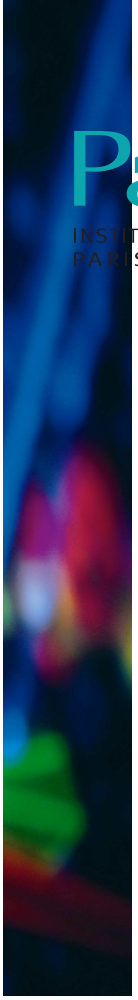


ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

EDITE - ED130

TELECOM ParisTech

2014-ENST-0006

Doctorat ParisTech

THÈSE

pour obtenir le grade de docteur délivré par

Télécom ParisTech

Spécialité : Signal et Images

Présentée et soutenue publiquement par

Emilie GUY

le 29 janvier 2016

Interactions Computationnelles pour la Modélisation de Formes 3D

Jury :

M. Marc ALEXA, Professeur, TU Berlin	Rapporteur
M. Loïc BARTHE, Professeur, Université Paul Sabatier	Rapporteur
M. Maks OVSJANIKOV, Maître de Conférences, LIX, Polytechnique	Examinateur
Mme. Catherine PELACHAUD, Directrice de Recherche CNRS, Télécom ParisTech	Examinatrice
M. Tamy BOUBEKEUR, Professeur, CNRS-LTCI, Télécom ParisTech	Directeur

THÈSE

TELECOM ParisTech
École de l'Institut Mines-Télécom - Membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

Interactions Computationnelles pour la Modélisation de Formes 3D

Computational Interactions for 3D Modeling

Emilie Guy

REMERCIEMENTS

Et voilà la fameuse page des remerciements... La page insolite qui arrive en premier alors qu'on l'écrit bien après toutes les autres!

Tout d'abord je tiens à remercier Tamy Boubekeur, mon directeur, pour avoir cru en moi dès le début et pour m'avoir soutenue pendant ces trois ans. Merci pour toutes ces discussions passionnantes qui m'ont souvent poussée à me dépasser et à ouvrir mes perspectives. Merci pour les bons moments passés aux conf' ou à la butte !

Je tiens aussi à remercier Pr. Marc Alexa et Pr. Loïc Barthe pour avoir accepté de relire mon manuscrit, ainsi que Pr. Catherine Pelachaud et Dr Maks Ovsjanikov pour leur remarques, leur questions, et pour m'avoir fait l'honneur d'assister à ma soutenance.

Merci à Lionel qui m'a fait découvrir et aimer le monde de la recherche; à Rinat et Eric pour ces supers projets à Toronto; à Christian, Alex, Gene et Lee pour m'avoir si bien encadrée et accueillie lors de mes différents stages, à Nicolas pour les discussions passionnantes et toute la gentillesse.

Un grand merci à mes collègues et amis. Jean-Marc et Noura, pour m'avoir si bien guidée dès mon arrivée, je n'en serais surement pas là sans vous. Merci énormément à Stéphane, Malik, Hélène et Gilles pour tous les bons moments dans le bureau; Pooran et Julien pour votre aide et votre gentillesse. Thierry, Charline, Leila, Sonia, Cécilia, Alasdair, Baptiste, Guillaume... pour les discussions, les cafés/thés, les conseils et bien sûr nos voyages à la butte.

Merci à tous les copains, de Grenoble, Toulouse, Labège, Paris, Rennes.. pour tous les bons moments, les discussions, les vacances! Merci Mathieu.

Je tiens aussi à remercier Alex, mon presque-colloc' et pote de thèse, c'était chouette de partager tout ça avec toi. Continue à rayonner d'énergie, aux Canaries ou ailleurs!

Un immense merci à ma famille. Le noyau dur: maman, papa, Manon.. il n'y a pas de mots pour vous remercier, c'est si bon de vous avoir. Mais aussi merci à Gisèle, Martine, Henry, Eveu, Phil (x2), Vero, Fred, Sabine, Claude, Marine, Pierre, Pauline, Victor... et puis à ma famille de cœur Catherine, Philippe, Simon, Marie, Elo, Pierre, Aod. C'est incroyable d'avoir la chance de vous avoir tous à mes cotés.

Enfin, le meilleur pour la fin. Un grand merci à Martin pour m'avoir accompagnée dans cette aventure et bien au delà. Tu m'as appris tant d'autres choses avec tes propres projets. Merci de croire en moi encore et toujours, et de m'insuffler toute cette énergie positive. Merci d'avoir partagé mes danses de la joie et mes moments de doutes. Prêt pour une nouvelle aventure?

ABSTRACT

The democratization of 3D scanners, 3D printers and virtual reality head-mounted displays has enlarged the spectrum of 3D modeling systems, making them accessible to a wider range of users, from novice to expert. To accommodate this broad range of usages, modeling tools have to set up intuitive and simple interactions, while leaving enough freedom to the users. In this manuscript we propose a set of techniques which allow the intuitive and interactive processing of complex virtual objects, often made of thousands or millions of primitives.

To this end, we introduce the concept of *computational interaction*, where shape analysis and user interactions are interleaved and are of mutual benefit. The idea is to redirect the computational power, usually dedicated to long off-line shape analysis processes, toward high level user interactions, that alleviate some of the inherent ambiguities of shape analysis methods. We analyze shapes on-the-fly to create new modeling applications with high-level user interactions. In this context, we focus on two different aspects of shape analysis, namely shape abstraction and similarity detection. The former aims at finding the principal characteristics of an object, while the latter aims at detecting and quantifying similarities between models.

In the first part of this manuscript, we use shape abstractions to control interactive processes. More precisely, the user body is first abstracted into a skeleton to navigate in a 3D scene using simple gestures. We then propose a new shape simplification method that works even at extreme simplification levels. We show that this abstraction is an efficient multi-resolution control structure for the deformation of 3D models, that is automatically built. In the second part of this manuscript, we focus on the detection of shape similarities. We propose to leverage the self-similarities naturally present in a shape to construct a new modeling application. Our interactive system is designed to help the user select and edit similar regions on a shape, the processing being automatically transposed to similar regions. Finally we introduce a method that combines both shape abstraction and similarity detection. The high-level non local knowledge coming from similarity detection is used to drive a local simplification process. We aim at building a multi-scale shape abstraction that inherently embeds the similarities of the original model.

In this manuscript we present complementary contributions to compose a 3D modeling ecosystem around high-level user interactions.

RÉSUMÉ EN FRANÇAIS

La démocratisation des scanners, des imprimantes 3D et des casques de réalité virtuelle a permis d'étendre le spectre des systèmes de modélisation 3D, les rendant accessibles aussi bien à des utilisateurs novices qu'experts. Pour s'adapter à cette grande variété d'usages, les systèmes doivent mettre en place des interactions à la fois simples et intuitives, tout en laissant une grande liberté aux utilisateurs. Dans ce manuscrit, on propose un ensemble de techniques permettant le traitement intuitif et interactif d'objets virtuels de plus en plus complexes, souvent composés de milliers ou de millions de primitives.

Pour cela on introduit le concept *d'interactions computationnelles*, où l'analyse de forme et les interactions utilisateur sont utilisées de concert et sont mutuellement bénéfiques. L'idée est de rediriger la puissance de calcul, généralement dédiée à une analyse de formes coûteuse et précalculée, vers des interactions utilisateurs de haut-niveau permettant ainsi de résoudre certaines ambiguïtés inhérentes à l'analyse automatique d'objets. On propose de créer de nouveaux outils de modélisation géométrique avec des interactions utilisateur de haut niveau grâce à l'analyse interactive de modèles 3D. Dans ce cadre, on se penche sur deux facettes de l'analyse géométrique que sont l'abstraction de forme et la détection de similarités. La première permet de révéler les caractéristiques principales d'un objet et d'en comprendre l'essence, alors que la seconde cherche à identifier et à quantifier des similitudes entre des modèles.

