figure 2.4: Tangent plane T_{p_0} and normal vector $\vec{n}$ of a surface point.

2.1.3 Surface curvature

One way to describe a surface visually is to describe it by telling how *curly* it is. In mathematics, *curvature* refers to a number of loosely related concepts in different areas of geometry and can be seen as a local measure of surface shape. Intuitively, curvature is the amount by which a geometric object deviates from being *flat*.

The *osculating circle* $C(p)$ of a curve α at a given point $p \in \alpha$ is the circle that best approximates the curve at p (see the left image of the Figure 2.5). Then, the osculating circle $C(p)$ is given by

$$C(p) = \lim_{p', p'' \to p} C(p, p', p'')\,,$$

where p' and p'' are points on α and $C(p, p', p'')$ is the circle that passes through p, p' and p''. The *radius of curvature* $r(p)$ of α at p is defined as the radius of $C(p)$, and the *curvature* is defined as

$$k(p) = \frac{1}{r(p)} .$$

If α is given by a regular parameterization ($\alpha'(t) \neq 0$) $\alpha(t) = (x(t), y(t), z(t))$ of class C^2 with $p = (x(t), y(t), z(t))$, the curvature $k(p)$ can be computed by

$$k(p) = \frac{\| \alpha'(t) \times \alpha''(t) \|}{\| \alpha'(t) \|^3} ,$$

where $\alpha'(t) = (x'(t), y'(t), z'(t))$ and $\alpha''(t) = (x''(t), y''(t), z''(t))$.

Let $\phi(u, v) = (x(u, v), y(u, v), z(u, v)$ be a chart parameterization of class C^2 of surface S. Let t be a unit vector in the tangent plane at $p = \phi(u, v)$. The *normal curvature* $k_n(t)$ is the curvature of the planar curve that results from intersecting S with the plane $\pi(t)$ through p spanned by n and t. The minimal normal curvature k_1 and the maximal normal curvature k_2 are called *principal curvatures* (see the right-hand image of Figure 2.5). The associated tangent vectors e_1 and e_2 are called *principal directions* and are always perpendicular to each other (if $k_1 = k_2$, it is sufficient to pick two arbitrary orthogonal tangent vectors). The normal curvature function $k_n(t)$ is a quadratic form and satisfies

$$k_n(t) = \cos^2(\theta) k_1 + \sin^2(\theta) k_2 ,$$

where θ is the angle between t and e_1. Moreover, in the coordinate frame with origin at p and axes determined by the principals directions e_1, e_2 and the surface normal n, surface S can be locally parameterized by

$$\left(x, y, \frac{k_1 x^2 + k_2 y^2}{2} \right) .$$

The *mean curvature* of S at p is defined by $H = \frac{k_1 + k_2}{2}$, and the *Gaussian curvature* of S at p is defined by $K = k_1 k_2$. The mean curvature is zero only for points on locally flat surfaces, while the Gaussian curvature can be zero for surfaces whose restriction into a plane is locally a straight line. For example, the mean and the Gaussian curvatures of a flat plane are zero, while for a cylinder of radius r the mean curvature is $1/r$ and the Gaussian curvature is zero. *Gauss's Theorema Egregium* states that the Gaussian curvature of a smooth surface in $\mathbb{R}^3$ is invariant under the local isometries. For example, since a sphere of radius r has constant Gaussian positive curvature $1/r^2$ and a flat plane has constant Gaussian curvature zero, these two surfaces are not isometric, even locally. Thus any planar representation of even a part of a sphere must distort the distances and

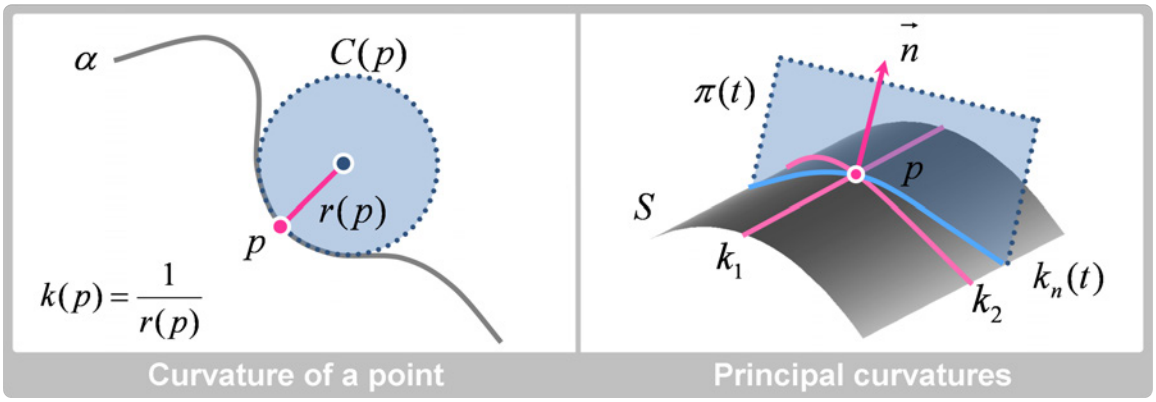

Figure 2.5: From left to right: curvature of a point in a curve and principal curvatures of a surface point.

no cartographic projection is perfect. On the contrary, perfect cartographic projections of cylinders do exist. Figure 2.6 shows a comparison between mean and Gaussian curvatures computed over a car bodywork model.

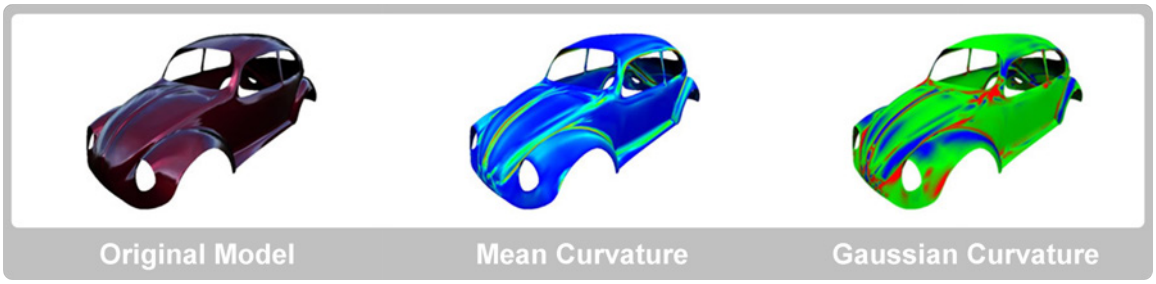

Figure 2.6: Mean and Gaussian curvature comparison.

2.1.4 Surface mesh representation

In computer science, polygonal meshes remain the most common and flexible way to approximate surfaces. A polygonal surface model, also known as a *mesh*, is a piecewise linear surface in three-dimensional Euclidean space $\mathbb{R}^3$. Without loss of generality, it can be assumed that the set of planar polygons defining a mesh consists entirely of triangular faces, since any non-triangular polygon may be triangulated in a pre-processing step ([Sei91, NM95]).

A mesh $M = (V, F)$ is a pair containing a list of vertices V and a list of triangular faces F. The vertex list $V = (v_1, v_2, ..., v_m)$ is an ordered sequence where each vertex may be identified by a unique integer i. The face list $F = (f_1, f_2, ..., f_n)$ is also ordered, assigning a

unique integer to each face. Every vertex $v_i = (x_i, y_i, z_i)$ is a vector in the Euclidean space $\mathbb{R}^3$. Each triangle $f_i = (j, k, l)$ is an ordered list of three indices identifying the corners (v_j, v_k, v_l) of f_i. Different surface mesh examples are shown in Figure 2.7.

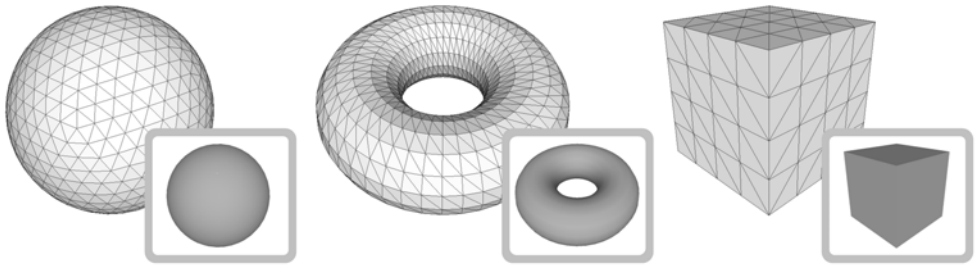

Figure 2.7: From left to right a sphere, a torus and a cube surfaces with their associated triangular meshes.

Extending the surface manifold definition, a polygonal surface is said to be a manifold mesh, called *closed mesh*, if every edge in the mesh is shared by exactly two faces and the neighbourhood of every vertex consists of a closed loop of faces. In exception, in a manifold mesh with boundary, called *open mesh*, boundary edges must have only one incident face and the neighbourhood of boundary vertices consists of a single fan of faces. Figure 2.8 illustrates different vertex neighbourhood configurations in a polygonal model.

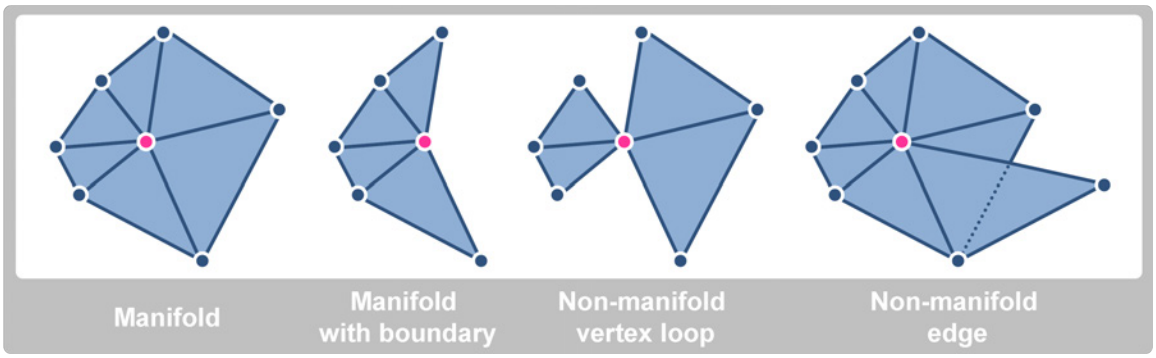

Figure 2.8: Neighbourhoods of a given vertex (in pink). On the left, two manifold neighbourhoods. On the right, two non-manifold neighbourhoods.

The orientation of a face is a cyclic order of the incident vertices. A manifold mesh is *orientable* if any two adjacent faces have *consistent orderings*. Let f_i and f_j be adjacent faces sharing the edge (v_i, v_j). If v_i and v_j occurs in this order for f_i, then they must occur in f_j in the order v_j followed by v_i. In Figure 2.9 consistent and inconsistent orderings are illustrated. The typical choice in computer graphic applications is based on visibility of the mesh from the camera location. The vertices are ordered counterclockwise in the plane of

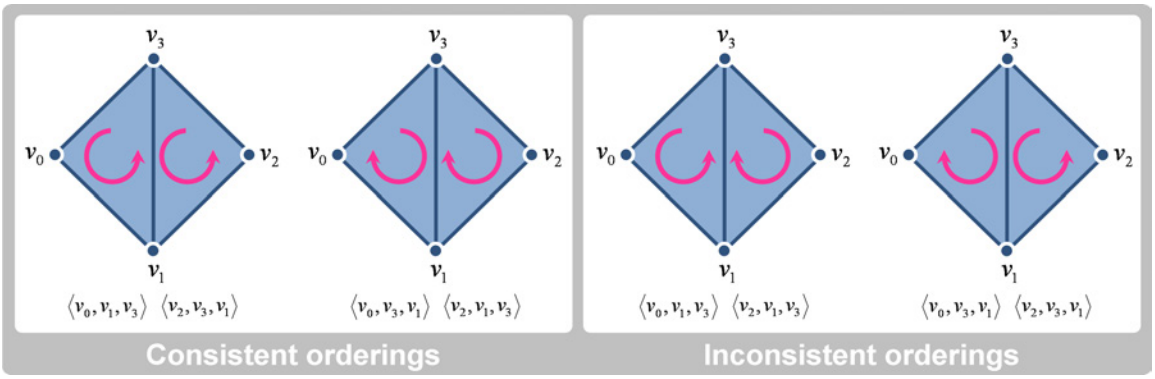

Figure 2.9: Consistency of the possible orderings of two incident faces. On the left, two consistent orderings. On the right, two inconsistent orderings.

the face viewed from outside the mesh. This assumption defines the normal vectors directed toward the eye point.

2.1.5 Mesh data structure: DCEL

Data structures play an important role in both numerical computations and geometric algorithms. A data structure is defined as a way of storing data in a computer for efficient search and retrieval. The efficiency of most of the geometric modelling algorithms crucially depends on the underlying mesh data structures and therefore have to be carefully chosen. A well-designed data structure allows a variety of critical operations to be performed using as few resources as possible. In geometry processing, all these available operations enable local and global mesh traversal. When using mesh data structures that can serve both numerical and geometric computations on parallel computers it is desirable to meet the following requirements: efficient in both time and space, neutral of programming languages, convenient for I/O, and easily extensible to support partitioned meshes.

Working with polygonal surfaces often requires information about the topological relations between vertices, edges and faces. Mesh data structures make it possible to iterate through the entities of a mesh, performe queries on incidence, adjacency, and classification (in particular, boundary classification) of entities, and modify the mesh efficiently. A variety of data structures has been described in the literature, and different implementations are available [Ket99]. One of the most convenient and flexible data structures in geometry processing is the *doubly-connected edge list* (DCEL) [M̈87]. A DCEL is used to explicitly store information from representations such as planar subdivisions, polyhedral surfaces and polytopes in $\mathbb{R}^3$. [PS85, dBvKOS97]. This data structure supports operations such as ob-

taining the adjacent regions, edges or vertices from a given region. In the DCEL structure, edges in the subdivision are represented by two directed half-edges such that one of the half-edges bounds one face incident to the edge and the other half-edge bounds the other face. If e is a half-edge, then $Twin(e)$ denotes the opposite half-edge that is part of the same edge. Every half-edge is oriented in such a way that the face to which it is incident lies to its left, following a consistent ordering. In this way, the incident half-edges for all bounded faces of a connected subdivision form a counterclockwise cycle around each face. The half-edges that bound the one unbounded face (the outer face) form a clockwise cycle. An example of the face, half-edge and vertex instances of a given subdivision is shown on the right in Figure 2.10.

A DCEL contains a record for each face, half-edge, and vertex of the subdivision stored in three tables, one for each of them. A vertex record stores the coordinates of the vertex position ($Point(v)$) and a single reference to a half-edge record ($IncidentEdge(v)$) that has the vertex as its origin. For any half-edge e (see Figure 2.10) its record stores one reference to the record of its vertex origin ($Origin(e)$), a reference to the record of the incident face which lies to its left ($IncidentFace(e)$), a reference to the half-edge that precedes e in the cycle of edges around the incident face ($Prev(e)$), a reference to the record of the half-edge that follows e in this cycle ($Next(e)$), and a reference to the record of the complementary half-edge ($Twin(e)$). The content of each face record depends on their connectivity. For a bounded face, a reference to a half-edge record of one of the half-edges incident to it is stored ($IncidentEdge(f)$). For an unbounded face a clockwise half-edge of its incident half-edges is stored.

The geometric and topological information stored in a DCEL makes it possible to perform basic operations like: walking around a face in counterclockwise order, accessing one

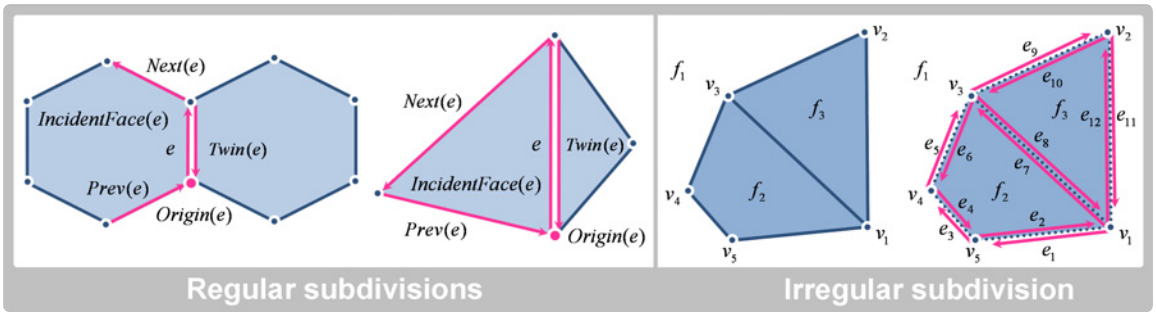

Figure 2.10: On the left, half-edge record for a hexagonal and a triangular configurations. On the right, an irregular subdivision with the corresponding face, half-edge and vertex instances.

face from an adjacent one given the common edge, and visiting all the edges around a given vertex. Besides the default information each record may also store additional information depending on the application needs.

2.1.6 Sharp features

Features are characterized by the way in which the unit surface normal varies along the surface. A smooth surface is a surface that has a tangent plane at each point, and for which the direction of its unit normal is a continuous function of the point of tangency. A point on a piecewise smooth surface is called a *sharp feature point* if the surface has no tangent plane at this point. The locus of the sharp feature points on a piecewise smooth surface has the structure of a graph whose edges are maximal smooth (at least C^1) curves. The edges of this graph are called *crease* or *sharp edges*. A crease edge where the surface terminates is called a *boundary edge*. A vertex of the graph is classified according to the number s of crease edges meeting it ([Ma05]):

- A *sharp vertex* has no meeting crease edge ($s = 0$). If the directional tangent of the limit surface at the vertex position does not vanish, the vertex is classified as a *cone-type vertex*. Otherwise, if the directional tangents of the limit surface at the vertex position vanish to a single vector, it is classified as a *cusp vertex*.

- A *dart vertex* is one where a crease edge terminates ($s = 1$).

- A *crease* or *boundary vertex* is located on the junction of two creases or two boundary edges, respectively ($s = 2$).

- A *corner vertex* has $s \geqslant 3$.

On the left side of Figure 2.11, a surface with different sharp features is illustrated.

For discrete representations like triangular meshes, we assume that they are linear approximations to piecewise smooth surfaces. Then, mesh vertices and edges inherit characteristics from their original sources. Notice that sharp features could be classified differently if the original surface was not taken into account to compute them. This can be seen with the cone mesh illustrated on the right of the Figure 2.11.

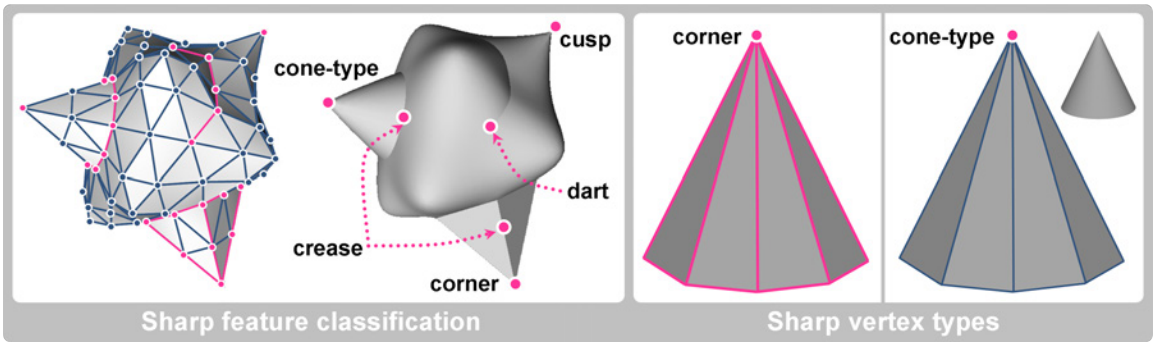

Figure 2.11: Illustration of sharp features (depicted in pink) over a surface model. On the right, the sharp vertex of the cone approximation is classified as a corner vertex or a cone-type vertex depending on the reference surface used to classify it.

2.2 Graphics Hardware

Graphics processing units (GPU) have long been used to accelerate gaming and 3D graphics applications. The increasing programmability and high computational rates of GPUs make them attractive as an alternative to CPUs for parallel general-purpose computing (GPGPU). In computational geometry several application fields take advantage of this powerful tool.

In this section, an introduction to the graphics hardware pipeline is given. A closely related concept, called mesh parameterization, is also summarized. Different mesh parameterization techniques are reviewed and some detail mapping applications are presented.

2.2.1 Graphics pipeline

The visualization of a mesh is achieved by sending basic primitives like points, segments or polygons as a set of vertices and indices and by setting the lights and the perspective desired through a graphic environment. The graphics, taking into account the parameters previously chosen, card is responsible for rendering all this primitives. In fact, the GPU could be seen as a black box with some basic controls providing an input for the geometry and an output for its visualization. As a result of the technical advancements in graphics cards, some areas of 3D graphics programming have become quite complex. New features were added to graphics cards to simplify the process. The GPU is a graphics accelerator that allows computations to be parallelized thanks to its parallel processing units. It obtains better real-time graphics and faster computations in non-real-time applications. Moreover, the ability to modify the rendering pipelines in programmable stages makes arbitrary com-

putations possible. So, if a problem can be transformed to an image-based algorithm, it can be solved (usually an approximated solution is obtained) by taking advantage of the GPU capabilities.

The graphics pipeline [FK03, SA04] is divided into several stages, which are implemented as separate pieces of hardware on the GPU. With a graphics application programming interface (API) it is possible to define objects using different 3D geometric primitives in CPU: points, segments, polygons, polygon strips, etc. The input to the graphics pipeline is a set of these geometric primitives expressed as vertices (and indices for face definitions) with associated attributes such as 3D coordinates, colour, etc., which enter the graphics engine one at time. The output is an image in the *frame buffer*, which is a collection of hardware buffers corresponding to two-dimensional grids whose cells are called *pixels*. Each pixel in the frame buffer is a set of some number of bits grouped together. Different buffers exist; the stencil buffer, the depth buffer and the colour buffer store stencil, depth, and RGB colour values, respectively. In order to store information in those images or textures, arrays in GPU are used. So, the use of textures is made possible by the graphics hardware. The texture coordinates associated with each vertex allow access to the information stored in the corresponding position. The maximal size of a texture, called *resolution*, and the number of textures that can be simultaneously used depend on the graphics card. The types of textures differ according to the number of channels stored per position and the size of these channels. For example, colour textures use RGBA textures consisting of four channels of a maximum of 16 bits for each one while depth textures store a single number per position, which can be up to 32 bits in current graphics cards. Then, each graphics pipeline stage is summarized.

The first stage of the graphics pipeline consists of per-vertex operations. Each input vertex is transformed from 3D coordinates to window coordinates (screen-space). Then, entire primitives as triangles or lines are processed in the geometry stage. In the third stage, rasterization takes place. It receives a set of fragments that, according to their x and y values, have to be attached to the appropriate pixels. Fragment attributes such as depth, colour, texture coordinates, etc., are obtained from the attributes associated with the vertices by linear interpolation. The next stage, called the fragment stage, computes the colour for each pixel according to the fragments falling on it. For that purpose, each fragment passes a series of tests (scissor, alpha, stencil, depth) and per-fragment operations (updating, blending, logical operations, etc.) to avoid rendering or to modify the appearance of some fragments before they are placed into their corresponding pixel of the frame buffer. Per-fragment operations can use values from textures to modify their depth, colour, etc.

Finally, the fragments that pass the tests and the operations of the previous stage are drawn or rendered on the screen or on a specified texture with the resulting colour. The graphics pipeline is schematically represented in Figure 2.12.

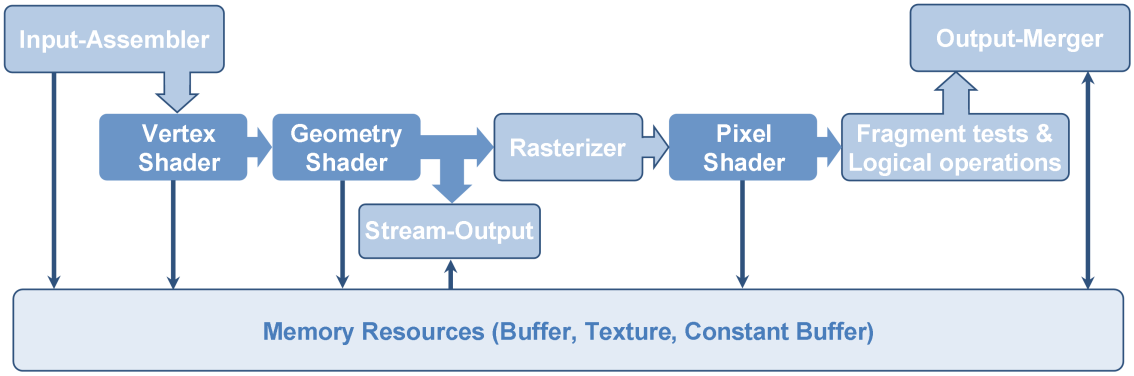

Figure 2.12: Graphics pipeline. The programable stages of the graphics hardware (vertex-shader, geometry-shader and pixel-shader) are depicted in dark blue.

Vertex, geometry and fragment shaders define the programmable parts of the graphics pipeline. *Vertex shaders* are executed on a per-vertex basis and allow 3D vertex coordinates to be modified. *Geometry shaders* are executed on full primitives and allow vertices to be instanced from the GPU. *Fragment shaders*, also called *pixel shaders*, are executed on a per-fragment basis and allow the appearance of the pixels to be changed by combining fragment values, such as colour and depth, with values stored in the fragment attributes or in textures, which can be sent to them as parameters.

2.2.2 Detail mapping applications

The representation of a detailed object without a significant increment in the time consumption has been an interesting area of research. Texture management together with the programmable capability of the graphics cards opens a wide range of possibilities in this area. In fact, different detail mapping applications that take advantage of the graphics hardware capabilities have become commonly used tools. The first detail mapping technique was introduced in computer graphics as a method for mapping textures onto surfaces [BVI91, MYV93]. This technique, known as *texture mapping*, enhances the visual quality of polygonal models by initially using colour textures (see Figure 2.13). Detailed objects can be efficiently represented by a coarse geometric shape with the details corresponding to each triangle stored in a separate 2D array. In traditional texture mapping the details are the colours of the respective pixels. Moreover, texture maps alone can enrich the appearance

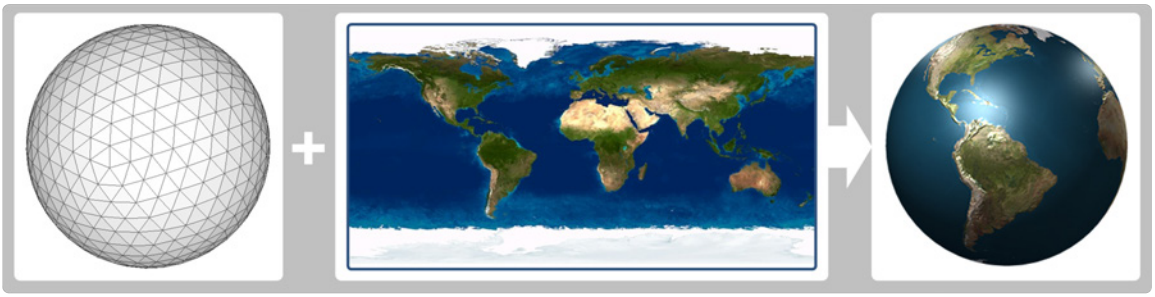

Figure 2.13: From a surface mesh and a texture, the texture mapping technique allows to obtain a textured model.

of a surface in a static picture, but since neighbouring pixels will have similar shadowing, objects may still look flat in animations with varying lighting conditions.

Later, new detail mapping techniques appeared and the rendering effects of the meshes improved. The computational cost of these techniques was improved by taking advantage of the programmable stages of the graphics pipeline, which was becoming more efficient. Next, three common techniques are summarized:

Bump mapping is used for simulating bumps and wrinkles on the surface of an object [Bli78]. This is achieved by storing small deviations of the point-wise normal from the original surface to the smooth underlying surface and perturbing the surface normals during shading. The result is an apparently bumpy surface rather than a perfectly smooth surface although the surface of the underlying object is not actually changed.

Normal mapping [SLMB05] is similar to bump mapping in that it replaces the normals directly rather than storing a perturbation. As the light direction changes, the shading variations produced by the normal perturbations simulate the shadows caused by small pits and dimples in the surface. Since the actual geometry of the object is not modified, the silhouettes still look polygonal or smooth.

Displacement mapping [Coo84, CCC87] addresses this problem by storing small local deformations of the surface, typically in the direction of the normal. In a vertex shader, the position of the points over the surface are displaced giving a great sense of depth and detail.

A comparison between the results obtained by these three detail mapping techniques is shown in Figure 2.14. As can be seen, displacement mapping is the only one that makes it possible to represent the surface details of the silhouette.

Figure 2.14: Comparison between detail mapping techniques (bump mapping, normal mapping and displacement mapping) applied on the Teapot model.

2.2.3 Mesh parameterization

The graphics hardware is used to totally or partially map two-dimensional arrays, called *textures*, onto the meshes to provide them with rendering effects. But, how are techniques such as detail mapping made possible? The mesh parametrization technique provides the solution.

The central objective of mesh parametrization techniques is to establish bijective mappings between surfaces and parametric domains [BVI91, FH05]. Therefore, in order to reduce distortion, each polygonal face of the mesh is mapped to a polygon in the plane with approximately the same shape (angles and area). Two different polygons of the mesh are mapped to the texture without overlapping. Computing a parameterization means finding the corresponding texture coordinate of each vertex of the mesh.

In recent years, numerous methods for parameterizing meshes have been developed, targeting diverse parameter domains and focusing on different parameterization properties. Two recent surveys [FH01, FH05] list more than 20 different planar parameterization techniques. Both surveys focus on the mathematical aspects of these techniques to avoid overlaps and study the suitability of the techniques for computer graphics applications in terms of distortion (type and amount), robustness and efficiency.

The planar parameterization of 3D surfaces inevitably introduces distortion in either angles or areas, as Gauss's Theorem Egregium states (see Section 2.1.3). So, a good mapping is one that minimizes these distortions in some sense. Planar parameterizations can be classified into four groups depending on the distortion treatment: ignore distortion, minimize angular distortion, minimize stretching or minimize area distortion. In addition, several other techniques provide tools for achieving a trade-off between different types of distortion.

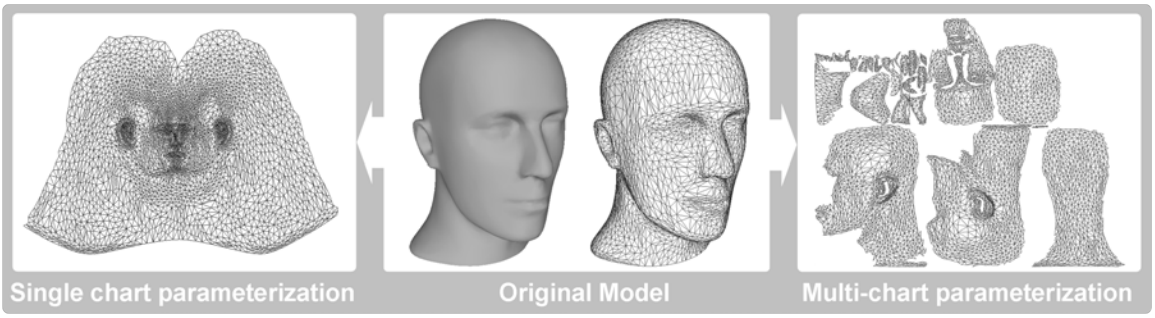

Figure 2.15: Two planar parameterizations from a head mesh. On the left, a single chart parameterization created by using a seam generation technique. On the right, a multi-chart parameterization created by using a segmentation technique.

In order to achieve parameterizations with acceptable distortion, the surface must be cut either by introducing seams or by generating charts. Since cuts introduce discontinuities into the parameterization, a delicate balance between the conflicting goals of small distortion and short cuts has to be achieved. Seams introduced to the parameterization are an obstacle for some surface processing techniques such as simplification or approximation. However, it is possible to use constrained parameterization techniques to reduce cross-cut discontinuities [KSG03, ZWT+05]. Cutting and chart generation are most commonly used when computing parameterizations for mapping textures onto the surface (see Figure 2.15). The techniques for cutting surfaces can be divided into two categories: segmentation techniques which partition the surface into multiple charts (*multi-chart techniques*), and seam generation techniques, which introduce cuts into the surface but keep it as a single chart. Multi-charts created by segmentation typically have longer boundaries than those created by seam cutting. However, they can often be more efficiently packed into a compact planar domain, which usually results in less distortion. A balance between the number of charts created and the distortion factor obtained has to be achieved. Next, two of the most common multi-chart parameterization techniques are summarized:

Least square conformal maps (LSCM) [LPRM02] is an automatic texture atlas generation method for polygonal models. This is a quasi-conformal parameterization method that minimizes angle deformations. It is proved that the minimum is unique, independent of a similarity in texture space and independent of the resolution of the mesh. The segmentation algorithm, driven by detected features, decomposes the model into charts with natural shapes, corresponding to meaningful geometric entities (see Figure 2.16).

Figure 2.16: Least square conformal maps.

Iso-charts [ZSGS04] was the first to consider stretching not only when parameterizing charts, but also when forming charts. It focuses on the distance distortion, specifically the geometric stretch defined in [SSGH01] (small texture distances mapped onto large surface distances), which measures the average and worst-case stretching of local distances over the surface. The algorithm creates texture atlases quickly, with fewer charts and lower stretch than previous methods (see Figure 2.17).

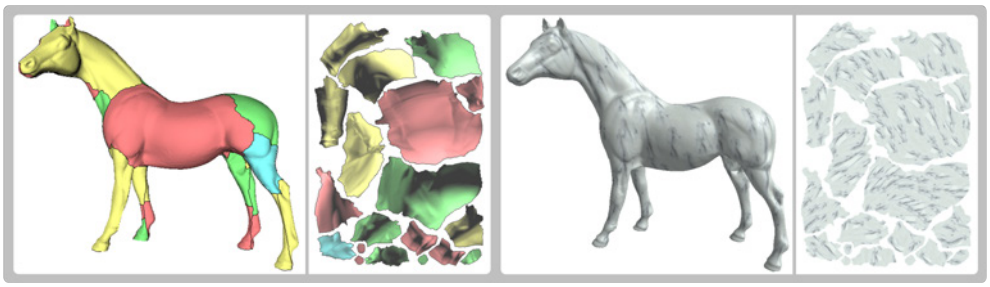

Figure 2.17: Iso-charts.

2.3 Level of Detail

Many applications in computer graphics and in related fields require polygonal surface models, one of the most common surface representations used for both simulation and display. Advances in technology have made that common data acquisition systems, such as laser range scanners, medical imaging devices, computer vision or computer-aided design (CAD) systems, require vast databases to mantain a convincing level of realism (see Figure 2.18). From these datasets, surface reconstruction and isosurface extraction methods can often produce very densely sampled meshes containing millions of polygons with a uniform

Figure 2.18: Scanning of the Michelangelo's David with the corresponding polygonal model of ≈ 56 million polygons.

distribution of points on the surface. Unfortunately, the complexity of these highly detailed models grows faster than the ability of our graphics hardware to render them interactively. Notwithstanding, the full complexity of such models is not always required, and since the computational cost of using a model is directly related to its complexity, it is useful to have different levels of detail of complex models. This gap motivated the graphics community to construct representations that would enable users to visualize and to interact with these datasets. The resulting hierarchical and multiresolution techniques offer a wide range of capabilities applicable to a wide variety of surfaces.

In this section, the related problem is introduced by describing the multiresolution models. The assessment of any method used in a multiresolution approach is always required. In that sense, some error metrics to measure the good performance of the approximations are then presented. Next, surface simplification is introduced as one of the bases of level of detail to obtain coarse meshes. Finally, mesh refinement techniques have to be applied over simplified models to get full multiresolution systems.

2.3.1 Multiresolution models

Level of detail (LOD) modelling focuses on the trade-off between fidelity and rendering performance. Its aim is to obtain a sequence of geometric approximations and switch them during rendering, selecting the best one for the current viewing conditions (see Figure 2.19). In interactively applications such as videogames, the distance between the object and the viewer can be a decisive factor for the chosen level. In order to manage the level of detail of an object maintaining a constant frame rate, a *multiresolution model* representation which allows the surface to adapt at run time is needed. A multiresolution model is a

Figure 2.19: Levels of detail (LODs) of the Bimba model used to reduce the rendering cost of small, distant, or unimportant geometry.

model representation that captures a wide range of approximations of an object and can be used to reconstruct any one of these on demand. The multiresolution representation must have roughly the same size as the most detailed approximation alone and the cost of reconstructing approximations should be low. There are two kinds of multiresolution models according to the continuity between different levels:

- ***Discrete Multiresolution.*** The simplest method for creating multiresolution surface models is to generate a set of increasingly simpler models. For any given frame, a renderer could select which model to use from a series of discrete levels of detail and render that model in the current frame [FS93]. The multiresolution model is defined by the set of levels and the threshold parameters used to control the switching between them. Detail blending or geomorphing techniques can be applied to avoid "popping" artefacts during level-of-detail transitions [FST92, Hop96]. When using a discrete multiresolution strategy, renderer would be forced to pick one of the pre-generated models, even if it needed an intermediate level. Thus the renderer would either have to pick a model without sufficient detail or choose a model with excessive detail, sacrificing quality or wasting unnecessary time. Notwithstanding, if an object is displayed such that the entire surface is at roughly the same scale, then discrete multiresolution models are an effective means of controlling level of detail. Walkthrough systems, commercial rendering systems and radiosity solutions are some examples of discrete multiresolution applications.

- ***Continuous Multiresolution.*** An approximation with a constant level of detail would either be too dense in the distance or too sparse near the viewpoint. An ap-

proximation where the level of detail is allowed to vary continuously over the surface is preferred. In particular, the level of detail of a particular neighbourhood would be like to be view-dependent. The result is an approximation which is specifically tailored to the current viewpoint. Thus, we are looking for a multiresolution representation that continuously adapts the surface at run time based on viewing conditions. The run-time adaptation can be combined with the geomorphing technique to produce smooth transitions. The need for adaptive level-of-detail control is particularly pronounced in terrains or flight simulator systems where a regular subdivision is commonly used.

In short, discrete methods are simpler and require less overhead while continuous methods are more flexible but have higher overhead. Despite their differences, both types of multiresolution models can be constructed using surface simplification and surface refinement methods. In general terms, level of detail is a widely used technique in several application fields such as games, films, architectural renders or virtual worlds (see Figure 2.20).

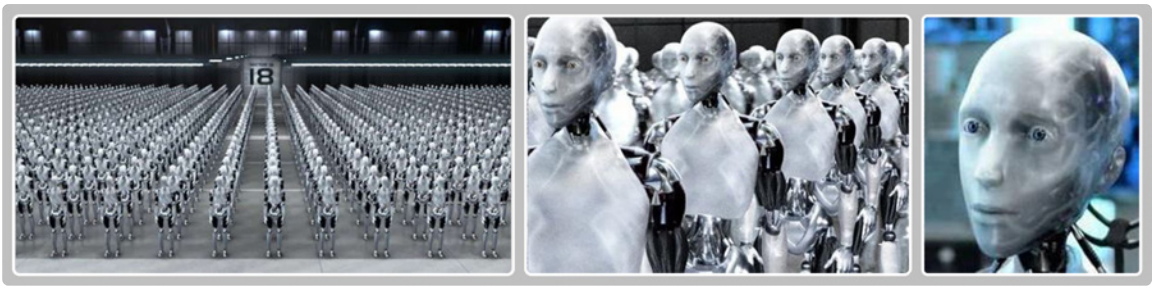

Figure 2.20: Application of the level of detail technique in the "I, Robot" film. From left to right, different views of a scene from a far view distance to a close view inspection. All images copyright ©2004 20th Century Fox.

2.3.2 Error metrics

In several surface processing techniques required for multiresolution an error measure is needed to measure the good behaviour of an approach. In order to assess the quality of an approximation, some means of quantifying the notion of similarity are needed. From a given polygonal model M and an approximation M' is defined an error metric $E : M \times M' \to \mathbb{R}$ of which the value $E(M, M')$ measures the approximation error of M'. The lower the error value assigned by E to M', the greater its similarity to the original model M.

The approximation error between surfaces can be measured in different ways according to the criteria chosen. While the preferred criteria are application-dependent, similarity of appearance is the natural choice for rendering applications. However, similarity of shape

in geometrical terms is one of the most used criterion for evaluating approximation quality. Shape-based metrics appear to be more computationally convenient and are also more appropriate in non-rendering applications.

2.3.2.1 Similarity of appearance

In rendering systems, similarity of appearance should be the ultimate criterion for evaluating the quality of an approximation.

The appearance of a model M under viewing conditions ξ is determined by the raster image I^{ξ} produced by the renderer. The similarity of appearance can be seen as an image error metric that measures the overall difference between two images. In that sense, it may say that two models M_1 and M_2 appear identical in view ξ if their corresponding images I_1^{ξ} and I_2^{ξ} are identical. If I_1 and I_2 are both $m \times m$ RGB raster images, the difference between them can be defined as the average sum of squared differences between all corresponding pixels

$$\|I_1 - I_2\|_{img} = \frac{1}{m^2} \sum_u \sum_v \|I_1(u,v) - I_2(u,v)\|^2 \tag{2.1}$$

where $\|I_1(u,v) - I_2(u,v)\|$ is the Euclidean length of the difference of the two RGB vectors $I_1(u,v)$ and $I_2(u,v)$. This simple metric makes it possible to measure the visual difference between a detailed input model and its approximation in a certain view. This can be viewed as the measure equivalent to human perception. Differential weighting for the colour channels, non-linear sensitivity to radiance, and spatial filtering are some factors that can be added to improve the measure. More elaborate metrics for comparing images have been presented in [RT98]. If M_2 is a good approximation of M_1 for the given view ξ, then $\|I_1 - I_2\|_{img}$ should be small. Given this image metric, the total difference in appearance between two models can be characterized by integrating $\|I_1^{\xi} - I_2^{\xi}\|_{img}$ over all possible views ξ.

The main advantage of an appearance-based metric is that it directly measures similarity of appearance, which is the interest of preservation in rendering systems. It also allows occluded details to be discarded. Moreover, these error metrics are useful when some finite set of viewpoints occurs. Unfortunately, in most cases adequate samples of viewpoints are not possible. If an incorrect set of samples is evaluated, significant features can be removed. Furthermore, each sample may involve an expensive rendering step.

2.3.2.2 Geometric approximation error

Geometric similarity can be used as a proxy for visual similarity. Approximations useful in application domains other than rendering can be obtained by producing geometrically faithful results. Geometrical measures give more accurate results for global similarity between whole meshes.

The two most commonly used geometric error metrics are the L_∞ and L_2 norms [Pre75]. Suppose a real-valued function $f(t)$, an approximation $g(t)$, and an interval of interest $[a, b]$. The L_∞ norm, which measures the maximum deviation between the original and the approximation, is defined by

$$\|f - g\|_\infty = \max_{a \le t \le b} |f(t) - g(t)| \tag{2.2}$$

The L_2 norm, which provides a measure of the average deviation between the two functions, is defined by

$$\|f - g\|_2 = \sqrt{\int_a^b (f(t) - g(t))^2 \, dt} \tag{2.3}$$

When L_2 is divided by $b - a$ it is also called *root mean square* (RMS).

In practice, neither of these error metrics is completely ideal. On one hand, the L_∞ norm provides strong error bounds, but can be overly influenced by noise and local deviations. On the other hand, the L_2 norm provides a better estimate of the overall fit and is more tolerant of noise, but it may discount local deviations. Consequently, a combination of these two metrics is preferable. Usually, they are desired approximations with small L_2 error for which the L_∞ error is bounded by some known threshold.

Surface-based analogs to L_2 and L_∞ function approximation norms can be formulated to evaluate surface approximations. When comparing general surfaces, there is no single distinguished direction along which to measure distances. Instead, we measure distances between closest pairs of points. If we denote the set of all points on the surface of a model M by $P(M)$, the distance from a point v to the model M is defined to be the distance to the closest point on the model:

$$d_v(M) = \min_{w \in P(M)} \|v - w\| \tag{2.4}$$

where $\|\cdot\|$ is the usual Euclidean vector length operator.

The Hausdorff distance [PS85], which corresponds closely to the L_∞ metric, is one commonly used geometric error measure. Based on the point-wise distance measure (2.4),

the Hausdorff error metric $\widehat{E}_{\max}(M_1, M_2)$ can be defined as

$$\widehat{E}_{\max}(M_1, M_2) = \max\left(\max_{v \in P(M_1)} d_v(M_2), \max_{v \in P(M_2)} d_v(M_1)\right) \tag{2.5}$$

The Hausdorff error measures the maximum deviation between the two models. If $\widehat{E}_{\max}(M, M')$ is bounded by ϵ, then we know that every point of the approximation is within ϵ of the original surface and that every point of the original is within ϵ of the approximation. An analog of the L_2 metric can also be defined as a measure of the average squared distance between the two models.

$$\widehat{E}_{avg}(M_1, M_2) = \frac{1}{w_1 + w_2}\left(\int_{P(M_1)} d_v^2(M_2)\, dv + \int_{P(M_2)} d_v^2(M_1)\, dv\right) \tag{2.6}$$

where w_1, w_2 are the surface areas of M_1, M_2. Instead of normalizing the sum of both integrals by the combined surface area, one could also consider normalizing each individual integral by its own corresponding area.

Observe that it is not sufficient to simply consider every point on M_1 and find the closest corresponding point on M_2. Closest distances are measured in both directions between M_1 and M_2 due to the differences in results depending on the direction of the computation (see Figure 2.21). As the geometric error metric measures the similarity between models independently of the order of the given models, note that both $\widehat{E}_{\max}$ and $\widehat{E}_{avg}$ present a symmetric construction.

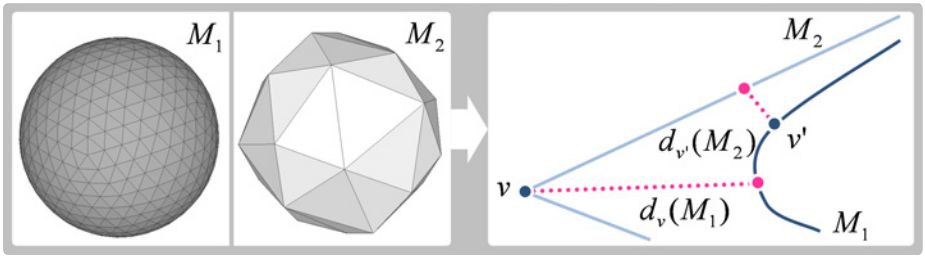

Figure 2.21: Distance between surfaces depending on the direction of the computation.

In practice, these error metrics can be prohibitively expensive to compute exactly. When working with meshes, it is common to formulate approximations of these ideal metrics based on sampling the distance d_v at a discrete set of points. Given $P(M_1)$ and $P(M_2)$, we can select two sets $X_1 \subset P(M_1)$ and $X_2 \subset P(M_2)$ containing m_1 and m_2 sample points, respectively. These sets should, at a minimum, contain all the vertices of their respective models. The reformulation of the geometric error measures 2.5 and 2.6 are

$$E_{\max}(M_1, M_2) = \max\left(\max_{v \in X_1} d_v(M_2), \max_{v \in X_2} d_v(M_1)\right) \tag{2.7}$$

and

$$E_{avg}(M_1, M_2) = \frac{1}{k_1 + k_2} \left(\sum_{v \in X_1} d_v^2(M_2)\, dv + \sum_{v \in X_2} d_v^2(M_1)\, dv \right) \tag{2.8}$$

As in the function approximation case, E_{avg}, also called E_{mean}, generally gives a better measurement of overall fit than $E_{\max}$ and is less sensitive to noise, even though it may over-discount localized deviations. Taking into account the two surface error metrics presented, the most commonly used measure is equivalent to a squared E_{avg} and is known as E_{rms}

$$E_{rms} = \left[\frac{1}{m} \sum_{k} \| S_k^1 - V_k^1 \|_2^2 \right]^{1/2} \tag{2.9}$$

As defined before, distance function d_v gives the distance of v to the closest point on M. Depending on the measure used to compute it, the results obtained can differ both in accuracy and evaluation cost. Next, the two main strategies to compute distance functions on meshes are summarized (see Figure 2.22):

- ***Vertex-Vertex.*** Given a vertex v of M_1, it consists of computing the nearest vertex v' of M_2. This is the faster distance function strategy but it gives the worst results. The precision of the measure directly depends on the resolution of the given meshes and is appropriated during topology changes.

- ***Vertex-Plane.*** A vertex to plane distance is evaluated for each vertex v to all defining planes of M_2. The nearest distance plane is restricted within the containing triangle. Although the computational cost increases, low error is obtained.

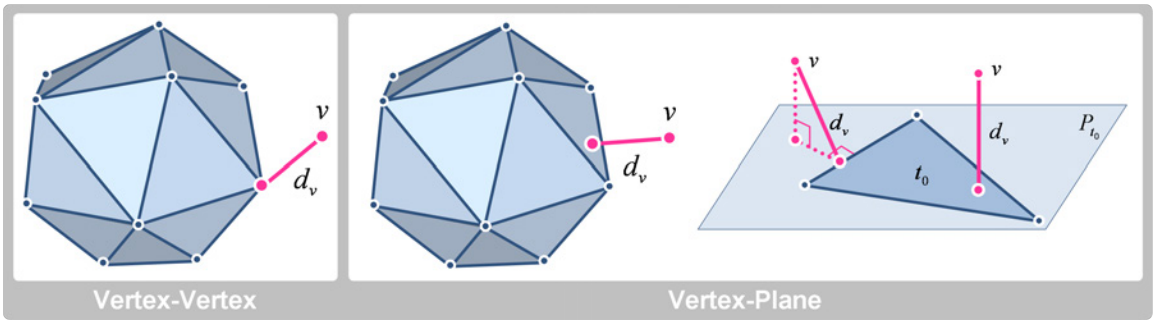

Figure 2.22: From a vertex, two main strategies to compute distance functions. On the left, the nearest vertex distance. On the right, the vertex-plane distance which depends on the projection of the vertex onto the plane with respect to the triangle.

2.3.3 Surface simplification

The complexity of geometric models used in several applications continues to increase because of fundamental advances in 3D modelling, simulation and data capture technologies. Having a model which captures very fine surface detail may be desirable to keep a convincing level of realism and to ensure sufficient and accurate data to process it. However, many applications will require far less detail than is present in full datasets. In polygonal surface applications, a trade-off exists between the accuracy with which a surface is modeled and the amount of time required to process it. If a simplified representation of those models is used, potential gains can be obtained. Eliminating redundant geometry, reducing model size or improving run-time performance of the scene being rendered are the main motivations for using surface simplification techniques. Different aspects such as geometry or visibility can be taken into account when simplifying a model.

Surface simplification approaches can be classified into two different strategies. The first consists of generating a single and static low-resolution approximation that resembles the original one but has far fewer faces. In this case, the models are generally acquisitions from real objects using 3D laser scans. Single approximations of these virtual models, commonly applied in realistic virtual environments, CAD systems or medicine, are composed of a fixed set of vertices and a fixed set of faces. Although they provide a single fixed resolution representation of an object, this single resolution may not be appropriate for all the contexts in which the model will be used. This restriction gives rise to the second strategy, which consists of using the surface simplification technique on a level-of-detail approach.

Suppose we have a polygonal model M and we would like an approximation M'. While this approximation will has fewer polygons than the original, it should also be as similar as possible to M and retain all its basic features. The goal of polygonal surface simplification is to automatically produce such approximations (see Figure 2.23). It is a valuable tool for tailoring large datasets to the needs of individual applications and for producing more economical surface models. Surface simplification is naturally targeted towards large and complex datasets that would be very hard to manipulate manually. So, this is a well-known problem that has been the subject of a great deal of research.

2.3.3.1 Classification of surface simplification methods

Surface simplification algorithms define a heuristic to find simplified representations that are close to the optimal. The computation of the minimal-facet approximation within a certain

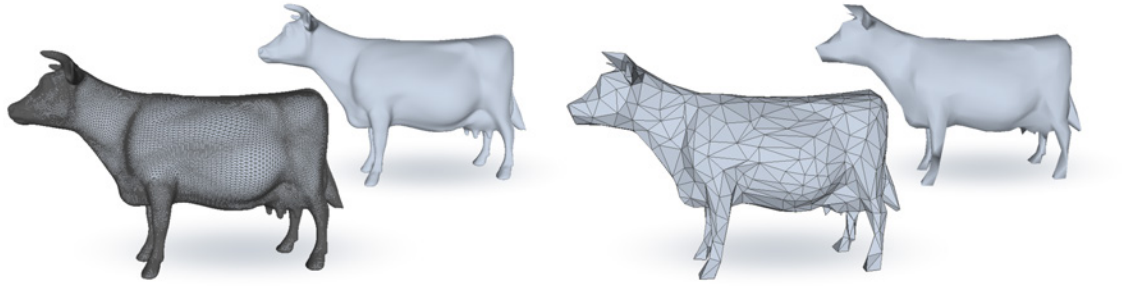

Figure 2.23: Simplification of a cow model.

error bound is a NP-hard problem for all objects. A characterization and classification of surface simplification methods is presented in [And99]. Following it, surface simplification methods can be classified into two types, depending on how they proceed:

- ***Decimation*** - *Top-down methods*
  In a decimation algorithm, unlike in refinement, a face reduction process is followed. It begins with the original surface and iteratively removes elements at each step using a face reduction strategy until the desired level of approximation is achieved. In contrast to refinement algorithms, accurate representations are produced first because coarse representations require more face reduction steps.

- ***Refinement*** - *Bottom-up methods*
  A refinement algorithm is an iterative algorithm that begins with an initial coarse approximation and uses a refinement strategy to add geometric elements at each step. Coarse representations are produced before accurate ones because this requires many more refinement steps. The main difficulty consists of constructing the base approximation, which must necessarily have the same topology as the original model. Even so, the sequence of refinement steps does not always lead to the original representation.

2.3.3.2 Decimation strategies

Most of the existing surface simplification approaches are defined by a decimation algorithm. In these cases, a face reduction strategy has to be followed during the process. Different face reduction strategies are summarized bellow:

- ***Vertex Clustering.*** The vertex clustering strategy consists of spatially partitioning the initial vertex set into a set of *clusters* and unifying all vertices inside a cluster

by a single vertex, called *cluster representant*. Vertex clustering often produces relatively poor quality approximations and tends to make substantial alterations to the topology of the original model. The results of this algorithm can be quite sensitive to the resolution and the placement of the grid cells, making it incapable of simplifying features larger than the cell size. Therefore, clustering methods tend to work well if the original model is highly over-sampled and the required degree of simplification is not too great. [RB93] is one of the most important methods of clustering simplification. The main steps of the approach are: weight computation, triangulation, vertex grouping, synthesis and elimination.

- ***Re-tiling.*** Re-tiling begins by introducing new vertices onto the original representation. And they are moved over maximal curvature locations. Then, a new triangulation is built, taking into account the original vertices and the newly added ones. Finally, original vertices are removed from the mesh by some local reduction operator. [Tur92] presents a surface simplification method that follows an original re-tiling approach.

- ***Wavelet decomposition.*** Simplification based on wavelet decomposition proceeds through two steps: re-meshing and wavelet parameterization. The surface is decomposed into a base mesh plus a sequence of successively finer surface details. Wavelet decompositions are generally unable to resolve creases on the surfaces and cannot construct approximations with a topology different from the original surface. A method that overcomes the subdivision connectivity[1] limitation is presented in [EDD+95].

- ***Incremental face reduction.*** Incremental face reduction strategies consist of the iterative removal of geometric entities through a local reduction operator chosen according to local geometric optimality criteria. In the literature, many incremental face reduction methods has been proposed. Some of the most important approaches presented are [HDD+93, CVM+96, GH97].

2.3.3.3 Local operators

Local reduction operators are the basis of the most common surface simplification methods. The main local operators for triangle meshes can be classified according to their purpose: *reduction operators*, which reduce mesh complexity; and *fitting operators*, which improve

[1] A mesh has subdivision connectivity when it can be obtained from a simple mesh through the recursive 4-to-1 subdivision.

the adjustment of the approximation. Next, the most common reduction operators are summarized:

- ***Vertex decimation.*** *Vertex decimation* operates on a single vertex by deleting that vertex and re-triangulating the resulting hole (see Figure 2.24). In each step of the decimation process, a vertex v is selected for removal, the operator eliminates it and all of their t incident triangles and the resulting hole is reconstructed with t-2 triangles. The vertex-removal operator preserves the topology of the mesh. The key ingredient of methods that use vertex decimation as operator is the selection of the vertex that will be removed.

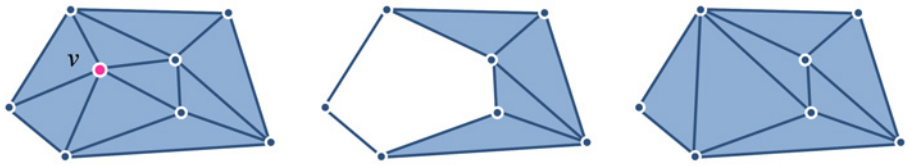

Figure 2.24: Vertex decimation operator. The elements reduction is: 2 faces, 3 edges and 1 vertex.

- ***Edge-collapse or edge-contraction.*** An *edge-collapse* takes as a parameter the edge to be collapsed or, equivalently, a pair of vertices sharing an edge $\{i, j\}$. The two vertices are collapsed in a single vertex h, updating all edges that are previously incident to i and j to reference h. As a result of the collapse, the triangles sharing the edge, degenerate in a segment and are removed (see Figure 2.25). The only computed parameter is the new vertex position, which is usually one of the two old vertices or a weighted average. Note that collapsing a vertex to one of the ends of an edge is equivalent to a vertex decimation operation without needing a re-triangulation step. Edge-contractions can alter the topology of a mesh, since repeatedly contracting edges can eventually close holes or join unconnected regions. Nonetheless, in most cases, the edge-collapse operator preserves the topology of the mesh. The selection of the edge that will be collapsed is the key ingredient in edge collapse methods. One of the benefits of iterative contraction is the hierarchical structure created. This quite naturally leads to a useful multiresolution surface representation.

- ***Vertex clustering or region merging.*** This can be seen as a sequence of *pair-contraction* operations. The pair-contraction operator takes as a parameter the two vertices to be collapsed. When these two vertices define an edge, this operation is equivalent to an edge collapse. Otherwise, when vertices (v_1, v_2) are disconnected, the

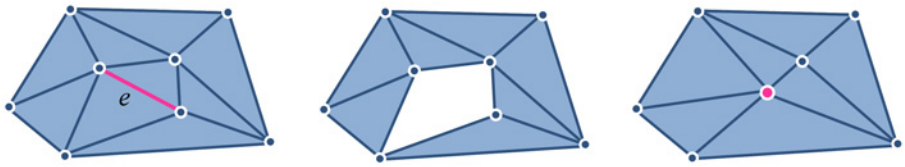

Figure 2.25: Edge-collapse operator. The element reduction is: 3 faces, 2 edges and 1 vertex.

process consists of moving the implied vertices to the new position, then connecting all the incident edges of one of the vertices to another and finally removing the vertex that was not actualized and edges or faces that have become degenerated (see Figure 2.26). The pair-contraction does not preserve the topology of the mesh because a sequence of these contractions can both close holes and connect regions originally disconnected. The key ingredient of methods that use vertex clustering is the selection of vertices to be collapsed.

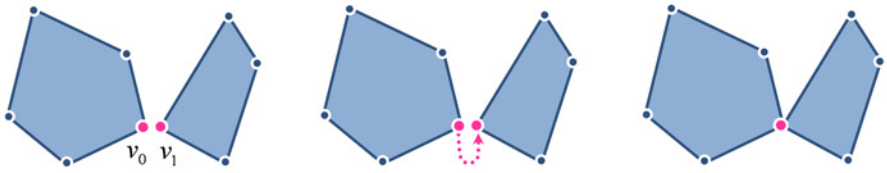

Figure 2.26: Pair-contraction operator. The element reduction is: 0-2 faces, 0-3 edges and 1 vertex.

- ***Face decimation.*** The face removal takes as a parameter a face F to be removed. This triangle and all its neighbours sharing one vertex with F are removed. The resulting hole is triangulated with the help of a new vertex (see Figure 2.27). The number of faces and edges which can be reduced with this operator is bigger than the others. Except in degenerate cases, the face decimation preserves the topology of the mesh.

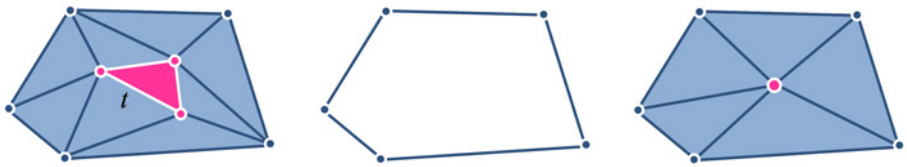

Figure 2.27: Face decimation operator. The element reduction is: 4 faces, 6 edges and 2 vertices.

In all face reduction operators, additional geometric tests for consistency checks are required to avoid fold-overs or self-intersections. Most surface simplification methods are based on the iterative contraction of edges, so they use the edge-collapse to perform the process.

As previously mentioned, in addition to reduction operators that reduce the number of geometric entities of the mesh, fitting operators allow the approximation to better match the original mesh. These operators preserve the topology of the mesh and the incident graph and also need additional geometric tests to avoid self-intersections. Next, two fitting operators are explained:

- ***Edge flip or edge swap.*** From two triangles sharing an edge, the non-planar quadrilateral result of merging together both triangles is triangulated using the opposite diagonal (see Figure 2.28). The edge flip operator is used for two purposes:
  - To improve the adjustment of the simplified model to the original surface especially in non-flat regions, and reduce the error that can be produced by other reduction operators previously applied.
  - To create well-shaped triangles, in flat regions.

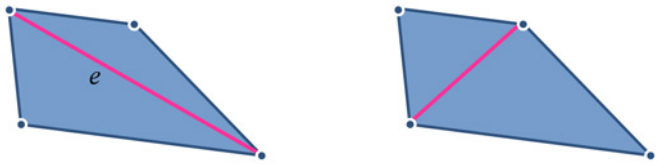

Figure 2.28: Edge flip operator.

- ***Vertex displacement.*** The operator that takes as a parameter the vertex to be moved and that only needs the new assigned coordinates, usually given as an offset vector (see Figure 2.29). Vertex displacement is used to improve the fitting of the simplification to the original model locally.

Figure 2.29: Vertex displacement operator.

All reduction operators that preserve topology can be derived from a series of *edge-collapses* and *edge-flips* and, moreover, they are the most commonly used.

2.3.4 Subdivision surfaces

Geometric mesh simplification is not the only form of LOD management. Subdivision surfaces are an alternative which have become a valuable tool in geometric modelling for computer graphics and computer aided geometric design (CAGD) due to their simplicity, efficiency and ease of implementation. The subdivision surface itself is defined as the limit of repeated recursive refinements (see Figure 2.30). The shape of the refined surface obtained from an iterative process is determined by a structured mesh of control points and a set of subdivision rules. A survey of subdivision surfaces can be found in [ZS00].

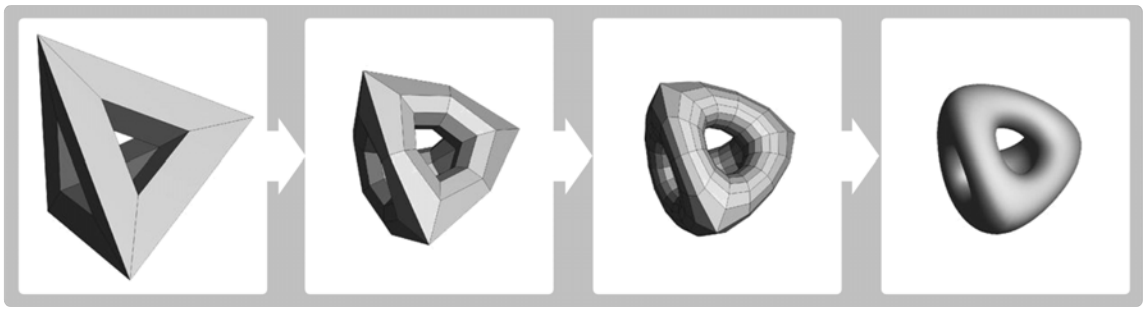

Figure 2.30: Recursive subdivision of a control mesh resulting in a subdivision surface.

The recursive nature of their definition makes subdivision surfaces suitable for many applications fields, such as animation (see Figure 2.31). These flexible modelling operators for two-manifold surfaces are used to construct smooth surfaces, multiresolution representations, model editors and mesh compression tools. Subdivision surfaces are ideal for interactive multiresolution mesh editing, where the overall shape of an object is controlled by a coarse mesh, while details are added by modifying the control points of a refined mesh. The computational efficiency, the use of arbitrary topology and the support of surface features and complex geometry are the main advantages from traditional splines. In addition to smooth surfaces, the management of boundaries and sharp features presented by some subdivision approaches allows more realistic objects to be represented.

In the literature several subdivision schemes have been proposed. One way to classify these schemes is based on their properties:

- ***Mesh type.*** Meshes can be of arbitrary topology but most subdivision schemes operate on triangular or quadrilateral meshes. A vertex is said to be ordinary if its

Figure 2.31: Subdivision surface model for Geri's Game [DKT98]. All images copyright ©1997 Pixar/Walt Disney Records.

valence is six in a triangular mesh or four in a quadrilateral mesh. A mesh, triangular or quadrilateral, is said to be regular if all their vertices are ordinary.

- ***Refinement rule.*** There are two main approaches used to generate a refined mesh: *face split* (vertex insertion) and *vertex split* (corner cutting). The schemes using face split can be applied to triangular and quadrilateral meshes. Each face is split into four faces of the same type (see the left-hand image of Figure 2.32). The schemes using vertex split can be applied to meshes of arbitrary topology. A new vertex is created for each old corner of each old face, as a linear combination of the old corners of that face. New faces are created linking these new vertices inside the old faces and across the old edges (see the right-hand image of Figure 2.32).

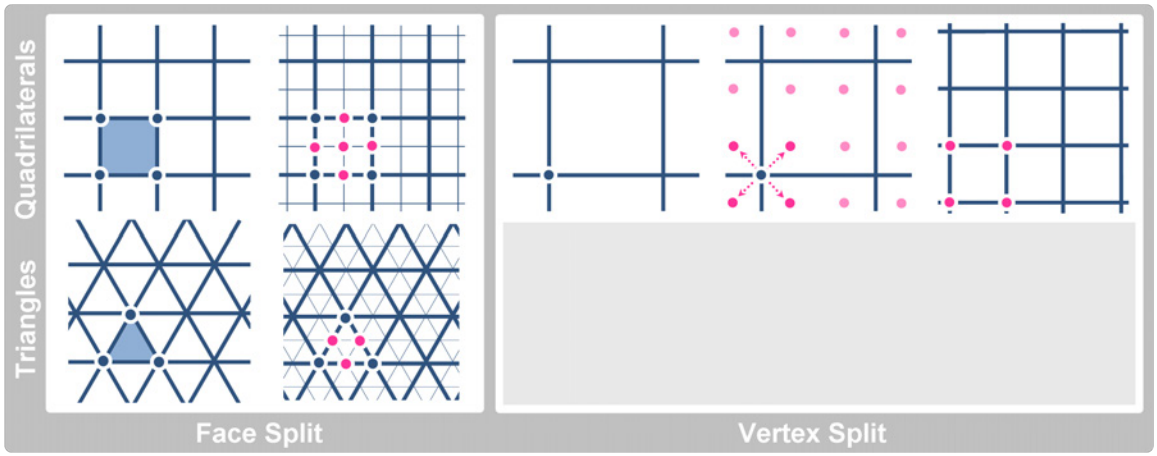

Figure 2.32: Face split and vertex split refinement rules according to the mesh type (triangles or quadrilaterals).

- ***Approximation vs. interpolation.*** A subdivision scheme is interpolating if all the control points in the original mesh are also control points in the subdivided meshes obtained. Otherwise, the scheme is approximating. Interpolation, the most commonly used, simplifies the computation and also allows the limit surface generation to be controlled more intuitively. Unfortunately, with respect to approximation schemes, the quality of the resulting surfaces is not as high and does not convergence as fast to the limit surface (see Figure 2.33).

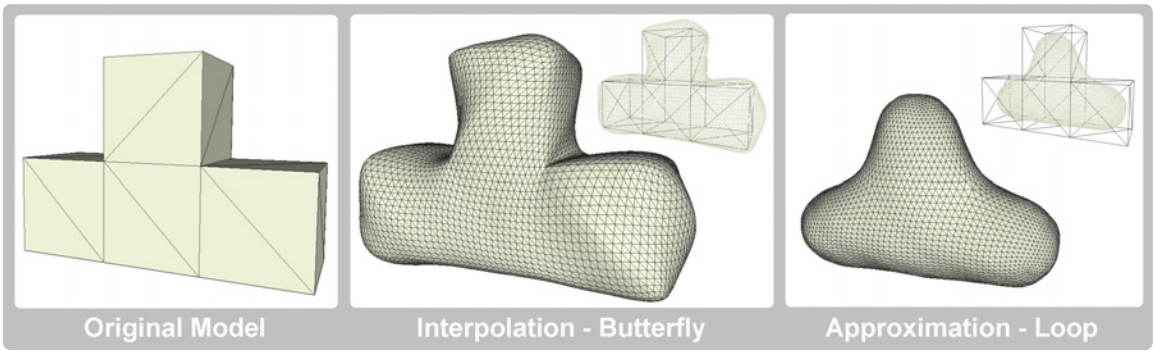

Figure 2.33: Comparison between interpolation and approximation schemes. In interpolation, the resulting mesh overcomes the control mesh while it is contained in approximation.

Tables illustrated in Figure 2.34 show a classification of the most popular subdivision schemes, taking into account the previous properties. The refinement technique used together with the placement of new vertices are the key points of a subdivision surface approach. Next, for each method, an analysis of the continuity for the generated surfaces is presented:

Face Split		
	Triangular Meshes	**Quadrilateral Meshes**
Approximating	Loop [C^2]	Catmull-Clark [C^2]
Interpolating	Butterfly [C^1]	Kobbelt [C^1]

Vertex Split
Doo-Sabin [C^1]

Figure 2.34: Classification of the subdivision schemes.

Doo-Sabin Scheme [DS98]. The subdivision is conceptually quite simple since there is only one mask used to compute the new vertices. Let p_i, $i = 0..n$, be the vertices of a face. The new vertex in the corner i is computed by $\sum_j a_j p_j$ where $a_i = \frac{n+5}{4n}$ and $a_i = \frac{3+2cos(2\pi(i-j)/n)}{4n}$ for $j \neq i$. In [PR98] is proved the C^1 continuity limit surface.

A special rule is required only for boundaries, where the limit curve is a quadratic spline.

Catmull-Clark Scheme [CC78]. The rules of this scheme are: a new face point is computed as the average of the vertices of the face; a new edge point as the average of the endpoints of the edge and the new face points of the adjacent faces; a new vertex point is computed by $\frac{F+2E+(n-3)V}{n}$ where V is the old vertex, n is the valence of V, F is the average of the new face points of all faces incident to V and E is the average of the midpoints of all edges incident to V. Special rules are needed for boundary points. The scheme produces surfaces that are C^2 continuous everywhere except at extraordinary vertices, where they are C^1 ([BS88] and [PR98]).

Loop Scheme [Loo87]. The rules are: a new edge point is computed by $\frac{3E+E'}{4}$ where E is the midpoint of the edge and E' the midpoint of the opposite edge; a new vertex point is computed by $(1-n\beta)V+\beta P$ where V is the old vertex, n is the valence of V, P is the sum of all n neighbours of V and $\beta = \frac{3}{16}$ for $n = 3$, or, $\beta = \frac{1}{n}(\frac{5}{8} - (\frac{3}{8} + \frac{1}{4}\cos(2\pi/n))^2)$ for $n > 3$. Special rules are needed for boundary points. Loop surfaces are C^2 at ordinary vertices and C^1 at the others. The original scheme was extended by [HDD+94] to incorporate sharp creases, darts and corner points. Next, [Sch96] further extended the scheme with conical and cusp points.

Butterfly Scheme [DLG90]. A new edge point is computed by $E + \frac{E'}{4} + \frac{P}{16}$, where E is the midpoint of the edge, E' the midpoint of the opposite edge and P is the sum of the four opposite vertices of the two faces adjacent to the edge. A special rule is needed for new boundary edge points. The original Butterfly scheme proposed by Dyn et al. is C^1 on regular meshes and is defined on arbitrary triangular meshes although the limit surface is not C^1 continuous at extraordinary vertices of valence $n = 3$ and $n > 7$ [Zor98]. In [ZSS96] a modified Butterfly scheme was presented which guarantees that C^1 continuous surfaces for arbitrary meshes are produced.

Kobbelt Scheme [Kob96]. The new points are computed in two steps. First, all new edge points are computed. Next, all new face points are computed by the same edge rule applied to the edge connecting new edge points on two opposite edges of the face. The edge rule for regular meshes of valence four is as follows. All mesh vertices are indexed by $V_{i,j}$. Then, the point on the edge $V_{i,j}V_{i+1,j}$ is computed by $\frac{9}{16}(V_{i,j}+V_{i+1,j}) - \frac{1}{16}(V_{i-1,j}+V_{i+2,j})$. When vertices $V_{i,j}$ or $V_{i+1,j}$ are not ordinary, this rule needs to be modified. See [Kob96] for more details of this scheme. C^1 continuity for interior vertices for all valences is proven in [Zor00].

2.4 Mesh Deformation

Mesh deformation is a valuable tool for editing techniques such as geometric modelling and computer animation (see Figure 2.35), since it provides a convenient way to edit the original mesh to meet various design requirements. In this section, an introduction to the shape deformation concept is first given by describing the main differences between surface-based and space-based deformations. Then, cage-based deformation methods are introduced as some of the most important space-based methods. The four main cage-driven approaches are described by giving a summary of their properties.

Figure 2.35: Mesh deformation applied to obtain different poses of the Nemo model. All images copyright ©2003 Pixar/Walt Disney Records.

2.4.1 Surface vs. space deformations

The wide range of applications such as industrial and artistic design makes mesh deformation an active area of research in geometric modelling. In recent years a wealth of research has been devoted to the deformation and manipulation of surfaces, specially those that are represented by triangular meshes [BS08, SB09].

Shape deformation techniques can be divided into two main groups: surface deformation and space deformation. Surface-based algorithms directly deform the surface. Because of their ability to develop detail-preserving techniques, they have been closely studied in the literature. In contrast, space deformation techniques, also known as free-form deformations, have received less attention in recent years. The deformation, in this case, is applied over a volume or some defined space. Nowadays, these methods have started to gain interest due to all the advantages presented with respect to surface-based techniques. An interesting discussion about the potential of these two approaches was presented in [CO09].

The main challenge in surface deformation techniques is handling non-trivial transformations while preserving the visual characteristics of the shape as much as possible at

interactive rates. Surface-based approaches achieve high quality shape-preserving deformation. However, these methods require solving large, often non-linear, systems of equations, making them hard to compute.

Space deformation techniques were introduced by Sedeberg and Parry [SP86] and further extended by Coquillart [Coq90], among others. The basic space deformation technique defines a lattice with too few control points enclosing the subject model. Manipulating the control points induces a smooth deformation of the space enclosed in the lattice, and hence the geometry of the enclosed model. Space deformation techniques can handle arbitrary inputs such as meshes, point sets or polygonal soups due to the deformation that can be applied over a volumetric space. The complexity depends mainly on the control object, not on the surface, making it easier to analyze. In fact, the deformation is only loosely aware of the shape that is being edited. Actually, some of the most important space deformation techniques are cage-based deformation methods [Flo03, JMD+07, LKCOL07, LLCO08].

Recently, in addition to these two mesh deformation techniques, hybrid methods have begun to emerge. They fuse both surface deformation and space deformation techniques in an efficient way. [SSP07], [XWXC08] or [HSL+06] are some examples of hybrid methods that combine the advantages of these two techniques.

2.4.2 Cage-based deformation methods

Cage-based deformation methods are space deformation approaches which define a general control polyhedron that encloses the model to gain more control over the whole interior (see Figure 2.36). This control element, called *cage*, is a rather low polygon-count polyhedron that typically has a topology and geometry similar to the enclosed object to obtain more accurate deformations. The deformation is performed by manipulating the cage vertices. Smooth deformations are therefore induced in all the volume inside the cage. The main advantages of these space deformation methods are their simplicity and speed. An enclosed mesh can be manipulated at a rather small computational cost since the deformation of its vertices involves only a linear combination of the cage geometry and the precalculated coordinates. Therefore, these techniques are independent of the surface representation and also free of discretization errors.

The points inside the cage are represented by affine sums of the cage elements (vertices or faces) multiplied by special weight functions called *cage coordinates*. Let $V = (v_1, v_2, \dots v_m)$ be the vertex set of the cage C. Most of the existing cage-based deformation methods

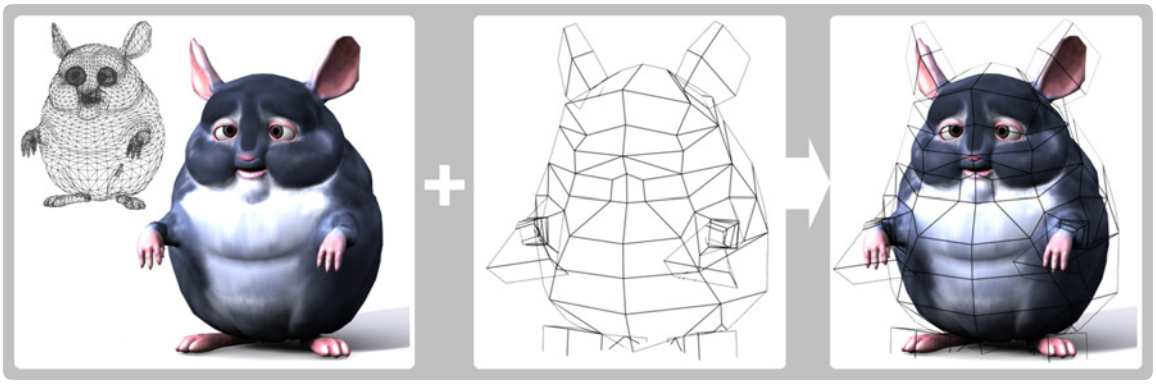

Figure 2.36: The original model enclosed by a control mesh in a cage-based deformation method.

express a point $p \in C$ as an affine combination of its vertices:

$$p = \sum_{v \in C} w(v, p) v \,, \tag{2.10}$$

where $w(v, p)$ are the coordinate basis functions, which are computed in different ways depending on the cage-based deformation method used. Then, the natural way to define a deformation in the cage C is by:

$$T(p) = \sum_{v \in C} w(v, p) T(v) \,, \tag{2.11}$$

where $T(v)$ are the deformed control cage vertices (see Figure 2.37). In this manner, transformations T can reproduce linear transformations. However, the resulting transformations, which are not shape-preserving, generate possible distortions on local surface details. Next, the three main cage coordinates, defined according to the previous nomenclature s a generalizations of the barycentric coordinates, are summarized:

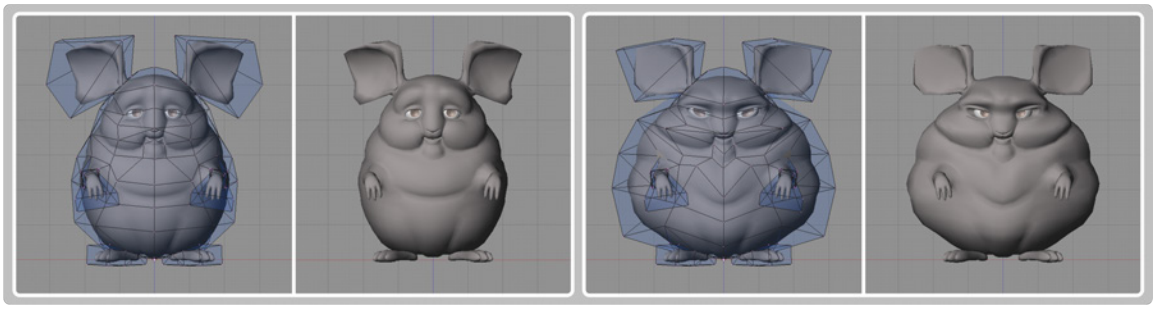

Figure 2.37: Two different deformations obtained by modifying the control mesh vertices.

Mean Value Coordinates [Flo03]. MVC are a simple and powerful method for creating functions that interpolate values assigned to the vertices of a closed mesh (see Figure

2.38). The coordinates $w(v,p)$ are computed as follows. Denote by F the set of the triangle faces of C and by N_v the union of the triangle faces $T \in F$ in the 1-ring neighbourhood of a vertex v of C. Let $\Phi_v(x)$ be the piecewise linear function defined on the boundary ∂C of C such that $\Phi_v(v) = 1$ and $\Phi_v(v') = 0$ for every vertex $v' \neq v$ of C. Given a point $p \in C$, let S_p be the unit sphere centred at p. The weight $w_v(p)$ is defined as follows:

$$w_v(p) = \sum_{T \in N_v} \int_{x \in \overline{T}} \frac{\Phi_v(p(x))}{|p(x) - p|} d\overline{T}\,,$$

where $\overline{T}$ is the projection of the triangle T onto S_p, and $p(x)$ is the projection of point x onto T in the direction $x - p$. Finally, the weight function $w(v,p)$ is computed by:

$$w(v,p) = \frac{w_v(p)}{\sum_{u \in V} w_u(p)}\,.$$

Then, MVC have a closed-form formulation and are C^∞ continuous and well defined both inside and outside of the control mesh but only C^0 continuous across the cage faces. Linear precision, interpolation and interior smoothness are the main properties related to MVC. However, MVC can be negative for non-convex cages and, consequently, can produce unacceptable deformations (see the left image of Figure 2.40).

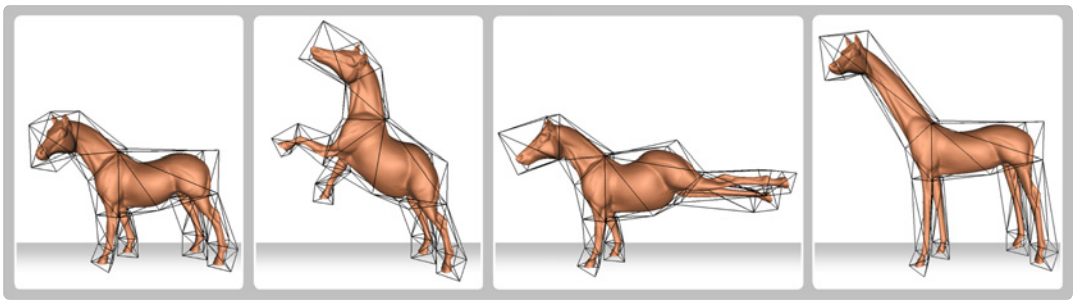

Figure 2.38: Mean value coordinates.

Harmonic Coordinates [JMD+07]. HC introduced were guaranteed to be non-negative on the interior of the cage, even in strongly concave situations (see Figure 2.39). A coordinate $w(v,p)$ satisfies:

$$\nabla^2 w(v,p) = 0, \quad p \in \mathrm{Int}(C)$$
$$w(v,p) = \Phi_v(p), \quad p \in \partial C\,.$$

Since these coordinates are solutions to the Laplace equation and these kinds of solutions are generically referred to as harmonic functions, these coordinates are called

harmonic coordinates, and the deformations they generate *harmonic deformations*. HC are non-negative, C^∞ continuous inside the cage, C^0 continuous on the boundary and have no definition outside the cage. However, the main limitation of the approach is the lack of a closed formed expression meaning a multi-grid finite difference must be used to compute the coordinates. The approximate nature of the method makes the accuracy of the results obtained depend on the cell size used in the computation. The computations and the memory required make it one of the costliest cage-based approaches.

Figure 2.39: Harmonic coordinates.

Positive Mean Value Coordinates [LKCOL07]. PMVC are alternative non-negative coordinates which are computed numerically by using a GPU-friendly approach. Only the visible portion of the cage with respect to a point p is considered to guarantee, like HC, that coordinates $w(v, p)$ are always positive. Although PMVC and HC perform similarly, the PMVC are computed much faster (see Figure 2.40). The main inconvenience of this approach is that PMVC are discontinuous on the visibility graph of the cage. Moreover, the quality of the results depends on the cage resolution used for the visibility computation carried out on the GPU.

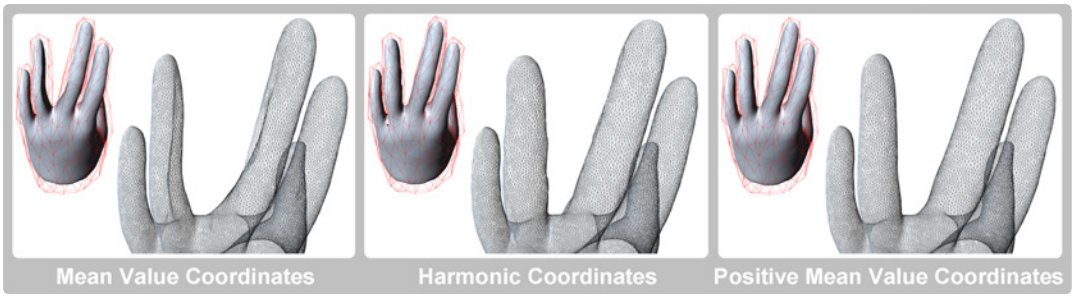

Figure 2.40: Positive mean value coordinates.

To preserve the shape and details of the enclosed surface, Lipman et al. [LLCO08] presented green coordinates (GC). In contrast to the previous work, this new cage-based approach induces conformal mappings in 2D that become quasi-conformal mappings in 3D (see Figure 2.41). Unlike the other methods presented before, the shape-preserving property is obtained because the cage face normals, rather than just vertex positions, are also taken into account to compute the cage coordinates. A point $p \in C$ is expressed by:

$$p = \sum_{v \in V} w(v,p)v + \sum_{T \in F} \Psi(T,p)n(T)\,,$$

where $n(T)$ is oriented outward normal to T. Let $G(x,p)$ be a fundamental solution of the Laplace equation in $\mathbb{R}^d$ $(d = 2,3)$, which has the following expression:

$$G(x,p) = \begin{cases} \dfrac{-1}{4\pi|x-p|} & d = 3 \\ \dfrac{\ln|x-p|}{2\pi} & d = 2 \end{cases}\,.$$

The vertex coordinate $w(v,p)$ is computed by:

$$w(v,p) = \sum_{T \in N_v} \int_{x \in T} \Phi_v(x) \frac{\partial G(x,p)}{\partial n(T)} dT\,,$$

while the face coordinate $w(T,p)$ is computed by:

$$\Psi(T,p) = -\int_{x \in T} G(x,p) dT\,.$$

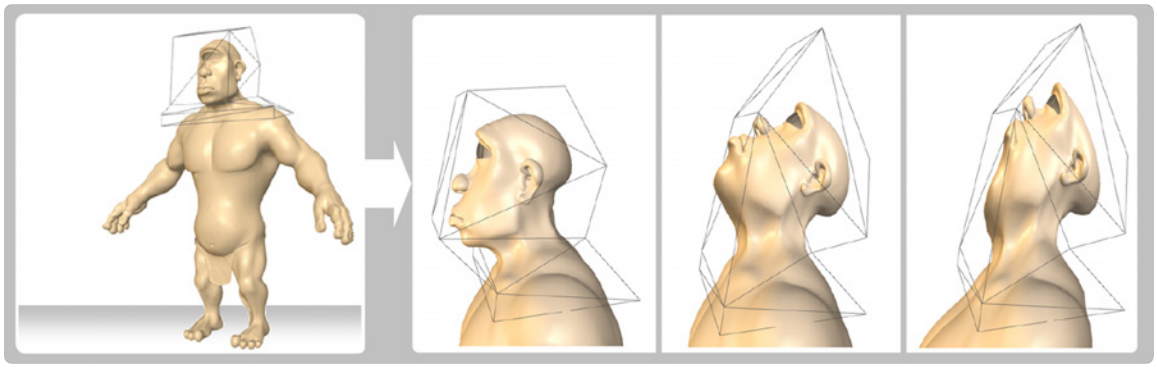

Figure 2.41: Green Coordinates.

GC are simple coordinates with a closed-form expression that achieves similar high quality shape-preserving deformations as surface-based approaches. The coordinates are C^1 continuous inside and outside the cage but discontinuous at the boundary. However, the deformation is not interpolatory, which can be considered a limitation in applications that require interpolation of the boundary of the cage.

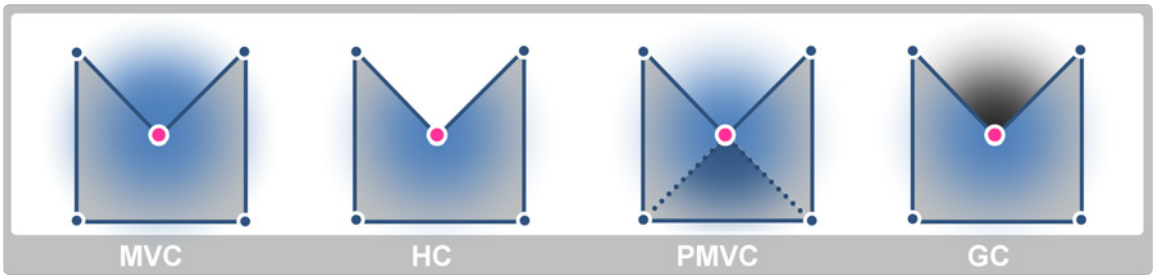

Figure 2.42: Scheme of coordinates continuity between cage boundaries for each cage coordinate function.

In general terms, the main drawback of cage-based deformation methods is the need to construct a cage or other structure around the surface to be manipulated. Moreover, the manipulation of the model could be limited by the geometry and the flexibility of the designed cage. All these cage coordinates are defined smoothly over the whole interior of the cage (see Figure 2.42). Besides this common point, the different properties of each one allow it to obtain different deformation results. The choice of which coordinate function to use depends on the application needs.

Chapter 3

Simplification of Multi-chart Textured Models

In this chapter we present the first stage of our study based on surface simplification techniques, previously introduced in Section 2.3.3, dealing with textured models. We have developed an accurate surface simplification method for the automatic reduction of a highly detailed multi-chart textured model into a single faithful approximation containing fewer polygons. The method consists on reparameterizing the model at each edge-collapse by local bijective mappings to avoid texture distortions and chart boundary artefacts on the simplified mesh due to the geometry changes. These mappings are performed on the GPU by using the standard hardware capabilities. Moreover, the simplification process is driven by a quadric error metrics weighted by a local area distortion measure not only to guarantee geometric fidelity, but also to better apply the appearance attributes. The main benefit of our approach is that realistic and accurate approximations of complex textured models can be generated.