Dans la première partie de ce manuscrit, on propose d'utiliser des abstractions spatiales de formes pour contrôler des processus interactifs. Plus précisément, l'abstraction du corps de l'utilisateur sous forme de squelette permet de contrôler la navigation dans une scène 3D grâce à des mouvements simples. On présente par la suite une nouvelle technique de simplification de formes permettant des simplifications extrêmes. Cette méthode est employée comme une nouvelle structure de contrôle multirésolution et haut niveau pour la déformation d'objets 3D. Dans la seconde partie de ce manuscrit, on s'intéresse à la détection de similarités. Dans un premier temps, on tire avantage des autosimilarités naturellement présentes dans les formes pour créer une nouvelle application de modélisation. Notre système interactif assiste l'utilisateur dans la sélection et l'édition de zones similaires sur des modèles 3D, en transportant automatiquement les traitements aux régions semblables. Enfin on présente une méthode permettant de combiner abstraction de formes et détection de similarités. Des informations non locales comme des symétries ou des similarités guident un processus local de simplification de forme. On construit ainsi une abstraction de formes multi-échelle qui contient intrinsèquement les informations de similarité du modèle.

Dans ce manuscrit on propose ainsi un ensemble de contributions complémentaires afin de mettre en œuvre un écosystème unifié de modélisation 3D basé sur des interactions utilisateur de haut-niveau.

CONTENTS

CHAPTER 1

INTRODUCTION

Make tools as transparent to the artists as special effects were made transparent to the public.

— Rob Cook, *2009*

Humans have always tried to depict their environment. Our ability to understand the world surrounding us is intimately linked to our capacity to represent it. In the same way, concepts and languages are connected. Humans do not have words for things they cannot imagine, and there is no existing idea than cannot be expressed or shared.

This might be a good reason why representation techniques such as drawing, reproduction or photography are such important domains. Computer Graphics is also part of this family and its goals are as diverse as the reasons we have to depict something. It is used for example to represent reality, to analyze, understand or simulate phenomena, to keep track of historical heritage, or to imagine entire new worlds.

Computer Graphics is usually divided in three major domains. *Modeling* aims at representing an object (real or not) in a virtual form. It deals with everything which affects the 3D shape along with the shape's intrinsic properties such as geometry, color or material. *Rendering* transforms the virtual 3D representation of a scene into a visible image displayed on the screen. Based on the objects' geometry and properties, the rendering process computes the response of the light on the virtual scene. The generated image depends on the scene objects, but also on the lights and camera position and properties. Finally, *Animation* adds a time dimension to rendering and modeling. We usually do not use a single image but rather multiple ones to visualize an animated scene. Animation is in charge of computing what happens to objects from frame to frame. It can be physically realistic, like simulation, or more artistic, like character animation.

In Computer Graphics users often play a central role. Of course, results of graphics processes such as images or 3D printed objects are visualized and used by people. However, in any interactive session, more than just being at the end, users are at the heart of the process. They continuously drive the process toward their goals through the use of interactive tools.

In this thesis, we focus on interactive modeling and we stand at the interface between shape, analysis and users. Our objective is to understand the relations between them and to create new possible exchanges to build novel interactive modeling tools. Figure 1.1 gives a very schematic overview of these relations, where different pipelines can be considered. A pre-process can be used to analyze the user or the model, abstract them and finally use the abstraction to create modeling

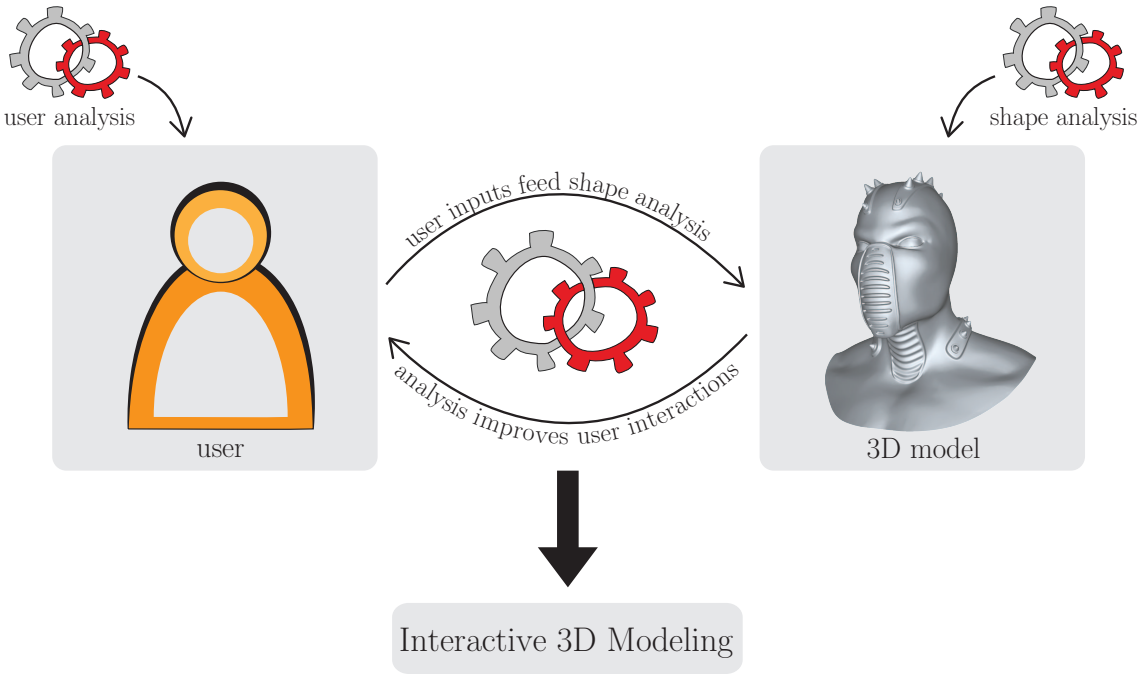


Figure 1.1 – Computational interactions for shape modeling.

tools. The exchanges can also be interactive and mutually beneficial. For example, the user can interact with the shape analysis process by providing high level inputs, while observing and using the analysis result to refine it and to perform modeling interactions on the shape. In the following we give a brief overview of our three main elements: user, shapes and analysis, and we propose a definition of *computational interactions* for modeling.

Users

From acquisition to printing, users may perform very different processes on a shape. In our tools, we do not want the user to only be the final spectator of a process, but rather we want to include her at its heart. To do this, we need a good enough understanding of the user abilities, difficulties, and objectives. We give here a few simple observations of human abilities that sound useful to us. Having a deeper understanding of human perception, interaction and shape recognition would be very interesting to build better modeling tools.

First, humans have a very good ability to aggregate data they seen. It is a deep instinct that allows us to understand and analyze the world around. The whole process of shape recognition is not perfectly understood yet, but we know that starting from a low-level observation of a 3D environment, made of lines and surfaces, humans have the ability to interpret shapes based on their knowledge. This capacity is used for example to classify a shape in a category, to retrieve an existing shape from an abstract representation or to imagine hidden parts of a model. All these unconscious, high level interpretations are difficult to reproduce by automatic processes. To become aware of our ability to

abstract shapes, the best solution is to trap it. For example, in the aforementioned image, known as the Rubin's vase, one can either see a vase or two human faces looking at each other.

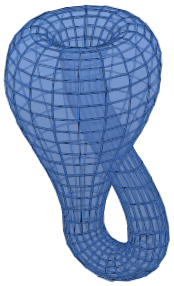
On the other hand, the user time is limited. 3D objects can have high complexity, and modeling is often a long and complex task. Users want efficiency, thus they would like to avoid repetitive, basic and boring tasks. Finally, in interactive modeling, having an interactive feedback allows trial-and-error processes, which are essential in creative tasks. In a perfect world, users would have a tool that learn and understand their gestures. The trade-off between the automation of all the tedious, repetitive tasks and the user freedom to fully control the process is of great importance.

3D shapes

Shape modeling is the creation or modification of a virtual representation of an object. In the discrete world of computer science, we need to encode and discretize the information of smooth shapes. Many representations have been created, and co-exist because of their different advantages and drawbacks. Users have to choose the representation that best fit their needs, depending on the application they have in mind. We give here a brief overview of the main shape representations. For a deeper introduction we refer the interested reader to the course of Botsch et al. [BPK+07].

Volume representations, using base elements such as tetrahedrons or voxels, are widely used in physical simulations. They represent the interior of a 3D object which has a finite and well define volume. Implicit representations also represent the volume of 3D objects thanks to a scalar field, with 0-isovalues at the boundary of the shape, and positive (resp. negative) values at its exterior (resp. interior).

For other applications in which there is no need to represent object interiors, only boundaries are encoded. The most common such encoding of 3D models is the mesh, a set of base planar polygons (usually triangles, quads or a mix of both) that represent the discretization of a smooth underlying surface. Surface representations allow more flexibility than volume ones. For example, objects that do not have a finite volume, like the famous Klein bottle shown on the opposite figure, can be displayed.



Finally the point set is the simplest representation of a 3D model. Scanners used to acquire a digital representation of real objects usually output a set of points, equipped with normals pointing toward the exterior of the shape. As scanners are getting more and more affordable, the number of digitalized objects is increasing. They usually contain a lot of small details and imperfections, tedious to create by hand. Unlike meshes, a point set does not encode the topology of an object. Thus, it is more difficult to retrieve some shape's properties from it than from a more structured representation.

Shape Analysis

Analyzing a shape is a common process in computer graphics. It can be useful for instance in *modeling* to simplify a shape while keeping its main features, in *rendering* to accelerate computations by factorizing similar objects or in *animation* to build deformation structures such as skeletons. Shape analysis is a broad domain divided into many sub-categories like shape segmentation, classification of objects in collections, symmetry detection or shape abstraction, among others. The

common factor between all these methods is that they aim at retrieving important information, aggregating and abstracting them into meaningful structures.

Shape analysis can be either used as an automatic preprocess or as part of an interactive tool. In a preprocessing step, it allows users to gain useful insights from a shape. Here, the final output cannot be modified by the user, and thus the result must have high fidelity to be useful. When the task is not well defined, achieving the desired result can be almost impossible. In this case, users might have different expectations based on their knowledge or cultural heritage. For instance, a shape analysis process would have difficulties to determine whether the aforementioned Rubin's vase is representing a vase or two faces.

On the other hand, using shape analysis in interactive methods gives more freedom to the user. They are able to change or modify the result of the process on the fly and they can even feed the analysis by high level inputs. However, three main types of difficulties arise in the construction of interactive systems based on shape analysis. First, the analysis should reach high frame rate while performing heavy computations to interactively produce meaningful results. Second a good tradeoff has to be found between the user freedom and the analysis efficiency. Third, the system should use good interactions, i. e. interactions that are easy to understand and perform for the user while being easy to exploit for the shape analysis.

Computational interaction for 3D modeling

In this thesis our goal is to understand user behaviors and shape signal in order to drive new intuitive user interfaces. We call a *computational interaction* the interleaving of human inputs and computational analysis tasks. The objective is not to have new cutting edge results in either human perception or shape analysis, but rather to interleave existing techniques and observations in a new way so that high level tasks, difficult for an automatic process are performed by users while repetitive and tedious tasks for humans are automatically computed during the interaction.

This manuscript is divided in two main sections.

In Part I we present the usage of shape abstractions in modeling. In this part, the abstraction of the shape occurs as a fast preprocess and is used to build high level structures. In Chapter 3 we build a navigation system based on the user gestures. The user body is analyzed to retrieve motions interactively and use them as the navigation interactions. In Chapter 4 we propose a new shape abstraction of 3D models, called *Sphere-Mesh*, and use it as high level, multi-resolution deformation tool.

In Part II we focus on shape similarity. In this section the main analysis of the shape is the detection of similarities. In Chapter 5, we first see how a good classification, user gestures, and shape similarity can be used to create a new selection tool, that allows to automatically select similar regions. We also address how this process would be a very interesting start to build similar editing tools. The user inputs and the analysis are really interleaved to create new metaphors for modeling. Finally, in Chapter 6 we interleave the detection of similarities with a decimation process, to create better abstraction of a model that inherently embeds the notions of similarity in its structure.

CHAPTER 2

BACKGROUND

In this chapter we present shape analysis notions that are used in the rest of the manuscript. In Section 2.2 we remind common topological and geometric properties of smooth and discrete surfaces. In Section 2.3 we give a definition of shape similarity and we survey techniques to detect and quantify the similarity among shapes. Finally, in Section 2.4, we focus on the notion of shape abstraction and we classify shape abstraction methods according to their primitives.

2.1 Notation

The following notation is adopted through this manuscript.

- $\mathbf{x} = (x_0, ..., x_{n-1})^T$ is a n-dimensional vector, with $||\mathbf{x}||$ its L_2 norm,
- $.^T \cdot .$ and $. \times .$ denote respectively the inner and the cross product of two vectors in any dimension,
- $\mathscr{M}_{mn}$ denotes the set of real m-by-n matrices and $\mathscr{S}^n$ the set of symmetric matrices of $\mathscr{M}_{nn}$,
- $\mathbf{M_{ij}}$ is the element at the i^{th} row and j^{th} column of a real m-by-n matrix, with $i \in [\![0, n-1]\!]$ and $j \in [\![0, m-1]\!]$,
- $\mathbf{M_{kl}^{ij}}$ denotes the $(k-i+1) \times (l-j+1)$-submatrix of M, whose top left corner element is $\mathbf{M_{ij}}$ and whose bottom right element is $\mathbf{M_{kl}}$.

2.2 Discrete Surface

In Computer Graphics the term *continuous surface* usually refers to a 2-manifold in $\mathbb{R}^3$, i. e. a topological space where the neighborhood of each point is homeomorphic to some set in the Euclidean plane (see Section 2.2.2). A closed 2-manifold encompasses a defined volume and is called a *watertight* surface. When each surface point is equipped with an inner product on its tangent space, the smooth surface is a *Riemannian* manifold.

2.2.1 Mesh

A mesh is a discrete representation of a smooth surface, composed by a set of planar polygons. As any planar polygon can be cut into triangles, we only talk about triangle meshes without loss of generality.

We define a triangle mesh $M = (V, T)$ as an ordered set $V = (v_1, v_2, ...v_n)$ of n vertices and an ordered set $T = (t_1, t_2, ...t_m)$ of m triangles. Each triangle stores the three indices of its vertices, and each vertex stores its position $p \in \mathbb{R}^3$. Most of the time, the set of mesh edges E is not stored and is assumed to be defined by the set of triangle edges. The set of triangles, edges and vertices indices give the combinatorial structure of the surface. The positions of the vertices define the embedding of the surface in $\mathbb{R}^3$.

We denote $V_1(v_i)$ the set of neighboring vertices of v_i, i. e. the set of vertices v_j such that there is an edge $e = (v_i, v_j) \in E$. In the same way, $E_1(v_i)$ and $T_1(v_i)$ are the set of neighboring edges and faces of v_i.

There are more sophisticated mesh data structures such as half-edges or directed edges. Their main benefit is the fast access to elements' neighborhood. We refer the interested reader to [BPK+07] for a survey of existing mesh data structures and libraries.

2.2.2 Manifoldness

A surface is a manifold with boundaries if every point has a neighborhood which is topologically equivalent to either a disk or half a disk. This continuous definition is easily expressed in the case of discrete triangle meshes. A mesh is manifold if:

- every interior edge has exactly two incident faces, and every boundary edge has exactly one incident face.
- the neighborhood of every interior vertex is made of a closed loop of triangles, and the neighborhood of every boundary vertex is a single fan of triangles.

Figure 2.1 shows examples of manifold and non manifold configurations for triangle meshes.

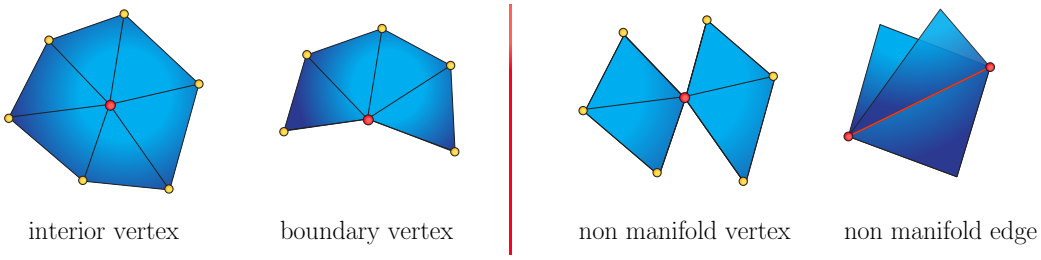


Figure 2.1 – Manifold configurations for the red vertices on the left, non manifold configurations for vertices and edges on the right.