3.1 Introduction

The surface simplification for automatically generating high-quality, appearance-preserving approximations of original models has been deeply studied in the last years. There are two different strategies in the simplification field. First, there is level-of-detail modelling, which focuses on the trade-off between fidelity and rendering performance (see Section 2.3.1). The aim is to obtain a sequence of geometric approximations that share the same texture map and switch them during rendering, depending on the distance between the object and the

viewer. The second strategy consists of generating a single, low-resolution approximation that resembles the original one but has far fewer faces. In this case, the models are generally acquisitions from real objects using 3D laser scans. Single approximations of these virtual models are commonly applied in several fields, such as realistic virtual environments. We present a new approach to this second strategy. The method proposed generates a single and accurate, low-resolution approximation of a multi-chart textured model that preserves geometric fidelity while avoiding texture distortions and artefacts.

3.1.1 Related work

In recent years, several simplification algorithms have been proposed. Most of the existing methods follow a face-reduction strategies that are commonly based on the edge-collapse operator. Edges are ordered according to a cost and in each iteration, the lowest-cost edge is contracted and the cost of its neighbouring edges is updated.

Geometric simplification of meshes was the first topic studied in this field. The simplification process was guided only by a geometric error, without taking into account the appearance attributes of the model. An extensive review of all existing methods proposed before 2002 can be found in [LWC+02]. Some of the main references are [HDD+93, Hop96, CVM+96, GH97, EM99, LT98].

More recently, simplifying *meshes with appearance attributes* has become the main area of interest. In this case, the local simplification operators try to generate faithful geometric approximations while minimizing texture deviation. The texture deviation of a mapped attribute is defined as the deviation of its position on the final surface from its position on the original surface. In [GH98], Garland and Heckbert extended the original Quadric Error Metric (QEM) scheme to account for a wide range of vertex attributes. Hoppe [Hop99] made several improvements to Garland and Heckbert's approach by computing the error based on the difference, in position and attribute, between a given point and its closest projection on the simplified surface. Cohen et al. [COM98] proposed a simplification algorithm guided by the texture deviation. Lindstrom and Turk [LT00] introduced the image-driven simplification concept, where multiple images were used to decide which portions of a model have to be simplified. Zhang and Turk [ZT02] proposed a new image-based algorithm that combines the QEM with a visibility function defined between the model and a surrounding sphere of cameras. New viewpoint-driven simplification approaches were presented in [CSCF07, CSCF08b, CSCF08a]. These methods use the variation in viewpoint quality to drive the simplification process. Previously to these image-based metrics, Sander et al. [SSGH01] presented a multi-chart parameterization, constrained to convex and straight

chart boundaries, to be used in a progressive mesh. Unfortunately, for a small number of charts, the chart boundaries do not usually follow the 'creases' of the model and, consequently, distorted results may be obtained.

These approaches guide the simplification process by measures that try to balance geometric fidelity and texture preservation. Nevertheless, significant texture distortions and artefacts can be observed on the simplified surfaces. The fundamental reason is that the texture map, even when it is very well parameterized, is accessed through linear interpolation of texture coordinates on the simplified triangles. One way to minimize the texture distortion is the texture adaptation technique, proposed by Chen and Chuang [CC06], which adapts the texture content for each edge-collapse. However, commonly used multi-chart parameterizations are not supported.

The presence of chart boundaries hinders the quality of the simplification due to the discontinuity in texture space. If boundary edges are not allowed to be collapsed, the resulting approximation presents a poor quality mesh with a lot of skinny triangles near the seams. Furthermore, texture distortions or artefacts appear near chart boundaries.

There are two main strategies to avoid getting seams. The first one consists of simplifying the geometric model with any of the surface simplification methods, and then parameterizing the simplified mesh and finally projecting the attributes, such as texture, of the high polygonal model onto the lower one ([TCS03]). The fidelity of the result depends on various factors, such as the complexity of the original mesh, the quality of the simplification method or the point-to-point correspondence strategy used to project the attributes. The most commonly used strategy is normal projection that needs the continuity of the normal field over each face and does not guarantee that every attribute of the texture is projected onto the simplified surface. The second strategy is to choose a texture domain with the same topology as the given mesh and a similar shape. The survey by Hormann et al. [HLS07] gives complete, detailed information on these seamless texturing parameterization techniques. One of the most powerful of these methods is PolyCube-Maps ([THCM04] and [LJFW08]). A polycube is a 3D shape, composed of many unit-sized cubes attached face-to-face, which is used as the texture domain. However, the main drawback of PolyCube-Maps is the fixed resolution that has to be carefully chosen to match the geometric features, which would give a parameterization with too many cubes for a surface with complex geometry or topology. [GP08] proposes a method that allows the use of a seamless parameterization for simplification purposes that is independent of the parameterization provided by the artist for texturing. The main drawback of this technique is the use of a common bijective parameterization for all the levels of detail.

3.2 Overview of our Proposal

When simplifying a textured surface, S, the triangles created for each edge-collapse have to be parameterized, that is texture coordinates have to be assigned to the vertices of the new triangles. However, this is not sufficient to prevent texture distortions due to the linearity of the texture mapping. Therefore, we propose not only to modify the texture coordinates, but also to modify the texture content in order to preserve the appearance of the model more accurately.

First the model is parameterized by an index texture, I, whose texels store texture coordinates referring to the original texture, T. This index texture, I, is modified in the course of edge collapsing. We denote by S^i and I^i the simplified surface and the index texture before the collapse of the edge, e_i. The edge-collapse operator modifies I^i and we obtain I^{i+1} from the triangles involved in S^i, their corresponding texture triangles in I^i and the triangles involved in S^{i+1}.

At first, it seems reasonable to assign to a point on S^{i+1} the texture attribute of its closest point on S^i. However this criterion does not produce a bijection between S^{i+1} and S^i and, consequently, texture attributes may be lost. In [CC06], this criterion is applied only for a reduced number of points in the following way: given two edges of the involved triangles, one of S^i and one of S^{i+1}, whose corresponding texture edges intersect, their closest points are computed and the texture attribute of the point on S^i is assigned to the point on S^{i+1}. These assignments create two cell partitions, one in I^i and one in I^{i+1}, and a cell correspondence between them. The texture modification is done by mapping a cell in I^i to its corresponding cell in I^{i+1}. The main drawbacks of this technique are that cell overlapping may be produced and that a large number of point assignments and cells are needed for computation when full-edge collapses are used.

In contrast, our method takes advantage of the orthogonal projections of the surface triangles involved in S^i and S^{i+1} onto the tangent plane, π, of the point of collapse, v_i. Only when these two projections are bijective is the edge-collapse allowed. The new triangles in I^{i+1} are filled with a fragment shader by first mapping the triangles in I^i to the projected triangles in plane π, and then mapping these latter triangles to the new triangles in I^{i+1}. In fact, these two mappings are linear bijections and can be merged into just one by using their composition. In this way, S^i and S^{i+1} have the same appearance from a viewpoint placed on the normal direction of v_i. Observe that, in contrast to point-to-point correspondence strategies, in our approach, every texture attribute will appear on the simplified surface.

Moreover, the texture deviation has to be minimized, that is the distance from the point on S^i of an attribute in texture I^i and the point on S^{i+1} of the same attribute in texture I^{i+1} must be minimal. Consequently, the more 'similar' a surface triangle is to its projected triangle onto the tangent plane and to its corresponding texture triangle, the more minimal the texture deviation is. Because the position of a point in a polygon can be derived from the areas of the triangles determined by the point and the polygon edges, we use the area as a measure of similarity between triangles. Consequently, our simplification process incorporates a measure of distortion between the area of the surface triangles and the area of their projections onto the tangent plane, and the new triangles in I^{i+1} are determined by the areas of the new involved surface triangles in S^{i+1}.

Therefore, the fundamentals of our method, which will be described in more detail in the following sections, are as follows:

- Using an index texture that avoids blurring (Section 3.3).
- A simplification process based on a modification of the QSlim [GH97] simplification method by weighting the quadrics with a local area distortion measure to preserve highly curved regions and consequently better apply the texture of the original model (Section 3.4).
- An edge management that (Section 3.5)
  - decides if a candidate edge can be collapsed by taking into account its adjacent surface triangles and their corresponding texture triangles and
  - determines the new surface triangles and the new texture triangles.
- For each edge-collapse, the mesh is reparameterized by local bijective mappings to avoid distortions to the appearance of the simplified mesh produced by geometric changes (Section 3.6). These bijective mappings are GPU-friendly: fully supported by the texture mapping hardware, the render-to-texture feature and the fragment shading. To avoid blurring artefacts due to an excessive number of resampling operations, the mappings are applied to an index texture. Each texel of this index texture stores the texture coordinates referring to the original texture. The technique supports any arbitrary multi-chart parameterization with the only requirement being that the index texture has to be empty near concave parts of its chart boundaries. In Section 3.7, we will see that this is not a strong requirement. Subsequently, our approach allows the chart boundary edges to be simplified, guaranteeing geometric fidelity and avoiding artefacts.

3.3 Index Texture

At the beginning of the process, an index texture, I, encoded with two float channels to get enough precision, is created with the same size as T, and each one of its texels stores its respective coordinates. Texture I can be created with the same parameterization as the original model or with another one. When using a parameterization method that minimizes the distortion between surface triangles and texture triangles, such as [LPRM02, SCOGL02, SWG$^+$03] or [ZSGS04], more accurate results are obtained (see Section 2.2.3). The whole process is applied to I, producing the successive modifications, I^i. In this way, the original texture, T, is not under-sampled and, consequently, blurring artefacts produced by the iterative resampling do not appear during the simplification process. Once the simplified level is obtained, we generate the texture of the output model by transferring the contents of the original texture via the I texture coordinates.

3.4 Weighted Quadric Error Metrics

The QSlim [GH97] algorithm is an incremental method that simplifies a mesh by iteratively collapsing edges ordered by increasing errors provided from the QEM.

Let F be the set of faces incident at vertex v. Each face $f \in F$ is contained in a plane π_f defined by $n \cdot u + d = 0$, where n is its unit normal. The squared distance of any point u to this plane is $(n \cdot u + d)^2 = u^T(n \cdot n^T)u + 2(dn)^T u + d^2$. The fundamental quadric Q_f of π_f is defined by $Q_f = (A, b, c) = (n \cdot n^T, dn, d^2)$, and the squared distance can be computed as $Q_f(u) = u^T A u + 2b^T u + c$. The quadric of the vertex, v, is defined as the weighted sum of these fundamental quadrics,

$$Q = \sum_{f \in F} Q_f w_f \,,$$

where w_f is the area of the triangle, f. Given a quadric Q, point $\overline{u} = -A^{-1}b$ minimizes the quadratic error $Q(u)$. After computing quadrics for all vertices, the contraction cost or the error of every edge, $e = (v_i, v_j)$, is computed by $(Q_i + Q_j)(\overline{u}_{ij})$, where point $\overline{u}_{ij}$ minimizes the quadratic error $(Q_i + Q_j)(u)$, where Q_i and Q_j are the quadrics of vertices v_i and v_j.

The algorithm has many advantages, which are justified theoretically in [HG99]. The most important quadric properties are the preservation of the Gaussian curvature of the model and the good aspect ratio of the simplified model triangles. The authors prove that, in the limit, the quadric error is minimized by triangulations with optimal aspect ratio. Notwithstanding all these advantages, important features of the objects may not be well

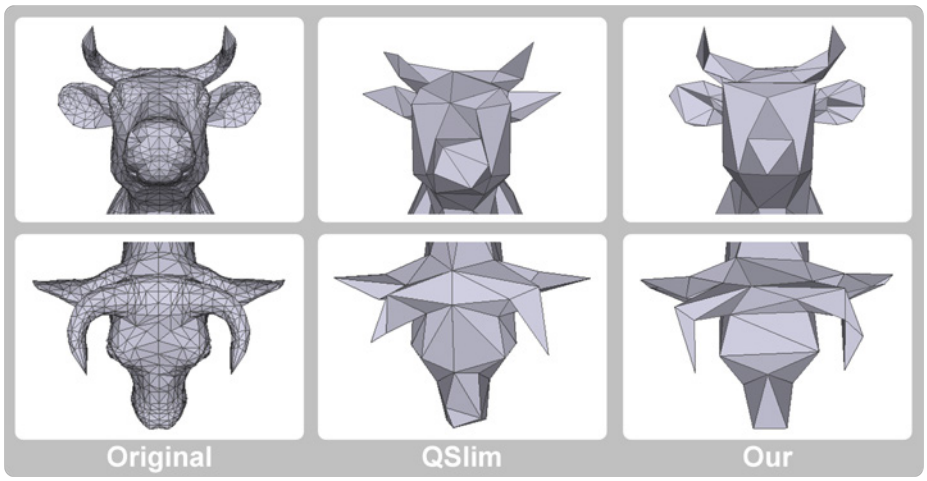

Figure 3.1: Curved regions (horns and ears) not preserved by the QSlim approach. In contrast, they are well preserved by our approach. Both simplified versions have the same number of faces.

preserved due to their low geometric error with respect to the whole model. To illustrate this, we show an example in Figure 3.1. In the image, we can see that the ears and the horns of the simplified cow are not well preserved. Moreover, this behaviour produces a poor visual appearance of a simplified textured model due to high texture distortions near these features.

Kho and Garland [KG03] introduced a user-guided simplification method by weighting the QEM of each vertex. The order of the edge contractions is manipulated by multiplying the quadric of each vertex by some scalar factor at the initialization step. By selecting different weights the user can control the relative importance of different surface regions to preserve the desired features. For view-dependent simplification, Zhang and Turk [ZT02] weight the QEM of each vertex by a visibility function. The visibility function is defined for a point on the surface of the mesh and it is the percentage of the camera space that can see the point giving more weight to views at better angles. Lee et al. [LVJ05] weighted the QEM of each vertex by its saliency. They define the mesh saliency using a centre-surround operator on Gaussian-weighted mean curvatures to capture what most would classify as visually interesting regions on a mesh. The main drawback of this method is the empirical and heuristic nature of the saliency measure, which needs user parameters.

Other metrics to guide the simplification process have been proposed. Among them, information-theoretic metrics: *viewpoint entropy* in [CSCF07], *viewpoint mutual information* in [CSCF08b] and *viewpoint f-divergences* in [CSCF08a]. These metrics measure the

amount of information from a scene that arrives at a certain viewpoint and use its variation to measure the edge-collapse error. Otherwise, Park et al. [PSC06] proposed an area-based metric. The contraction cost of every edge is computed by the absolute difference between the areas of the mesh before and after the contraction. In contrast, we want to minimize the distortion between the area of surface triangles and the area of their projections onto the tangent plane. For this, we propose to preserve the curved features by weighting the quadrics by a *local area distortion measure*. In the following sections, we will define this measure and how it is used in our simplification process.

3.4.1 Local area distortion measure

First, we need to recall some theoretical aspects of the QEM ([HG99]) that are used to define and justify the proposed *local area distortion measure*. Let n be the surface normal at vertex v, let k_1, k_2 be its principal curvatures and e_1, e_2 their corresponding principal directions. In the orthogonal coordinate frame with origin at v and axes e_1, e_2, n, the second-order local approximation of the surface (see Section 2.1.3) is a patch of the form

$$S(x,y) = \left(x, y, \frac{k_1x^2 + k_2y^2}{2}\right),\ (x,y) \in [-\epsilon_1, \epsilon_1] \times [-\epsilon_2, \epsilon_2]\,.$$

By considering only the lower terms of the Taylor series approximation, we have

- A is a diagonal matrix with entries

$$a_{11} = \frac{4}{3}\epsilon_1^3\epsilon_2k_1^2,\ a_{22} = \frac{4}{3}\epsilon_1\epsilon_2^3k_2^2,\ a_{33} = 4\epsilon_1\epsilon_2 - \frac{a_{11}+a_{22}}{2}\,.$$

- The area of the surface patch is

$$a_{11} + a_{22} + a_{33} = 4\epsilon_1\epsilon_2\left(1 + \frac{\epsilon_1^2k_1^2 + \epsilon_2^2k_2^2}{6}\right).$$

Thus, the distortion factor between the area $4\epsilon_1\epsilon_2$ of the domain and the area of the surface patch is

$$\frac{a_{11}+a_{22}+a_{33}}{4\epsilon_1\epsilon_2} = 1 + \frac{\epsilon_1^2k_1^2 + \epsilon_2^2k_2^2}{6}\,.$$

Observe that the area distortion depends on the term $\frac{\epsilon_1^2k_1^2+\epsilon_2^2k_2^2}{6}$, which we call the *local area distortion measure* and can be computed by

$$\frac{a_{11}+a_{22}}{a_{11}+a_{22}+2a_{33}} = \frac{\mathrm{trace}(A) - a_{33}}{\mathrm{trace}(A) + a_{33}}\,.$$

Due to the invariance of the eigenvalues of a matrix under bijective linear transformations, the local area distortion measure can be computed in the canonical coordinate frame by

$$\frac{\text{trace}(A) - \lambda}{\text{trace}(A) + \lambda}\,.$$

where λ is the A eigenvalue corresponding to the eigenvector closest to the surface normal at v.

3.4.2 Simplification process

We guide the order of the edge-collapses (see Section 2.3.3) using the weight derived from the local area distortion measure to guarantee a balance between the minimization of the geometrical error and the minimization of the texture deviation. This weight can be computed for the initial quadrics and for the newly created quadrics. Intuitively, recomputing the weights during the process seems a better solution. Experimental results have confirmed this and we have modified the simplification process in this way. At the initial step each vertex, v_i, has assigned to it the quadric $Q_i = (A_i, b_i, c_i)$. Then, we have to compute the contracting cost of each edge $e = (v_i, v_j)$ to create a keyed heap. We use the quadric $w_iQ_i + w_jQ_j$, where the weights w_i are computed by

$$w_i = \frac{\text{trace}(A_i) - \lambda_i}{\text{trace}(A_i) + \lambda_i}\,.$$

Consequently, the contraction cost of every edge $e = (v_i, v_j)$ is computed by

$$(w_iQ_i + w_jQ_j)(\overline{u}_{ij})\,,$$

where point $\overline{u}_{ij}$ minimizes the quadratic error $(w_iQ_i + w_jQ_j)(u)$. Moreover, the quadric assigned to the new vertex $\overline{u}_{ij}$ has to be

$$\frac{w_iQ_i + w_jQ_j}{w_i + w_j}$$

in order not to overestimate its weight since it will be recomputed.

This is a simple and efficient solution which allows us to preserve the curved features of the models. Consequently, although the global error increases slightly, the texture will be better mapped to the simplified model. Figure 3.1 shows an example of the results obtained by our approach using the cow model. There are evident differences between our results and those obtained by the QSlim approach. With our approach, the horns and the ears are preserved without losing the correct shape. Figure 3.2 shows a whole view and a close-up of the Armadillo model simplified by the QSlim approach and ours. It can be seen that, in

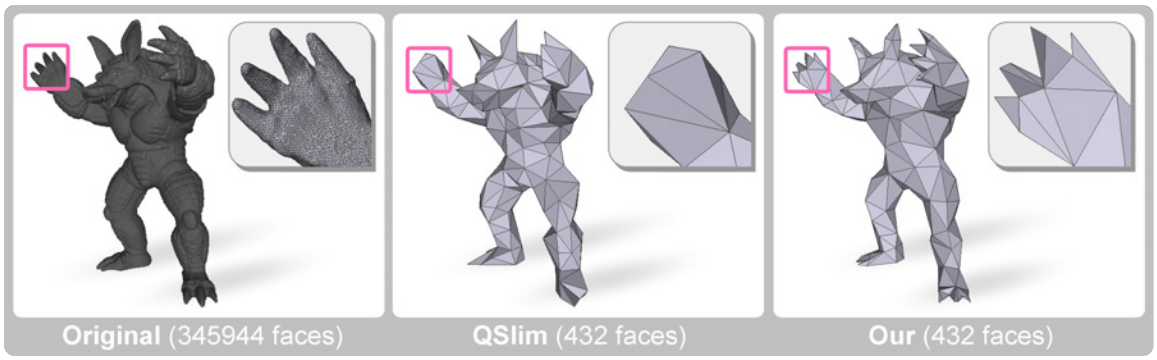

Figure 3.2: Simplification of the Armadillo model with QSlim and our approach. Comparison of the whole model and a close-up detail of its hand.

contrast to QSlim, our approach preserves all the fingers even when the approximation has only 0.125% faces of the original one. Figure 3.3 shows different levels of simplification of the Lucy model. The fingers disappear at low levels of detail by using QSlim approach as can be seen in close-up views. Finally, to illustrate the good behaviour of the simplification method Figure 3.4 shows a sequence of simplified models for pegasus and a gargoyle scanned models.

3.5 Edge Management

The edges have to be collapsed while both the surface and the parameterization topology are preserved. Thus, necessary consistency checks must be carried out ([DEGN99]).

In the following a triangle on S^i is denoted by t, its corresponding triangle in I^i by $\bar{t}$, a triangle on S^{i+1} by t' and its corresponding triangle in I^{i+1} by $\bar{t}'$. The same notation is applied for edges and vertices.

Let e be a candidate edge to be collapsed and v be its point of collapse with tangent plane π. Let $\{t_0, \ldots, t_k\}$ and $\{t'_0, \ldots, t'_{k-2}\}$ be the triangles involved in the collapse on S^i and on S^{i+1} respectively.

Edge e is not allowed to be collapsed if at least one of the following constraints holds:

- e has only one endpoint on a chart boundary;
- e is a non- chart boundary edge with both endpoints on a chart boundary;
- e has an endpoint lying on more than two charts;

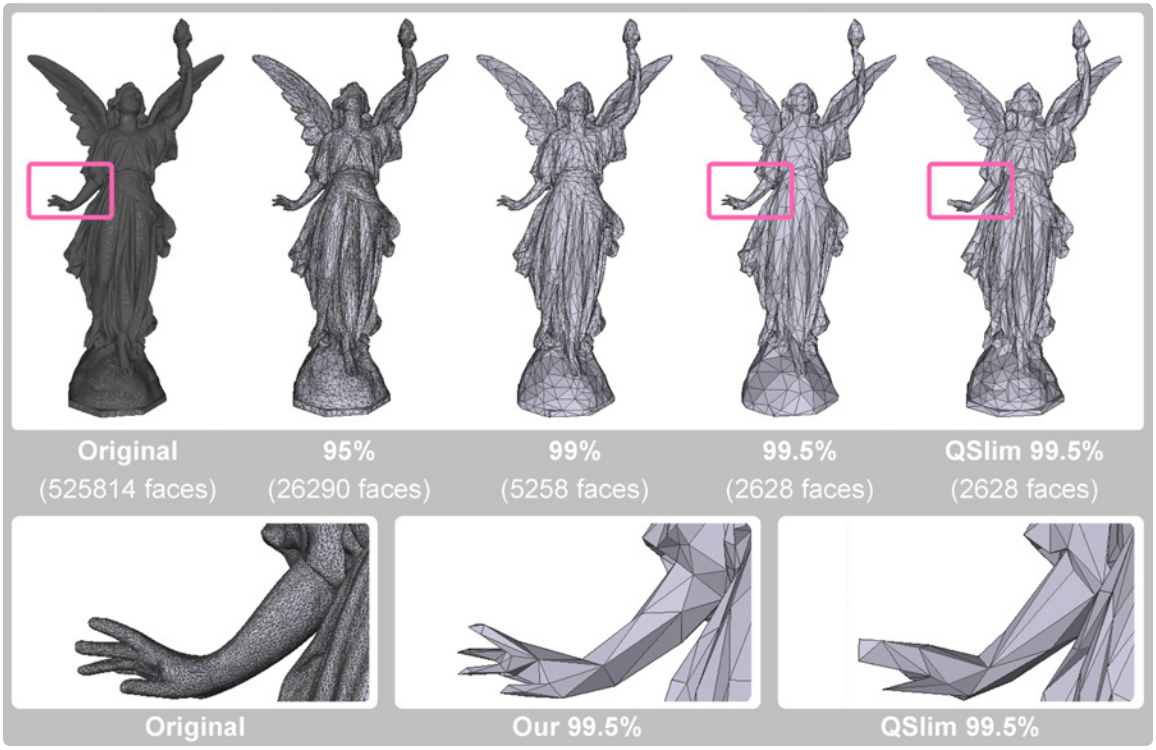

Figure 3.3: Lucy model simplification at different levels of detail by our method and QSlim. QSlim does not preserve the fingers at the low level.

- the orthogonal projection onto π of triangles $\{t_0, \ldots, t_k\}$ or triangles $\{t'_0, \ldots, t'_{k-2}\}$ overlap.

These constraints enforce chart compliance and ensure a bijection between S^i and S^{i+1}.

When an edge e does not fulfil any of the previous constraints, we need to determine the new triangles $\{\overline{t}'_0, \ldots, \overline{t}'_{k-2}\}$ in I^{i+1}. There are two possible cases depending on whether e lies on a chart boundary or not.

- **Chart interior edge**. Let $\overline{U}$ be the union of texture triangles $\{\overline{t}_0, \ldots, \overline{t}_k\}$. For each boundary edge b_j of $\overline{U}$, let $\overline{a}_j(x)$ be the relative area of the triangle determined by b_j and a point $x \in \overline{U}$ with respect to the area of $\overline{U}$, and let a_j be the relative area of the triangle t'_j with respect to the total area of the triangles $\{t'_0, \ldots, t'_{k-2}\}$. Because we want to minimize the area distortion measure between triangles in S^{i+1} and triangles in I^{i+1}, we take as the corresponding point of v in I^{i+1} the point $\overline{v} \in \overline{U}$ that minimizes the quadratic function

$$\sum_j \left(\frac{\overline{a}_j(x)}{a_j} - 1 \right)^2 .$$

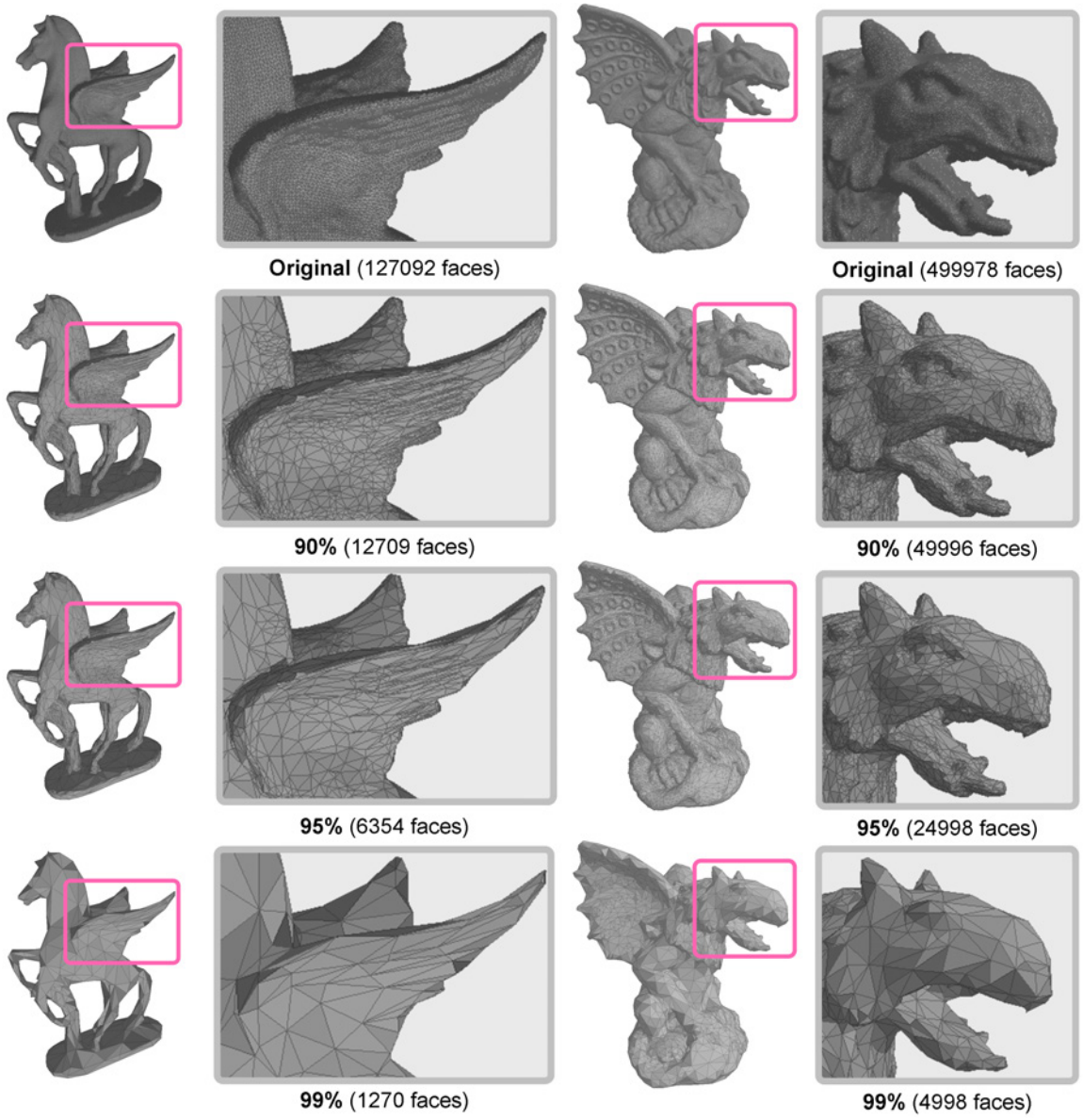

Figure 3.4: Simplification results on a pegasus and a gargoyle models.

Areas $\overline{a}_j(x)$ can be expressed by

$$\overline{a}_j(x) = \frac{1}{2}(x - b_{j1}) \cdot n_j = \frac{1}{2}(n_j^T x - n_j^T b_{j1}),$$

where b_{j1} is the initial point of edge b_j and n_j is the normal vector to b_j satisfying $|n_j| = |b_j|$. Then, the function to be minimized is

$$\sum_j \left(\frac{1}{2} \frac{n_j^T x - n_j^T b_{j1}}{a_j} - 1 \right)^2 .$$

Setting equal to zero the partial derivatives leads up to the equation:

$$\left(\sum_j \frac{1}{2}\frac{n_j n_j^T}{a_j^2}\right)\overline{v} = \sum_j \frac{1}{2}\frac{n_j n_j^T}{a_j^2} b_{j1} + \sum_j \frac{n_j}{a_j},$$

from which point $\overline{v}$ is obtained.

Taking into account the point $\overline{v}$ and the boundary edges b_j, we obtain the triangles $\{\overline{t}'_0, \ldots, \overline{t}'_{k-2}\}$. If these new triangles overlap, the edge e is also not allowed to be collapsed.

- **Chart boundary edge** (see Figure 3.5). There are two texture edges $\overline{e}_1$, $\overline{e}_2$ corresponding to surface edge e. Each one is a boundary edge of its own chart. For each subset of triangles of the same chart, we apply the same process as in the first case, restricting the optimal point to lie on the boundary edge, $\overline{e}_1$ or $\overline{e}_2$. Observe that, when a chart is not convex, the new triangles intersect the exterior of concave parts of the chart. For this reason, we need the texture to be empty near concave parts of the chart boundaries. Therefore, in the worst case, the convex hull of each chart should be empty.

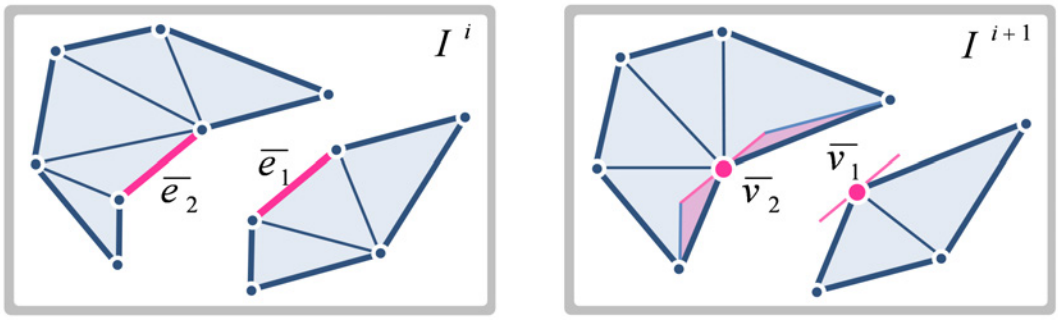

Figure 3.5: New texture triangles of a chart boundary edge.

3.5.1 Edge flip

To improve the fitting of simplified models to original models and to create well-shaped triangles in flat regions, we use an edge flip operator. It takes two adjacent triangles as parameters and swaps the shared edge with its opposite diagonal. This operator has to be guided by an accurate measure to decide which of the two edges, the actual or the opposite one, must be kept in order to better approximate the mesh to the initial one.

For each edge-collapse, non-chart boundary edges of triangles $\{t'_0, \ldots, t'_{k-2}\}$ are flip-tested. The criterion used to decide the flip operation is the one proposed by Jiao et al.

[JCNH06], which uses the restriction of the QEM onto the tangent plane of each vertex as a local metric, and applies the modified Delaunay criterion proposed by Bossen and Heckbert [BH96]. The non-overlapping checks are carried out on each tangent plane of the four vertices for the surface triangles and on the texture map for the texture triangles.

3.6 Bijective Mappings

Once an edge-collapse has been carried out, the texels in triangles $\{\bar{t}'_0, \ldots, \bar{t}'_{k-2}\}$ must be filled with the content of texels in triangles $\{\bar{t}_0, \ldots, \bar{t}_k\}$. As we briefly described in Section 3.2, our method takes advantage of the orthogonal projections of the involved surface triangles in S^i and S^{i+1} onto the tangent plane π of the point of collapse v. Thus, our basic idea consists of transferring the content of a texel $\overline{p}$ to a texel $\overline{p}'$ having the same projection onto π of their corresponding surface points. Figure 3.6 illustrates the situation and we will now describe how to implement it.

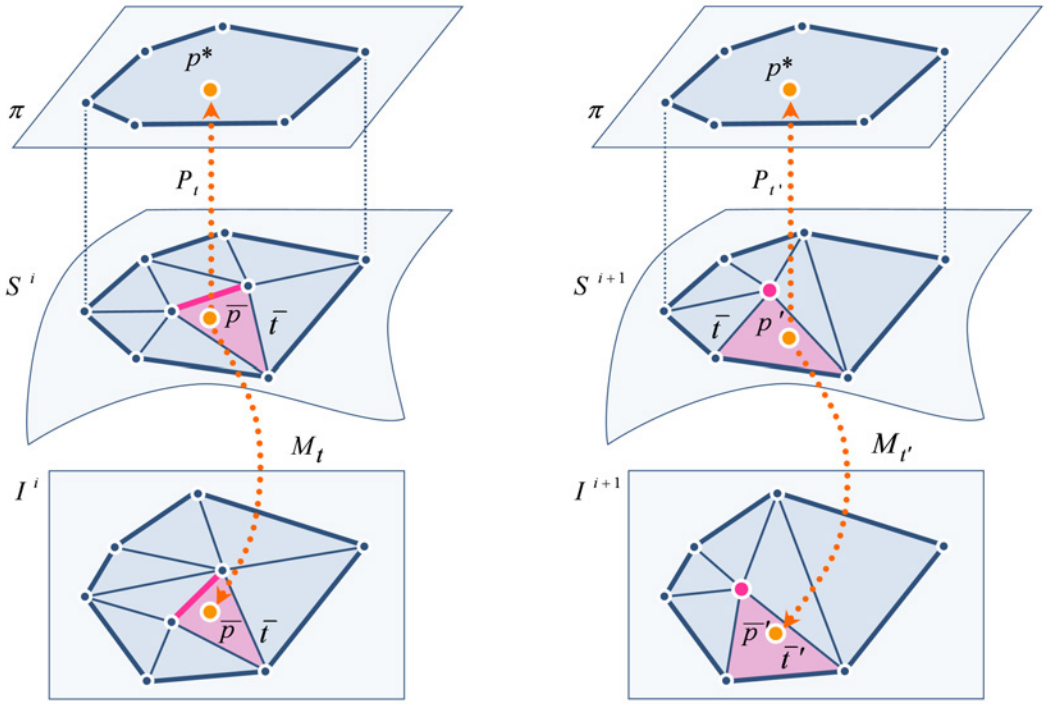

Figure 3.6: Mappings for filling the new texture triangles.

On plane π, we consider an orthogonal frame centred at v. For any triangle t or t' on the surface, we have two linear transformations, M_t and P_t. The transformation of M_t

maps t to its corresponding triangle $\bar{t}$ in the texture, while the transformation of P_t maps t to the orthogonal projection of t onto π. Let PM_t be the linear transformation $P_t \circ M_t^{-1}$. For each texel $\overline{p}'$ of a triangle $\bar{t}'$, we have the point p' on t' satisfying $M_{t'}(p') = \overline{p}'$ and the point p contained in a triangle t of S^i satisfying $P_t(p) = p^* = P_{t'}(p')$, where $p^* \in \pi$. Let $\overline{p}$ be the point in $\bar{t}$ determined by $\overline{p} = M_{t'}(p)$. Thus we have

$$PM_t(\overline{p}) = (P_t \circ M_t^{-1})(\overline{p}) = (P_{t'} \circ M_{t'}^{-1})(\overline{p}') = PM_{t'}(\overline{p}')\,.$$

Our approach consists of filling texel $\overline{p}'$ with the content of texel $\overline{p}$ without computing it explicitly.

Due to the bijective nature of the defined linear transformations for a fixed triangle $\bar{t}'$ we have

- If $\bar{t}_j \neq \bar{t}_l$, $(PM_{t'}^{-1} \circ PM_{t_j})(\bar{t}_j) \cap (PM_{t'}^{-1} \circ PM_{t_l})(\bar{t}_l) = \emptyset$.
- $\bigcup_j \bar{t}' \cap (PM_{t'}^{-1} \circ PM_{t_j})(\bar{t}_j) = \bar{t}'$.

Consequently, by using standard hardware capabilities, we can fill the texels in a triangle $\bar{t}'$ by mapping onto $\bar{t}'$ the triangles $(PM_{t'}^{-1} \circ PM_{t_j})(\bar{t}_j)$ without explicitly computing their common intersection. However, because we are dealing with a multi-chart atlas, the transformation that, for each I^i texel, maps the T texture coordinates stored in it, is not continuous. Thus, the standard hardware bilinear interpolation of T texture coordinates might cause invalid T texture coordinates. To solve this problem, we map the triangles $(PM_{t'}^{-1} \circ PM_{t_j})(\bar{t}_j)$ with a fragment shader. For each fragment within $\bar{t}'$ with I^i coordinates (u, v), we compute the four closest texels to (u, v), with a being the closest. Then, we compute the index of the fragment by a bilinear interpolation of the I^i indices stored at these four texels, taking into account only those the indices that lie in the chart of the index stored at a. Figure 3.7 shows the good performance in reducing blurring artefacts resulting from using an index texture.

3.7 Results

Our approach allows us to simplify large models while preserving their most important details without losing fidelity. We obtain single approximations as close as possible to the original models, which may be useful in various fields of application where the frame rate is more important than insignificant details. Several tests have been done on different models to study the accuracy of our method. All experiments were carried out on a quad core duo (2.83GHz) with a GeForce GTX 280.

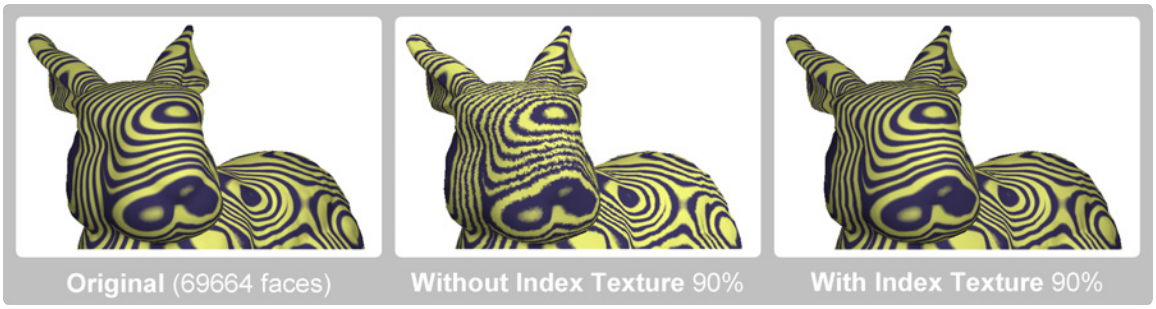

Figure 3.7: Blurring artefacts disappear when using an index texture during the simplification process.

Figure 3.8 compares our textured cow approximation result with the results obtained using the open-source tool MeshLab [CCR08, CCC+08], the commercial package Polygon Cruncher [Moo] and Hoppe's approach [Hop99]. The first two tools also allow us to preserve the boundary edges by simplifying chart interiors only. Activating this option generates a poor quality, low-resolution model full of seams is generated that cannot achieve the desired simplification level. To test how well our approach behaves, we have filled in the empty space of the atlas with red. Observe how our approach is the only one able to generate simplified models without texture distortions or artefacts near the chart boundaries by using only one additional texture during the simplification process.

Another comparison is presented in Figure 3.9. As you can see, the bunny model generated with [SSGH01] presents perceptible texture distortions because of the reparameterization, with convex and straight chart boundaries, that is carried out to be used on a progressive mesh. In contrast, with our approach we obtain an accurate approximation with a high-quality mesh and correctly preserved texture without distortions.

Figure 3.10 shows the Buffle model parameterized with a high number of small sized charts. A close-up view of the model shows that neither texture distortions nor artefacts appear in the simplified version, despite the presence of chart boundaries.

Parameterizations with a lot of charts reduce the level of simplification because the charts must all be preserved. Figure 3.11 shows a scanned Vase model parameterized with LSCM [LPRM02] and Iso-charts [ZSGS04] (see Section 2.2.3). Notice that the first one has a huge number of charts, whereas the second one only has 24 charts. In Figure 3.12, we can see the results of our simplification of these two parameterizations of the Vase model at the same level (90% simplification). Observe that parameterizations with less charts allow us to obtain better geometric quality and also to substantially increase the simplification level

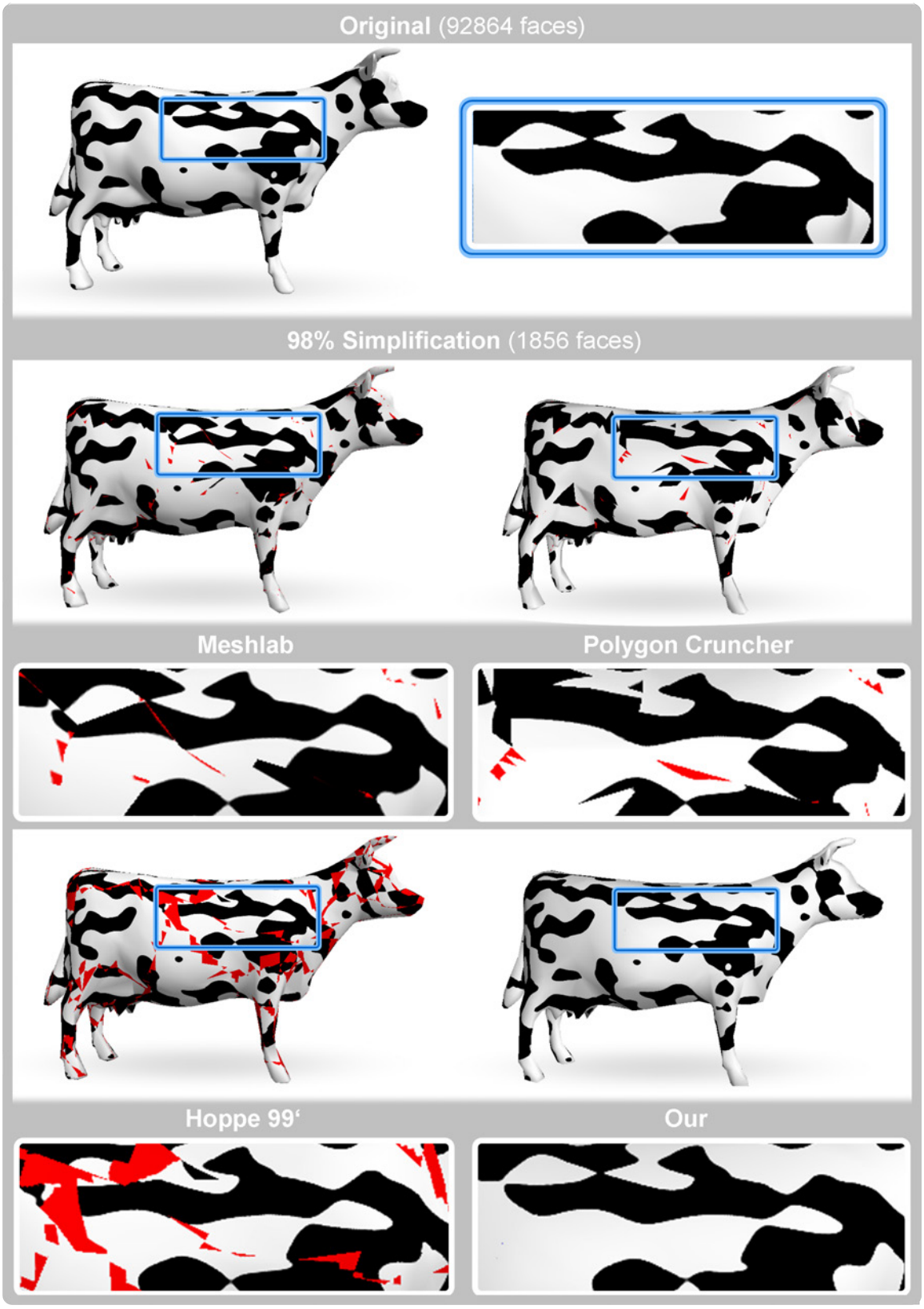

Figure 3.8: Simplification of a textured cow model. Comparison between our approach and three different texture preserving methods. The empty space of the atlas has been filled in with red.