2.2.3 Topological Invariant and Genus

One important topological invariant of a mesh is its *Euler characteristic*, noted χ. It coarsely describes the topological structure of the surface, regardless of its embedding. Based only on the number of mesh elements, we have the following relation $\chi = |V| - |E| + |T|$ with $|V|$ the number of vertices, $|E|$ the number of edges, and $|T|$ the number of triangle faces of the mesh.

When M is a surface without boundaries, χ is even, and we define the genus of the surface as the integer g such as $\chi = 2 - 2g$. The genus of a closed-surface describes its number of handles. For example, a sphere has a genus 0, whereas a torus (or every other surface topologically equivalent) has a genus 1. More generally, the genus of a surface which boundaries are composed of r curves is equal to $\chi = 2 - 2g - r$.

2.2.4 Geometric Properties

In this section we review some fundamental notions of differential geometry. In particular, we define intrinsic and extrinsic properties of smooth surfaces. Intrinsic properties only depend on lengths measured along the surface, while extrinsic properties rely on the surface embedding. This section is inspired by Keenan Crane's course notes [Cra12] and by [Mel12] (Chapter 2). Like them, for the sake of clarity, we highlight the connections between notions defined on 2D curves and on surfaces.

Induced metric: A curve can be expressed as a map $c : D \subset \mathbb{R} \to \mathbb{R}^2$ (or $\mathbb{R}^3$ for a 3D curve), where D is a parameter space. The differential of the curve d_c characterizes how a tangent vector $\mathbf{w} \in D$ get stretched out by the curve.

The induced squared length of $\mathbf{w}$ is equal to $|d_c(\mathbf{w})|^2 = d_c(\mathbf{w}) \cdot d_c(\mathbf{w})$. The *curved length* l between two points of D is the length of the curve between them:

$$l = \int_{u_1}^{u_2} |d_c(u)| du.$$

l defines the *intrinsic metric* of the curve. We usually choose a *unit-speed* parametrization, i. e. a parametrization which preserves length ($||\mathbf{w}|| = |d_c(\mathbf{w})|$).

A similar notion of metric holds on smooth surfaces. On a Riemannian manifold we can locally find a map $s : D \to \mathbb{R}^3$ that goes from a region $D \subset \mathbb{R}^2$ to a subset of $\mathbb{R}^3$, and we can locally express the surface in a coordinate system: $s(u,v) = [s_1(u,v),\ s_2(u,v),\ s_3(u,v)]^T$.

Once again we can define the differential $\mathbf{d_s}$ of the surface which characterizes how vectors on the parameter space get stretch out by the mapping. The induced squared length of $\mathbf{d_s}(\mathbf{w})$ is then $||\mathbf{d_s}(\mathbf{w})||^2 = \mathbf{d_s}(\mathbf{w})^T \cdot \mathbf{d_s}(\mathbf{w})$.

The *metric tensor* of the surface, also called its *first fundamental form*, is defined as

$$\mathbf{I}(\mathbf{w_1}, \mathbf{w_2}) = \mathbf{d_s}(\mathbf{w_1})^T \cdot \mathbf{d_s}(\mathbf{w_2}).$$

Using the coordinate system, the differential $\mathbf{d_s}(\mathbf{w})$ of a vector $\mathbf{w} \in D$ is equal to $\mathbf{Jw}$ where $\mathbf{J}$ is the Jacobian matrix:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial s_1}{\partial u} & \frac{\partial s_1}{\partial v} \\ \frac{\partial s_2}{\partial u} & \frac{\partial s_2}{\partial v} \\ \frac{\partial s_3}{\partial u} & \frac{\partial s_3}{\partial v} \end{bmatrix} = [\mathbf{s_u}, \mathbf{s_v}]$$

The squared length of $\mathbf{d_s}(\mathbf{w})$ is equal to $(\mathbf{Jw})^T(\mathbf{Jw}) = \mathbf{w}^T\mathbf{J}^T\mathbf{Jw}$ and we can express $\mathbf{I}$ as a symmetric matrix:

$$\mathbf{I} = \mathbf{J}^T\mathbf{J} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} = \begin{bmatrix} \mathbf{s_u}^T \cdot \mathbf{s_u} & \mathbf{s_u}^T \cdot \mathbf{s_v} \\ \mathbf{s_u}^T \cdot \mathbf{s_v} & \mathbf{s_v}^T \cdot \mathbf{s_v} \end{bmatrix} \tag{2.1}$$

The first fundamental form measures the inner product between any two tangent vectors. It completely characterizes the intrinsic properties of the surface, especially the length element ds on the surface expressed as $ds^2 = Edu^2 + Fdudv + Gdv^2$, and the area element dA equals to $dA = \sqrt{EG - F^2}dudv$.

Orientability: We say that a vector $\mathbf{n}$ is normal to the surface at p if, for all tangent vectors $\mathbf{w}$ at p, we have:

$$\mathbf{n}^T \cdot \mathbf{d_s}(\mathbf{w}) = 0.$$

For a discrete mesh, we usually compute a normal for each vertex as the weighted combination of its neighboring triangles.

If we can make a global and consistent choice for the normal direction in every point, we say that the surface is *orientable*. By convention, we call $\mathbf{n}$ the normal unit vector going outward the surface. For a manifold without boundary, being orientable also means that we are able to define the volume inside and outside the object. The Klein bottle, shown in the introduction, or the Möbius strip are two examples of non orientable surfaces.

Curvatures: Going back to a smooth curve $c(u)$ with a unit-speed parametrization, we measure the way it bends by looking at its tangent vector variation. As the unit tangent vector is the first derivative $d_c(u) = \mathbf{c}'(\mathbf{u})$ of the curve, its variation is simply defined by the second derivative $\mathbf{c}''(\mathbf{u}) = \kappa\mathbf{n}$, and the *curvature* κ is equal to $||\mathbf{c}''(\mathbf{u})||$. κ can also be seen as the inverse of the radius of the osculating circle, i. e. of the circle approximating best the curve at a point p, as shown in Figure 2.2. The bigger κ, the more bended is the curve.

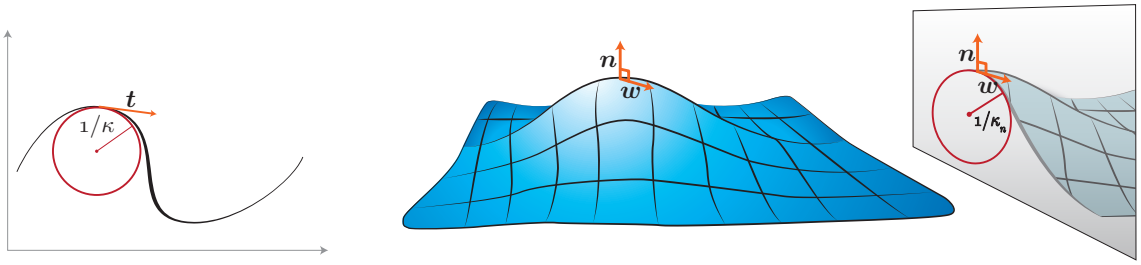


Figure 2.2 – Curvature for a curve embedded in the euclidean plane (left), and normal curvature on a smooth surface (right)

On smooth surfaces we can define the same notion in any tangent direction. The normal curvature $\kappa_n(\mathbf{w})$ along a unit direction $\mathbf{w}$ in the tangent plane of p is the curvature of the curve defined by the intersection between the surface and the plane spanned by $\mathbf{w}$ and $\mathbf{n}$, as shown in Figure 2.2. The normal curvature on a surface is signed. If the tangent vector variations and the normal have the same orientation, the curvature is negative and corresponds to a concavity. On the opposite, for a convexity the curvature is positive. Given the coordinate system, we can express the normal curvature in a given direction as:

$$\kappa_n(w) = \frac{\mathbf{w}^T\ \mathbf{II}\ \mathbf{w}}{\mathbf{w}^T\ \mathbf{I}\ \mathbf{w}},$$

where $\mathbf{II}$ is the second fundamental form, i. e. a quadratic form on the tangent plane defined by:

$$\mathbf{II} = \begin{bmatrix} e & f \\ f & g \end{bmatrix} = \begin{bmatrix} < \mathbf{s_{uu}}, \mathbf{n} > & < \mathbf{s_{uv}}, \mathbf{n} > \\ < \mathbf{s_{uv}}, \mathbf{n} > & < \mathbf{s_{vv}}, \mathbf{n} > \end{bmatrix},$$

with $\mathbf{s_{uu}}$, $\mathbf{s_{uv}}$ and $\mathbf{s_{vv}}$ the partial second order derivatives,

$$\mathbf{s_{uu}} = \frac{\partial^2 \mathbf{s}}{\partial u^2}, \quad \mathbf{s_{uv}} = \frac{\partial^2 \mathbf{s}}{\partial u \partial v}, \quad \mathbf{s_{vv}} = \frac{\partial^2 \mathbf{s}}{\partial v^2}.$$

Taking all the possible directions on the tangent plane, the maximum and minimum values of κ_n are called the **principal curvatures** κ_1 and κ_2. They are associated with the principal directions $\mathbf{e_1}$ and $\mathbf{e_2}$ that are orthogonal.

The **mean curvature** H is the integral of $\kappa_n(\theta)$ over all possible directions θ on the tangent plane. By integrating $\kappa_n(\theta) = \cos^2(\theta)\kappa_1 + \sin^2(\theta)\kappa_2$ over $[0, 2\pi]$ we obtain:

$$H = \frac{\kappa_1 + \kappa_2}{2},$$

The **Gaussian curvature** G is defined as the product of the two principal curvatures:

$$G = \kappa_1 \kappa_2 = \frac{\det \mathbf{II}}{\det \mathbf{I}}.$$

The *Gauss-Bonnet theorem* links the geometric and topological properties of the surface by connecting the total Gaussian curvature of a surface to its Euler characteristic. More specifically, for any smooth closed surface:

$$\int_M G dA = 2\pi\chi(M).$$

Finally, it can be shown that the Gaussian curvature depends only on the first fundamental form and its derivative. This is a remarkable property as it states that G is an intrinsic property of the surface, invariant to isometric deformations.

We can define discretized versions of the above geometric properties for a mesh representing an underlying Riemannian manifold. The interested reader is referred to [Rus04] for a derivation of the computations.

2.2.5 Laplacian Operator

The Laplacian characterizes the irregularity of a function. It is locally equal to the weighted average of the difference between the function value at a vertex and at its neighbors. Over a discrete mesh, the Laplacian is a n-by-n matrix defined as:

$$(\mathbf{Lf})_i = b_i^{-1} \sum_{j \in V(i)} w_{ij}(f_i - f_j), \tag{2.2}$$

where $\mathbf{f} = (f_0, ..., f_{n_1})^T$ is a n-dimensional vector, $V(i)$ is the local neighborhood of vertex v_i and w_{ij} is the (symmetric) weight associated to the edge $e = (v_i, v_j)$, b_i is a real positive number.

We can also write the Laplacian in a matrix form $\mathbf{L} = \mathbf{B}^{-1}\mathbf{S}$ where $\mathbf{B}^{-1}$ is diagonal with $\mathbf{B_{ii}} = b_i^{-1}$ and $\mathbf{S}$ is symmetric with:

$$\mathbf{S_{ij}} = \begin{cases} \sum_{j \in V(i)} w_{ij} & \text{if} \quad i = j \\ -w_{ij} & \text{if} \quad j \in V(i) \end{cases}$$

Several Laplacians can be defined on a discrete surface. Among them, the Laplace Beltrami operator is commonly called the Laplacian of the surface. Combinatorial Laplacian operators characterize topological properties of the mesh, while discretizations of the Laplace Beltrami operator are based on its geometry. The Laplace Beltrami operator is widely used in shape analysis as it reflects the surface geometry, its spectrum allows to define the harmonic basis of the manifold and to use classical tools from harmonic analysis.

Figure 2.3 – Some eigen vectors of the geometric laplacian on a raptor shape

The combinatorial Laplacian of a mesh is entirely determined by the mesh topology. Its matrix $\mathbf{K}$ is defined by $\mathbf{K} = \mathbf{D} - \mathbf{W}$, where $\mathbf{W}$ is the adjacency matrix of $M = (V, E, T)$:

$$\mathbf{W_{ij}} = \begin{cases} 1 & \text{if} \quad (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

and $\mathbf{D}$ is the diagonal *degree* matrix, where each d_i represents the degree of the associated vertex, i. e. the size of its one ring neighborhood $d_i = |V_1(v_i)|$.

$\mathbf{K}$ is simply defined from Equation (2.2) by setting $b_i = w_{ij} = 1$ for all i, j. $\mathbf{K}$ is positive and semi definite since the sum of each line of the matrix is null. Thus λ_0, the smallest eigen value of $\mathbf{K}$, is null and its associated eigen vector is constant. We can also show that the multiplicity of λ_0 is equal to the number of connected components of the graph.

The Laplace Beltrami operator is a second order differential operator. It is defined on a Riemannian manifold surface S as the divergence of the gradient of a real smooth function ϕ defined over S,

$$\Delta\phi = div \nabla\phi,$$

Here we focus on the approximation of the Laplace Beltrami operator over a discrete mesh M approximating S. As shown in [WMKG07] there is no discrete representation that fulfills all the properties of the continuous Laplace Beltrami operator. Thus, the choice of the discretization to use should depend on the properties needed for a particular application. We review few of the most classical discretizations here.

Pinkall and Polthier [PP93] describe the geometric Laplacian $\mathbf{P}$ using cotangent weights. For every scalar function defined over the mesh vertices, we have:

$$(\mathbf{Pf})_i = \sum_{j \in V_1(i)} \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij})(f_i - f_j), \tag{2.4}$$

where α_{ij} and β_{ij} are the angles defined in Figure 2.4. As for the combinatorial Laplacian, we can express $\mathbf{P}$ from Equation (2.2) by setting $b_i = 1$ and $w_{ij} = \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij})$. The matrix $\mathbf{P}$ is positive, semi definite and symmetric. Thus the discrete Laplacian is also self-adjoint, such as the smooth operator.

One disadvantage of this discretization is that $(\mathbf{Pf})_i$ represents the integral of the smooth Laplacian over a neighborhood and not its sampling on one point.

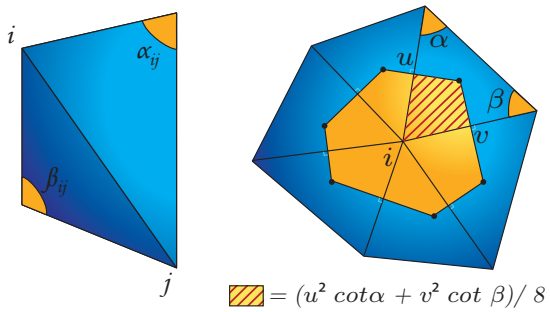


Figure 2.4 – Angles used to define the geometric Laplacian (left), Voronoï area used to weight the Laplacian (right).

To solve this problem Meyer et al. [MDSB02] propose to divide the expression by the area of the local neighborhood, i. e. by the Voronoï area of the vertex Ω_i, as depicted in Figure 2.4. The operator thus becomes:

$$(\bar{\mathbf{P}}\mathbf{f})_i = \frac{1}{\Omega_i} \sum_{j \in V_1(i)} \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij})(f_i - f_j). \tag{2.5}$$

However, using this discretization the Laplacian is not symmetric anymore.