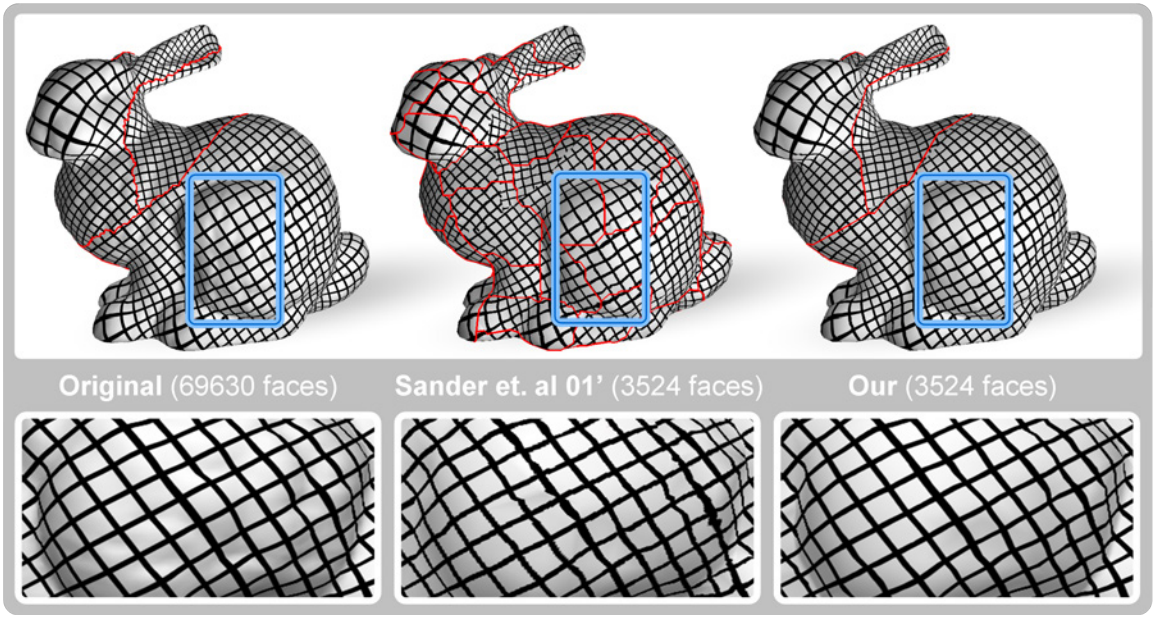

Figure 3.9: Simplification of a textured bunny model. Comparison between [SSGH01] and our approach.

(98% simplification). The flexibility that results from using parameterizations that admit non-convex charts is an important advantage with respect to [SSGH01].

Figure 3.13 shows a close-up view of the Vase model parameterized with Iso-charts. Chart boundaries have been properly simplified to give accurate approximations without seams between charts. Observe that, even though charts have interchanged part of their contents, their global shapes are preserved due to the low stretching of the parameterization used. For this reason, if we use a good parameterization method to generate the index texture, empty texture near concave parts of chart boundaries is not a strong requirement.

Figure 3.14 and Figure 3.15 are representative examples of large meshes obtained from real objects. For each of these, we would like to have a simple version that shares the same special characteristics. Not all the details of these models are represented in their texture maps, since small but important geometric features are lost at low resolutions with any simplification method. To capture all these details, their normal maps were generated before applying our method. Faithful and realistic approximations are obtained by applying bijective mappings on colour and normal maps even at a simplification level of 95% for the Imperia and of 99% for the Lion.

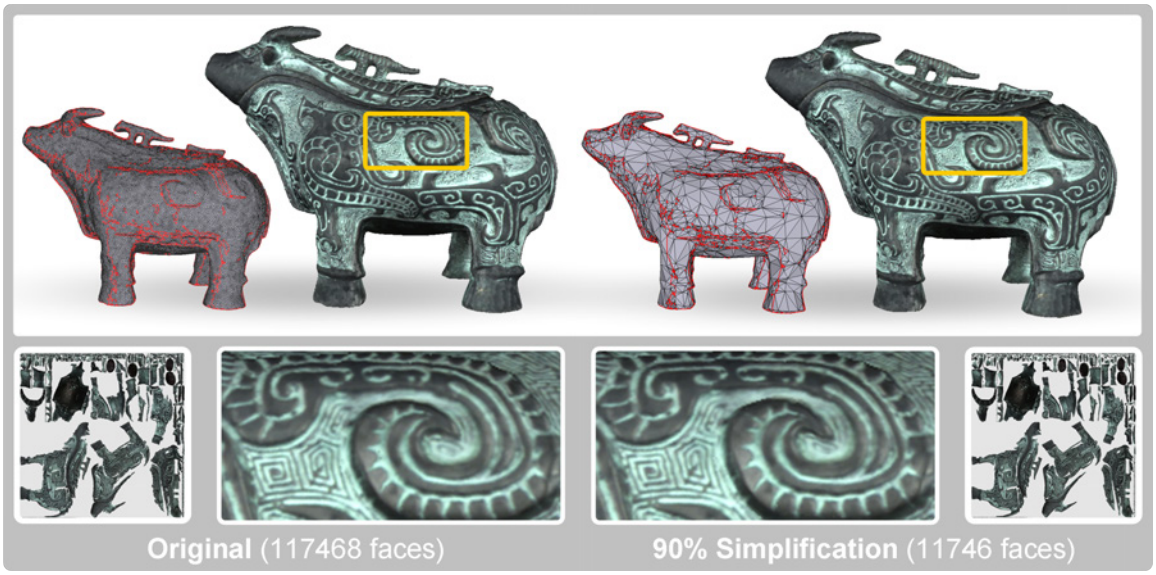

Figure 3.10: Buffle model simplification. For original (left) and simplified (right) models: at the top, the mesh with chart boundaries depicted in red and the textured model; at the bottom, the atlas and a close-up view.

Figure 3.11: Two different parameterizations for the Vase model.

Figure 3.12: Vase model simplification results according to the parameterizations showed in Figure 3.11.

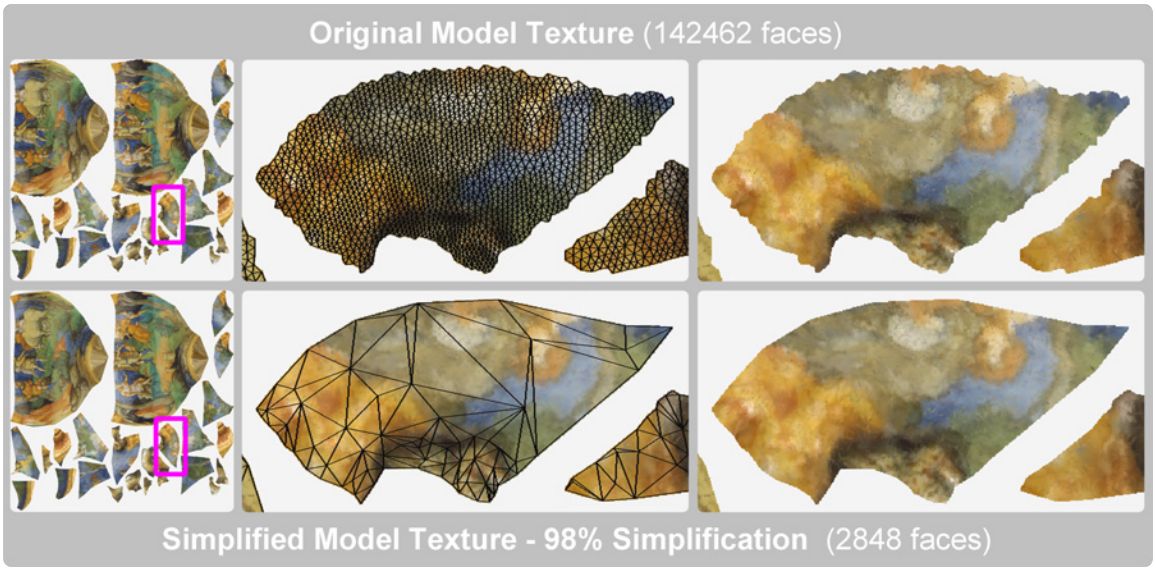

Figure 3.13: Close-up view of the Vase model texture before and after applying our bijective mapping approach.

			Without Flips			With Flips		
	Orig.	**Simplif.**	**CPU**	**GPU**	**Total**	**CPU**	**GPU**	**Total**
Bunny	69630	3524	9.03	10.47	19.50	101.27	30.93	132.20
Cow	92864	1856	12.61	19.79	32.40	129.65	33.26	162.91
Buffle	117468	11746	15.11	17.80	32.91	157.53	88.88	246.41
Vase LSCM	142462	14246	20.12	21.78	41.90	199.11	118.59	317.70
Vase Isochart	142462	14246	19.36	20.74	39.30	200.49	62.95	263.44
		2848	20.18	21.70	41.88	215.11	61.69	276.80
Imperia	199978	9997	26.76	41.30	68.06	278.20	76.67	354.87
Lion	309148	3090	45.09	48.18	93.27	462.67	114.74	577.41

Table 3.1: Time consumption of the simplification process with and without edge flips for some results presented in the chapter measured in seconds.

Although the proposed simplification method can be considered a part of a preprocess, in Table 3.1 we present a quantitative analysis of the time consumptions of some results presented in this chapter to give an idea about the complexity of the approach. The two first columns show the number of faces of original and simplified models. The following columns distinguish between CPU and GPU operations to show the time required for the simplification process with and without edge flips. Observe that the total time greatly increases when the edge flip operator is used. The elevated cost of the edge flip operator in the CPU is due to the high number of tests required to check all possible flips and their consistency. Notwithstanding, using the edge flip operator improves the results obtained and, taking into account that the presented approach is a preprocess, the time increment can be considered acceptable. The complexity of the parameterization affects the resulting time, as can be seen in the Vase results. The presented approach is almost independent of the texture resolution because only the faces affected for the edge collapse or the edge flip are sent to be drawn. Moreover, using the index texture allows all the desired textures (colour, normal, relief, ...) to be generated after the process without increasing the simplification cost.

Finally, to illustrate the good behavior of our approach Figure 3.16 shows a set of simplification results obtained from a collection of scanned multi-chart textured models. All models have been parameterized using the LSCM parameterization method.

Figure 3.14: Imperia model simplified by our approach with texture and normal maps. Observe that, in both global and close-up views, the visual appearance is preserved from the original one to the simplified one. Chart boundaries (in red) have been simplified without producing artefacts.

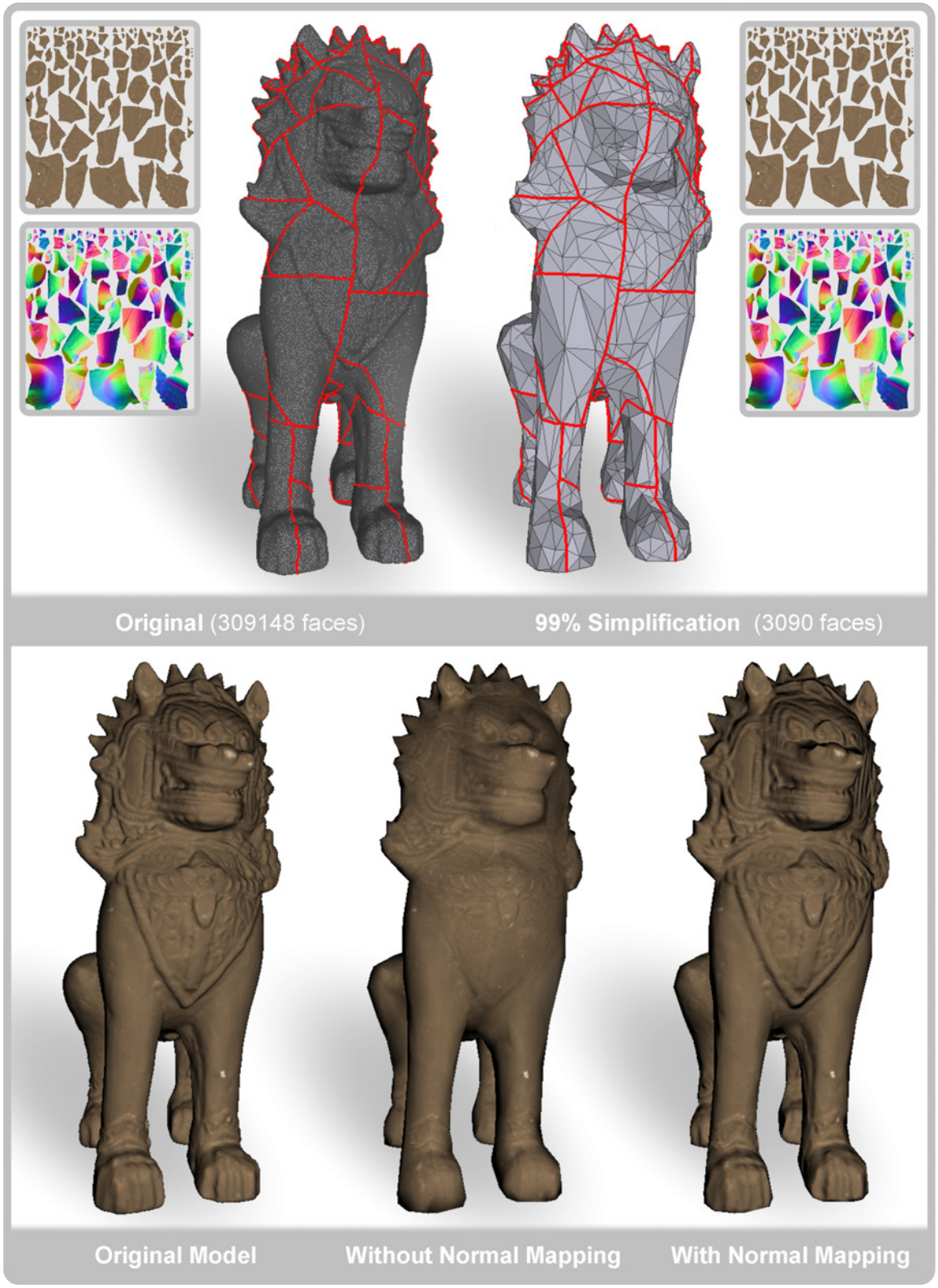

Figure 3.15: Lion model simplification results with and without normal mapping.

Figure 3.16: Simplification results obtained for a set of multi-chart textured models. From left to right, the two first columns correspond to the original models while the next two columns correspond to simplified models. The grey background columns illustrate the wireframes with chart boundaries depicted in red.

Chapter 4

Compact Models

Surface simplification are not always enough for all application fields. The increasing demand for realism together with the need of speed make that new surface approximation techniques begin to arise. In this chapter we present a new mesh structure, called *Compact Models* (CMs), that allows dense triangular meshes of arbitrary topology to be approximated by preserving their original shapes. The structure, based on local surfaces, is constructed from the information gathered by each vertex taking into account the sharp features detected during a simplification process. Thus, the input model can be approximated from its CM by blending the local surfaces at its vertices and refining each triangle of the simplified model. CMs make possible adaptive reconstructions, textured models are also supported and the whole approximation process can be completely parallelized. The versatility of the method combined with the simplicity of the computations makes it a powerful approach.

4.1 Introduction

Level of detail (LOD) is an extensively used technique for many computer graphics applications. The correct balance of memory space, transitions between models and quality of the approximations is key to the success of a method. As we stated as the key point of all content of Section 2.3, for a wide range of applications, original highly detailed surfaces are too expensive to support and their processing is made difficult. Unfortunately, simplified models are not always enough because of the increasing demand for realism.

In this work, we present a new mesh structure that allows dense triangular meshes of arbitrary topology to be approximated. We call such structures *Compact Models* (CMs).

A CM is a small model from which we are able to generate, whenever we want, an approximation of the original mesh at a desired level and preserve its original shape. The resulting mesh can be adaptively reconstructed by different criteria. Surfaces with sharp features are also faithfully reconstructed.

Surface fitting is achieved by blending local surfaces. These surfaces are stored at each vertex of the CM and are generated by gathering the required information during a simplification process. The combination of the idea of blending local primitives and LOD techniques, results in a simple and easy to implement method that produces adaptive approximations of the original surface close to it. Next, we summarize the main contributions of our approach:

- Approximation with controlled error.
- Sharp feature preservation.
- Least-squares fitting that involves solving only 5×5 linear systems of equations.
- Simple parallelizable reconstruction.
- Adaptive reconstruction based on a desired criterion.
- Local shape deformations can be integrated to the reconstruction process.

Observe that the last three contributions are the main advantages of our approach over those based on subdivision surfaces introduced in Section 2.3.4. A CM can be used in many different application fields such as visualization, adaptive reconstruction, surface fitting, surface smoothing and surface deformation.

4.1.1 Previous work

Interactivity provided by compactness of detailed geometric models is a desired goal in several application fields. Increasingly, we want to work with more detailed surfaces that consume much less time. A correct balance between the memory space and the execution time required has to be achieved to obtain the desired frame rate.

In light of this idea, different techniques have been proposed. In one group, the main aim is to find a simpler geometry and a set of scalar values which together are equivalent to the original model. Krishnamurthy and Levoy [KL96] presented a scheme for encoding an arbitrary mesh using a manually constructed B-spline patch network together with a

vector-valued displacement map. The *Displaced Subdivision Surfaces* introduced by Lee et al. [LMH00] consist of a control mesh and a scalar field that displaces the associated Loop subdivision surface [Loo87] locally along its normal. The control mesh is obtained by simplifying the original mesh using the quadric error metric (QEM) technique proposed in [GH97]. Guskov et al. [GVSS00] pursue a similar goal, called *Normal meshes*. A normal mesh is a multiresolution mesh where each level can be written as a normal offset from a coarser version. Their construction allows most of the vertices to be encoded by scalar displacements.

In a second group, the main goal is to fit the original model to a simpler surface by minimizing a geometric error. Ohtake et al. [OBA+03, OBA05] present shape representations (MPU and SLIM) that allow to approximate surface models from a set of points. These techniques, driven by hierarchical structures (octree and ball tree, respectively), locally fit piecewise implicit quadratic functions to the data and use weighting functions (partitions of unity) to blend these functions together. Because shapes are described by implicit functions, some shape modelling operations are simple to perform. However, an isosurface extraction process is needed to obtain a polygonal surface approximation, and they are not capable of representing correctly surfaces with boundaries. Moreover, correct reconstruction of the input data is not guaranteed. *Least-squares meshes (LS-meshes)* presented by Sorkine and Cohen-Or [SCO04] are meshes with a prescribed connectivity that approximate in a least-squares sense a set of strategically placed control points. An initial LS-mesh is computed and new control points are placed at the vertices whose location in the LS-mesh have maximal error compared to their location in the original mesh. Because of their prescribed connectivity, LS-meshes can only approximate a given smooth mesh. *Moving Least Squares (MLS)* is a classical method for point set surface approximations ideally designed to reconstruct smooth surfaces [Lev03, ABCO+03]. Fleishman et al. [FCOS05] use the MLS methodology for reconstructing surfaces with sharp features. The technique is based on an iterative refitting algorithm that locally classify regions of the point set to outlier-free smooth regions expressed by bivariate polynomials of degree two. The outliers are detected by applying a robust statistics framework. The result of this approach is a piecewise quadratic surface. *T-splines* defined on manifolds [HWW+06] or on polycube maps [WHL+08] are capable to approximate smooth surfaces. However, these approaches have a high computational cost due to the intrinsic complexity of T-splines. *Subdivision surfaces* have also been used in shape approximation. Several algorithms [MMTP04, MK05, CWQ+07, LWY08] for fitting subdivision surfaces to dense triangular meshes or dense point clouds have been presented. With an initial control mesh, these algorithms employ an iterative optimization method that performs the following two

steps until convergence occurs: finding foot points of the input vertices to compute fitting errors, and updating the control mesh points to further reduce the fitting errors. Marinov et al. [MK05] and Cheng et al. [CWQ+07] only deal with smooth fitting surfaces, while fitting subdivision surfaces with sharp features is considered by Ma et al. [MMTP04] and Ling et al. [LWY08]. Variational shape segmentation techniques aim to segment a shape into patches that can be well approximated by a parametric surface. Two iterative steps are used: mesh partition and fitting a surface, called proxy, to each partitioned region. Cohen-Steiner et al. [CSAD04] only use plane proxies, Wu and Kobbelt [WK05] use planes, spheres, cylinders and rolling ball patches as proxy types, while Yan et al. [YLW06] consider quadric surface proxies. However, these kinds of approaches are only adequate for models which inherently consist of clear geometric structures.

4.2 Overview of the Algorithm

The presented approach allows to generate at any time surface approximations at different levels of detail with the desired characteristics thanks to the flexibility of the local surfaces stored. A CM can be used for a wide range of applications as we will see in Section 4.6. To introduce CMs we have to explain how they are constructed, stored and applied. The following sections has been organized into the next three stages of processing (illustrated in Figure 4.1):

- **Simplification process:** A simplification process is applied to the original highly detailed surface. Original points are gathered and stored in simplified vertices during simplification. Finally, we obtain a simplified model in which each vertex represents a set of original points.

- **CM generation:** Taking into account the information collected during the simplification process, we first detect the sharp edges and then generate local surfaces at each vertex of the simplified model. We call CM the simplified model plus the necessary information to generate these local surfaces. A CM can be efficiently stored in a file.

- **CM reconstruction:** From a CM we can easily obtain a faithful reconstruction of the original model by blending local surfaces stored at each vertex. Thanks to the codification done in the CM file format, we can properly reconstruct sharp features.

Moreover, the original models can be adaptively reconstructed by different criteria. For example, it is possible to only reconstruct parts of a model in a region of interest.

4.3 Simplification Process

From a dense triangular mesh we use a simplification process to compact the surface model information. Since we want to obtain accurate results, we use our previously developed simplification method presented in Chapter 3 which properly preserves geometric fidelity between original and simplified models. We take advantage of the simplification process to collect the necessary information to generate the CM. Before the simplification process we create a set of points R_i for each vertex v_i of the original model. Each set R_i is initialized with v_i and its adjacent vertices. The simplification process proceeds as follows. If v_iv_j is an edge to be collapsed and v_k is the vertex of collapse, the set R_k is computed by $R_i \cup R_j$. At the end of the simplification process we obtain a simplified model with a set of points in each of its vertices (see Figure 4.1(b)). In case a vertex v_i of the simplified model is not an original vertex, vertex v_i is substituted by the point in R_i closest to v_i.

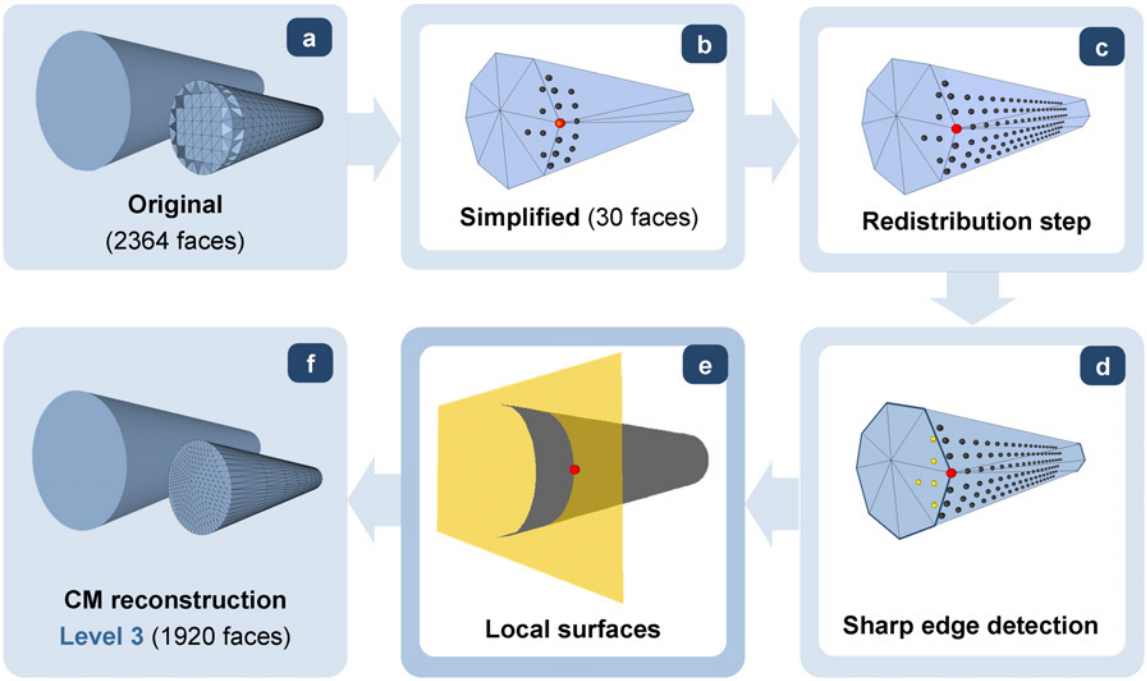

Figure 4.1: Steps of the CM generation and reconstruction of a cylinder model. Local surfaces incident to the red point are obtained by the corresponding yellow and black point regions after a redistribution step and a sharp edge detection.

The resulting mesh can be seen as a set of vertices that represent the regions defined by the corresponding set of points. Each set of points allows the shape of the surface at each simplified vertex position to be determined. As the simplified vertices are optimally distributed to guarantee geometric fidelity, the correct union of the shapes defined by the collected points at each vertex can result in an accurate approximation of the original model.

When a model has regions of nearly constant curvature, points of the original model can be incorrectly distributed to sets R_i due to the arbitrary order of the edge-collapses. To prevent this and really capture the shape of the original surface at each vertex, we apply a *redistribution step* (see Figure 4.1(c)).

This step consists of adding points from one point set to others. Given two adjacent vertices v_i and v_j, a point $p \in R_i$ is added to the set R_j if distance p to v_i is greater than distance p to v_j and the angle between normal vectors at p and v_j is less than a certain threshold. At the beginning of the whole process, the normal vector at each vertex of the original model is computed by the area-weighted average of the normal vectors of its adjacent faces.

4.4 CM Generation

The basic idea of CMs is to create a continuous representation of the shape of a highly detailed surface with a much smaller mesh. The generation process is key to the good behaviour of the result. We have to extract the most important information about the model to faithfully reconstruct the original model. The process consists of two steps:

1. Sharp edge detection.
2. Local surfaces generation at each vertex of the simplified model.

In the following sections, we describe these two steps in detail. But, first we need to introduce some terminology and some notation about the least-squares technique.

4.4.1 Terminology and notation

Given a point p in R_i, the face of the simplified model adjacent to v_i closest to p along the normal direction at p is called the *foot face* of p. The points of $v_iv_jv_k$ are those in

$R_i \cup R_j \cup R_k$ whose foot face is $v_i v_j v_k$. The *foot point* of a point p on $v_i v_j v_k$ is the point of $v_i v_j v_k$ nearest to p.

We use the least-squares technique to fit each set R_i by a local quadratic surface as follows. Let v be the vertex where we want to construct a local quadratic surface and R be the set of points collected by v. Let N be the unit normal vector assigned to v. From $N = (n_1, n_2, n_3)$ we construct two additional orthogonal unit vectors U and V as follows:

$$U = \begin{cases} (n_2, -n_1, 0)/\sqrt{n_1^2 + n_2^2} & n_1 \neq 0 \text{ or } n_2 \neq 0 \\ (1, 0, 0) & n_1 = 0 \text{ and } n_2 = 0 \end{cases}$$

$$V = N \times U.$$

Vertex v and vectors U, V and N define a local coordinate frame where we use (x_1, x_2, x_3) as local coordinates of a point p. At this local frame a quadratic function Q is given by

$$Q(x_1, x_2) = ax_1^2 + bx_1x_2 + cx_2^2 + dx_1 + ex_2 \,.$$

The unknown coefficients a, b, c, d and e are determined by minimizing

$$E(a, b, c, d, e) = \sum_{p \in R} \left(Q(x_1, x_2) - x_3\right)^2 \,.$$

Setting equal to zero the partial derivatives $\frac{\partial E}{\partial a}$, $\frac{\partial E}{\partial b}$, $\frac{\partial E}{\partial c}$, $\frac{\partial E}{\partial d}$ and $\frac{\partial E}{\partial e}$ leads up to the equation system:

$$\left(\sum_{p \in R} X^T \cdot X\right) C^T = \sum_{p \in R} x_3 X^T \,,$$

where

$$X = \begin{pmatrix} x_1^2 & x_1x_2 & x_2^2 & x_1 & x_2 \end{pmatrix} \,,$$

and

$$C = \begin{pmatrix} a & b & c & d & e \end{pmatrix} \,.$$

The solution of this linear system defined by the previous equations provides the local function Q which determines a local surface S at v defined by

$$S(x_1, x_2) = v + x_1U + x_2V + Q(x_1, x_2)N \,,$$

and the normal vector $NS = N - dU - eV$ of S at v.

Notice also that a local conical surface can be fitted by minimizing

$$\sum_{p \in R} \left(Q(x_1, x_2) - x_3^2\right)^2 \,,$$

and then the local surface S at v is defined by

$$S(x_1, x_2) = v + x_1 U + x_2 V + \sqrt{Q(x_1, x_2)} N \, .$$

4.4.2 Sharp edge detection

At this point we have only one set of points for each simplified vertex. If we imagine that each set of points can be represented by a surface, we can only generate smooth surfaces by assembling the surfaces at each vertex. For this reason, we have to detect sharp edges to be able to reconstruct different kinds of meshes, even surfaces with sharp features.

There are different approaches to detecting sharp features [HG01, GWM01, HPW05, YBS05, DHOS07, ZGM09] over the meshes. The most common sharp edge detection method consists of computing the angle between the normals of the two incident faces of the studied edge. If the obtained angle is larger than a certain threshold, the edge is marked as a sharp edge. This method is not useful in our approach because the simplified faces do not strictly follow the shape of the original model. For example, if we simplify a cylinder at a high level we can obtain a prism. If this method were used, the vertical edges would be incorrectly marked. We have to use a method that takes into account the original shape of the model. For this reason, we define a new sharp edge detection process. For each edge e of the simplified model we apply the following steps:

1. Fit by the least-squares technique the points of both incident faces to e with a quadratic surface centered at the foot point m of the midpoint of e.
2. Compute the normals NS_0 and NS_1 at m of the two quadratic surfaces.
3. Compute the angle between NS_0 and NS_1. If it is bigger than a threshold, the edge e is marked as a sharp edge.

In Figure 4.1(d) we can see depicted in blue the sharp edges of the cylinder.

4.4.3 Local surface generation

Given a vertex v of the simplified model and its corresponding set R of collected points, we want to fit a local surface to R. First, we analyze if at least one sharp edge is incident to v. In case of only one sharp edge (*dart vertex*), the incident edge with bigger angle between NS_0 and NS_1 is also classified as sharp edge. Then, the set R is split into subsets R^n

according to the incident sharp edges as follows. Two adjacent faces are in the same subset if their shared edge is not a sharp edge. Then, each subset R^n corresponds to the points whose foot face is within the same group of adjacent faces (see Figure 4.1(d)). After this, the sets R^n are fitted by the least squares technique with a quadratic surface S^n centered at v (see Figure 4.1(e)). When any sharp edge is incident to v, the set R is fitted by the quadratic surface or the conical surface that gives minimum fitting error. When the second case holds, the vertex is classified as *cone-type vertex*.

After generating the CM we obtain a mesh with a set of n lists of coefficients $[a, b, c, d, e]$ and the corresponding normal N at each vertex.

A CM can be easily saved by storing vertices, least-squares coefficient list, normal vectors, cone-type vertex list and by describing the faces by three sets (one per vertex) of three indices: vertex, coefficient list and normal vector. Sharp edges are implicitly saved by only using this data structure. From this reduced information we are able to reconstruct the original model.

4.5 CM Reconstruction

The original model is reconstructed by refining the simplified model. Each triangle of the simplified model is subdivided into smaller triangles and each vertex p of the refined triangulation is substituted by a point $\Phi(p)$ computed by blending the local surfaces at each vertex of the triangle. The following sections are dedicated to describing these two processes. Notice that each triangle can be refined independently of the others. Consequently, adaptive refinements are allowed and the whole approximation process can be completely parallelized.

4.5.1 Blending local surfaces

Let v be a vertex of the simplified model. As explained in the previous section, the adjacent faces of v are grouped according the sharp edges, and a local surface S^n has been assigned to each group. For each point p on an adjacent face of v we define a projection $\overline{S}(p)$ onto the local surfaces as follows. If p lies on a sharp edge e, we use the two local surfaces S^n and S^m corresponding to the two groups of faces adjacent to e. Let π the orthogonal plane to e passing through p. Then, point $\overline{S}(p)$ is defined as the point on $S^n \cap S^m \cap \pi$ closest to p. Observe that $\overline{S}(p)$ needs to be computed by a Newton iteration scheme. Notice also that it may not exist if a bad fitting of surfaces S^n and S^m to the original model is done. However, in practice this not happen due to the nature of the surfaces and the simplification levels

used. For a point p not lying on a sharp edge, we use the local surface S^n corresponding to the group of the p face. Two different projections onto S^n are possible. The simplest one is the projection along the normal direction N at v. If point p is expressed by its local coordinates (x_1, x_2, x_3), $\overline{S}(p)$ is determined by $S^n(x_1, x_2)$. The second possibility consists of determining $\overline{S}(p)$ by the point on S^n closest to p. In this case, we compute $\overline{S}(p)$ using a Newton iteration scheme. Naturally this second option increases the computational time but offers more accurate results. The two options were tested on various models and we experimentally observed that second option multiplied by a factor of three the running times. This is because Newton's method requires three iterations in average. In spite of this, all the examples shown in Section 4.6 were computed using the second option because running times were low enough (see Table 4.1).

Next, we describe how we compute $\Phi(p)$. Let $(\alpha_1, \alpha_2, \alpha_3)$ be the barycentric coordinates of point p with respect to its face $v_1v_2v_3$. Then we have $p = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$ with $\alpha_1 \in [0, 1]$, $\alpha_2 \in [0, 1 - \alpha_1]$ and $\alpha_3 = 1 - \alpha_1 - \alpha_2$. From coordinates α_1 an α_2 we consider the following weight functions:

$$W_1(p) = \frac{\alpha_1^3}{\alpha_1^3 + \alpha_2^3 + (1 - \alpha_1 - \alpha_2)^3}\,,$$

$$W_2(p) = \frac{\alpha_2^3}{\alpha_1^3 + \alpha_2^3 + (1 - \alpha_1 - \alpha_2)^3}\,,$$

$$W_3(p) = \frac{(1 - \alpha_1 - \alpha_2)^3}{\alpha_1^3 + \alpha_2^3 + (1 - \alpha_1 - \alpha_2)^3}\,.$$

Observe that:

- $W_i(v_i) = 1\,, \quad W_i(v_j) = 0 \quad (i \neq j)\,.$
- $\dfrac{\partial W_i}{\partial \alpha_k}(v_j) = 0, \quad \dfrac{\partial^2 W_i}{\partial \alpha_1 \partial \alpha_2}(v_j) = 0.$

Then, we determine the point $\Phi(p)$ by blending the three projections $\overline{S_1}(p)$, $\overline{S_2}(p)$ and $\overline{S_3}(p)$ as follows:

$$\Phi(p) = W_1(p)\overline{S_1}(p) + W_2(p)\overline{S_2}(p) + W_3(p)\overline{S_3}(p)\,.$$

Surface Φ satisfies the following properties:

- $\Phi(v_i) = v_i.$
- $\dfrac{\partial \Phi}{\partial \alpha_k}(v_i) = \dfrac{\partial \overline{S_i}}{\partial \alpha_k}(v_i), \quad \dfrac{\partial^2 \Phi}{\partial \alpha_1 \partial \alpha_2}(v_i) = \dfrac{\partial^2 \overline{S_i}}{\partial \alpha_1 \partial \alpha_2}(v_i).$

Then, surface Φ is approximately equal to the local surfaces at each vertex. This is why we use such kind of approximations. In Figure 4.1(f) we can see the final approximation of the cylinder.

4.5.2 Triangle subdivision

The triangles of the simplified model can be subdivided using two different strategies:

Regular subdivision. Each triangle is recursively subdivided into four triangles by joining the midpoints of the edges until a certain level of accuracy is achieved.

Adaptive subdivision. Each triangle is recursively subdivided into two, three or four triangles by joining the midpoints of the edges that satisfy some criterion. Several criteria can be applied. We have used the following criteria for a given edge p_1p_2:

- *Silhouette.* At least one of the points $\Phi(p_1)$ or $\Phi(p_2)$ is detected to be on the silhouette of the model with respect to a given point of view.
- *Region of interest.* At least one of the points $\Phi(p_1)$ or $\Phi(p_2)$ is within a region of interest.
- *Length.* The length of $\Phi(p_1)\Phi(p_2)$ is greater than a certain threshold defined by the user.

In the following, the number of subdivision steps is called level.

4.5.3 Controlling the error

The reconstruction process provides a natural way to control the error between the original mesh and the reconstructed mesh. Finding the foot points of the reconstructed mesh vertices and computing their fitting errors, we can update the CM by adding in a straightforward way the foot points whose fitting error is greater than a certain bound introduced by the user.

4.5.4 Textured models management

With the aim to obtain more realistic approximations CM also supports textured model. We take advantage of the accurate results obtained by our simplification approach over

multi-chart textured models (Section 3) in the CM generation process. For each of the vertices of the CM we also store its texture coordinates which are directly obtained from the simplification method. In the CM reconstruction process, the texture coordinates assigned to each of the reconstructed vertices are computed by performing a linear combination taking into account their corresponding barycentric coordinates.

4.6 Results and Applications

The presented technique can be seen as a surface approximation method that obtains faithful reconstructions. Our approach provides the versatility required to adapt the result according to user needs. It can be achieved by changing the simplification level, the sharp edge threshold or by making more or less subdivisions in the reconstruction process. Moreover, we can reconstruct the model regularly or adaptively by a desired criterion. We present some results to demonstrate the effectiveness of our technique. A quantitative and a qualitative analysis have been made to show all the characteristics of the method. All experiments were carried out on a quad core duo (2.83GHz) with a GeForce GTX 280.

4.6.1 Smooth surfaces

Figure 4.2 shows a reconstruction of a knot model. We generate a well smoothed model by using a very simplified mesh. We compare our result with that obtained by a smoothing method ([DLG90]). The error distribution maps (notice that the range of colours depends on the corresponding maximum error of each model) shows how our technique obtains better approximations than only applying a smoothing method over the simplified model.

Figure 4.3 and Figure 4.4 are representative examples of big meshes obtained from scanning processes. As can be appreciated in close views, the original shapes are faithfully reconstructed despite the substantial storage space reduction.

4.6.2 Sharp feature preservation

Figure 4.5 and Figure 4.6 present the reconstruction of models with the presence of sharp features. The difference between original models and their reconstructions is inappreciable even though the simplified mesh have a small number of faces. The error map distribution illustrated for the Fandisk model shows the good performance of our method when dealing with sharp feature models. Figure 4.7 presents an example with both sharp edges and

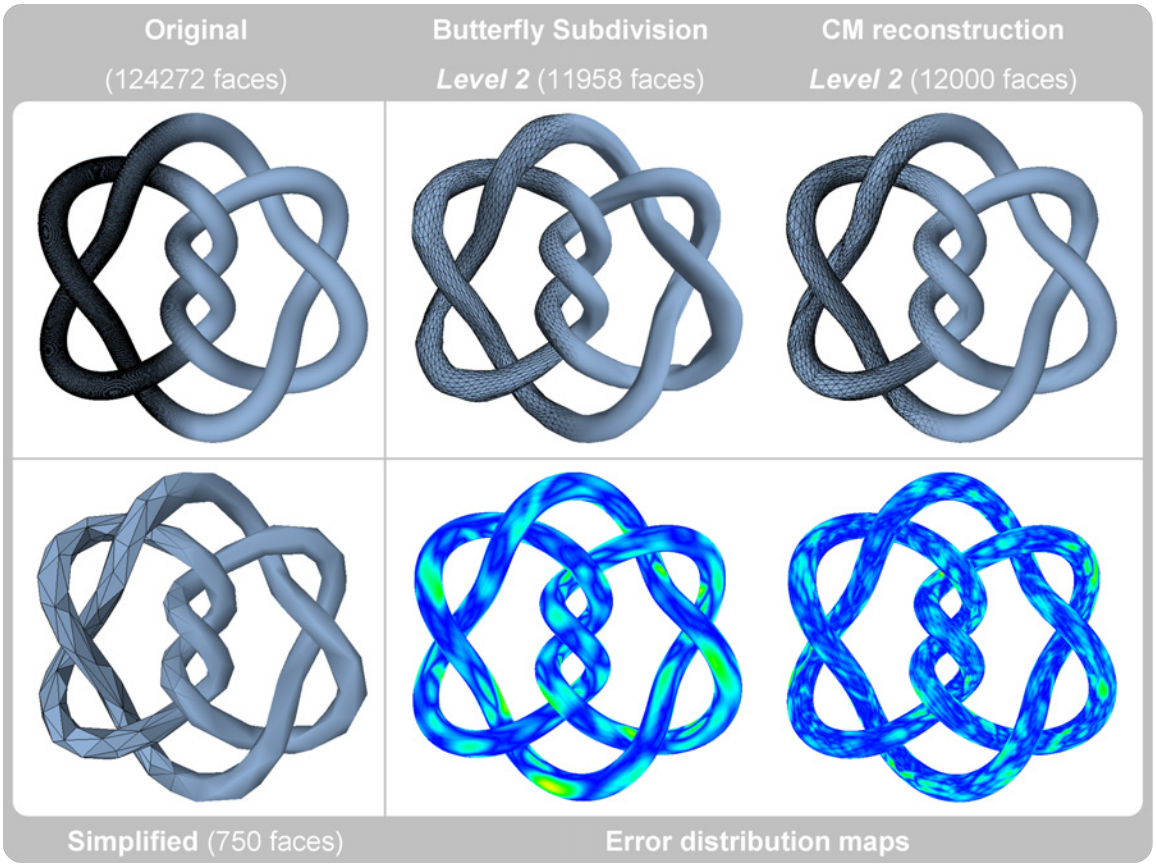

Figure 4.2: Comparison between our approach and the Butterfly subdivision method of the reconstruction of a smooth knot model.

cone-type vertices. Another example is showed in Figure 4.8 formed by sharp edges and two dart vertices. As we can see, we are able to correctly detect, preserve and reconstruct all the features of the models.

4.6.3 Adaptive reconstruction

Figure 4.9 shows the reconstruction results on the Max Planck model. Regular reconstruction demonstrates the power of the local surfaces stored at each vertex since the facial features have been recovered. Moreover, we illustrate the results obtained with the three different adaptive reconstruction criteria implemented. The first generates a model with regular size edges that are 20% of the longest edge of the simplified model. The second subdivide takes into account a point of interest situated in the tip of the nose (point depicted in red). As we move away from the selected point the faces become bigger. Depending

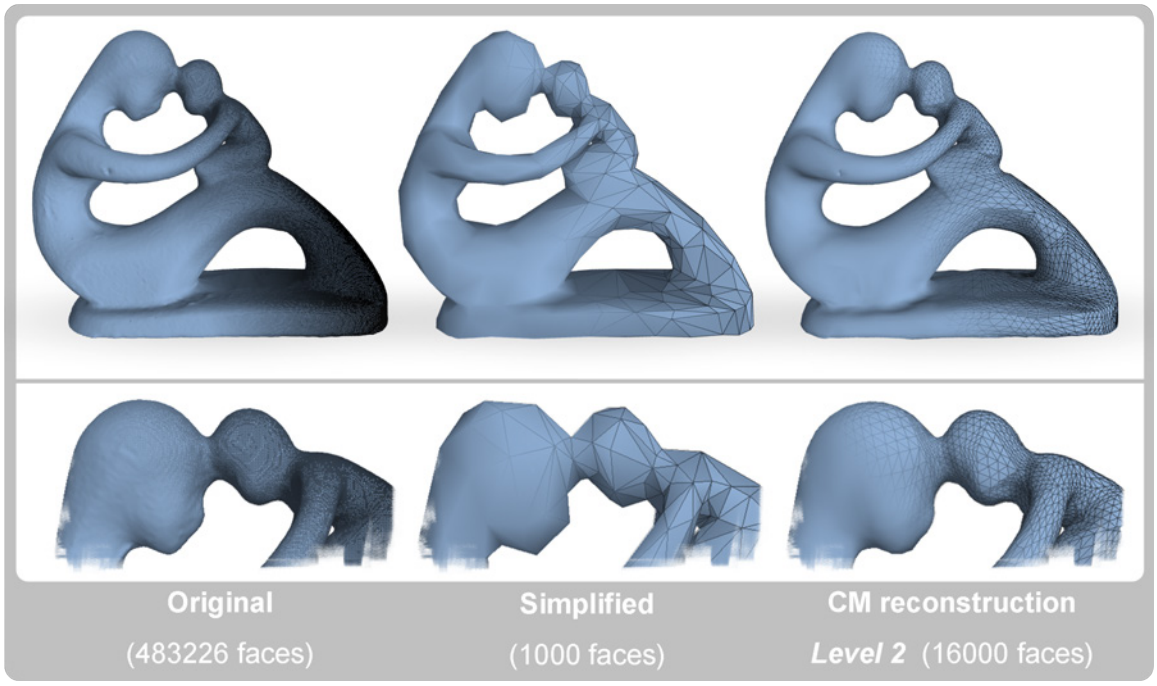

Figure 4.3: From left to right: the original Fertility model, the simplified model and the CM reconstruction. The CM generation takes 163.55 seconds and only 0.28 seconds to reconstruct it at level 2. The size of the CM file is only 0.3% of the original model.