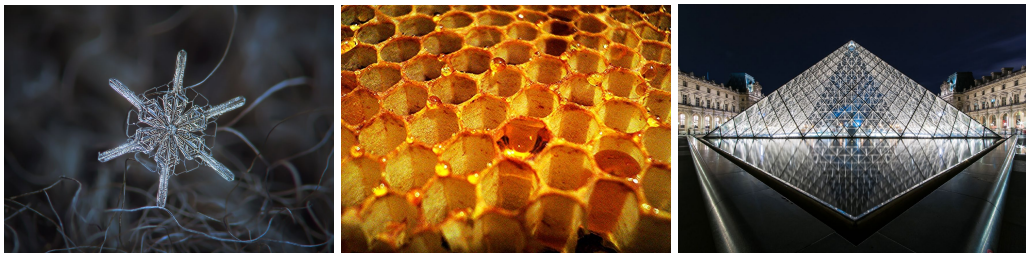
Finally, using finite elements we can define an other discretization, called the FEM Laplacian, $\mathbf{F} = \mathbf{B}^{-1}\mathbf{P}$, where $\mathbf{B}$ is the mass matrix as usually defined in finite element methods (see [LZ09] for calculus derivation).

Reuter et al. [RBG+09] propose an analysis of the different discretizations of the Laplace Beltrami operator and evaluate the differences between their eigenfunctions.

We give a special focus on the Laplace Beltrami operator and its frustum since it allows to use classical tools from harmonic analysis on meshes. Many shape analysis methods presented in the next two sections are based on the above surface's properties, especially on the curvature and the Laplace Beltrami operator.

2.3 Shape Similarity

We find repetitive patterns and symmetries everywhere in nature, art and science. For example, in chemistry, we classify molecules according to their symmetry since it explains many of their chemical properties. Looking at a honeycomb, its repetitive hexagonal structure (Figure 2.5) minimizes the material quantity needed to tile a given volume. In biology, most animals, including human, have a bilateral symmetry while most plants have radial symmetries. Scientists has also proved that symmetrical shape are more appealing to humans and animals. Thus, symmetry is present in arts and especially in architecture and music. Beside aesthetic regards, symmetric configurations are often the minimizer of some energy functions and, since they are efficient to design, they are omnipresent in manufactured products and architecture.



a) snowflake b) honeycomb c) Louvre museum

Figure 2.5 – Similarity patterns and symmetries.

The ubiquity of symmetry and the human strong ability to detect similarities make them a hot topic in computer graphics. Similarity detection is used in many different tasks such as shape segmentation, editing, morphing, classification or retrieval. However, assessing shape similarity is not easy. It depends on the models being analyzed and on the properties considered as relevant for a particular comparison. *Being similar* is context dependent and there is no single best similarity measure. For example, given a set of human shapes, some might be similar because they depict people in the same pose, but others might also be similar as they represent the same person.

In this chapter we give a brief overview of what is similarity, how we quantify it and how we can classify existing similarity detection algorithms. We focus on algorithms which use the geometric properties of the shape to detect similarity. Other approaches, that take into account additional information such as texture, color, or labeling of functional structures, are not reviewed.

2.3.1 General definition

We call *shape similarity detection* the process of quantifying how a shape resembles to another. We aim at finding a function $d : X \times X \to \mathbb{R}$, that takes two input objects A and B in the same universe X and outputs a real number representing their similarity score. Hereafter d is referred as the *similarity function*. It is worth noting that most techniques do not actually define the similarity but rather the opposite *dissimilarity* function between shapes, where a higher dissimilarity score stands for a lower similarity, and vice versa.

2.3.2 Query input & output

Some techniques aims at finding *global* similarities between shapes, meaning that the query inputs are two single and well-defined entities. However, in some cases, a shape is only partially similar to another one. We thus want to find *partial* similarities, i. e. to detect the similar regions of the shapes. This is a more complicated task, since it requires to find similar areas, and thus to segment the input shapes. Here, the query is not a whole shape, but rather a region on a model that has to be found.

All algorithms output a score assessing the similarity, but some of them also provide additional information. In particular, some techniques aim at finding a map between A and B, which can be either dense (mapping all the points of the shape) or sparse (mapping only few relevant points). Other methods, especially when looking for symmetry, aim at finding the set of transformations to go from A to B.

2.3.3 Desired properties

An important question while looking for a similarity detection algorithm is the desired properties of the method depending on the resulting application. Most of the time, the similarity function should fulfill the following properties:

Robustness d should be robust to missing data, to different discretizations of the shapes, and to small geometric and topological noise.

Invariance A similarity measure is invariant to a group of transformations G, if the similarity function is left unchanged for every transformation $g \in G$ and for all shapes $A, B \in X$, namely if $d(A, g(B)) = d(A, B)$. Different kinds of invariance are needed depending on the application. *Extrinsic* methods are invariant to some rigid transformations such as translation and rotation, while *intrinsic* ones are invariant to isometric or conformal deformations. An isometric deformation preserves the geodesic distances on the shape, while a conformal deformation preserve the angles.

Metric Some techniques require the function d to be a metric, i. e. to fulfill the metric properties:

- non-negativity $\quad d(A,B) \geq 0$
- reflexivity $\quad d(A,B) = 0 \iff A = B$
- symmetry $\quad d(A,B) = d(B,A)$
- triangle inequality $\quad d(A,B) + d(B,C) \geq d(A,C)$

Symmetry and non-negativity are usually desired properties for the similarity function. The triangle inequality postulate is more questionable, as shown in this famous example: a centaur and a man share some similarity, such as a centaur and a horse, but a horse and a man are totally dissimilar. Biasotti et al. [BCBB14] give more references and details on the metric postulate for similarity function.

Continuity The similarity function should smoothly increase as the inputs are getting more and more dissimilar.

Similarity and Symmetry

We included symmetry detection in the more general group of similarity detection techniques. We recall that an object M is symmetric with respect to a transformation T if it is invariant under its action, i. e. if $M = T(M)$. Thus detecting a symmetry is equivalent to find T such that $d(M, T(M)) = 0$. The particularities of symmetry detection techniques are that they act on one single shape, and that they aim at finding the set of transformations which leaves it unchanged. Mitra et al. [MPWC13] provide a complete overview of state of the art techniques in symmetry detection and classify existing approaches as *partial* vs *global*, *intrinsic* vs *extrinsic*, and *exact* vs *approximate*. Here, the notion of partial symmetry slightly differs from the one of partial similarity and has two different aspects. As before, the word *partial* applies for the input query, i. e. we look only at some regions and not at the whole shape. But it also applies for the set of transformations. Indeed, the set of transformations for a global symmetry has a group structure, but not for a partial symmetry. As shown in Figure 2.6, translational symmetries are good examples of partial symmetries.

complete symmetry group on part of a shape

partial translational symmetry

partial rotationnal symmetry

Figure 2.6 – Examples of partial symmetries.

2.3.4 Generic Pipeline

Most of the similarity detection methods share a similar workflow. A common strategy is to use a concise description of the shape, called a *descriptor*, instead of the whole model. The similarity detection problem is thus recast into the definition and comparison of the shape descriptors. Doing this, the complexity and size of the problem can be significantly reduced.

Most of the existing descriptors are computed per vertex, and represent some local properties of the shape as a real value or vector. To be relevant a descriptor should have the same properties than the similarity measure, especially the same invariance and robustness. It should also have a high descriptive power, i. e. a high ratio between the quality of the description and the descriptor size. Over the years a large number of descriptors have been proposed. In Section 2.3.5, we review and classify some of them in 3 categories namely statistics, regression and diffusion-based descriptors.

After the descriptor computation, most methods have a feature detection stage where the most relevant points on the shape are found. Usually the detection of features is made by analyzing the

variation or distribution of the descriptors along the shape. Working on the reduced set of feature points is one more strategy to reduce the size of the problem.

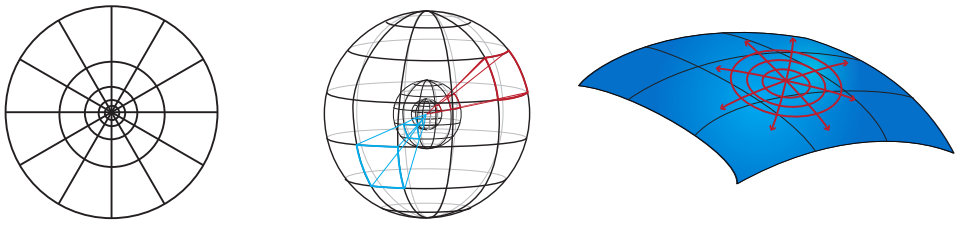
The last step in similarity detection techniques is to find the most relevant match(es) between shapes (or part of shapes) and to compute a similarity score between them. To do so, several strategies have been used, In Section 2.3.6 we classify them in 3 generic categories namely: energy minimization, meaningful structures and functional mapping.