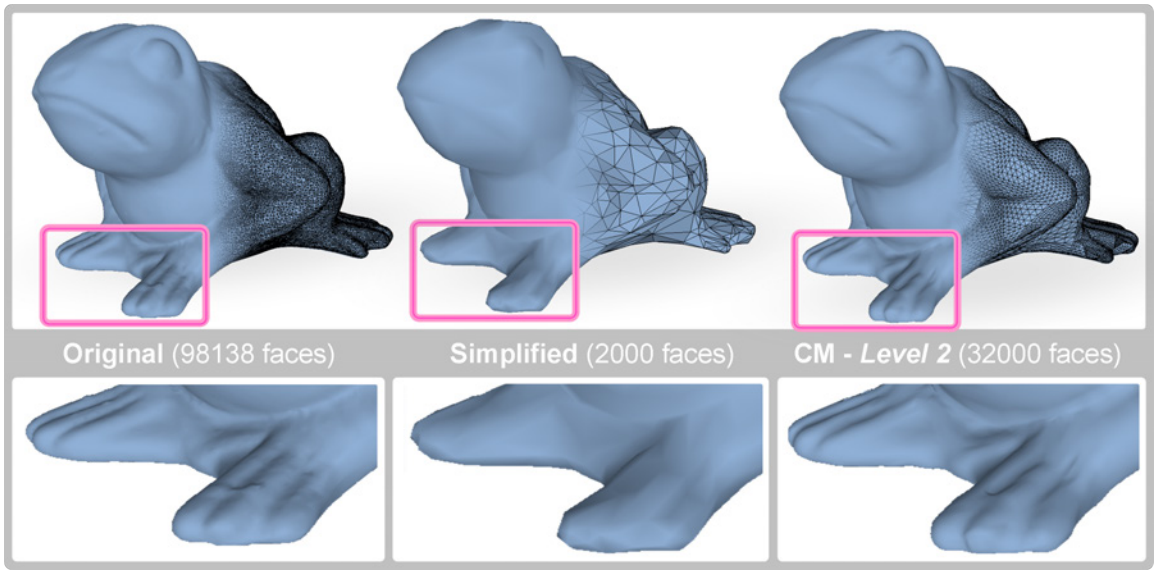

Figure 4.4: CM reconstruction of a frog model.

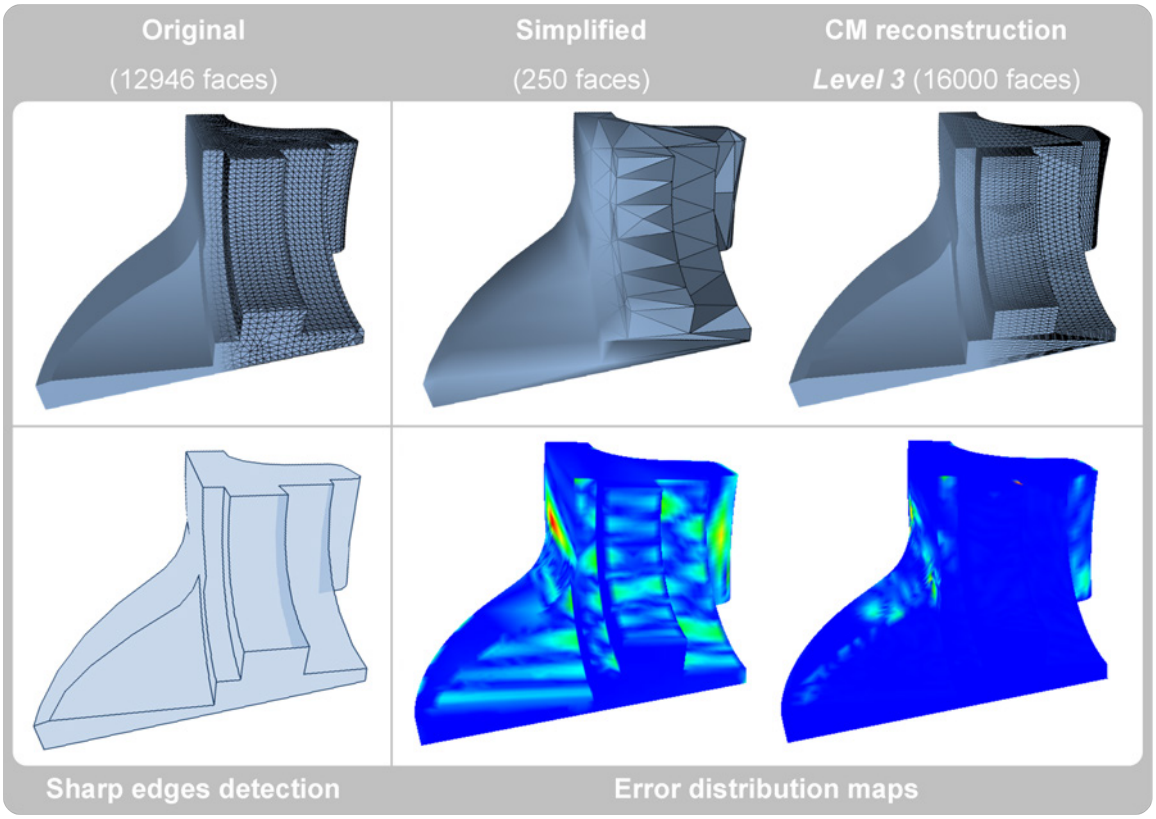

Figure 4.5: CM reconstruction of the Fandisk model with the corresponding error maps.

on the point of view, the last one allows the silhouette to be refined. As we can see, the versatility of our technique allows users to adapt the result to their needs.

4.6.4 Textured models

Textured models are also allowed to be reconstructed by a CM. The more similar the original model is to the simplified model the better applied is the texture on the reconstructed approximation. This ability allows to enrich the simplification approach developed previously obtaining thus more realistic results. The detail information can be added adaptively to the textured simplified model depending on our needs.

Figure 4.10 shows the approximation of the lion model illustrated in the previous chapter (Figure 3.15). As can be seen, the adaptive approximation obtained from a CM by adding more geometric detail on the silhouette allows to improve the fidelity to the original model with respect to the simplified version with only one level. The two lion poses shows how the silhouette are refined depending on the point of view.

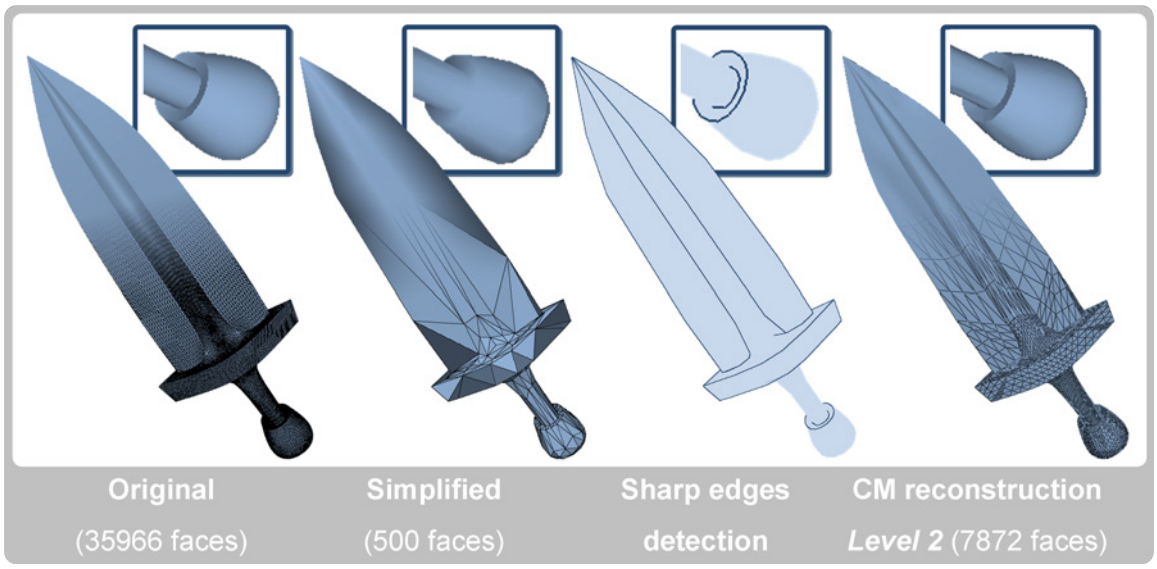

Figure 4.6: CM reconstruction of a sword model with sharp edge preservation.

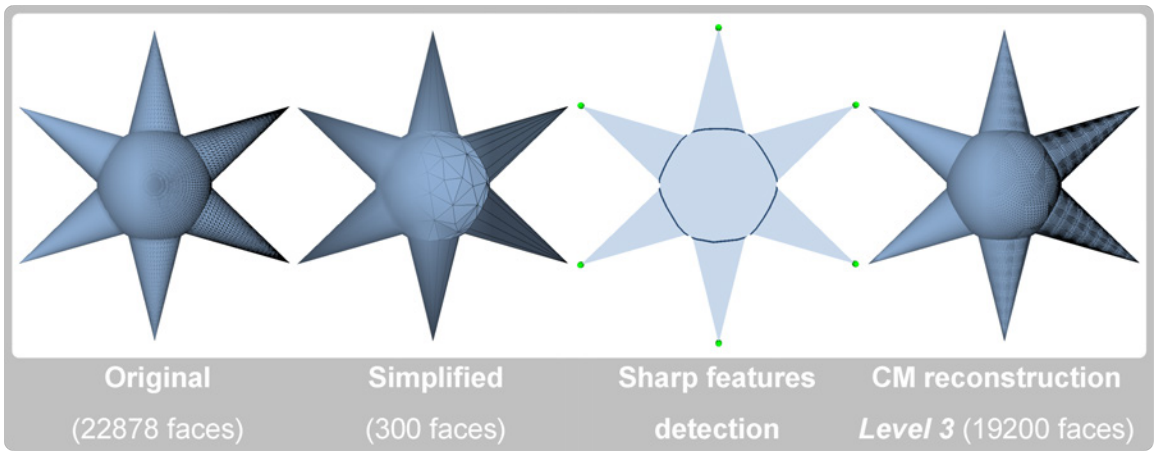

Figure 4.7: CM reconstruction of a star model with sharp edge and cone-type vertex (depicted in green) preservation.

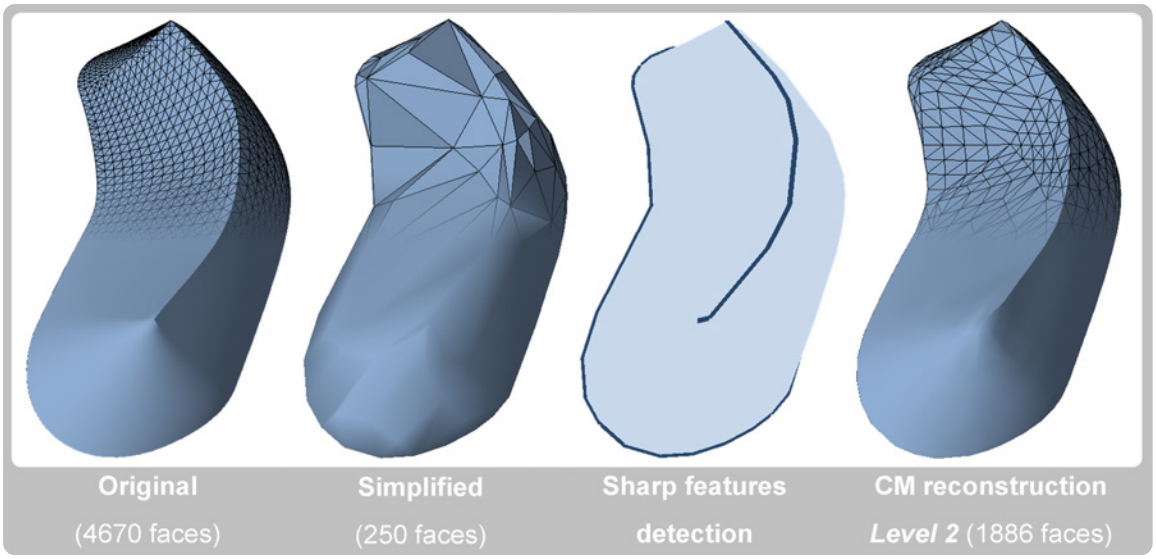

Figure 4.8: CM reconstruction of a model with sharp edge and dart vertex preservation.

In Figure 4.11 a buddha model is reconstructed regularly and adaptively according to the silhouette. Moreover, we show the results obtained for two simplification levels. The rounded shape of the model is achieved and the texture is correctly applied although the simplified version is too poor. Therefore, the approximations produced are almost indistinguishable to the original model. The adaptive solution allows to obtain similar results with fewer triangles but depending on the point of view.

To show the accuracy of the reconstructions obtained for textured models we present Figure 4.12. The correct preservation of the lines defined on the texture allows to illustrate the high precision of the method. In the figure we also show the reconstruction obtained adaptively with regular size edges. The number of faces is too much lower than in regular reconstruction.

4.6.5 Strategies for LOD

The general idea of the LOD technique is to visualize models at different levels of detail according to the distance to the observer. A CM can then be incorporated into a LOD technique using one of two different strategies. Figure 4.13 illustrates the first strategy: the model is simplified at different levels and each simplified model is reconstructed at the same level. Observe that as we reduce the number of faces of the simplified model, the geometric error increases. Figure 4.14 illustrates the second strategy: the model is

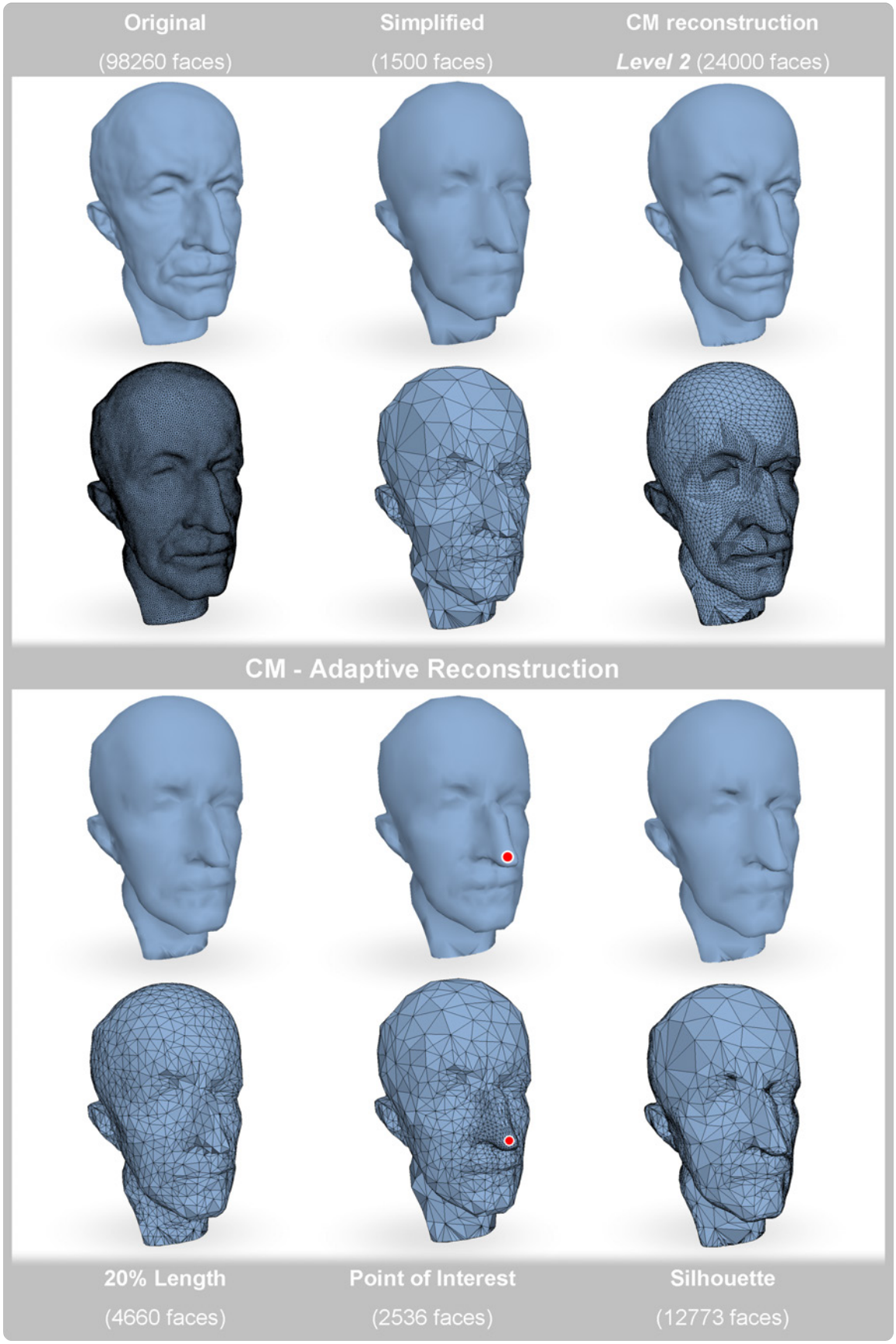

Figure 4.9: Regular and adaptive reconstructions of the Max Planck model.

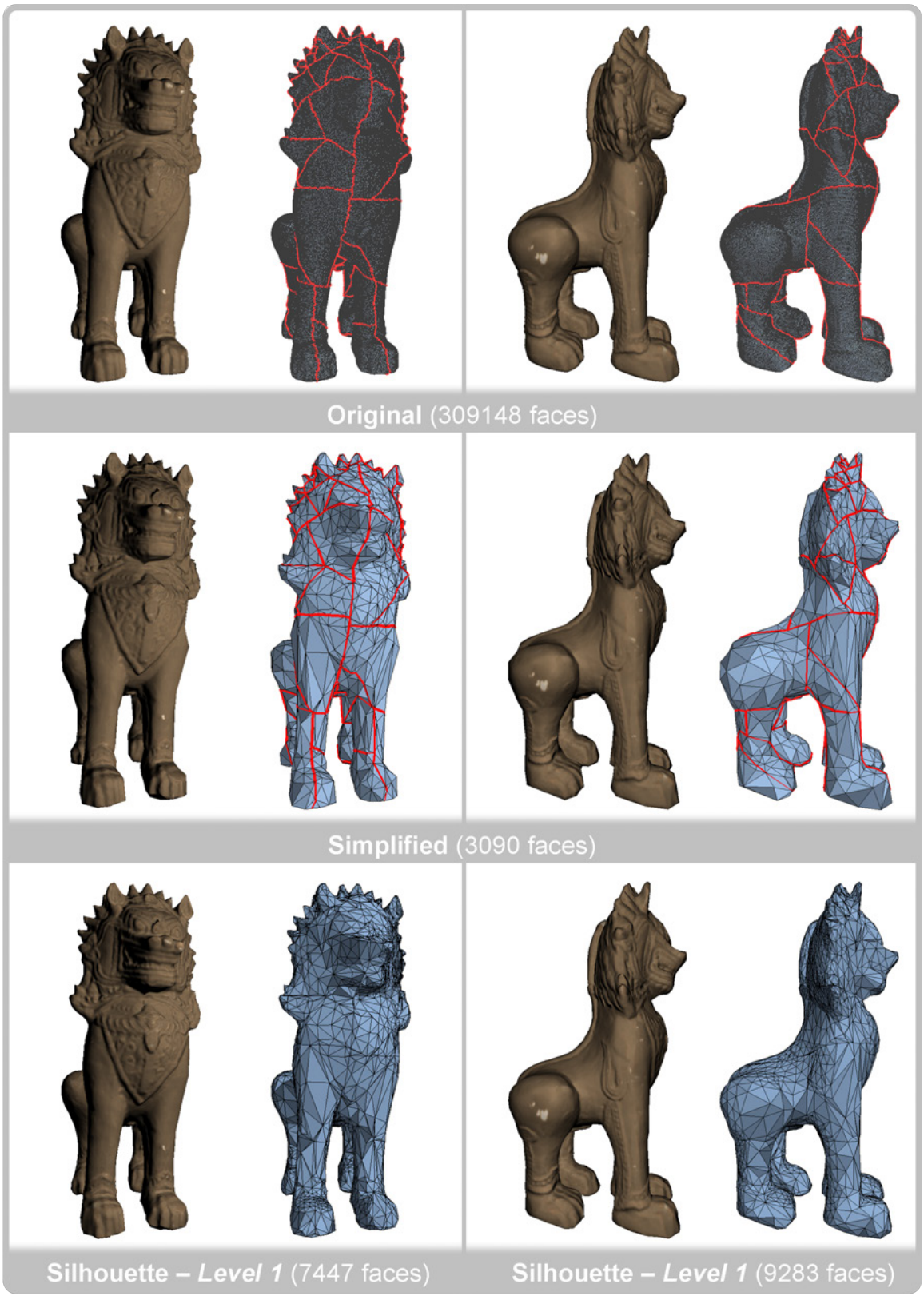

Figure 4.10: Approximations of the lion model from two points of view: regular and adaptive by the silhouette.

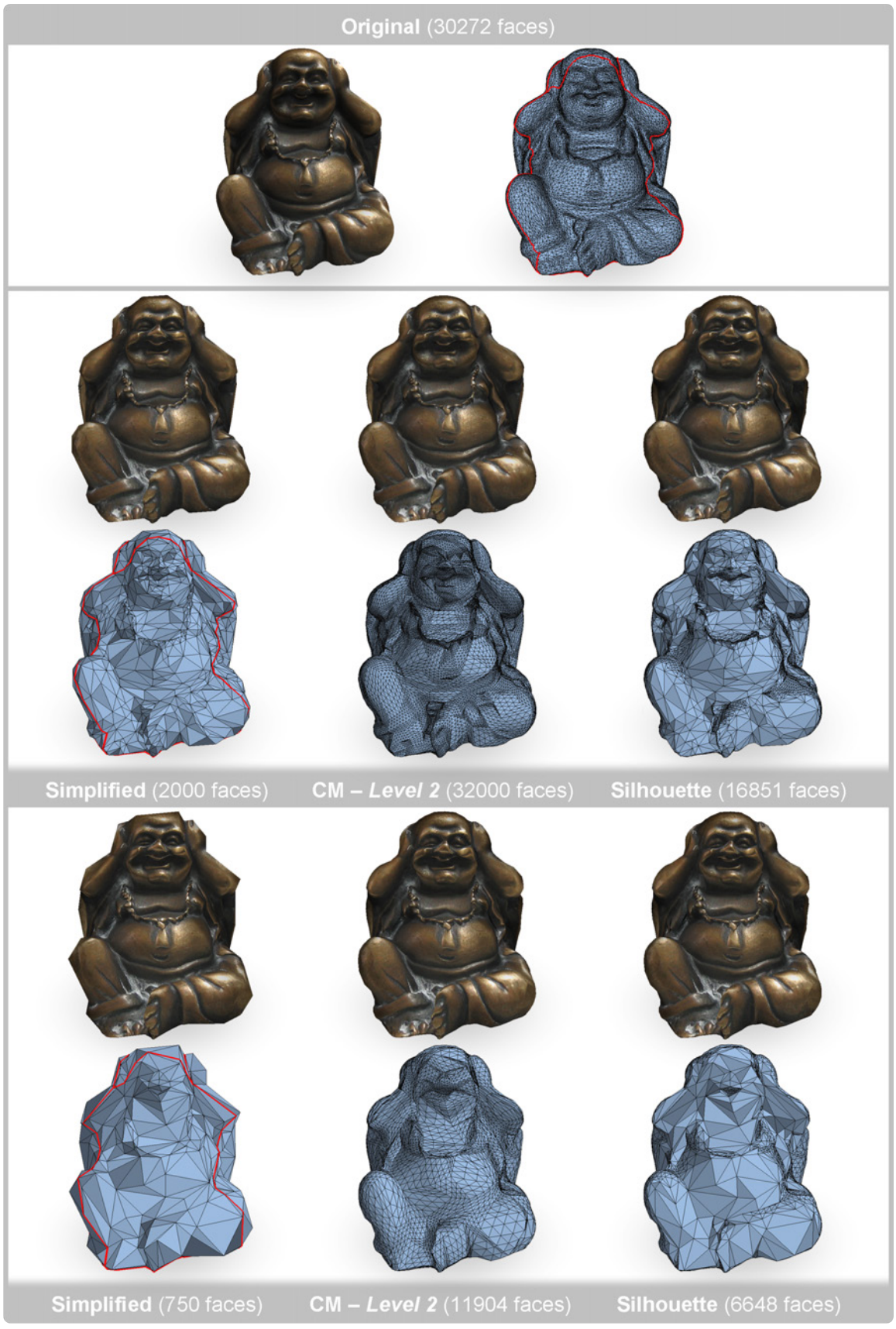

Figure 4.11: Regular and adaptive reconstructions of a buddha model for two simplification levels as a CM base.

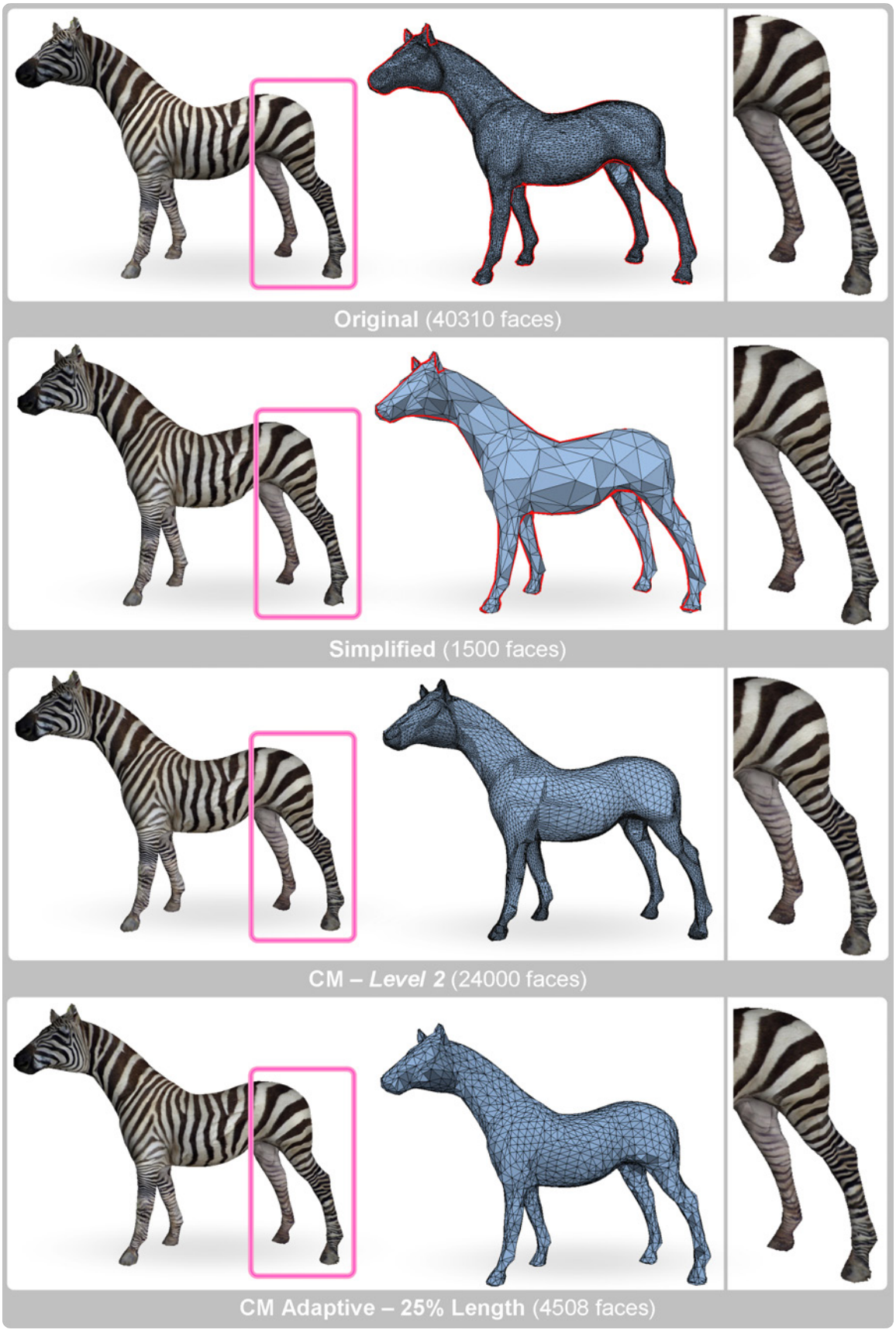

Figure 4.12: Zebra results for two reconstructions: regular and adaptive by a regular edge size (25% of the longest edge).

simplified at a desired level and reconstructed at different levels. Observe that as we increase the reconstruction level, the geometric error decreases.

4.6.6 Quantitative results

Table 4.1 shows the memory space and the computation time related to most of the examples used. In first column we compare the size of the original file with the size of the CM file. As we can see the reduction is considerable using our method. We can obtain really good approximations with much less space. Observe that for horse and Max Planck models the size of CM are equal in all variants due to the CM used is the same. Figure 4.15 illustrates some comparison results graphically. In second column we present the computation time for the two principal steps, CM generation and CM reconstruction (without parallelization). The cost of gathering points during the simplification can be underestimated due to in all our experiments it only represents an increment of 3%. As we can see, the CM generation time depends on the size of the original model, the simplification level and the presence of sharp edges. The CM reconstruction time depends on the size of the simplified model, the subdivision level and the presence of sharp edges. For example, a simplified model of 1000 faces can be reconstructed at level 2 in less than 0.3 s. Figure 4.16 shows graphically the results obtained.

Finally, we would like to make some comparisons with related approaches based on subdivision surfaces (see Section 2.3.4). Figure 4.17 and Figure 4.18 compare the results obtained by [CWQ$^+$07] and our approach for Igea and Ball Joint models, respectively. As we can see, our reconstruction obtains more accurate approximations of the original models. Due to the nature of the [CWQ$^+$07] method, their results are too smoothed and do not fit the real feature details of the models even though they are reconstructed at the limit. Table 4.2 shows the geometric errors obtained for these two models with the two methods. It supports the good behaviour of our method and shows how we obtain smaller RMS and Max errors for the two models. For models with sharp features we compare our results with [LWY08] for the Fandisk model. In Table 4.3 we can see how substantially smaller error values are obtained even though the simplified mesh has fewer vertices.

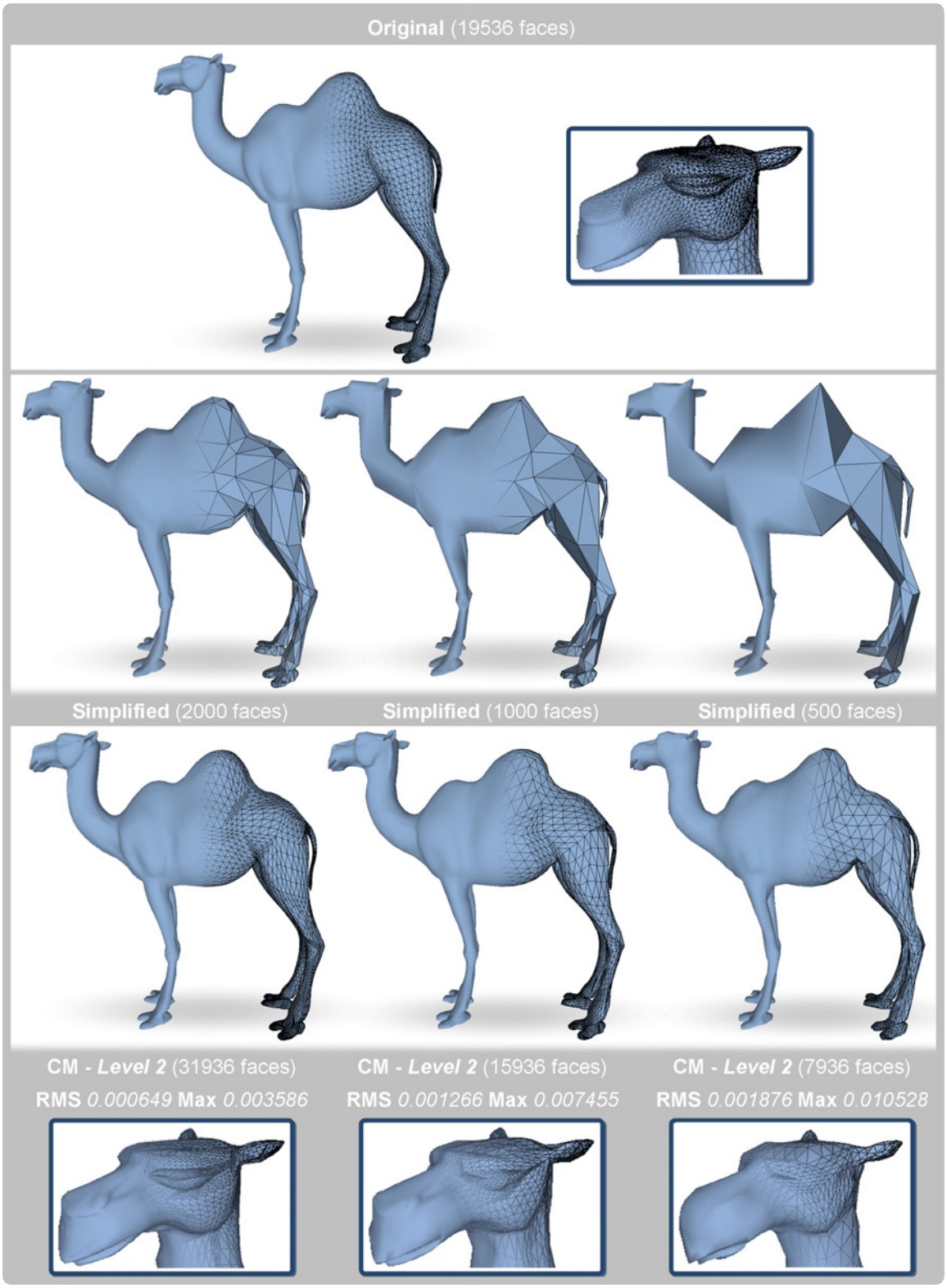

Figure 4.13: CM reconstruction of a camel model at different levels of simplification specifying the corresponding RMS and Max geometric errors.

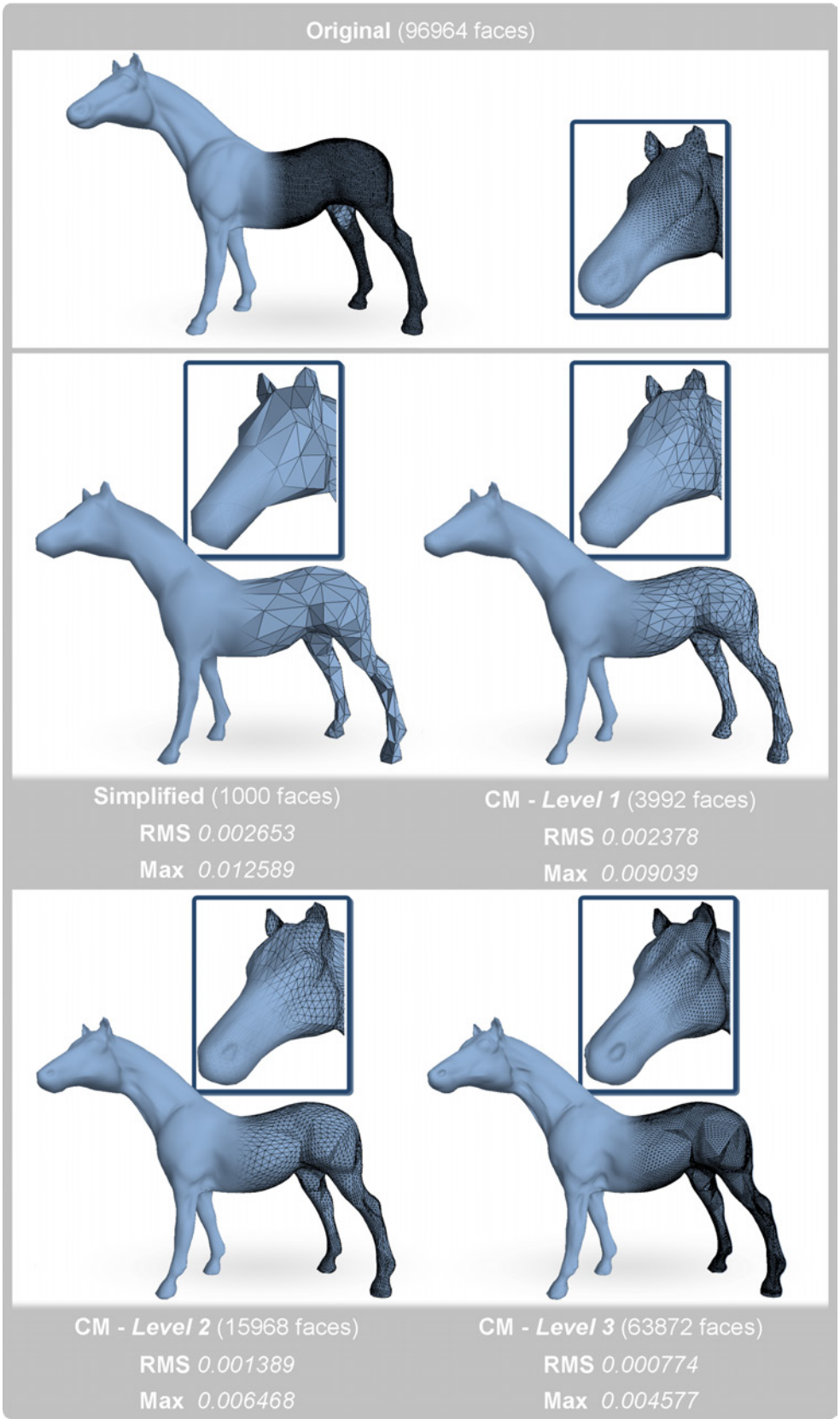

Figure 4.14: CM reconstruction of a horse model at different levels of subdivision specifying the RMS and Max geometric errors in each case.

	Memory Space (MB)		**Computation Time** (sec)	
	Original	CM	Generate	Reconstruct
Fandisk	0.64211	0.02656	1.29	0.42
Camel				
2000 faces	0.96891	0.13751	1.37	0.49
1000 faces	0.96891	0.10890	1.37	0.25
500 faces	0.96891	0.05455	1.37	0.14
Sword	1.78455	0.04766	3.82	0.17
Horse				
Level 1	4.80865	0.08220	10.21	0.06
Level 2	4.80865	0.08220	10.21	0.24
Level 3	4.80865	0.08220	10.21	1.03
Frog	4.86687	0.16422	7.70	0.46
Max Planck				
Level 2	4.87292	0.12321	9.32	0.35
20% Length	4.87292	0.12321	9.32	0.22
Point of interest	4.87292	0.12321	9.32	0.14
Silhouette	4.87292	0.12321	9.32	0.17
Knot	6.16278	0.06151	13.54	0.21

Table 4.1: Memory space occupied by the original model and the CM files in MB and computation time of the CM generation and CM reconstruction steps in seconds.

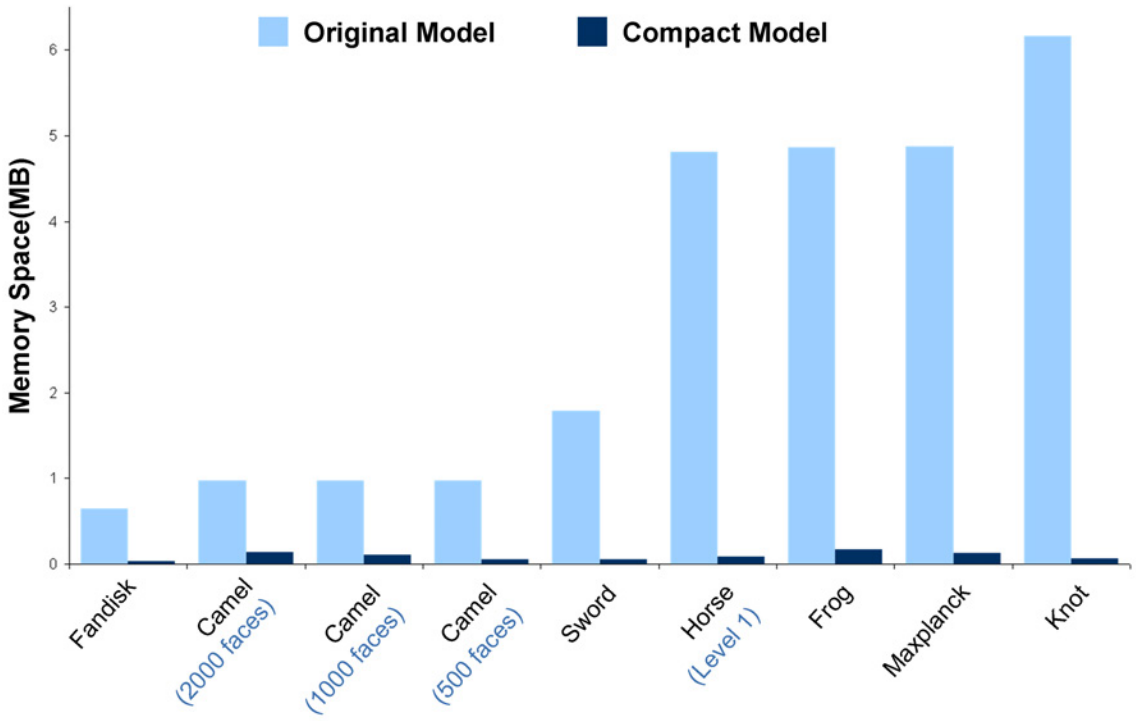

Figure 4.15: Graphic illustration of some memory space results showed in Table 4.1.

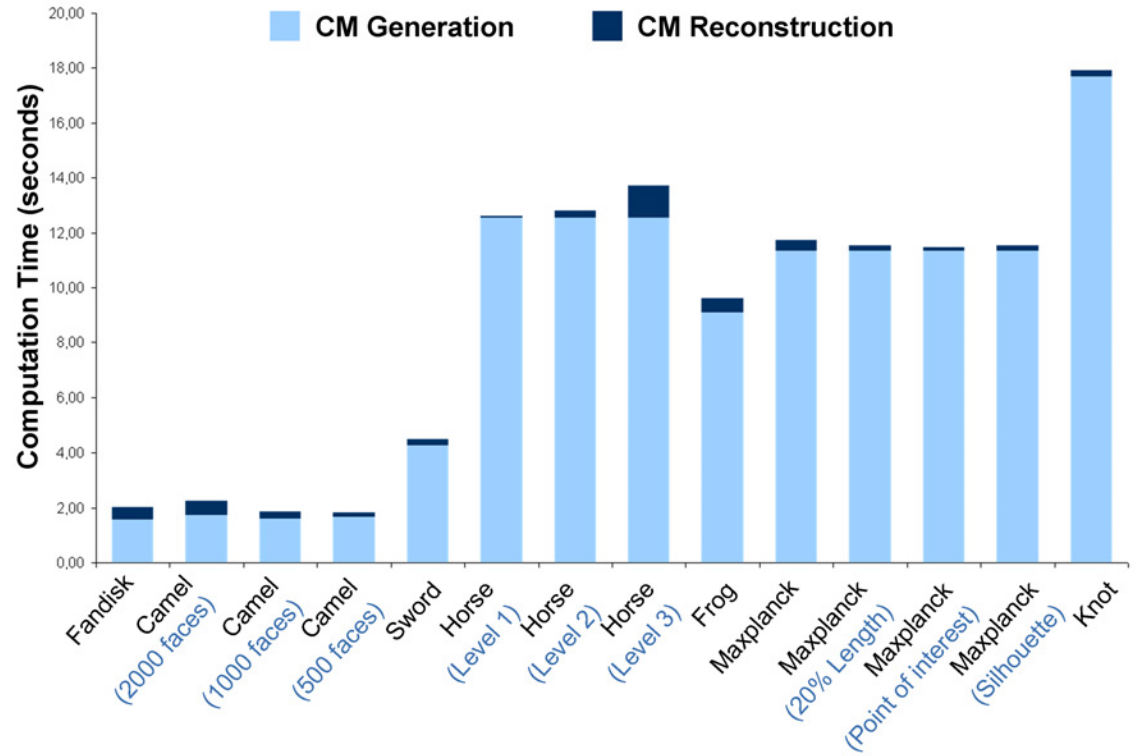

Figure 4.16: Graphic illustration of the computational time results showed in Table 4.1.