2.3.5 Descriptors

Statistical descriptors: Statistical descriptors aim at computing a distribution of some values on vertex neighborhoods. They usually build histograms on each sampled points and compare them to find similar points.

The Spin Image descriptor [JH99] evaluates the distribution of the neighbors positions. Given a vertex p on a mesh and a predefined neighborhood $N(p)$, all the vertices in the neighborhood are projected on the tangent plane estimated at p. The Spin Image is the 2D histogram which measures the elevation of the neighbors (the distance between the vertices and their projections) and their radial distance (the distance between p and the projected points). A scale invariant version of the Spin Image was created by automatically computing the support size of the descriptor [DK12].

Shape context are other well-known representatives of statistical descriptors. They were originally applied on sampled 2D contours [BJP00]. The main idea is to notice that the set of vectors connecting a reference point to all the others is a rich description of the shape. It depicts the shape configuration seen from the reference point. The space of all these vectors is very large, so we rather consider their distribution by binning them in a log polar histogram (as show in Figure 2.7). This representation has several advantages: it is compact, robust to noise, and yet discriminative. The same descriptor can be applied on 2D images by aggregating values of a field $I(x,y)$. Going to 3D, [KPNK03] proposed a global rigid matching using 3D histograms for k sampled points on the surface, as depicted in Figure 2.7. As in the original shape context, the $k-1$ vectors formed by connecting one sample to all the others are binned into a 3D histogram. The descriptor is invariant to global rotation, translation and scale. More recently, Kokkinos et al. [KBLB12] proposed the intrinsic shape context which is invariant to isometric deformations. They build a 2D log polar histogram around each point by sampling geodesics starting from the reference point and going in some directions on the surface (see Figure 2.7).



2D Shape Context 3D Shape Context Intrinsic Shape Context

Figure 2.7 – Representations of the different histograms used for the original shape context, the 3D shape context, and the intrinsic shape context (from left to right).

Statistical descriptors are compact and discriminative. However several parameters are difficult to set such as the number of bins, the size of the neighborhood or the distance between histograms.

Regression descriptors: In regression-based techniques the computation of descriptors is approached by using an intermediate smooth representation to compute geometric properties of the shape. A continuous primitive is first fitted to approximate a local neighborhood on the surface and then used to compute geometric attributes like curvature. Most regression based algorithms work on any surface which can be expressed as a set of points, and do not use topological information.

Early methods fit a plane to approximate the local neighborhood of a point and then approach geometric properties such as curvature [PKG03]. Unfortunately, these techniques are highly sensitive to noise. A higher order primitive can be used to better approximate second order properties. For example in [GCO06], the authors fit a quadric patch on which differential properties are analytically computed to get an estimation of the curvature map of the local neighborhood.

Finally, the GLS descriptor [MGB+12] has drawn inspiration from the moving least squares algorithm which reconstruct a surface from point sets by locally fitting an algebraic sphere (see [GG07], [GGG08] for the fitting procedure). The authors do not aim at reconstructing a surface but rather at finding a new tool to analyze it. For each sampled point, they look at growing neighborhoods and fit the best algebraic sphere for each scale. Based on the fitting scalar field, they derive a number of parameters depicted in Figure 2.8 to create a descriptor: the algebraic offset distance τ between the reference vertex position p and the 0-isosurface of the sphere, the normal η of the scalar field at p, the signed curvature κ of the sphere and a fitness parameter φ that measures the quality of the fit. The GLS is fast to compute, multi-scale, highly informative, and can be derived in both scale and space. It is also invariant to translation, rotation, and can be made invariant to scale. Nonetheless, it is not invariant to isometric deformations.

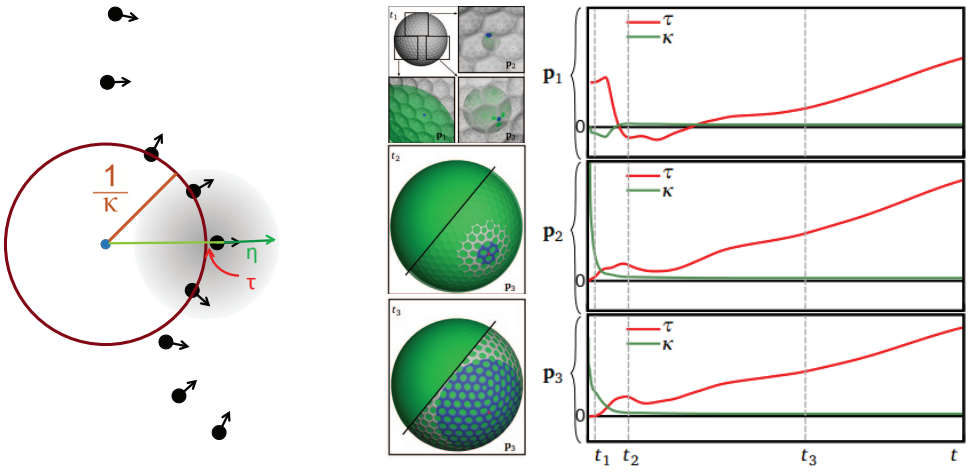


Figure 2.8 – Schematic view of the descriptor parameters for a given scale (left). The red circle represents the 0-isosurface of the scalar field. Scale variations of the GLS for three different points on a golf ball and for different scales t (right - image from [MGB+12]). The points neighborhood are depicted in blue while the fitted sphere are in green.

Regression based techniques lead to powerful descriptors. They are able to detect changes in curvature but also more complicated features, like feature lines or T-junctions. Yet, most of them only

compute extrinsic properties. Finally, their success is closely related to the robustness of the fitting process.

Diffusion descriptors: A family of intrinsic descriptors is based on diffusion processes on surfaces. Coifman and Lafon [CL06] first suggested to use the eigenpairs of the Laplace Beltrami to construct intrinsic metrics known as *diffusion distances*. Their technique is based on the heat diffusion, defined on a compact Riemmanian manifold M as:

$$\Delta_M u(x,t) = -\frac{\partial u(x,t)}{\partial t}. \tag{2.6}$$

We denote H_t the heat operator. $H_t(f)$ represents the heat distribution at time t, it respects the Equation 2.6 for all t and its limit as t approaches 0 ($lim_{t\to 0} H_t(f)$) is equal to an initial heat distribution $f : M \to \mathbb{R}$. The Laplace Beltrami and the Heat operator are linked by $H_t = e^{-t\Delta_M}$. Thus they share the same eigenfunctions Φ_i and their eigenvalues are linked by the same relation. For any compact Riemannian manifold, we call the *heat kernel* $k_t(x,y)$ the minimum function satisfying $H_t f(x) = \int_M k_t(x,y) f(y) dy$. The heat kernel is equal to:

$$k_t(x,y) = \sum_{i=0}^{\infty} e^{-\lambda_i t} \Phi_i(x) \Phi_i(y), \tag{2.7}$$

and corresponds to the amount of heat going from x to y in a time t given a unit heat source at x. The heat kernel is symmetric, invariant to isometries and multi-scale.

Based on these properties, Sun et al. [SOG09] define the Heat Kernel Signature, a function going from $\mathbb{R}^+$ to $\mathbb{R}$ such as $HKS(x) = k_t(x,x)$. It is a restriction of $k_t(x,.)$ on the temporal domain, but the authors show that it keeps all the properties of the heat kernel under mild assumptions. Finally the descriptor of each vertex v is a vector representing a sampling of $HKS(v)$ on the temporal domain (see Figure 2.9). The HKS has led to many similar descriptors such as the Scale invariant HKS [BK10] and the more discriminative Wave Kernel Signature [ASC11].