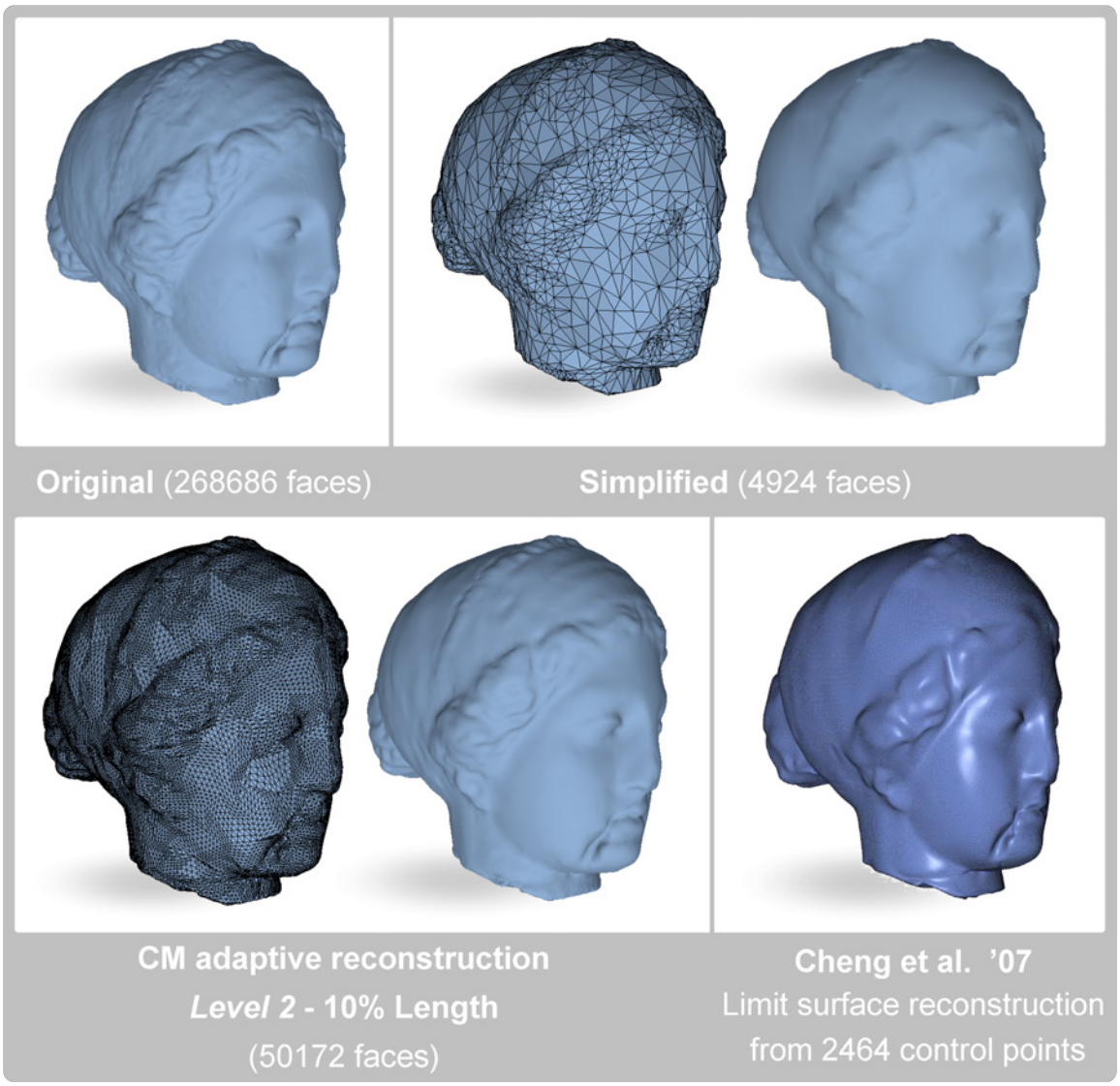

Figure 4.17: Comparison of the approximation result of the Igea model between the approach of Cheng et al. and our method. The reconstructions are build from the same number of faces. The image of the result of Cheng et al. 07' has been borrowed from [CWQ+07].

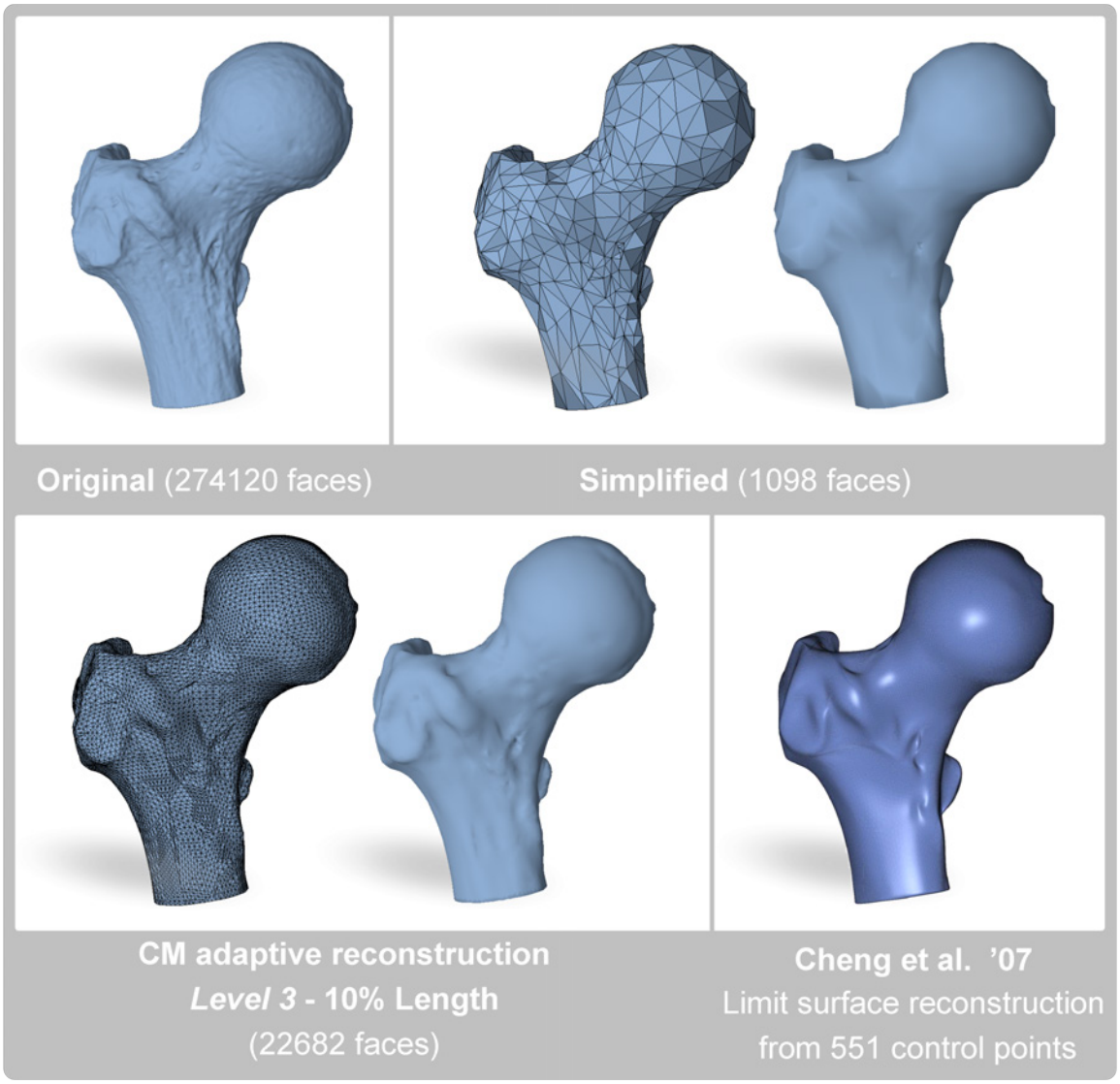

Figure 4.18: Comparison of the approximation result of the Ball Joint model between the approach of Cheng et al. and our method. The reconstructions are build from the same number of faces. The image of the result of Cheng et al. 07' has been borrowed from [CWQ+07].

	Igea		**Ball Joint**	
	RMS	Max	RMS	Max
Cheng et al. '07	0.0005	0.0036	0.0009	0.0064
CM reconstruction	0.00035	0.00221	0.00071	0.00424

Table 4.2: Geometric error comparison in % with respect to the BBox diagonal for the Igea and Ball Joint models showed in Figure 4.17 and Figure 4.18, respectively).

	Fandisk		
	#Vertices Simplified	RMS	Max
Ling et al. '08	346	0.000581	0.0135
CM reconstruction	127	0.000271	0.00208

Table 4.3: Geometric error comparison in % with respect to the BBox diagonal for the Fandisk model.

Chapter 5

∗Cages for Mesh Deformation

Geometric modelling and editing tools play an important role in geometric processing. Mesh deformation techniques, which provide a convenient way to edit a model to meet various design requirements (see Section 2.4), are especially important. The main advantages of cage-based deformation techniques are their simplicity, relative flexibility and speed. However, up to now there has been no widely accepted solution that provides both user control at different levels of detail and high quality deformations. In this chapter, we present ∗*Cages*, a mesh deformation system which allows multiple cages enclosing the model to be used for easier manipulation while preserving the smoothness of the mesh in the transitions between them. The proposed deformation scheme allows heterogeneous sets of coordinates and different levels of deformation to be used, thus obtaining fast evaluations and a reduced memory footprint. This results in a extremely flexible and versatile tool very useful for a wide range of applications from industrial design to computer animation.

5.1 Introduction

Shape deformation plays a central role in computer graphics, both in two and three dimensions. Space deformation techniques have received a lot of attention, especially in cage-based methods as a practical means to manipulate 3D models [Flo03] [JMD+07] [LLCO08] [WBCG09]. A cage is a low polygon-count polyhedron, which typically has a similar shape to the enclosed object. The object points inside the cage are represented by affine sums of the cage elements (vertices or faces) multiplied by special weight functions called coordinates. The main advantage of these space deformation techniques is their simplicity, relative flexibility and speed on applying the deformation. Manipulating an enclosed ob-

ject, for example a surface mesh, requires transforming a point by a linear combination of the cage geometry using a set of precalculated coordinates. Moreover, since each point is transformed independently, these techniques are indifferent to the surface representation and in general free of discretization errors.

However, up to now there is no widely accepted solution that provides both user control and high quality deformations. It is commonly accepted that an ideal deformation system should allow user intervention when it is required but infer all the missing data automatically. For instance, given a user-chosen set of constraints, the system should find the best deformed shape that satisfies them. There are several possible alternatives, like Mean Value Coordinates (MVC) [Flo03], Harmonic Coordinates (HC) [JMD+07] or Green Coordinates (GC) [LLCO08], but they present some problems. They can be classified as global deformation methods because they are defined in terms of a single cage which affects all mesh vertices. Also, the construction of these single cages around the entire model may not always be easy, while generating specific smaller cages around a region of interest is rather simple. Even more, unnecessary memory space is used and more time is consumed if local deformations are to be applied. Moreover, all of them present continuity problems on the cage boundaries, from lack of smoothness to discontinuities. From these aspects we can see that their use was limited to being applied to monolithic single cages, without any possibility of combining their strengths and, even less, to be used in a multi-level deformation system. Even more, currently there is no method that allows all the advantages of these powerful coordinates to be combined in order to apply finer-detail deformations together with large-scale ones. A combination of different cages at different levels of granularity could provide an interesting tool, which would offer both flexible mesh deformation and would require minimal computational resources (memory, computational power) as they would consume only what is necessary for each cage in isolation.

In this work we present ∗Cages (pronounced *star-cages*), a cage-based deformation method that involves a hierarchical set of cages where the leaf cages bound the object in a piece-wise manner. Cage coordinates can be individually defined for each leaf cage, and blended among neighbouring cages to produce a smooth (class C^1) deformation, thus offering localized deformation control with fast computation. The hierarchy further allows deformation control to take place at multiple levels. In this sense, we can say that ∗Cages *complements* the existing techniques instead of *competing* with them. Hence the reason for its name: ∗Cages can accommodate *any* coordinate system inside a cage, and smoothly combine *any* number of cages to get a flexible general deformation system at *any* level of detail. The main contributions of the proposed technique are:

- We can have any number of cages at different levels of granularity, and use each one to smoothly deform the base mesh.
- As far as we know, this is the first system that allows the use of heterogeneous sets of coordinates: We are able to define different coordinates for different cages and use them together in combination.
- We allow a multi-level deformation scheme, where different cages are used to locally control different levels of detail in the model, from a whole-model deformation to a very localized one.
- As a consequence, our method has small memory and computational requirements.

Together this gives rise to an extremely flexible and versatile deformation scheme, which is much more intuitive and user-friendly.

5.2 Previous Work

Cage-based deformation methods are considered one of the most important space deformation techniques, driving the deformation by a control cage which encloses the fine-detailed model to be deformed. The first method based on three dimensional regular lattices was introduced by Sederberg and Parry [SP86]. Later, this method was extended to handle general lattices [Coq90] and LOD management [SMT00]. In recent years, new deformation methods have been proposed based on the use of coordinates with respect the vertices of a single enclosing cage. Floater and co-workers [Flo03] [FKR05] [JSW05] introduced Mean Value Coordinates (MVC) as a method for constructing an interpolant for closed triangular meshes. MVC have a closed-form formulation and allow linear functions to be reproduced. They are well defined both inside and outside the control mesh (C^∞ continuous) but they are only C^0 continuous across the cage faces. Later, Joshi et al. [JMD+07] proposed Harmonic Coordinates (HC) for character articulation. In contrast to MVC, HC are guaranteed to be positive everywhere in the interior of the cage, while their influence decreases with distance as measured within the control mesh. However, as they do not have an explicit expression, they force the use of a multi-grid finite difference to compute the coordinates. HC are C^∞ continuous inside the cage, C^0 continuous on the boundary and have no definition outside the cage. Lipman et al. [LKCOL07] presented an alternative non-negative coordinate definition to MVC (PMVC). The coordinates are computed numerically by using a GPU-friendly approach. Later, Lipman et al. [LLCO08] proposed

a new shape-preserving space deformation approach called Green Coordinates (GC). The work, motivated by Green's third integral identity, produces conformal mappings, and extends naturally to quasi-conformal mappings in 3D by using both vertex positions and face orientations of the cage. GC are C^∞ continuous inside and outside the cage but discontinuous at the boundary, although some extension mechanism can be defined. ∗Cages is a technique that provides smoothness to any of these coordinates across multiple cages, allowing them to be used in combination. Also, the introduction of a multi-level system of cages provides the artist with much finer control for specific local deformations.

Langer et al. [LBS08] developed criteria for the construction of smooth maps, called Bézier maps, that are piecewise a homogeneous polynomials in generalized barycentric coordinates. For that purpose, they had to increase both the number of control points and the order of the polynomials to avoid discontinuities, with the corresponding added computational cost. In the work by Ben-Chen et al. [BCWG09], the challenge was to find a harmonic map from a domain such that it satisfies constraints specified by the user, and it is detail-preserving and intuitive to control. Huang et al. [HCLB09] presented a mesh deformation technique using modified barycentric coordinates with a tetrahedron control mesh that avoids first order discontinuities across the cage boundaries. Unlike them, we are not restricted in the nature of the cages we can use.

Other deformation techniques take advantage of the previously mentioned cage coordinates to define new deformation approaches. A GC-based technique to locally deform a mesh contained by an automatically generated umbrella-shaped cell was presented by Li et al. [LLD$^+$10]. Even though their cage is local, they need to bind coordinates for all mesh vertices, increasing the memory consumption. Ju et al. [JZvdP$^+$08] introduced skinning templates as a solution to share and reuse skinning behaviours for similar joints and similar characters. The skinning templates were implemented using cage-based deformations, and thus they can benefit from all the features of our approach. A hybrid approach that combines surface-based and cage-based deformations was presented by Borosan et al. [BHZN10]. There, an "as-rigid-as-possible" method was applied on a region of interest of the cage and then the obtained deformation was transferred to the mesh using MVC. As they note, too coarse meshes limited the effectiveness of the method making their approach suitable only for local mesh deformations. Moreover, the resulting deformation of the mesh that falls on the boundary of the cage is not smooth. ∗Cages, instead, is able to provide smoothness to an arbitrary combination of different coordinates. Recently, Landreneau and Schaefer [LS10] introduced a Poisson-based method to reduce the storage needs of the coordinates for animated meshes. Our method can be used in combination with theirs, as

both consider different aspects of the deformation: while ∗Cages aims at creating a hierarchical set of cages and cage-coordinates can be individually defined for each cage, the work by Landreneau and Schaefer aims at making a coordinate system local while still using a global cage, thus reducing storage and computational needs.

Cage-based deformations have also been applied to planar domains. One related work was introduced by Meng et al. [MSW+09], who designed a method to keep the shape of images during the deformation of a region of interest. In their approach, different types of discontinuities can appear over the deformed image depending on the cage coordinates used (MVC, HC or GC). Later, Weber et al. [WBCG09] generalized the concept of barycentric coordinates from real numbers to complex numbers, only applicable to two dimensional shape deformations. Let us remark that our method can be applied to planar domains as well as 3D domains.

As far as we know, none of the existing cage-based methods can be used in a multi-level approach to perform both local and global deformations on the same framework if a smooth mesh must be obtained after applying the desired deformations because of the discontinuity problems already mentioned.

All these space deformation techniques require a construction of a cage or other 3D structures around the manipulated object. One of the first steps to automatically generate coarse bounding cages while keeping the main features of the original model was presented by Xian et al. [XLG09]. The construction and manipulation of the cages is closely related to the deformations obtained, thus this approach may not be suitable for any situation. Moreover, this method can generate an unnecessary amount of vertices, giving as a result not only more complex-to-use cages, but also more time-consuming deformations. Let us note that the aim of our work is not to build cages.

5.3 ∗Cages

As we mentioned in the introduction, the main objective of ∗Cages is to allow deformations at different levels of detail on the model while preserving the smoothness of the enclosed mesh. This is achieved by defining a set of non-intersecting cages that share boundaries and are responsible for directly modifying the mesh. Each cage can have its own coordinate system, and ∗Cages provides smoothness at cage boundaries and between coordinate systems. The basic definitions are presented in Section 5.3.1. Then, a new transformation to obtain smooth transitions between this set of neighbouring cages is introduced in Section

5.3.2. Next, an enriched transformation, presented in Section 5.3.3, allows different cage coordinates to be used on each cage while smoothly combining them.

Because of the local nature of the presented transformations, a multi-level deformation scheme can be used to incorporate different levels of deformation, from a whole-model deformation to a very localized one. These new transformations allow extremely fast interactive deformations because they only need to compute affine combinations of a low number of precalculated coordinates corresponding to the affected cages. In Section 5.3.4, a multi-level system is introduced by taking advantage of the powerful transformations previously defined, where the already presented cages are called *leaf cages* and the rest of the upper-level cages, which own the vertices of cages at lower levels, are called *internal cages*. Here, intersections between internal cages are allowed, and smooth transitions between the internal-cage transformations do not need to be enforced.

5.3.1 Definitions

Let $C = (C_0, C_1, \dots C_n)$ be a set of control cages satisfying $\text{Interior}(C_i) \cap \text{Interior}(C_j) = \emptyset$ for all $i \neq j$. Let $V = (v_0, v_1, \dots v_m)$ be the vertex set of C, and let VC_i be the set of vertices of cage C_i. By $Adj(C_i)$ we denote the adjacent cages of cage C_i. For a cage $C_j \in Adj(C_i)$, let $B_{ij} = C_i \cap C_j$ be the border between C_i and C_j, and let VB_{ij} be the vertices of B_{ij}. Let the *boundary* of C_i, noted ∂C_i, be the union of all borders B_{ij}. These definitions are illustrated in the left image of Figure 5.1.

Most cage-based deformation methods such as MVC, PMVC and HC express a point $p \in C_i$ as an affine combination of cage vertices $v \in VC_i$:

$$p = \sum_{v \in VC_i} w_i(v, p) v \,,$$

where $w_i(v, p)$ are the coordinate basis functions. Then the natural way to define a deformation in each cage C_i is by:

$$T_i(p) = \sum_{v \in VC_i} w_i(v, p) T(v) \,,$$

where $T(v)$ are the deformed control cage vertices.

In this manner, the piecewise transformation defined on C by transformations T_i is only C^0, i.e., it can generate first-order discontinuities across common faces of adjacent cages (see Figure 5.2(a) where classic MVC/PMVC/HC were used). Transformations T_i could be also defined by the GC technique, which uses both vertices and face normals

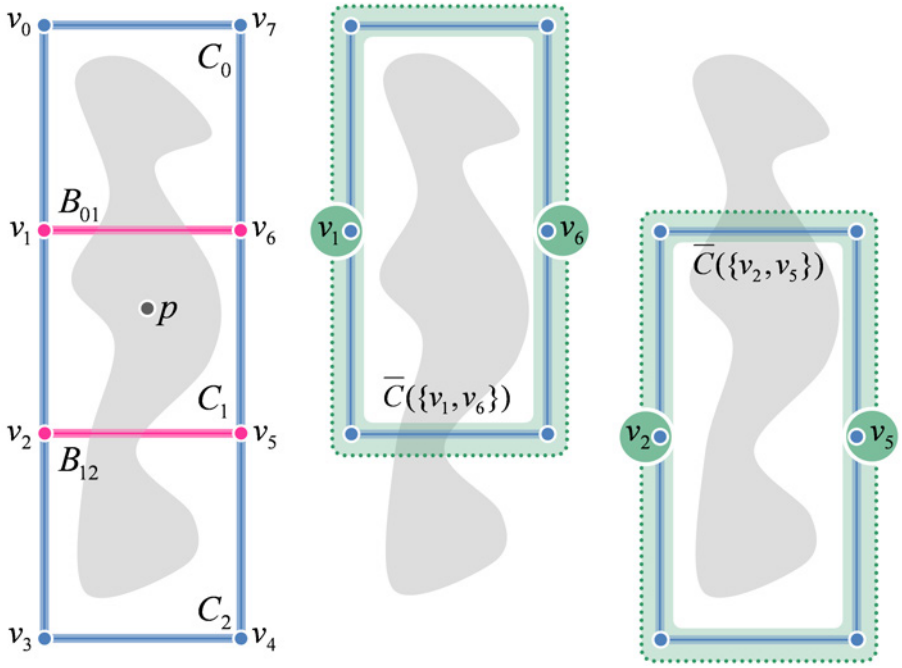

Figure 5.1: Left: $C_0 = v_0v_1v_6v_7$, $C_1 = v_1v_2v_5v_6$, $C_2 = v_2v_3v_4v_5$, $\partial C_0 = B_{01}$, $\partial C_1 = B_{01} \cup B_{12}$, $\partial C_2 = B_{12}$. Middle: Cage $\overline{C}(\{v_1, v_6\}) = v_0v_1v_2v_5v_6v_7$. Right: Cage $\overline{C}(\{v_2, v_5\}) = v_1v_2v_3v_4v_5v_6$.

of the cages. Unfortunately, the corresponding piecewise transformation is discontinuous at cage boundaries (see Figure 5.2(b)). The insets in the figures show a detail of these discontinuities.

To address this continuity problem, we first obtain a C^1 transformation in each cage C_i whose associated piecewise transformation on C, which we call *Join Transformation* (J), is C^1 on C. Then, the J transformation is blended inside each cage C_i with transformations T_i obtaining another C^1 transformation on C, which we call *Smooth Transformation* (S). In this manner, we can apply any of the existing cage coordinates at each cage to combine their strengths to generate different transformations, while preserving smooth transitions between cages.

5.3.2 Join transformation

Our proposal consists of defining a transformation J_i for each cage C_i. This transformation is created by blending transformations defined on cages resulting from joining C_i with its adjacent cages. This blending is done through a partition of unity, which is constructed by the use of coordinate functions with respect to the cage vertices. There are two main

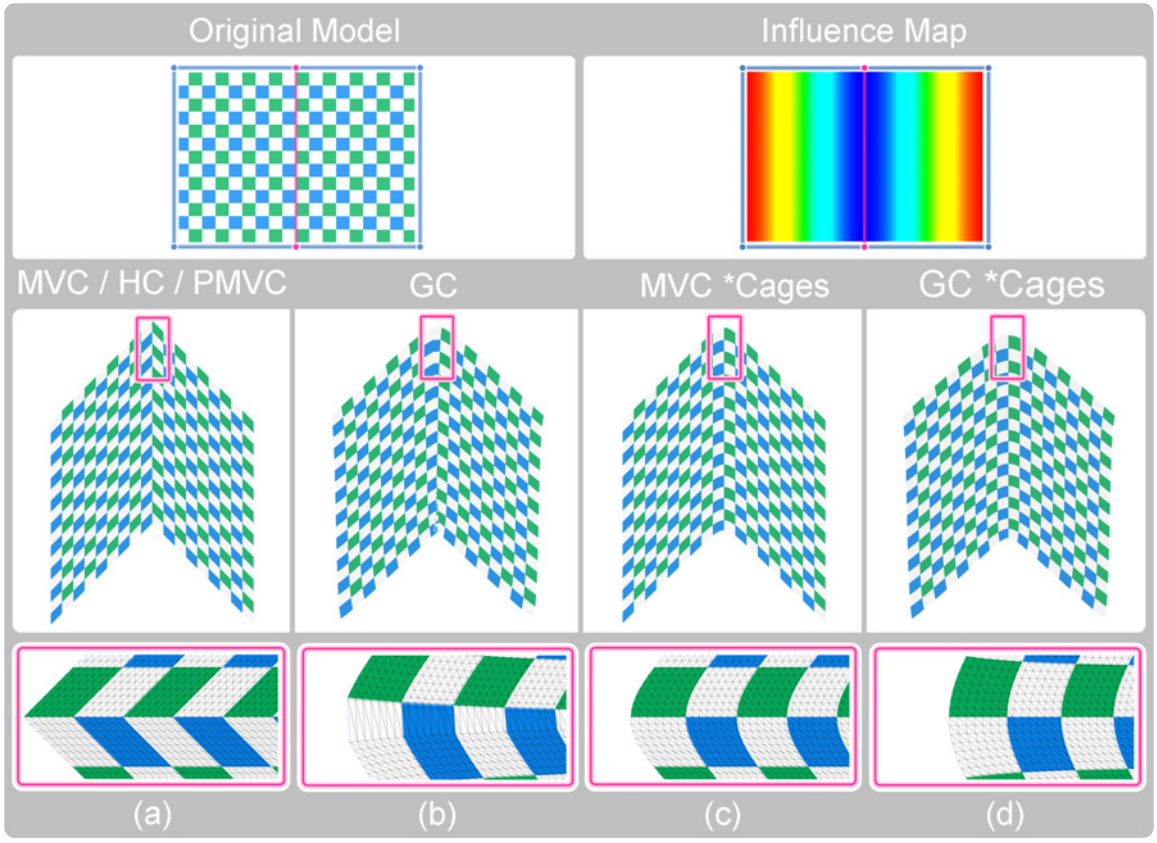

Figure 5.2: A comparison between piecewise deformations. At the top, the original model enclosed in two cages and its influence map. (a) MVC/PMVC/HC deformation. (b) GC deformation. (c) ∗Cages with MVC deformation, both for J and T. (d) ∗Cages with GC deformation, both for J and T. The second row shows close views of the deformed model. Notice that only (c) and (d) are C^1.

reasons for this: the coordinates are of class C^1 inside the cages and they are equal to zero on the cage faces that do not contain the vertex itself.

We proceed as follows. Let $\overline{C}(v)$ be the union of the incident cages to a vertex $v \in \partial C_i$, which we call the *Join Cage* of v. Then, the set of vertices of ∂C_i is partitioned in subsets according to having the same join cage. From now this partition will be denoted by $\overline{\partial C}_i$, their elements by $[u]$ and the join cage of the vertices belonging to $[u]$ by $\overline{C}([u])$ (see the middle and right images of Figure 5.1). Given a join cage $\overline{C}([u])$ we consider the transformation $T_{[u]}$ (MVC/HC/GC) determined by the displacement of its vertices, and let $\overline{w}(v,p)$ be the coordinate function with respect to $v \in [u]$ in cage $\overline{C}([u])$. The function $\overline{w}(v,p)$ needs to be computed by the MVC or HC approaches to ensure it is of class C^1 in $\overline{C}([u])$.

Then, we define the join transformation J_i in C_i by:

$$J_i(p) = \frac{\sum_{[u]\in\overline{\partial C}_i} f(\overline{W}_{[u]}(p))T_{[u]}(p)}{\sum_{[u]\in\overline{\partial C}_i} f(\overline{W}_{[u]}(p))},$$

where

$$\overline{W}_{[u]}(p) = \sum_{v\in[u]} \overline{w}(v,p)\,,$$

and $f : [0,1] \to [0,1]$ is a *smoothing* function that satisfies $f(0) = f'(0) = 0$, $f(1) = 1$ and $f'(x) \geq 0$. In our implementation we have used $f(x) = \frac{1}{2}sin(\pi(x - \frac{1}{2})) + \frac{1}{2}$. To see an example, transformations

$$J_0(p) = \overline{T}_{\{v_1,v_6\}}(p)\,,$$

$$J_1(p) = \frac{f(\overline{W}_{\{v_1,v_6\}}(p))\overline{T}_{\{v_1,v_6\}}(p) + f(\overline{W}_{\{v_2,v_5\}}(p))\overline{T}_{\{v_2,v_5\}}(p)}{f(\overline{W}_{\{v_1,v_6\}}(p)) + f(\overline{W}_{\{v_2,v_5\}}(p))},$$

$$J_2(p) = \overline{T}_{\{v_2,v_5\}}(p)\,,$$

(being $\overline{W}_{\{v_1,v_6\}}(p) = \overline{w}(v_1,p) + \overline{w}(v_6,p)$, $\overline{W}_{\{v_2,v_5\}}(p) = \overline{w}(v_2,p) + \overline{w}(v_5,p)$) are the join transformations of the cage system illustrated in the left image of Figure 5.1.

Transformation J_i is C^1 in the interior of C_i and for a point $p \in B_{ij}$ we have:

$$J_i(p) = J_j(p), \quad \text{Jac}(J_i)(p) = \text{Jac}(J_j)(p)\,,$$

being Jac the Jacobian.

5.3.3 Smooth transformation

Once we know how to generate a smooth transformation between neighbours in C, we are interested in solving the problem of being able to combine a *different* coordinate system for each cage in a way that suits the user needs. The only requirement for the coordinate system is that it must be defined inside the cage. In this section, we explain how to combine the previous smooth transformation with others (e.g. MVC, HC, GC) defined on each cage.

For a point $p \in C_i$ we define a weight with respect to each border B_{ij} and a distance with respect to the boundary ∂C_i as follows:

- Weight of p with respect to B_{ij}:

$$w_{ij}(p) = \sum_{v\in VB_{ij}} w_i(v,p)$$

- Distance of p with respect to ∂C_i:

$$d_i(p) = f_{h_i}(\prod_j (1 - w_{ij}(p)))$$

where f_{h_i} is a smoothing function depending on a parameter $h_i \in (0, 1]$ satisfying $f_{h_i}(0) = f'_{h_i}(0) = f'_{h_i}(h_i) = 0$, $f_{h_i}(x) = 1$ if $x \in [h_i, 1]$, and $f'_{h_i}(x) \geq 0$. In our implementation we have used $f_{h_i}(x) = \frac{1}{2} sin(\pi(\frac{x}{h_i} - \frac{1}{2})) + \frac{1}{2}$ for $x < h_i$.

Observe that $d_i(p) \in [0, 1]$ is equal to 0 when $p \in \partial C_i$ and is equal to 1 on the faces of cage C_i that are not incident to any vertex of ∂C_i. Moreover, $d_i(p)$ is set to be 1 in case that $\partial C_i = \emptyset$, so when C_i does not have any neighbouring cage, its own transformation is fully applied.

Let T_i be a cage deformation (e.g. MVC, HC, GC) defined on C_i. Then, we define a smooth transformation S_i in C_i by:

$$S_i(p) = d_i(p)T_i(p) + (1 - d_i(p))J_i(p)$$

Since for a point p on B_{ij} we have:

$$S_i(p) = J_i(p) = J_j(p) = S_j(p)\,,$$

$$\mathrm{Jac}(S_i)(p) = \mathrm{Jac}(J_i)(p) = \mathrm{Jac}(J_j)(p) = \mathrm{Jac}(S_j)(p)\,,$$

the piecewise transformation S defined on neighbouring cages of C by transformations S_i is C^1 in C. Figures 5.2(c) and 5.2(d) illustrate the effects of a smooth transformation. In Figure 5.2(d) a MVC deformation has been performed by using it to compute both the cage transformations and the join transformation while in Figure 5.2(d) GC are used.

Moreover, transformation S inherits properties of transformations T_i and $\overline{T}_{[u]}$. Then, if all transformations T_i and $\overline{T}_{[u]}$ are affine invariant or perform boundary interpolation, transformation S also is.

The distance $d_i(p)$ is a measure of the influence of the transformation $T_i(p)$ in S_i, and can be adjusted by changing the parameter h_i. To facilitate this task we use an influence map (see the top-right image of Figure 5.2), where the model is painted in blue-red gradation according to the distance $d_i(p)$. The effect of the h_i variation can be appreciated in Figure 5.3. At the right, results obtained from three different values of the h_i corresponding to the front ear cage are shown.

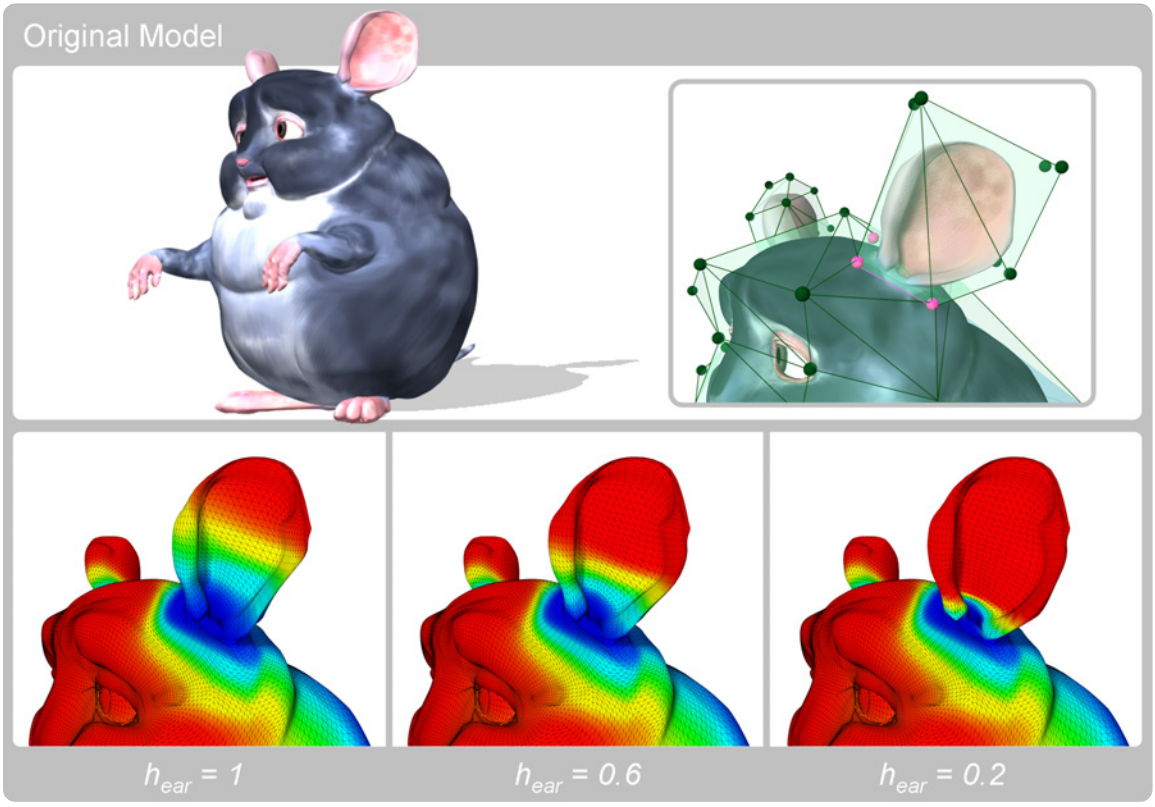

Figure 5.3: Influence map variation on the chinchilla model. At the top, the original model and a close-view with the initial cages. At the bottom, the original model with its initial cages (left) and the results obtained by using different h_i values for the front ear cage (right).

5.3.4 Multi-level deformations

We can use ∗Cages to build a multi-level system which gives flexibility, versatility, interactivity and control over the deformations to be applied to a part of the model. In our scheme, upper-level cages can own an arbitrary set of vertices of lower-level cages, being the only restriction that cages must have a hierarchical relation (e.g. a *Directed Acyclic Graph* or a tree) and that a given cage vertex cannot be controlled by more than one parent cage. An example can be seen in Figure 5.4, where the two kinds of cages can be distinguished. On the left, the leaf cages that directly control the mesh cannot intersect each other and the transformations applied have to ensure smooth transitions between them. On the right, the internal cages that control cage vertices of lower-level cages. Because of the transformations applied to them do not directly affect the mesh, these upper-level cages can intersect and smoothness does not need to be enforced. In the figure, the cage C_3 controls all the vertices

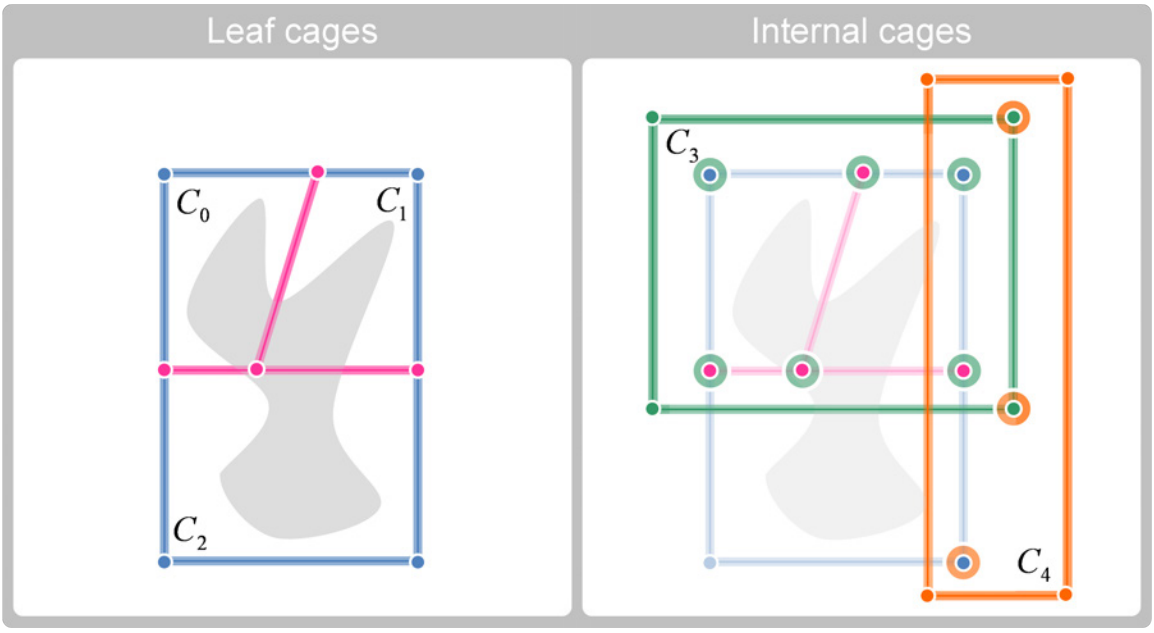

Figure 5.4: Multi-level scheme. Colours indicate the correspondence between cages and their controlled vertices.

of C_0 and C_1, and the cage C_4 controls two vertices of the cage C_3 and one vertex of the cage C_2.

Our multi-level system relies on a simple yet effective observation: When a cage in the multi-level system changes, the effects of this change should be propagated only downwards, not upwards, in the hierarchy. This means that, when a vertex v of a cage is changed, the positions of all the vertices contained in the cage should be updated, but not their coordinates with respect to the cage. Thus, the parent cage C_i containing v would not be affected. However, if C_i changes later on, v should be updated according to the new position of C_i, which implies recomputing its coordinates with respect to the owning cage. This means recalculating only cage vertices, not the mesh vertices. To save unnecessary work, we perform this recomputation only in this specific situation. If coordinates that are defined everywhere are used, like MVC, then the above implementation works as described.

However, in the case of coordinates *not* defined outside (e.g. HC), special measures should be taken. We propose an easy but effective solution without the need for any cage recomputation. We can express any new position for v as $v' = T(v) + t_v$ with t_v being the user-generated displacement, and $T(v)$ the transformation with respect to the parent cage C. We can express $t_v = \lambda_v st_v$, where st_v is a displacement small enough to satisfy that point $sv = T(v) + st_v$ within cage C at its current position. Now, if cage C_i suffers

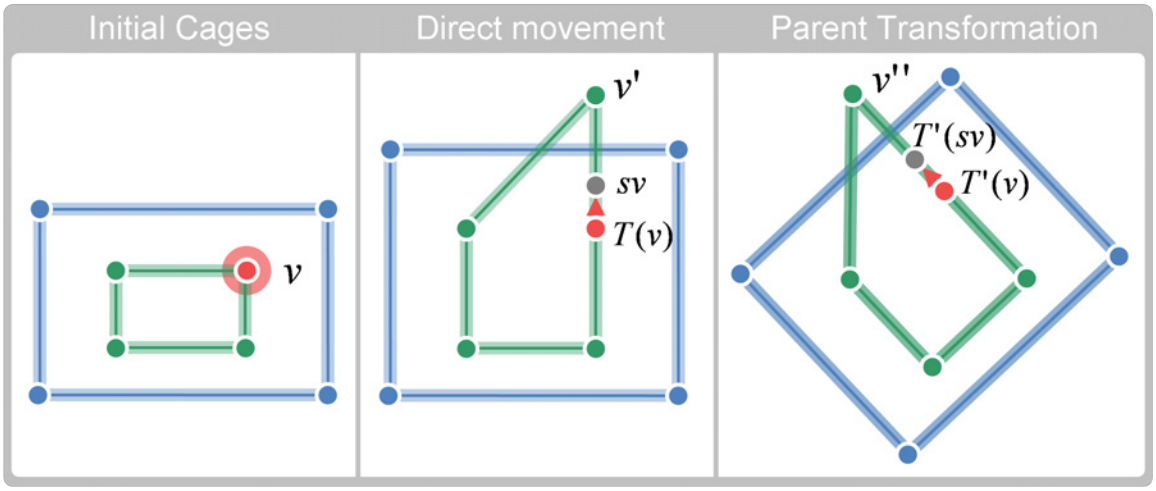

Figure 5.5: Multi-level deformation for coordinates not defined outside the cage. (a) Initial cages. (b) Direct cage vertex movement. (c) Parent transformation.

another transformation T' that converts $T(v)$ into $T'(v)$ and sv into $T'(sv)$, we will update the current position of v by $v" = T'(v) + \lambda_v(T'(sv) - T'(v))$ (see Figure 5.5).

5.4 Results

For the whole results, we have used the following colouring convention: blue cages use MVC, red ones use HC and the green ones use GC. We have also drawn the boundaries between cages in pink.

Figure 5.6 shows a comparison between single cage approaches and ∗Cages. The prism model is twisted (each cage level is rotated by $\pi/2$) using four cages by ∗Cages and the union of them as a single cage. See the large similarity between the results obtained by MVC and GC (Figure 5.6(a) and Figure 5.6(d)) and ∗Cages (Figure 5.6(b) and Figure 5.6(e)). Both cage and join transformations have been computed with the same coordinate systems (MVC/GC). The corresponding error maps are shown in Figure 5.6(c) and Figure 5.6(f) with the maximum (depicted in red) and RMS errors. Note that the differences between the single cage and ∗Cages approaches are more noticeable at the center of the middle cages. Another example of a single cage and ∗Cages comparison is shown in Figure 5.7. Again, the camel model is deformed in our approach by using four cages (Figure 5.7(a)), and in MVC by using the union of them as a single cage. Two deformations have been applied for MVC (Figure 5.7(c) and Figure 5.7(d)) and ∗Cages (Figure 5.7(e) and Figure 5.7(f)). The resulting error maps (Figures 5.7(g) and Figure 5.7(h)) show the differences and the low

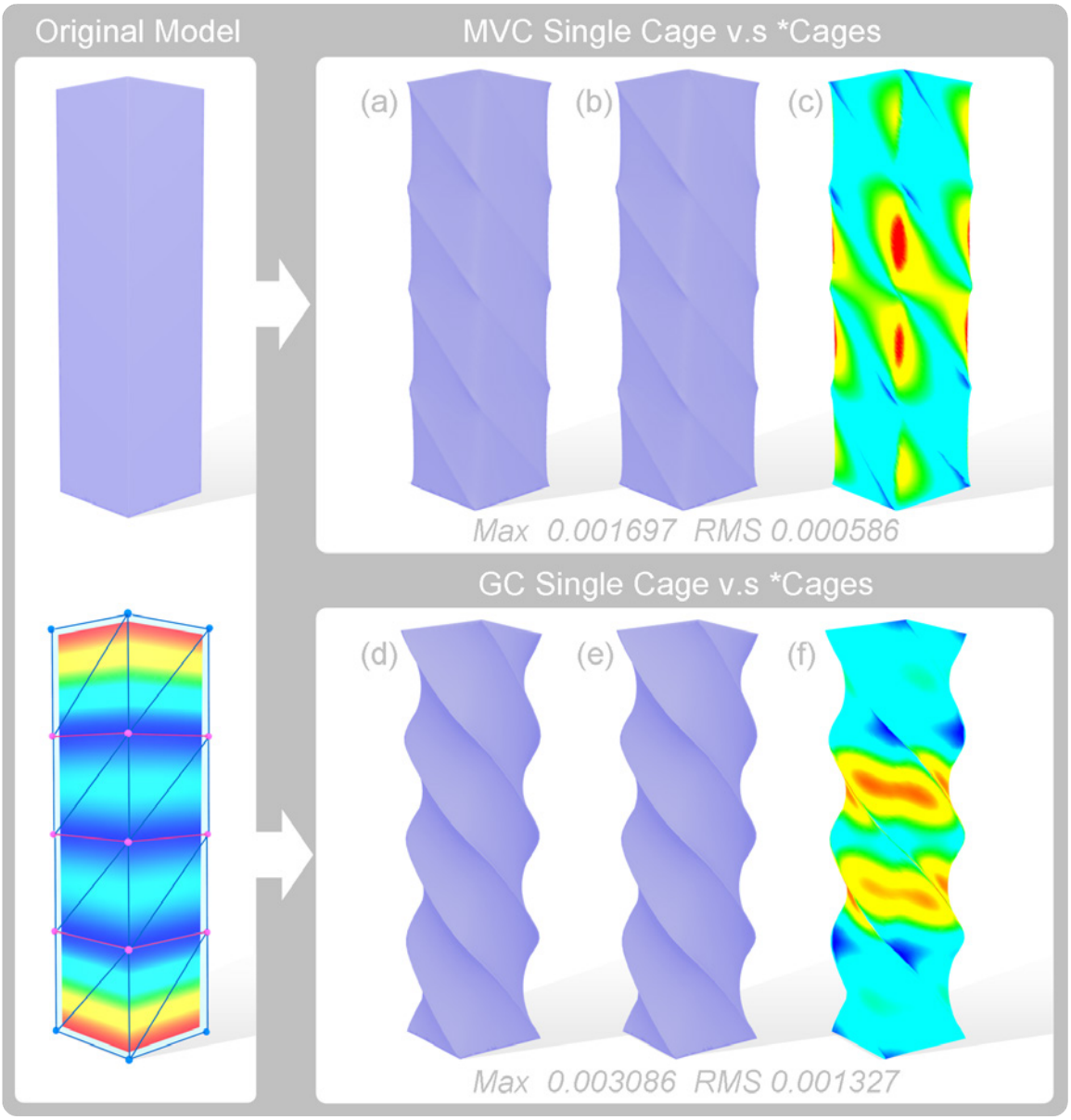

Figure 5.6: Twisting a prism using MVC (left) and GC (right). See the large similarity between single cage (a,d) and ∗Cages (b,e) results. (c,f) show the corresponding error maps.

level of error obtained. Please note that the cages defined for the camel's front and back legs are different, as the front ones have articulations also for the camel's knees resulting in differences in the influence map (Figure 5.7(b)).

Figure 5.8 illustrates that ∗Cages supports the combination of different types of coordinates in different cages. Two different combined transformations have been applied to the elk toy model. HC have been used for the deformation of the body cage for both deforma-

tions, while the head and wheel cages have been deformed by MVC in the left image and by GC in the right image. Observe the differences between the results depending on the coordinates used and how ∗Cages provides a way to smoothly combine them.

∗Cages is able to handle any number of cages meeting at a boundary cage vertex. Figure 5.9 shows a deformation obtained from a flower model enclosed with 13 cages using different coordinate types. Observe the good behaviour of the method despite the presence of cage boundary vertices with more than two incident cages.

An example of a multi-level deformation is illustrated in Figure 5.10. The squirrel model has been enclosed by four leaf cages: teeth, face, left ear and right ear. There are also two internal cages (coloured in grey). The ears' cage englobes some vertices of the left ear and right ear cages. The head cage englobes all unbinded vertices of the previous cages (see Figure 5.10(a)). The sequence of deformation is as follows: the teeth have been deformed in Figure 5.10(b), the ears in Figure 5.10(c), the entire head in Figure 5.10(d) and the face in Figure 5.10(e). Observe the degree of control achievable by the multi-level system.

In the right of the Figure 5.11 we show a final render of the deformations applied over the squirrel and the chinchilla models. Note that the chinchilla model has 9 leaf cages and the squirrel model has 11 leaf cages and 3 internal cages, as can be seen on the left of the image. Let us remark that different coordinates have been used for different cages.

Memory and time requirements are listed in Tables 5.1, 5.2 and 5.3. In Table 5.1 we show the influence of the parameter h_i in the boundary distance function $d_i(p)$ (see Section 5.3.3). We compare the results obtained for three different h_i values constant for all cages of the chinchilla model both for MVC and GC. Let us remark that cage and join transformations have been computed with the same coordinate types. Observe that the memory usage and the computational cost is nearly proportional to h_i. This is because as the h_i values decrease, the own transformations T_i are fully applied on more mesh vertices and then, the join transformations do not need to be computed and are stored on them.

In Table 5.2 we compare ∗Cages with MVC and GC on the prism model (with 4 cages) of Figure 5.6, the camel model (with 4 cages) of Figure 5.7 and the squirrel model (only the 11 leaf cages) of Figure 5.11. Observe that in our experiments ∗Cages consumes from the same up to a half of the memory (column 2) depending on the number of cages and their distribution. The best results are obtained when the number of cages is high and the degree of a cage adjacency is low (squirrel model). The total time required for the preprocess is shown in column 3, specifying the amount of time dedicated to compute the coordinates with respect to the parent cages. Observe that ∗Cages takes less time to compute cage

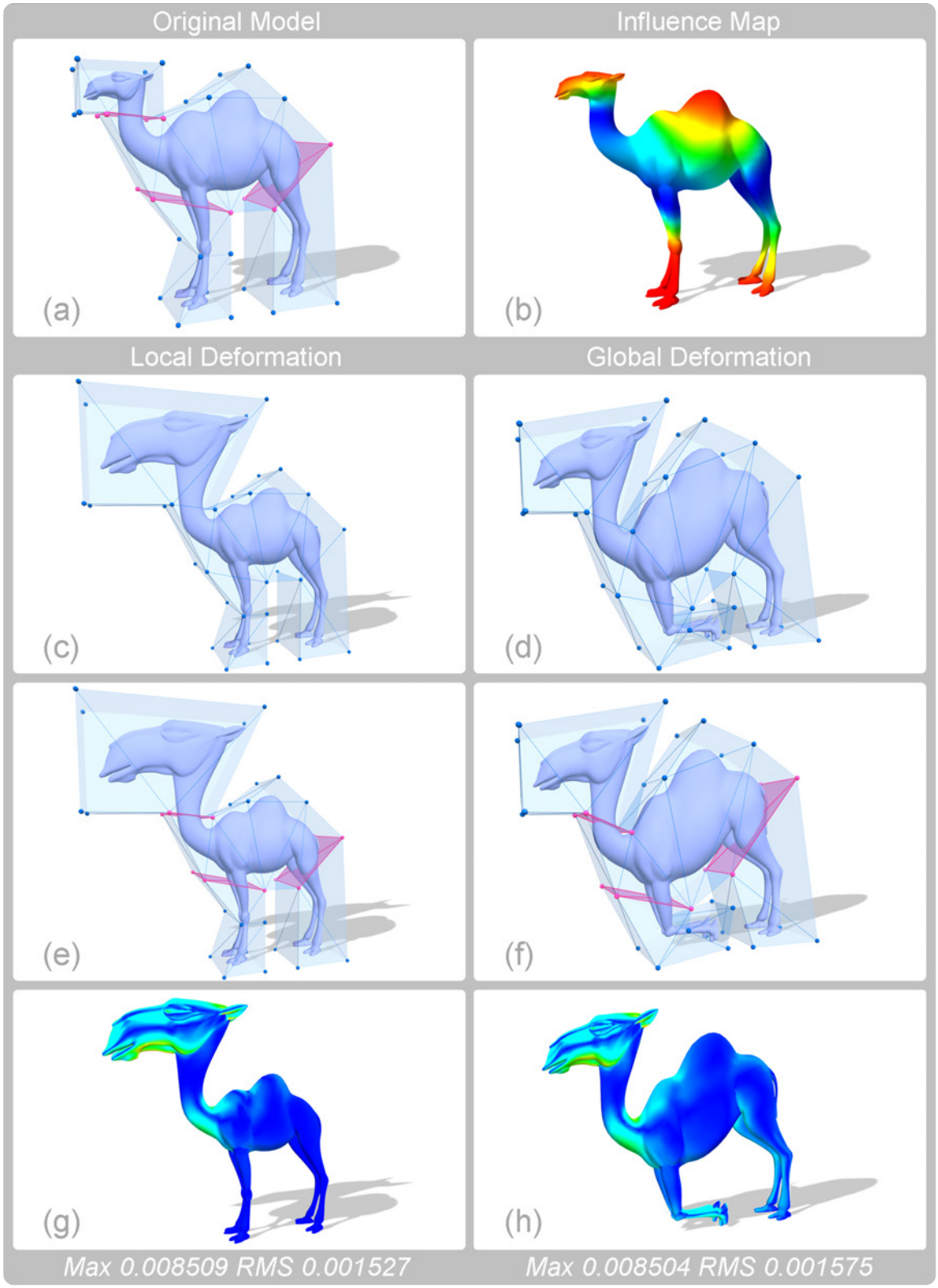

Figure 5.7: Comparison between a single cage approach and ∗Cages on the camel model. (a) Cages and original model. (b) Influence map. (c,d) Two different MVC deformations applied to the union of the cages. (e,f) The corresponding ∗Cages deformations. (g,h) Error maps showing the large similarity between them.

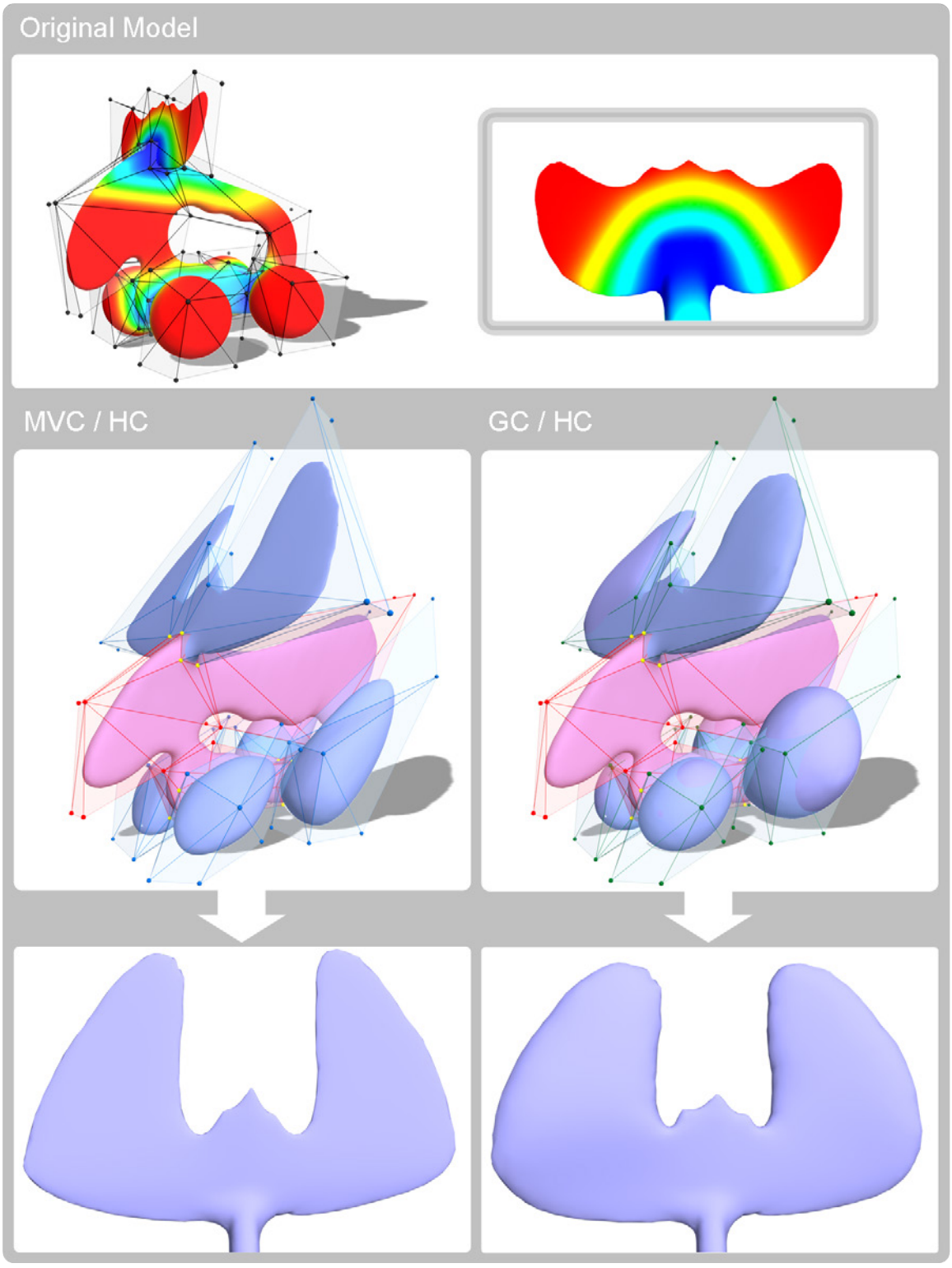

Figure 5.8: Combined deformations of the elk toy model. The first row shows the cages and its influence map. The second row shows two different deformations. The body cage uses HC for both deformations. Head and wheel cages use MVC for the deformation on the left and GC for the deformation on the right. The third row shows close views of the deformed antlers.

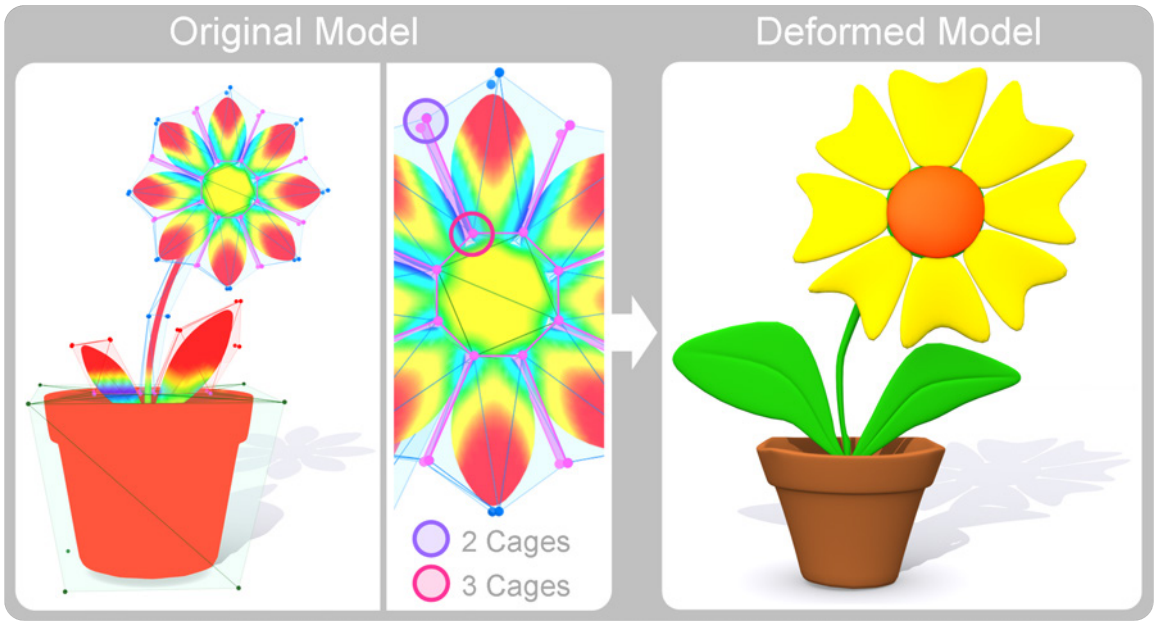

Figure 5.9: Multiple number of cages meeting at a cage vertex. Left: Original model with 13 cages using different coordinates and a close view. Right: Deformation obtained with ∗Cages.

coordinates because the cages used are simpler than a single cage. The extra amount of time is needed to compute join cages and coordinates with respect to them. In case of using GC, ∗Cages spends less preprocess time because of the nature of their computations [LLCO08]. The deformation time (column 4) is the average of the time needed for a deformation of a cage vertex. Observe that our approach is significantly faster, especially for the squirrel model where we achieve about 2 times the acceleration of MVC and about 8 times with respect to GC.

Finally, in Table 5.3 we show the requirements of the other 3D models illustrated in the chapter. Observe that when HC are involved, the preprocess increases time considerably due to the way they are computed. In all our experiments ∗Cages deforms the models in less than 0.2 seconds.

We have implemented ∗Cages using the Ogre3D engine and all our experiments were carried out on a quad core duo (2.83GHz).

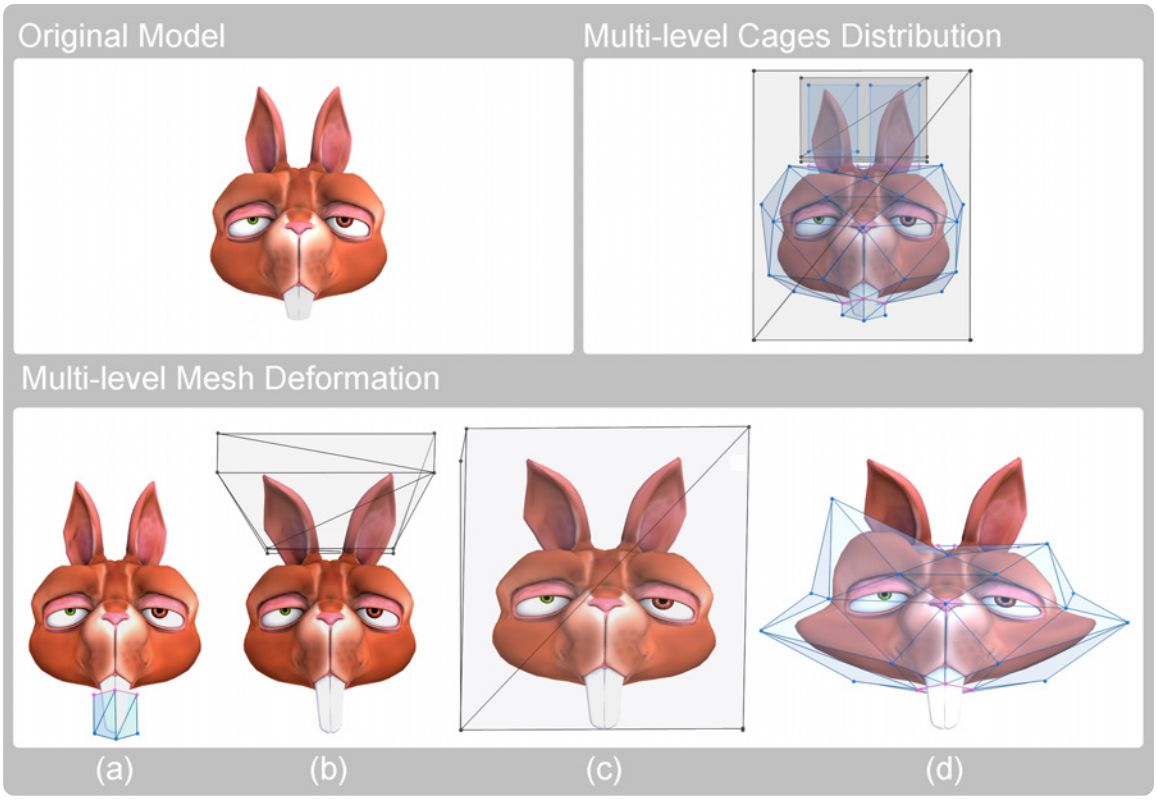

Figure 5.10: Multi-level deformation of the squirrel model. (a) Multi-level cages. (b) Leaf deformation: teeth cage. (c) Internal deformation: ears' cage. (d) Internal deformation: head cage. (e) Leaf deformation: face cage.

Chinchilla	**Memory** (MB)	**Preprocess** (sec)		**Deform** (sec)
		Cage Coord.	Total	
MVC				
h = 1.0	45.45	3.8481	103.2508	0.5277
h = 0.6	29.75	3.7938	46.2589	0.2477
h = 0.2	14.05	3.7944	21.3310	0.1064
GC				
h = 1.0	131.27	10.1164	130.5485	1.1849
h = 0.6	84.35	10.0846	72.0995	0.5012
h = 0.2	37.43	10.0876	47.5258	0.2086

Table 5.1: Memory and time requirements for the chinchilla model using different h_i values for MVC and GC.

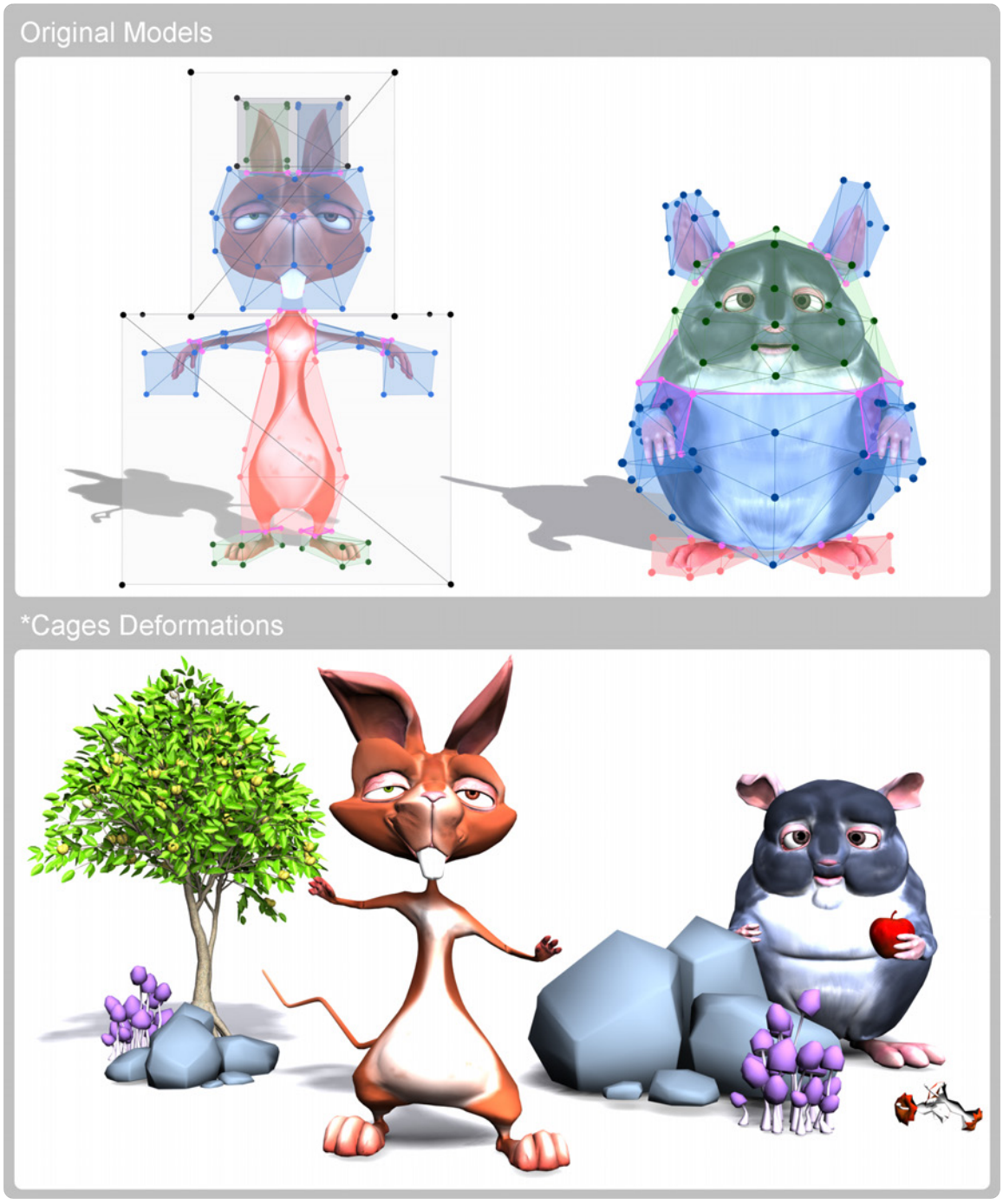

Figure 5.11: Deformation of the squirrel and chinchilla models using ∗Cages. Top: The models and their corresponding multi-level cages at binding time. Bottom: Composition of the two resulting poses. Cage colouring: Green - GC, Blue - MVC, Red - HC.

Model	**Memory**	**Preprocess** (sec)		**Deform**
	(MB)	Cage Coord.	Total	(sec)
Prism				
MVC Single Cage	3.75	1.5164	1.5164	0.0531
MVC ∗Cages	3.99	1.0292	5.8322	0.0936
GC Single Cage	10.50	7.0746	7.0746	0.1871
GC ∗Cages	9.24	2.7612	9.9643	0.1552
Camel				
MVC Single Cage	2.68	1.0634	1.0634	0.0256
MVC ∗Cages	2.80	0.6317	4.2333	0.0554
GC Single Cage	7.75	5.2271	5.2271	0.1159
GC ∗Cages	7.18	1.8454	7.4331	0.0960
Squirrel				
MVC Single Cage	25.55	7.1640	7.1640	0.1029
MVC ∗Cages	11.26	3.4819	11.8422	0.0433
GC Single Cage	63.30	39.3373	39.3373	0.7428
GC ∗Cages	30.39	8.2603	27.9253	0.0903

Table 5.2: Memory and time requirements: comparison between ∗Cages ($h_i = 0.5$) and single cage-based methods.

Model	Vertices	Cages	Memory (MB)	Preprocess (sec)	Deform (sec)
Elk toy					
MVC/HC	26216	6	22.20	305.0652	0.19289
GC/HC	26216	6	30.42	309.7331	0.2067
Squirrel's head	9724	6	6.87	8.7232	0.07821
Flower	10978	13	11.06	31.0083	0.1436
Chinchilla	19274	11	51.64	64.5301	0.1943
Squirrel	19030	15	12.04	48.8736	0.0449

Table 5.3: Memory and time requirements for several models included in the chapter with $h_i = 0.5$.

5.5 Discussion

We have presented ∗Cages, a multi-level (e.g. hierarchical) cage-based system for spatial mesh deformations. It allows heterogeneous sets of coordinates to be combined, allowing the user to define different coordinates for different neighbouring cages and smoothly use them together in combination while preserving their properties (e.g. linear precision, affine invariance, quasi-conformality, boundary interpolation, etc). With ∗Cages, any change the user makes in one cage is kept local to the cage being modified. This is the main advantage with respect to other mechanisms that try to obtain more localized deformations. Moreover, ∗Cages allows the local use of any coordinate, even those that do not allow such usage when used in isolation.

∗Cages allows a multi-level deformation scheme, where upper-level cages are used to locally control deformations in lower-level cages, allowing the passage from a whole-model deformation to a very localized one. Also, it avoids recomputing vertex weights for all vertices by keeping these computations local to the contents of a cage. Observe that the multi-level nature of the binding relationship between vertices of one cage and the parent cage can be re-defined vertex-wise by the user, binding or unbinding vertices according to the specific deformation needs for that model and situation. This allows the user to have the flexibility of an arbitrarily shaped cage while preserving a simpler one for the deformations. Using ∗Cages considerably reduces the management cost of the hierarchy while performing deformations. Thus, ∗Cages is an extremely flexible and versatile deformation tool, which

results in a much more intuitive and user-friendly approach than the current state of the art.

∗Cages reduces computational and memory costs when cages used to deform the model are high and they have a small adjacency degree, that is, when one cage is connected to a reduced number of other ones. Although ∗Cages can handle any number of cages sharing a vertex, as in Figure 5.9, the evaluation cost increases due to the number of join transformations taken into account. This also can be seen in Figures 5.6 and 5.7, where the computational time of ∗Cages is about twice that for a single MVC global cage. Memory requirements are roughly the same with ∗Cages, though. If the number of adjacent cages to a vertex is reduced, however, as in Figure 5.11, ∗Cages outperforms global cage deformations both in memory footprint and speed of evaluation. Moreover, the user can have some degree of control over this behavior by adjusting the influence map parameter h_i. As explained in Table 5.1, changing this variable has a drastic impact on ∗Cages requirements, but also using a value too small for h_i in extreme deformation conditions could introduce visible non-smooth transitions. Also, it is important to mention that the value for h_i can be set in an easy but independent manner for each border, for each cage or for the whole model. In Figure 5.3 we have used the second approach, while for all the tables we have used a single h_i value for the entire model to make comparisons fairer. In our system, the user is provided with a simple slider to control this parameter for each selected cage independently.

Also, the memory and computational needs of ∗Cages can be reduced if used in combination with the coordinate compression technique presented by Landreneau and Schaefer [LS10]. This compression could be used for both cage and join transformations. In the latter case, it would benefit ∗Cages the most, as join cages are more computationally demanding to evaluate.

As a space deformation approach, ∗Cages can be used in the same domains as previous methods. For instance, the lowest-level cages of our hierarchy could be deformed by a simple skeleton, as did Ju at al. [JZvdP$^+$08]. Thanks to the local behavior of our approach, we could provide a finer degree of control over the skeleton, and as a result, a smoother final animation.

Finally, we must also mention that ∗Cages has problems when dealing with cage vertices that become interior vertices of a join cage. One possible way to solve this is to create a very small cage around the vertex and define its join cage as the difference between the union of its incident cages and this small cage. Then, the vertex weight is computed by the sum of the coordinate functions with respect to the vertices of the small cage. However, for

the case of interior vertices in the 2D case, there is always the simple solution of extruding the cages to a set of 3D cages with an infinitesimal width, as shown in Figure 5.12. There, the right finger, hand and eye cages use GC, body and head cages use MVC and the mouth cage uses HC.

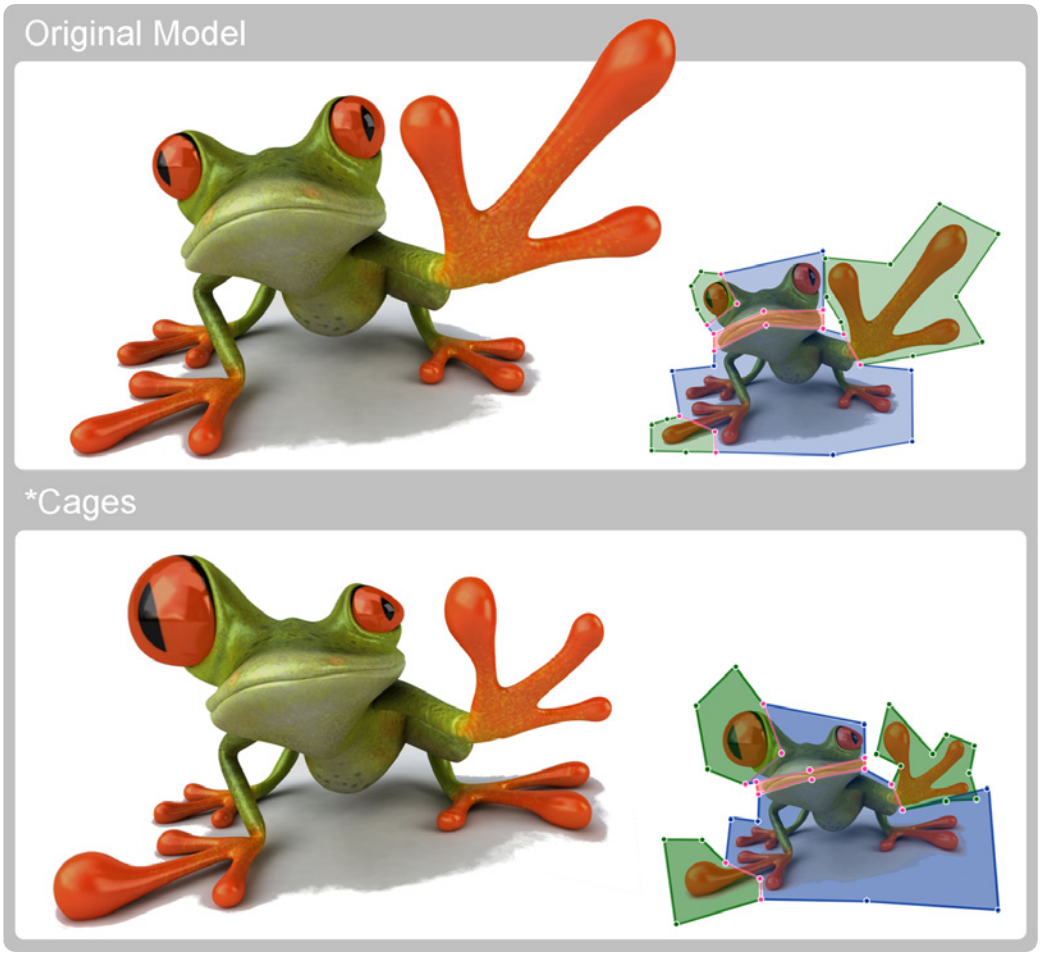

Figure 5.12: 2D deformation. The eye, hand and finger cages use GC. The mouth cage uses HC and the head and body cages use MVC.

Chapter 6

Deformable Compact Models

In this last chapter we combine the good properties of the different methods developed throughout the thesis to provide a powerful tool for geometric modelling applications. From the simplification approach (Chapter 3) we construct a compact model (CM) (Chapter 4) which enriches the approximations by obtaining more faithful and realistic results. Next, the versatile cage-based deformation approach (Chapter 5) introduces us to the mesh editing tools. In order to create a versatile tool that combines the advantages of mesh modelling and mesh editing techniques we provide the CM with the ability to be deformable. In this manner, any of the existing deformation techniques can be applied to large models. Faithful approximations at different poses will be obtained with a reduced memory footprint and a high performance.

6.1 Introduction

Satisfying the increasing desire for realism in several computer graphic applications depends heavily on the ability to accurately represent geometric detail of the objects rendered. In geometric modelling, scanned models provide a high resolution input to achieve this goal. However, the manipulation of such models is not trivial. The challenge is to provide efficient tools, in terms of performance and accuracy, for the edition and management of large models.

A wealth of research has recently been devoted to the deformation and manipulation of surface meshes. On one hand, surface-based deformation methods allow the user to fix one region of a mesh while moving in a small and local region. On the other hand, space-based deformation methods allow users to edit a shape indirectly via a control structure that exerts

a prescribed influence on the enclosed space when deformed. The combined advantages of these two techniques make them the natural choice to obtain real-time applications which require highly detailed models.

We present the *Deformable CMs*, a versatile representation which allows us to obtain good quality deformations on accurate approximations of highly detailed models. The main idea of this hybrid solution consists of applying the deformations over the base of the CM (the simplified model) presented in Chapter 4, and then transferring it to the reconstruction in a simple way. As a result, we obtain high fidelity deformations quickly. Next, we summarize the main contributions, or improvements with respect to the actual state of the art, of the approach presented.

- **Generality of the applicable deformations.** Any kind of deformation can be applied to the CM: affine transformations, surface-based, space-based, direct manipulation, etc. So, both global deformations over the whole model and surface manipulations of localized regions can be performed.

- **Detail preserving.** High quality approximations are obtained thanks to the accurate simplification approach used as a base method, even when a textured model is used. The preservation of surface details after applying a deformation directly depends on the deformation method used and the simplification level of the CM base. So, the linear reproduction property is always fulfilled.

- **Compact representation.** All the properties of the CM are preserved. The approximation is automatically adjusted to satisfy the handle position constraints defined by the deformation applied.

- **Sharp feature preservation.** Models possessing sharp features are correctly approximated and consequently deformed.

- **Adaptive deformations.** The resulting approximation can be adaptively generated according to the desired conditions. Any criteria, even some information related with the deformation applied, can be used to adaptively reconstruct the final mesh.

- **Reduced memory footprint.** The information required to be stored depends on the CM and the deformation method used. In any case, this is much smaller than needed to directly deform the original model.

- **High performance.** Good quality deformation results can be achieved with high performance due to the simplicity of the computations required to transfer the defor-

mations while reconstructing. Moreover, it can be suitable for GPU implementation, so the computations are simple to parallelize.

The hybrid solution proposed represents a versatile tool that can be used in several application fields, from mesh editing and mesh compression to mesh approximation.

6.1.1 Related work

Due to the wide availability of very detailed scanned meshes, recent research has focused on high quality mesh editing. Detail preservation and high performance are the central goals of such algorithms. In Chapter 5, we introduced some of the most important free form deformation approaches. They perform object deformations indirectly by manipulating a control mesh. However, these deformation techniques are usually not viable when large models are used.

Multiresolution methods have been developed for detail-preserving deformations by decomposing the surface into a smooth base representation and the corresponding surface details [ZSS97, KCVS98, GSS99, BK04]. The deformation is applied directly to the base representation and later the high frequency details are added back as displacement vectors.

Surface deformation techniques are also applied on smooth surfaces. In [QMV98], a dynamic framework for the Catmull-Clark subdivision surfaces is presented. It allows the smooth limit surface to be directly manipulated by applying forces. A variational approach to deform subdivision surfaces has been proposed in [MRB05]. Surface details are preserved by optimizing the energy of a deformation vector field instead of the deformation energy of vertex positions.

Energy minimization has long been used for surface deformations. In gradient domain techniques [Ale03, SCOL$^+$04, YZX$^+$04, LSLCO05, ZHS$^+$05, ZRKS05, HSL$^+$06] the deformation is cast as an energy minimization problem, where the energy function incorporates position constraints as well as terms for detail preservation. Algorithms based on differential representations extract local shape properties such as curvature, orientation or scale and strives to preserve them while editing. The detail preservation term is non-linear as it also depends on the position constraints, and various strategies can be used depending on how it is approximated. Minimization of this energy distributes errors globally over the mesh leading to high quality deformation results. The user can directly manipulate the surface mesh and use the region of interest to control the scale of manipulation. In [SYBF06] a multigrid technique for gradient domain mesh deformation is presented to improve the running times.

A combination of gradient domain techniques and subdivision surfaces is presented in [ZHX+07]. Displaced subdivision surfaces and subdivision surfaces with geometric textures are combined in an algorithm for interactive deformation of subdivision surfaces. The main goal is to achieve visually pleasing deformations with high performance. Sumner et al. [SSP07] built a space deformation represented by a collection of affine transformations organized in a graph to manipulate an embedded object. Detail preservation and independent shape representation are the main properties of this useful approach.

In the mesh editing field, geometric modelling and mesh deformation are active research areas in which a lot of previous work has been published. The above review only summarizes some of the most relevant techniques.

6.2 Overview of the Proposal

The main aim of this work is to build an efficient system that will allow applying any kind of deformation from global to local over a highly detailed model. We want to obtain visually pleasing deformations with a high performance. The presented approach generates accurate results while preserving the most important details of the model according to the deformation applied. Next, we summarize the way we achieve it by taking advantage of the previously developed techniques.

We propose deforming the CMs to obtain high quality approximations of deformations of large-scale meshes. The idea is very simple. The base mesh of the CM is deformed first and for each vertex of this mesh we compute a local deformation. Then, every refined point of a CM face is deformed by blending the local deformations of the vertices of the face applied to it. In this manner, the details are added simultaneously with the deformation thanks to the local surfaces stored in the CM and, moreover, the resulting process can be parallelized due to the independent nature of the operations done.

Sharp-feature preservation, textured model management, adaptive approximation and parallelization of the reconstruction process are some of the good properties of CMs that, combined with the ability to be deformable, make our proposal a powerful tool for the interactive manipulation of high resolution models. Moreover, the versatility of the resulting representation allows it to be very useful in several application fields.

The following sections are dedicated to explaining how we compute the local deformations and how they are blended.

6.3 Local Deformations

For each vertex v_i of the CM with deformed position $\overline{v}_i$ we compute a local deformation T_{v_i}. Let v_i^j be the adjacent vertices of v_i and $\overline{v}_i^j$ be their deformed positions. Deformation T_{v_i} is selected as the best affine transformation in the least square sense that transforms points v_i^j into points $\overline{v}_i^j$. In consequence, we compute the transformation

$$T_{v_i}(x) = M(x - v_i) + \overline{v}_i$$

that minimizes

$$\sum_j |T_{v_i}(v_i^j) - \overline{v}_i^j|^2 .$$

Accordingly, matrix M is computed by $M = U^{-1}V$, where

$$U = \sum_j (v_i^j - v_i) \cdot (v_i^j - v_i)^T$$

and

$$V = \sum_j (\overline{v}_i^j - \overline{v}_i) \cdot (v_i^j - v_i)^T .$$

6.4 Transferring Deformations to CMs

From a CM, the original model is reconstructed by refining the simplified base model. In the reconstruction process, each triangle of the simplified model is subdivided into smaller triangles and each vertex p of the refined triangulation is substituted by a point $\Phi(p)$ computed by blending the local surfaces stored at each vertex of the triangle (see 4.5). Let $(\alpha_1, \alpha_2, \alpha_3)$ be the barycentric coordinates of p with respect to its containing face $v_1v_2v_3$. Let T_{v_1}, T_{v_2} and T_{v_3} be the local transformations applied to the vertices v_1, v_2 and v_3 respectively. The natural way to compute the deformed point $T(\Phi(p))$ of $\Phi(p)$ is given by:

$$T(\Phi(p)) = W_1(p)T_{v_1}(\Phi(p)) + W_2(p)T_{v_2}(\Phi(p)) + W_3(p)T_{v_3}(\Phi(p)) ,$$

where

$$W_1(p) = \frac{\alpha_1^3}{\alpha_1^3 + \alpha_2^3 + (1 - \alpha_1 - \alpha_2)^3} ,$$

$$W_2(p) = \frac{\alpha_2^3}{\alpha_1^3 + \alpha_2^3 + (1 - \alpha_1 - \alpha_2)^3} ,$$

$$W_3(p) = \frac{(1 - \alpha_1 - \alpha_2)^3}{\alpha_1^3 + \alpha_2^3 + (1 - \alpha_1 - \alpha_2)^3} .$$

Notice that the deformation method is totally applied over the vertices of the base mesh and it is properly propagated to the reconstructed model. Thus, the level of locality in the deformations depends on the level of detail of the base mesh of the CM.

6.5 Results and Applications

The proposed approach takes advantage of the previously developed methods to present a very useful tool for the deformation of large meshes. Providing the CM with the ability to generate deformed reconstructions makes it a powerful representation for many applications. The accuracy of the simplification method together with the versatility of the CM representation allow different versions of high quality deformations to be obtained depending on user needs. Moreover, we make it possible to apply any of the existing deformation schemes to large models without any change in them. The memory space required and the time consumed are widely reduced with respect to directly deforming the original mesh. The level of reduction obtained depends on the deformation method used, the simplification level of the CM base and the reconstruction level chosen. Next, we present some results to demonstrate the good behaviour of our proposal. All experiments were carried out on a quad core duo (2.83GHz) with a GeForce GTX 280.

Figure 6.1 presents a set of deformation results obtained from the Vase model. It shows how different kinds of deformations are supported, from local deformations that affect a certain region to global deformations that manipulate the whole model. Affine transformations can also be applied to guarantee the linear reproduction property. Figure 6.2 shows a deformation result obtained from a zebra model. In this case, although global deformation has been performed, the resulting approximation obtained from the simplified version correctly preserves the original features of the model.

A representative example of a large mesh is illustrated in Figure 6.3. It shows two different poses obtained from a CM after the original position was manipulated. Observe the good quality of the resulting approximations with only one level of reconstruction.

In Figure 6.4, we compare the results obtained from the same deformation applied to the original, the simplified, the reconstructed and the CM for the Camel model. The performed deformation corresponds to the one presented in Figure 5.7(f) of the last chapter. A comparison of the two last deformation results illustrated at the bottom of the image shows that the differences between them are almost inappreciable. The one labeled *CM Reconstruction* was generated by first reconstructing the model from the CM and then

Figure 6.1: Set of deformation results from a CM of the Vase model reconstructed at level 2.

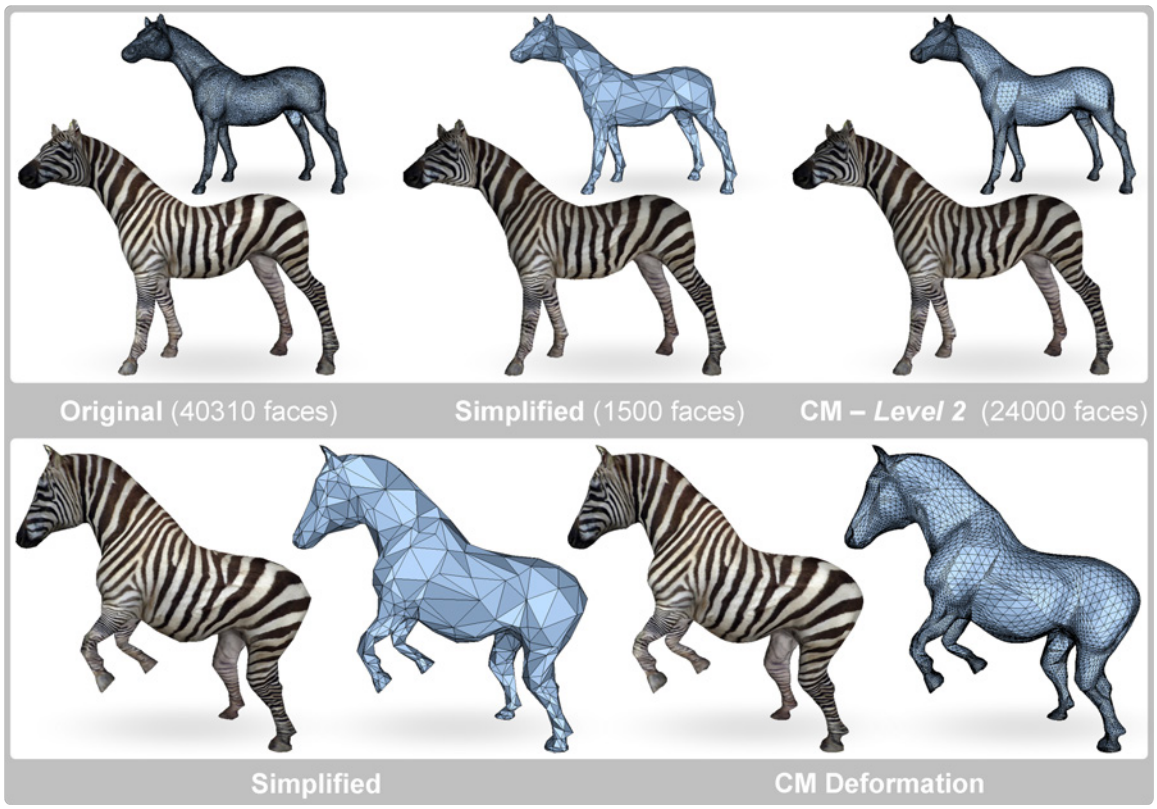

Figure 6.2: Deformation of a zebra model applied on the simplified and the CM models.

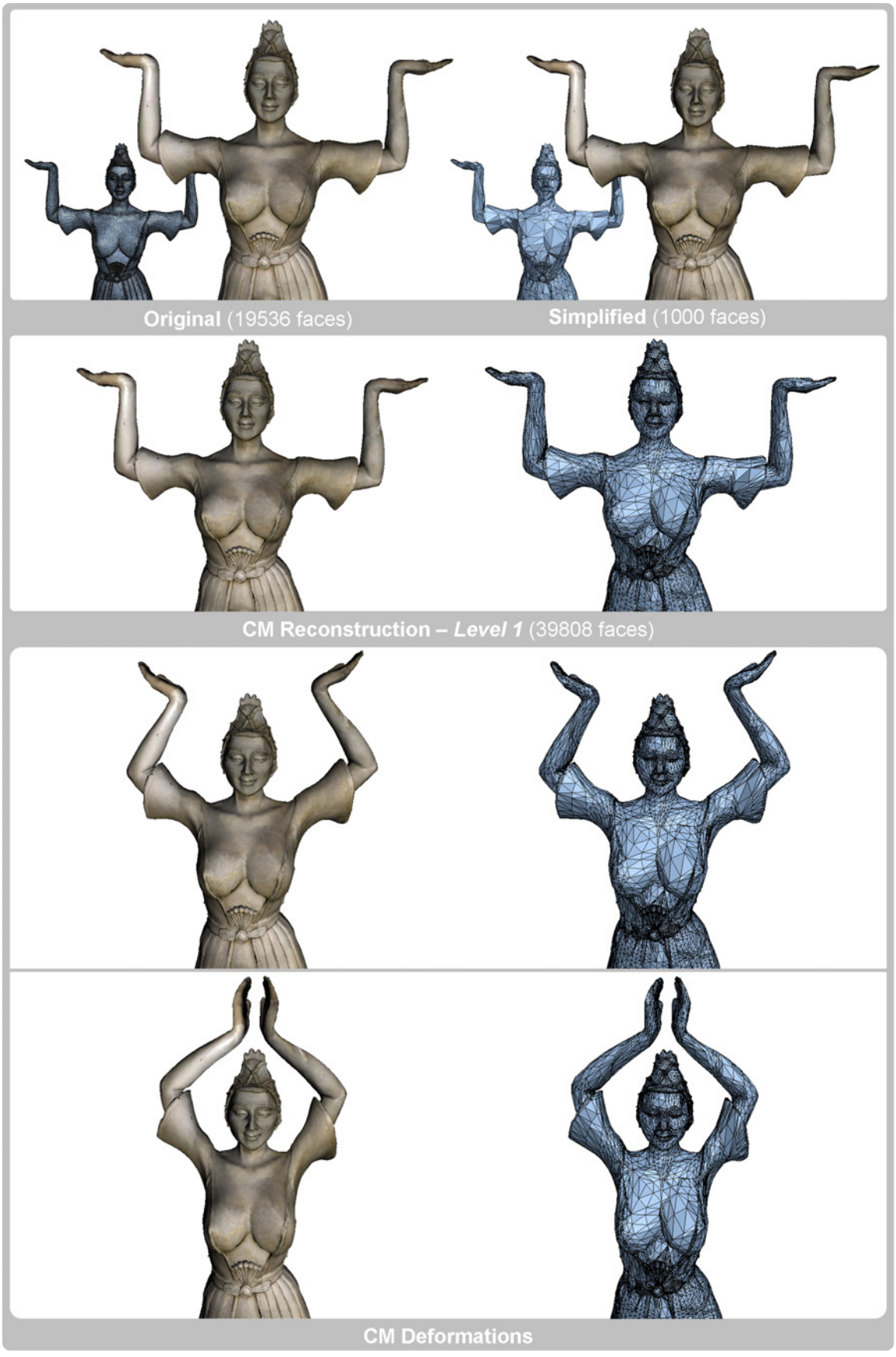

Figure 6.3: Two deformation poses of a CM of the Imperia model reconstructed at level 1.

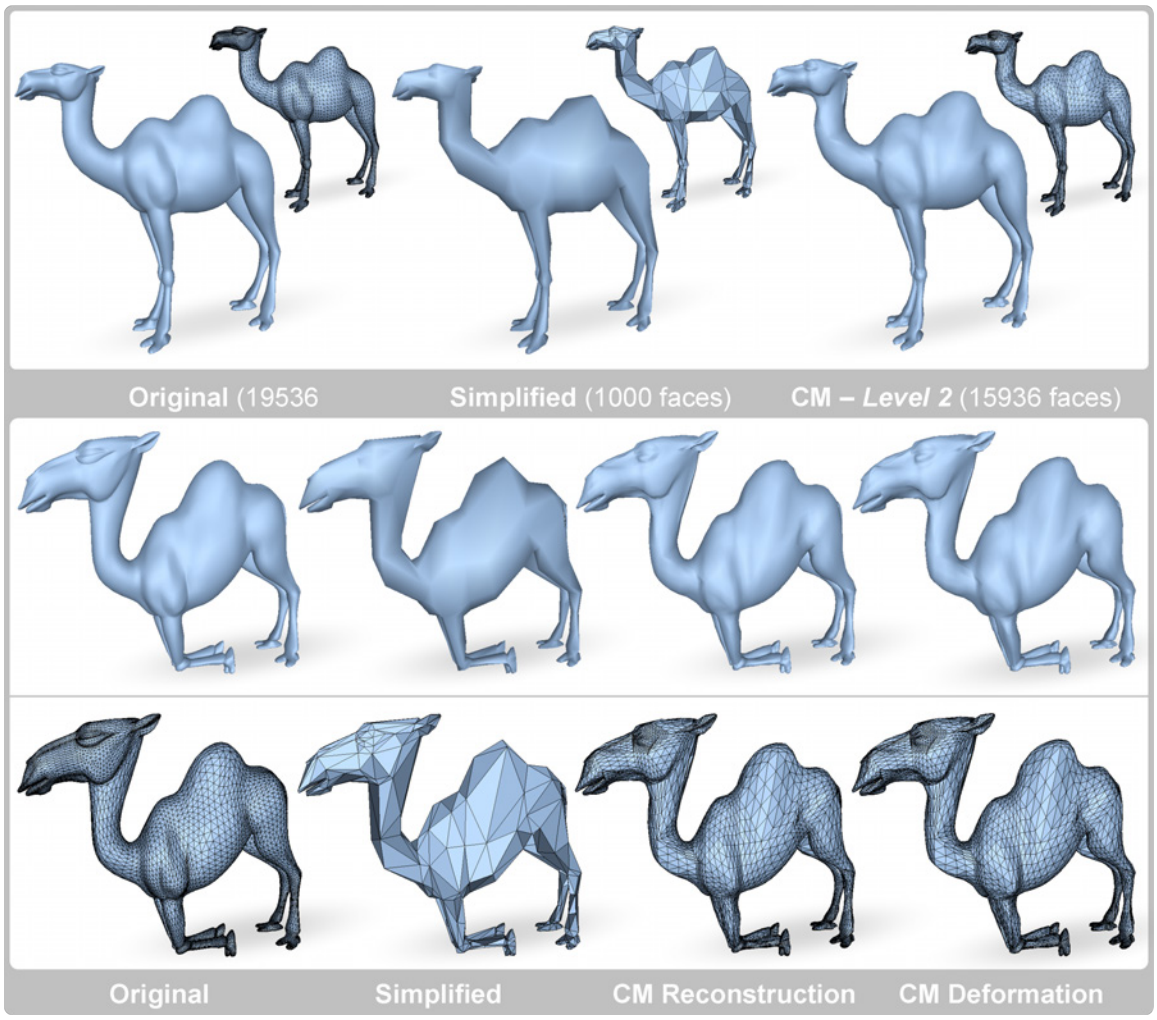

Figure 6.4: Comparison of the same deformation results applied on different Camel model versions.

applying the deformation to all the reconstructed vertices. The second one labeled *CM Deformation* corresponds to the result obtained by our proposal. The memory space required and time consumption are the main differences between them. Moreover, observe that the deformed version of the original model is too similar to the deformation of the CM which has been generated more efficiently. The cost of the CM deformation is equivalent to the deformation cost of the simplified model plus the local affinity computations that depend on the reconstruction level chosen.

After a deformation of a CM the sharp features are also preserved depending on the effects of the deformation method applied. To illustrate that, Figure 6.5 shows different deformation results of a sword model. Because the manipulation is performed over the

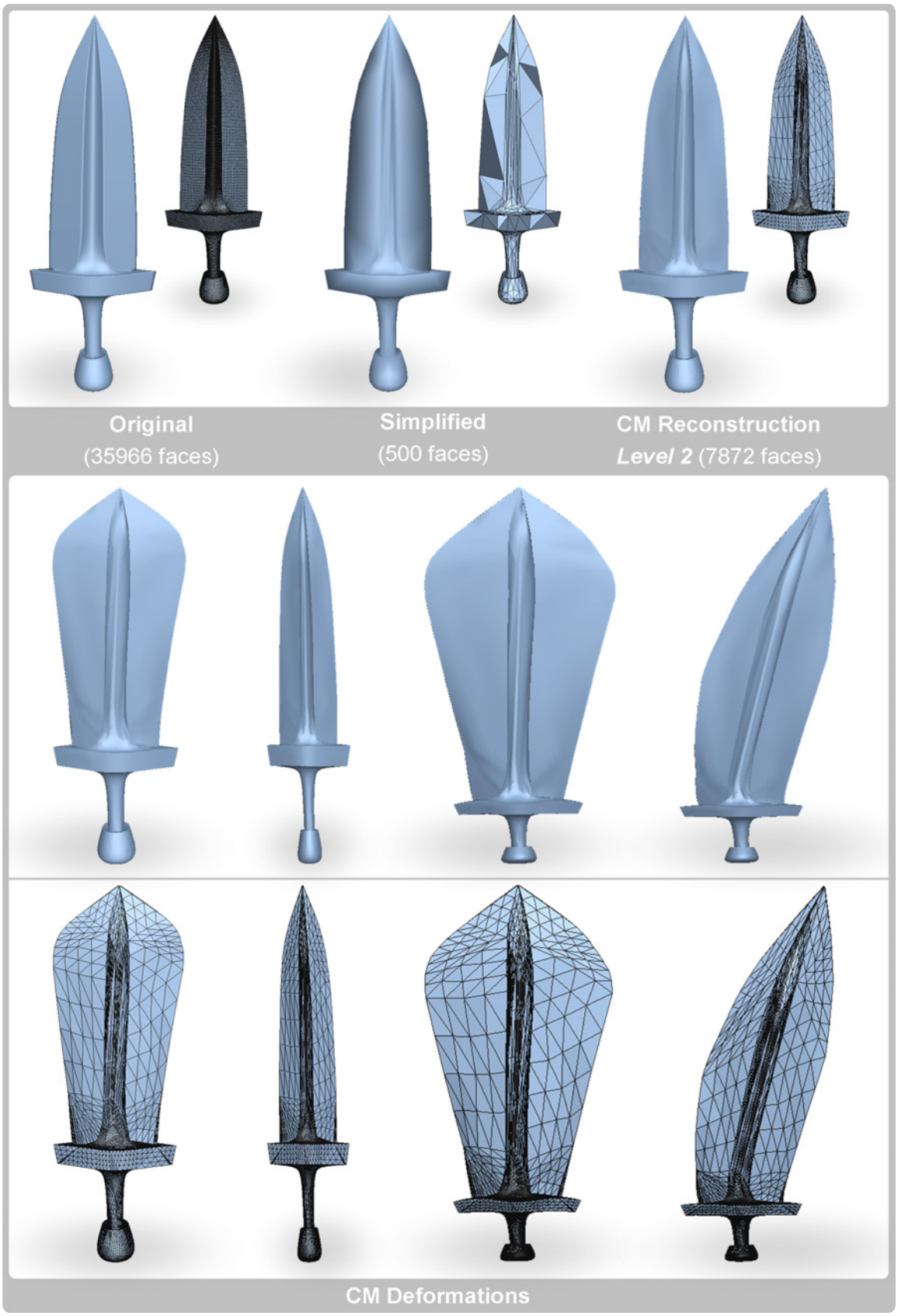

Figure 6.5: Set of deformation results with sharp-feature preservation from the CM of a sword model.

vertices of the simplified model, the accuracy of the deformation results depends on the level of detail of the CM base.

Just as with CM, the detail in the reconstruction process after a deformation can be applied adaptively according to user needs. Figure 6.6 shows how different deformed versions of the zebra model can be obtained depending on the criteria used to approximate it. In the figure, we show four different adaptive approximations: regular size edges (25% of the longest edge), two different points of interests and the silhouette according to the point of view.

Next, we perform a quantitative analysis over the Imperia and the Vase models that supposes we use a cage-based deformation approach for their manipulation. For a wider study, we show the results obtained for MVC and GC.

Table 6.1 shows the memory space occupied by the original, the simplified and the CM models for both mesh files and deformation coordinates needed. Notice that the increment of the CM with respect to the simplified version is due to the local surface coefficients stored. As the deformation is performed over the CM base, the memory space required for the deformation coordinates is the same for both simplified and CM models. The experimental results show the high reduction obtained on the memory footprint.

Table 6.2 and Table 6.3 show the computational time results obtained for the original, the simplified and the CM at two different reconstruction levels for each model. For the preprocess time we distinguish between the time required to compute the cage coordinates and the time dedicated to the reconstruction process. Observe that the reduction obtained is significant even though it takes taking into account the sum of the two values. The execution time column is the average of the time needed for the deformation due to the displacement of one cage vertex. The difference between the two columns is only noticeable in CM as it represents the amount of time required to compute the local affine transformations and the new positions of the reconstructed vertices. Therefore, the deformation cost for any CM is the same as for the simplified, and as we increment the reconstruction level the total execution time increases accordingly. Observe that deformable CMs are significantly faster than directly deforming the original model in both preprocess and execution times.

Both in memory and time results, the differences are much greater with the GC deformation approach. So, the reduction level will depend on the deformation method used.

As can be seen with some results, textured models are also supported. The experiments show how the texture is correctly preserved and applied in the deformed approximations. In this way, the results obtained are much more realistic. Thanks to the high geometric quality,

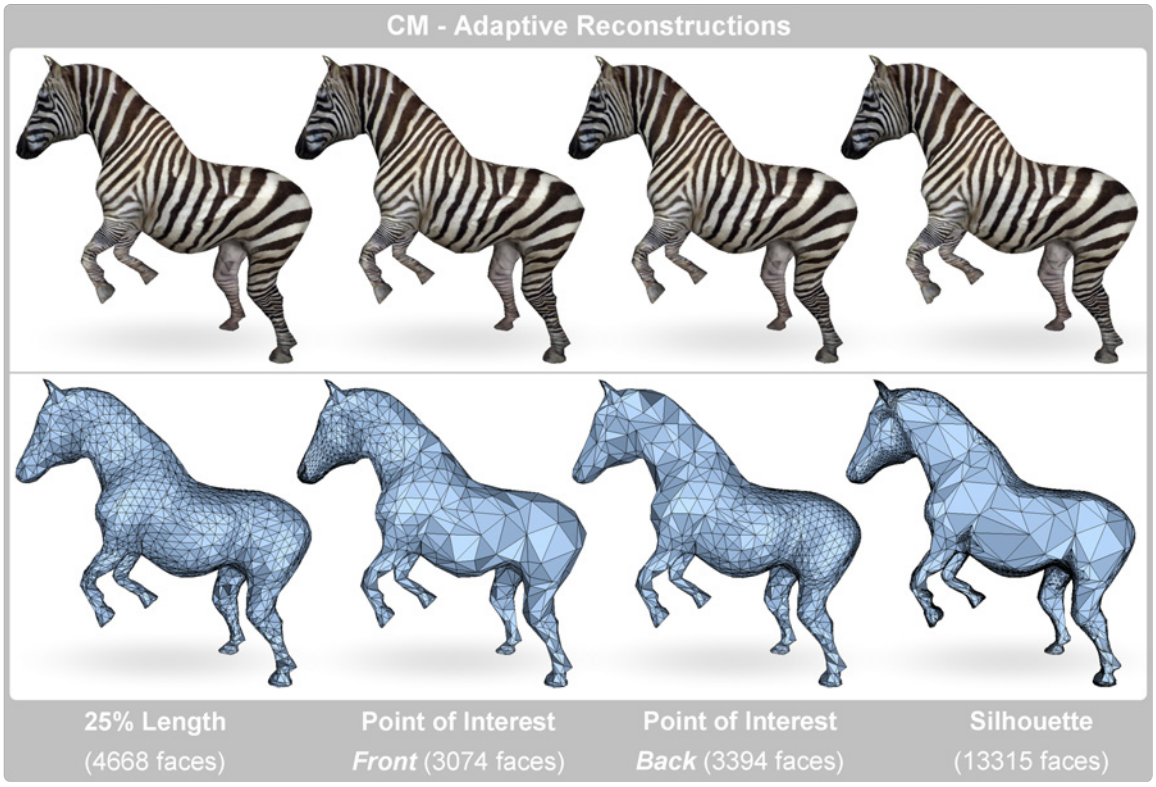

Figure 6.6: Adaptive reconstructions of a deformed zebra model.

	Model			**Cage**		**Def. Coord.** (MB)	
	Faces	Vert.	Memory (MB)	Faces	Vert.	MVC	GC
Vase							
Original	142462	71229	7.06473	36	20	10.8686	30.4323
Simplified	2848	1422	0.14112	36	20	0.21698	0.60754
CM	2848	1422	0.81929	36	20	0.21698	0.60754
Imperia							
Original	199978	9992	9.91728	116	60	45.7727	134.2666
Simplified	9996	4988	0.49517	116	60	2.2833	6.6977
CM	9996	4988	0.81929	116	60	2.2833	6.6977

Table 6.1: Memory space occupied by the original, simplified and CM Imperia and Vase models both for mesh files and for deformation coordinates (MVC and GC).

Vase		**Preprocess** (sec)		**Execution time** (sec)	
	Faces	Cage Coord.	Reconstruction	Deformation	Total
MVC					
Original	142462	3.18	-	0.22175	0.22175
Simplified	2848	0.08	-	0.00023	0.00023
CM - Level 1	11384	0.08	0.17	0.00023	0.03944
CM - Level 2	45536	0.08	0.77	0.00023	0.09602
GC					
Original	142462	25.15	-	0.78417	0.78417
Simplified	2848	0.50	-	0.01286	0.01286
CM - Level 1	11384	0.50	0.17	0.01286	0.05282
CM - Level 2	45536	0.50	0.77	0.01286	0.10685

Table 6.2: Computational times for different versions of the Vase model applying a MVC and GC deformation approach. Cage information defined in Table 6.1.

Imperia		**Preprocess** (sec)		**Execution time** (sec)	
	Faces	Cage Coord.	Reconstruction	Deformation	Total
MVC					
Original	199978	13.22	-	0.44146	0.44146
Simplified	9997	0.68	-	0.01987	0.01996
CM - Level 1	39808	0.68	0.58	0.01987	0.14198
CM - Level 2	159232	0.68	2.62	0.01987	0.34971
GC					
Original	199978	108.11	-	2.90368	2.90368
Simplified	9997	5.56	-	0.14031	0.14031
CM - Level 1	39808	5.56	0.58	0.14031	0.25779
CM - Level 2	159232	5.56	2.62	0.14031	0.48929

Table 6.3: Computational times for different versions of the Imperia model applying a MVC and GC deformation approach. Cage information defined in Table 6.1.

the correct texture adaptation and the accurate reconstructions, faithful approximations can be generated. In addition to all these good properties, the reduced memory footprint and the high performance of the computations makes deformable CM a potential tool in several application fields.

Chapter 7

Conclusions and Future Work

In this dissertation we addressed the simplification, approximation and deformation of large models, some of the main geometry processing techniques. We proposed a set of new robust and efficient techniques that represent a step forward with respect to the state of the art.

We first developed a method for the automatic simplification of a highly detailed polygonal surface model into a single faithful approximation containing fewer polygons. The method consists, on the one hand, of weighting the QEM by a local area distortion measure and, on the other hand, of bijective mappings that properly modify an index texture for each edge-collapse to avoid distortions on the appearance of the simplified mesh. In this way, the chart boundary edges are not penalized. The main benefit of this approach is that realistic and accurate approximations of complex textured models can be generated.

Then, we presented a novel technique to encode model information in order to approximate it while preserving its original shape. The method consists of two principal steps: CM generation and CM reconstruction. The first computes local surfaces at each vertex following the original shape of the model in a simplification process. The second exploits all stored information and joins it properly to obtain an accurate approximation. The method also allows sharp features to be preserved and an adaptive reconstruction to be performed. Moreover, textured models are supported, allowing the application of colour textures and normal mappings to recover all the original details. The versatility of the method combined with the simplicity of the computations makes it a very useful tool in different fields of application.

After the study of simplification and approximation techniques we dealt with the mesh editing field. In this context, we have presented $*$Cages, a multi-level (hierarchical) cage-

based system for spatial mesh deformations. It combines heterogeneous sets of coordinates, allowing the user to define different coordinates for different neighbouring cages and smoothly use them together in combination. With ∗Cages, any change the user makes in one cage is kept local to the cage being modified, not requiring recomputations for the contents of a different cage. This is the main advantage with respect to other mechanisms that try to obtain more localized deformations. Moreover, in the multi-level deformation scheme proposed, upper-level cages are used to locally control deformations in lower-level cages in the system, allowing the passage from a whole-model deformation to a very localized one. The use of multiple cages to control the mesh makes the use of local effects possible and thus the computational costs are greatly reduced with respect to traditional approaches. In general terms, ∗Cages is an extremely flexible and versatile deformation tool, which results in a much more intuitive and user-friendly approach than the current state of the art.

Finally, we combined all the knowledge acquired previously to develop a hybrid solution for the efficient deformation of large models. When meshes become large and complex, the performance of the deformation methods becomes a bottleneck of the entire system. Deformations on large meshes are desirable in applications designed to obtain real-time performance. In recent years, several deformation methods to achieve the necessary requirements for the manipulation of highly detailed models have begun to emerge. In this context, we presented deformable CMs, a truly powerful tool for modelling and editing surfaces that preserve the original surface details efficiently. Giving the CM the ability to be deformable, we allow any of the existing deformation techniques to be applied in large models. The deformations are transferred to the approximation by computing and blending local transformations at each vertex of the deformed CM base. In this simple manner, the resulting representation provides flexibility to the results obtained since, as with CM, textured models are supported, sharp features are preserved and adaptive approximations can be generated. Therefore, faithful approximations at different poses of highly detailed models are obtained with a reduced memory footprint and a high performance.

7.1 Future work

After this thesis, several new avenues for future research are open. Next, we summarize some of them.

- In the simplification context, extending our approach to obtain a progressive mesh [Hop96, SSGH01] of the model with all the intermediate meshes sharing a common

texture parameterization could be significant. Because the bijective mappings can be inverted, our simplification technique could be applied on the fly while rendering a change in the level of detail. However, we believe that this strategy can only be applied for simple models. Consequently, dealing with progressive meshes on large models will be an important part of our future work.

- Study the feasibility of integrating the simplification of the scene objects on the GPU, thus simplifying the stored information needed for data structures while preserving the global shape. The key point to performing the simplification process under the GPU consists of the parallelization of the edge-collapses. In this manner, each edge-collapse would be carried out independently of the rest of the collapses during the simplification. Moreover, the texture, the appearance attributes and the sharp features of the model would be preserved.

- In order to provide more flexibility to the CM when dealing with models possessing sharp features, we could define more kinds of local surfaces. In this manner, we could better approximate the surfaces by using less information.

- Taking into account the techniques developed for the CMs, a new mesh smoothing approach could be studied.

- After studying deformation techniques we thought about the possibility of developing a novel parameterized mesh deformation approach which would allow a set of deformed models to be obtained, taking into account semantic attributes. In addition, we could also define a set of restrictions to limit the deformation in favour of more "meaningful" ones. Most of the current approaches use a database of manually parameterized scanned models to obtain different versions of them. In contrast, we would automatically generate the required models without the need for any kind of database, thus avoiding manual fittings and registrations that could introduce surface artefacts. As a consequence, the generation of crowds would be easier, faster and more user-friendly than the current techniques.

- Current space deformation techniques require the construction of a cage or other 3D structures around the entire manipulated model or part of it. Both the construction and the manipulation of the subject model are limited to the geometry and flexibility of the cage. The problem of automatic cage generation is an open issue that is worth further study. A framework which would allow the construction of suitable cages without the need for user interaction is an interesting field of study.

- The use of ∗Cages as a way to generate some sort of mesh deformation compression by gathering crucial information from the model over the cages could be another interesting possibility to study. Also, along the same line of thought, we could re-use information gathered on cages for different, but similar models using the same set of cages. In this way, we would be able to transfer deformations between different models, which is usually quite a complex task.

- Several steps of the algorithms presented are completely parallelizable: both the CM generation and the CM reconstruction, in the case of CM, or local transformations for deformable CMs. The parallelization of the computations related to the deformations depends on the deformation method used. Taking advantage of the programmable capabilities of the graphics hardware, we would improve (significantly decrease) the computation time of such processes.

Bibliography

[ABCO+03] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9:3–15, 2003.

[Ale03] M. Alexa. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, V19(2):105–114, May 2003.

[And99] C. Andújar. *Octree-based Simplification of Polyhedral Solids.* PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, June 1999.

[BCWG09] M. Ben-Chen, O. Weber, and C. Gotsman. Variational harmonic maps for space deformation. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–11, New York, NY, USA, 2009. ACM.

[BH96] F. J. Bossen and P. S. Heckbert. A pliant method for anisotropic mesh generation. In *5th International Meshing Roundtable*, pages 63–74, Oct. 1996.

[BHZN10] P. Borosan, R. Howard, S. Zhang, and A. Nealen. Hybrid mesh editing. In *to appear in Proceedings of Eurographics 2010 (short papers)*, 2010.

[BK04] M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics*, 23:630–634, August 2004.

[Bli78] J. F. Blinn. Simulation of wrinkled surfaces. *SIGGRAPH Computer Graphics*, 12:286–292, August 1978.

[BPR+06] M. Botsch, M. Pauly, C. Rossl, S. Bischoff, and L. Kobbelt. Geometric modeling based on triangle meshes. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[BS88] A. A. Ball and D. J. T. Storry. Conditions for tangent plane continuity over recursively generated B-spline surfaces. *ACM Transactions on Graphics*, 7:83–102, April 1988.

[BS08] M. Botsch and O. Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):213–230, 2008.

[BVI91] C. Bennis, J. M. Vézien, and G. Iglésias. Piecewise surface flattening for non-distorted texture mapping. *SIGGRAPH Computer Graphics*, 25:237–246, July 1991.

[CC78] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-aided Design*, 10:350–355, 1978.

[CC06] C.-C. Chen and J.-H. Chuang. Texture adaptation for progressive meshes. *Computer Graphics Forum*, 25(3):343–350, September 2006.

[CCC87] R. L. Cook, L. Carpenter, and E. Catmull. The Reyes image rendering architecture. *SIGGRAPH Computer Graphics*, 21:95–102, August 1987.

[CCC+08] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: an open-source mesh processing tool. In *Sixth Eurographics Italian Chapter Conference*, pages 129–136, 2008.

[CCR08] P. Cignoni, M. Corsini, and G. Ranzuglia. MeshLab: an open-source 3D mesh processing system. *ERCIM News*, 73:45–46, April 2008.

[CO09] D. Cohen-Or. Space deformations, surface deformations and the opportunities in-between. *Journal of Computer Science & Technology*, 24:2–5, January 2009.

[COM98] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 115–122, New York, NY, USA, 1998. ACM.

[Coo84] R. L. Cook. Shade trees. *SIGGRAPH Computer Graphics*, 18:223–231, January 1984.

[Coq90] S. Coquillart. Extended free-form deformation: a sculpturing tool for 3D geometric modeling. *SIGGRAPH Computer Graphics*, 24:187–196, September 1990.

[CSAD04] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3):905–914, 2004.

[CSCF07] P. Castelló, M. Sbert, M. Chover, and M. Feixas. Viewpoint entropy-driven simplification. In *International Conference in Central Europe on Computer Graphics and Visualization*, 2007.

[CSCF08a] P. Castelló, M. Sbert, M. Chover, and M. Feixas. Viewpoint-based simplification using f-divergences. *Information Sciences*, 178:2375–2388, June 2008.

[CSCF08b] P. Castelló, M. Sbert, M. Chover, and M. Feixas. Viewpoint-driven simplification using mutual information. *Computers & Graphics*, 32(3):451–463, 2008.

[CVM+96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. *Computer Graphics*, 30(Annual Conference Series):119–128, 1996.

[CWQ+07] K. D. Cheng, W. Wang, H. Qin, K. K. Wong, H. Yang, and Y. Liu. Design and analysis of optimization methods for subdivision surface fitting. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):878–890, 2007.

[dBvKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, 1997.

[DEGN99] T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev. Topology preserving edge contraction. *Publications de l'Institut Mathématique (Beograd)*, 66:3–45, 1999.

[DH07] M. Dehn and P. Heegaard. *Analysis situs*, volume III.1.1. Enzyklopäedie d. Math. Wiss., 1907.

[DHOS07] J. II Daniels, L. K. Ha, T. Ochotta, and C. T. Silva. Robust smooth feature extraction from point clouds. In *SMI '07: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007*, pages 123–136, Washington, DC, USA, 2007. IEEE Computer Society.

[DKT98] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 85–94, New York, NY, USA, 1998. ACM.

[DLG90] N. Dyn, D. Levine, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9:160–169, April 1990.

[DS98] D. Doo and M. Sabin. *Behaviour of recursive division surfaces near extraordinary points*, pages 177–181. ACM, New York, NY, USA, 1998.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics*, 29(Annual Conference Series):173–182, 1995.

[EM99] C. Erikson and D. Manocha. GAPS: general and automatic polygonal simplification. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, I3D '99, pages 79–88, New York, NY, USA, 1999. ACM.

[FCOS05] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics*, 24(3):544–552, 2005.

[FH01] M. S. Floater and K. Hormann. Parameterization of triangulations and unorganized points. In *Principles of Multiresolution in Geometric Modelling*, pages 127–154. Springer, 2001.

[FH05] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 157–186. Springer Verlag, 2005.

[FK03] R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Professional, 2003.

[FKR05] M. S. Floater, G. Kós, and M. Reimers. Mean value coordinates in 3D. *Computer Aided Geometric Design*, 22(7):623–631, 2005.

[Flo03] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[FS93] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 247–254, New York, NY, USA, 1993. ACM.

[FST92] T. A. Funkhouser, C. H. Séquin, and S. J. Teller. Management of large amounts of data in interactive building walkthroughs. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, I3D '92, pages 11–20, New York, NY, USA, 1992. ACM.

[GH97] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics (Proceedings of SIGGRAPH 1997)*, 31(Annual Conference Series):209–216, 1997. http://graphics.cs.uiuc.edu/ garland/software/qslim.html.

[GH98] M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *IEEE Visualization '98*, pages 263–270, 1998.

[GP08] I. García and G. Patow. IGT: inverse geometric textures. *ACM Transactions on Graphics*, 27(5):1–9, 2008.

[GSS99] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 325–334, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[GVSS00] I. Guskov, K. Vidimče, W. Sweldens, and P. Schröder. Normal meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 95–102, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[GWM01] S. Gumhold, X. Wang, and R. Macleod. Feature extraction from point clouds. In *Proceedings of the 10 th International Meshing Roundtable*, pages 293–305, 2001.

[HCLB09] J. Huang, L. Chen, X. Liu, and H. Bao. Efficient mesh deformation using tetrahedron control mesh. *Computer Aided Geometric Design*, 26(6):617–626, 2009.

[HDD+93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Computer Graphics*, 27(Annual Conference Series):19–26, 1993.

[HDD+94] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction.

In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 295–302, New York, NY, USA, 1994. ACM.

[HG99] P. S. Heckbert and M. Garland. Optimal triangulation and quadric-based surface simplification. *Computational Geometry*, 14(1-3):49–65, 1999.

[HG01] A. Hubeli and M. Gross. Multiresolution feature extraction for unstructured meshes. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 287–294, Washington, DC, USA, 2001. IEEE Computer Society.

[HLS07] K. Hormann, B. Lévy, and A. Sheffer. Mesh parameterization: Theory and practice. In *ACM SIGGRAPH Course Notes*, 2007.

[Hop96] H. Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 99–108, New York, NY, USA, 1996. ACM.

[Hop99] H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *VISUALIZATION '99: Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*, pages 59–66, Washington, DC, USA, 1999. IEEE Computer Society.

[HPW05] K. Hildebrandt, K. Polthier, and M. Wardetzky. Smooth feature lines on surface meshes. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, pages 85–90, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[HSL+06] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum. Subspace gradient domain mesh deformation. *ACM Transactions on Graphics*, 25:1126–1134, July 2006.

[HWW+06] Y. He, K. Wang, H. Wang, X. Gu, and H. Qin. Manifold T-spline. In *Proceedings of Geometric Modeling and Processing*, pages 409–422, 2006.

[JCNH06] X. Jiao, A. Colombi, X. Ni, and J. C. Hart. Anisotropic mesh adaptation for evolving triangulated surfaces. In *Proceedings, 15th International Meshing Roundtable, Springer-Verlag*, pages 173–190, September 2006.

[JMD+07] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. *ACM Transactions on Graphics*, 26(3):71, 2007.

[JSW05] T. Ju, S. Schaefer, and J. D. Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, 2005.

[JZvdP+08] T. Ju, Q.-Y. Zhou, M. van de Panne, D. Cohen-Or, and U. Neumann. Reusable skinning templates using cage-based deformations. *ACM Transactions on Graphics*, 27(5):1–10, 2008.

[KCVS98] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 105–114, New York, NY, USA, 1998. ACM.

[Ket99] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry: Theory and Applications*, 13:65–90, May 1999.

[KG03] Y. Kho and M. Garland. User-guided simplification. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 123–126, New York, NY, USA, 2003. ACM Press.

[KL96] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 313–324, New York, NY, USA, 1996. ACM.

[Kob96] L. Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Computer Graphics Forum*, pages 409–420, 1996.

[KSG03] V. Kraevoy, A. Sheffer, and C. Gotsman. Matchmaker: constructing constrained texture maps. *ACM Transactions on Graphics*, 22:326–333, July 2003.

[LBS08] T. Langer, A. Belyaev, and H.-P. Seidel. Mean value Bézier maps. In *GMP'08: Proceedings of the 5th international conference on Advances in geometric modeling and processing*, pages 231–243, Berlin, Heidelberg, 2008. Springer-Verlag.

[Lef49] S. Lefschetz. *Introduction to Topology.* Princeton University Press, Princeton, New Jersey, 1949.

[Lev03] D. Levin. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization*, 3:37–49, 2003.

[LJFW08] J. Lin, X. Jin, Z. Fan, and C. C. L. Wang. Automatic PolyCube-Maps. In *GMP*, pages 3–16, 2008.

[LKCOL07] Y. Lipman, J. Kopf, D. Cohen-Or, and D. Levin. GPU-assisted positive mean value coordinates for mesh deformations. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 117–123, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[LLCO08] Y. Lipman, D. Levin, and D. Cohen-Or. Green coordinates. *ACM Transactions on Graphics*, 27:78:1–78:10, August 2008.

[LLD+10] Z. Li, D. Levin, Z. Deng, D. Liu, and X. Luo. Cage-free local deformations using green coordinates. *Visual Computer*, 26(6-8):1027–1036, 2010.

[LMH00] A. Lee, H. Moreton, and H. Hoppe. Displaced subdivision surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 85–94, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Loo87] C. Loop. Smooth Subdivision Surfaces Based on Triangles. Department of mathematics, University of Utah, Utah, USA, Aug 1987.

[LPRM02] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21:362–371, July 2002.

[LS10] E. Landreneau and S. Schaefer. Poisson-based weight reduction of animated meshes. *Comput. Graph. Forum*, 29(6):1945–1954, 2010.

[LSLCO05] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Transactions on Graphics*, 24:479–487, July 2005.

[LT98] P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualization*, pages 279–286, 1998.

[LT00] P. Lindstrom and G. Turk. Image-driven simplification. *ACM Transactions on Graphics*, 19(3):204–241, 2000.

[LVJ05] C. H. Lee, A. Varshney, and D. Jacobs. Mesh Saliency. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24, No. 3:659–666, 2005.

[LWC+02] D. Luebke, B. Watson, J. D. Cohen, M. Reddy, and A. Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[LWY08] R. Ling, W. Wang, and D.-M. Yan. Fitting sharp features with loop subdivision surfaces. *Computer Graphics Forum*, 27(5):1383–1391, 2008.

[M̈87] M. Mäntylä. *An introduction to solid modeling*. Computer Science Press, Inc., New York, NY, USA, 1987.

[Ma05] W. Ma. Subdivision surfaces for CAD-an overview. *Computer Aided Design*, 37(7):693–709, 2005.

[MK05] M. Marinov and L. Kobbelt. Optimization methods for scattered data approximation with subdivision surfaces. *Graphical Models*, 67(5):452–473, 2005.

[MMTP04] W. Ma, X. Ma, S. K. Tso, and Z. Pan. A direct approach for subdivision surface fitting from a dense triangle mesh. *Computer Aided Design*, 36(6):525–536, 2004.

[Moo] Mootools. Polygon Cruncher. *http://www.mootools.com*.

[MRB05] I. Martin, R. Ronfard, and F. Bernardini. Detail-preserving variational surface design with multiresolution constraints. *Journal of Computing and Information Science in Engineering*, 5(2):104–110, June 2005.

[MSW+09] W. Meng, B. Sheng, S. Wang, H. Sun, and E. Wu. Interactive image deformation using cage coordinates on GPU. In *VRCAI '09: Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 119–126, New York, NY, USA, 2009. ACM.

[MYV93] J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 27–34, New York, NY, USA, 1993. ACM.

[NM95] A. Narkhede and D. Manocha. Fast polygon triangulation based on Seidel's algorithm. In *Graphics Gems V*, pages 394–397, Boston, 1995. Academic Press.

[OBA+03] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics*, 22(3):463–470, 2003.

[OBA05] Y. Ohtake, A. Belyaev, and M. Alexa. Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, pages 149–158, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[O'N66] B. O'Neill. *Elementary Differential Geometry.* Academic Press, NYC, USA, 1966. 411 pages.

[PR98] J. Peters and U. Reif. Analysis of algorithms generalizing B-spline subdivision. *SIAM Journal on Numerical Analysis*, 35:728–748, April 1998.

[Pre75] P. M. Prenter. *Splines and Variational Methods.* John Wiley & Sons, New York, 1975.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry. An Introduction.* Springer, New York, 1985.

[PSC06] I. Park, S. Shirani, and D. W. Capson. Mesh simplification using an area-based distortion measure. *Mathematical Modelling and Algorithms*, 5(3):309–329, 2006.

[QMV98] H. Qin, C. Mandal, and B. C. Vemuri. Dynamic Catmull-Clark subdivision surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 4:215–229, July 1998.

[RB93] J. Rossignac and P. Borrel. Multiresolution 3D aproximations for rendering complex scenes. *Modeling in Computer Graphics*, 1993.

[RT98] Y. Rubner and C. Tomasi. Texture metrics. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 4601 – 4607, 1998.

[SA04] M. Segal and K. Akeley. The OpenGL graphics system: A specification (version 2.0), October 2004.

[SB09] O. Sorkine and M. Botsch. Tutorial: Interactive shape modeling and deformation. In *Eurographics*, 2009.

[Sch96] J. E. Schweitzer. *Analysis and application of subdivision surfaces.* PhD thesis, University of Washington, 1996. AAI9704545.

[SCO04] O. Sorkine and D. Cohen-Or. Least-squares meshes. In *Proceedings of Shape Modeling International*, pages 191–199. IEEE Computer Society Press, 2004.

[SCOGL02] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski. Bounded-distortion piecewise mesh parameterization. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 355–362, Washington, DC, USA, 2002. IEEE Computer Society.

[SCOL+04] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '04, pages 175–184, New York, NY, USA, 2004. ACM.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1(1):51–64, 1991.

[SLMB05] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov. ABF++: fast and robust angle based flattening. *ACM Transactions on Graphics*, 24:311–330, April 2005.

[SMT00] H. Seo and N. Magnenat-Thalmann. LoD management on animating face models. In *Proceedings of the IEEE Virtual Reality 2000 Conference*, VR '00, pages 161–168, Washington, DC, USA, 2000. IEEE Computer Society.

[SP86] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Computer Graphics*, 20:151–160, August 1986.

[SSGH01] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 409–416, New York, NY, USA, 2001. ACM.

[SSP07] R. W. Sumner, J. Schmid, and M. Pauly. Embedded deformation for shape manipulation. *ACM Transactions on Graphics*, 26:80:1–80:7, July 2007.

[ST67] I. M. Singer and J. A. Thorpe. *Lecture Notes on Elementary Topology and Geometry*. Scott, Foresman and Company, Glenview, Illinois, 1967.

[SWG+03] P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. Multi-chart geometry images. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 146–155, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[SYBF06] L. Shi, Y. Yu, N. Bell, and W.-W. Feng. A fast multigrid algorithm for mesh deformation. *ACM Transactions on Graphics*, 25:1108–1117, July 2006.

[TCS03] M. Tarini, P. Cignoni, and R. Scopigno. Visibility based methods and assessment for detail-recovery. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 457–464, Washington, DC, USA, 2003. IEEE Computer Society.

[THCM04] M. Tarini, K. Hormann, P. Cignoni, and C. Montani. PolyCube-Maps. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 853–860, New York, NY, USA, 2004. ACM.

[Tur92] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics (Proceedings of SIGGRAPH 1992)*, 26(2):55–64, 1992.

[WBCG09] O. Weber, M. Ben-Chen, and C. Gotsman. Complex barycentric coordinates with applications to planar shape deformation. *Computer Graphics Forum (Proceedings of Eurographics)*, 28(2):587–597, 2009.

[WHL+08] H. Wang, Y. He, X. Li, X. Gu, and H. Qin. Polycube splines. *Computer Aided Design*, 40(6):721–733, 2008.

[WK05] J. Wu and L. Kobbelt. Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum*, 24(3):277–284, 2005.

[XLG09] C. Xian, H. Lin, and S. Gao. Automatic generation of coarse bounding cages from dense meshes. In *Proceedings of IEEE Shape Modeling International*, pages 21–27, 2009.

[XWXC08] K Xu, Y.-Z. Wang, Y. Xiong, and Z.-Q. Cheng. Interactive shape manipulation based on space deformation with harmonic-guided clustering. In *Proceedings of Computer Animation and Social Agent 2008*, 2008.

[YBS05] S. Yoshizawa, A. Belyaev, and H.-P. Seidel. Fast and robust detection of crest lines on meshes. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 227–232, New York, NY, USA, 2005. ACM.

[YLW06] D.-M. Yan, Y. Liu, and W. Wang. Quadric surface extraction by variational shape approximation. In M.-S. Kim and K. Shimada, editors, *GMP*, volume 4077 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2006.

[YZX+04] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics*, 23:644–651, August 2004.

[ZGM09] M. Zhihong, C. Guo, and Z. Mingxi. Robust detection of perceptually salient features on 3D meshes. *Visual Computer*, 25(3):289–295, 2009.

[ZHS+05] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum. Large mesh deformation using the volumetric graph Laplacian. *ACM Transactions on Graphics*, 24:496–503, July 2005.

[ZHX+07] K. Zhou, X. Huang, W. Xu, B. Guo, and H.-Y. Shum. Direct manipulation of subdivision surfaces on GPUs. *ACM Transactions on Graphics*, 26:91:1–91:9, July 2007.

[Zor98] D. Zorin. *Stationary subdivision and multiresolution surface representations.* PhD thesis, California Institute of Technology, Pasadena, CA, USA, 1998. AAI9842343.

[Zor00] D. Zorin. A method for analysis of C1-Continuity of subdivision surfaces. *SIAM Journal on Numerical Analysis.*, 37:1677–1708, May 2000.

[ZRKS05] R. Zayer, C. Rössl, Z. Karni, and H.-P. Seidel. Harmonic guidance for surface deformation. *Computer Graphics Forum*, 24:601–609, 2005.

[ZS00] D. Zorin and P. Schröder. Subdivision for Modeling and Animation. Technical report, SIGGRAPH 2000, 2000. Course Notes.

[ZSGS04] K. Zhou, J. Synder, B. Guo, and H.-Y. Shum. Iso-charts: stretch-driven mesh parameterization using spectral analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '04, pages 45–54, New York, NY, USA, 2004. ACM.

[ZSS96] D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 189–192, New York, NY, USA, 1996. ACM.

[ZSS97] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 259–268, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[ZT02] E. Zhang and G. Turk. Visibility-guided simplification. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 267–274, Washington, DC, USA, 2002. IEEE Computer Society.

[ZWT+05] K. Zhou, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum. TextureMontage. *ACM Transactions on Graphics*, 24:1148–1155, July 2005.