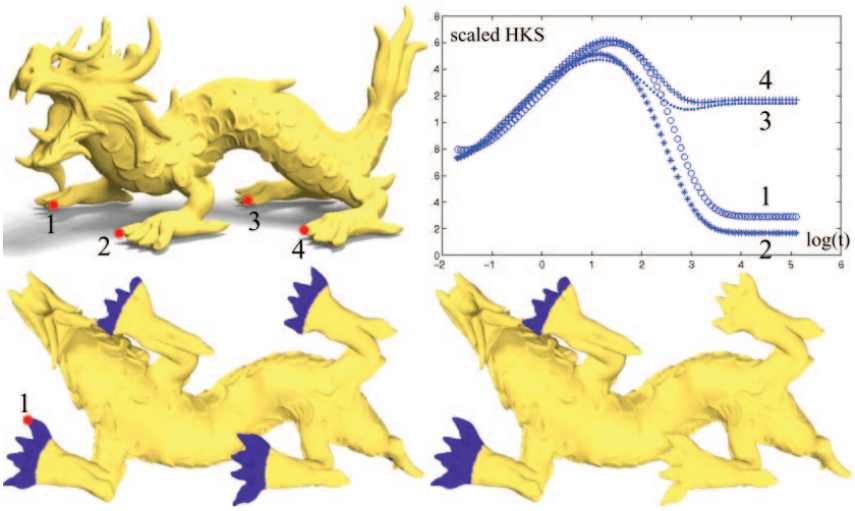


Figure 2.9 – HKS descriptor at different time step on a logarithmic scale, depending on the scale the four foot or only the front ones are detected as similar, image from [SOG09].

All these descriptors are multi-scales and invariant to isometries. Thus, they perform well for isometric similarity detection, but they still have some drawbacks. First, there is no established link between the time parameter t and the size of the local region. Then, these diffusion descriptors analyze better the low levels of a shape and are not suitable for detecting similar small scale details. Indeed, the HKS equation,

$$HKS(x,t) = k_t(x.x) = \sum_{i=0}^{\infty} e^{-\lambda_i t}\Phi_i(x)\Phi_i(x),$$

is approximated by the truncated sum $\sum_{i=0}^{k} e^{-\lambda_i t}\Phi_i(x)\Phi_i(x)$. Thus, computing the HKS requires to find the first k eigenpairs of the Laplace Beltrami operator, $\Delta_M\Phi_i = \lambda_i\Phi_i$. The higher the precision, the more eigenpairs are needed. Yet computing the first eigenpairs of an n-by-n sparse matrix is time expensive and non linear with respect to k.

2.3.6 Matching and Similarity Measure

Energy Minimization: Some methods express the similarity function d as an energy to minimize, usually made of two terms. The first one measures the dissimilarity of the matched points and the second one the distortion induced on the shapes. The Iterative Closest Point (ICP) [BM92], and all its derivatives ([RL00], [BR07]) are representative examples of this kind of methods (see [VKZHCO11] for a survey of other techniques). When dealing with intrinsic or partial matching, the energy to minimize usually becomes non convex, leading to a time consuming minimization that can fall into local minimums.

Meaningful Structures: Many methods aggregate shape descriptors into different meaningful structures to assess the dissimilarity of shapes. Among them, voting algorithms, Bag-of-Words paradigm and Skeleton based graphs are the more common.

The detection of similarity can be phrased as a voting scheme problem ([MGP06], [PMW+08], [LF09]). Point descriptors are expressed in a transformation space, in which close descriptors agreed on the same transformation. A clustering algorithm is then performed to retrieve the best set of transformations according to the descriptor votes.

Another way to aggregate descriptors is to take the topological structure of the shape into account. Most techniques based on matching skeletons [SSGD03] or Reeb Graphs ([TVD09], [MH12]) aggregate geometric descriptors into relevant topological parts and then work on the graph space to perform matching. It allows to easily perform part-based matching. However, one drawback of topological techniques is that they are by definition not robust to topological changes.

Finally, the Bag-of-Words model uses the dictionary paradigm and quantify descriptors into words. The procedure works in two steps: first, using a set of features, a *dictionary* of descriptors is created. All the descriptors are quantized into a set of k representatives, called the *words*. Then, given a shape and it corresponding set of descriptors, the method builds an histogram which represents the words of the shape. Each descriptor is approximated by the closest word in the dictionary. The histograms of two shapes are finally compared to evaluate their similarity. The Bag-of-Words paradigm is used in many algorithms, see e. g. [BBGO11], [DK12] or [Lav12]. Having a representative set of descriptors for the building step is essential in order to have meaningful classifications, also the number of words is a significant parameter. One disadvantage of Bag-of-Words techniques is that they lose the spatial relations between words, and only consider their statistics.

Functional Mapping: The most recent methods are based on a novel mathematical framework to compare descriptors formulated as real-valued functions on shapes [OBCS+12]. Given two manifolds M and N, a bijective mapping $T : M \rightarrow N$, and a real-valued function f on M, the function corresponding to f on N is $g = f \circ T^{-1}$. The transformation T_F induced on the two spaces of functions is a linear map called the *functional representation* of T. If both function spaces are equipped with basis T_F can be represented by the matrix of change of basis $\mathbf{C}$ between the two function spaces. To be efficient the selected basis on the function spaces should be compact, representative and stable; authors use the eigenvectors of the Laplacian. In a nutshell, the similarity detection is recast in finding the matrix C which represent an unknown mapping. This boils down to a linear problem to solve with isometric invariant functions used as constraints. This mathematical formulation has many advantages, especially linear algebra tools can be used such as matrix multiplications, inversions, or PCA analysis and the applications on maps become much easier.

Several techniques have used this framework to perform complex study on maps. [OMPG13] use functional maps to match similar shapes with symmetries, and Soft Maps [SNB+12] build a probabilistic relaxation of point-to-point maps. While, in [ROA+13] differences between a set shapes are localized and quantified. Finally, [BEKB15] are able to compute a new shape embedding to respect some similarities with other shapes.

2.4 Shape Abstraction

2.4.1 Broad definition

A shape abstraction is a representation of a 3D model that is as compact as possible while retaining the primal characteristics of the shape. This last notion is essential and application dependent. It is often linked to the final function of the 3D model as well as to some human perception or comprehension of the original shape. Preserving primal characteristics can mean for example preserving the visual appearance, the semantic meaning, the function, or the ability to recognize or classify the original shape.

The possible applications for shape abstractions are countless. Abstracting a shape is often used in the process of shape matching, retrieval, stylization, fabrication, editing or deformation to name a few examples out of many.

Most of the shape abstraction algorithms are based on a multi-resolution scheme. They aim at providing different levels of abstraction. This multi-resolution property is used in different contexts. For rendering, for example, the abstraction to render can be selected in real-time depending on the view point of the camera, and especially on the distance between the rendered object and the camera. The multi-resolution scheme implies to rank the shape characteristics, that is to be able to order them from primal features to details.

Shape abstraction is also present in art where artists like Picasso, for example, have tried to depict the essence of a shape. In his famous piece of art, called *Bull*, Picasso started from a complex representation of a bull and performed a progressive analysis of the shape to simplify and abstract it until he reached a final abstraction of the original model. This work is a suite of 11 lithographies showing the multi-resolution analysis performed by Picasso. The last lithography preserves all the meaning and even have a more expressive power that the original detailed version (see Figure 2.10).

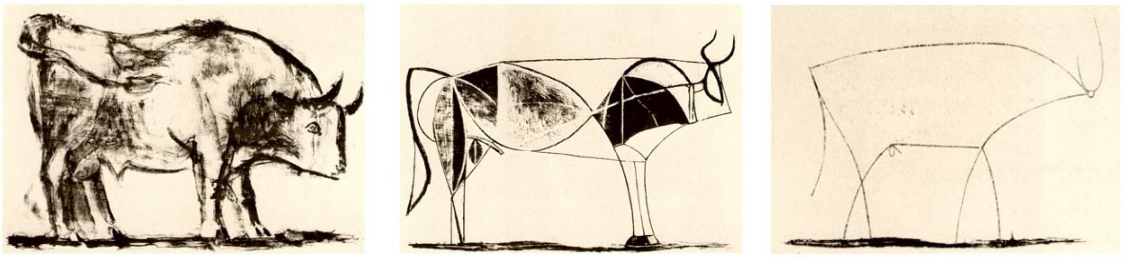


Figure 2.10 – different levels of abstraction, Picasso *"Bull"*, lithograph plates 1, 7 and 11 - ©*Picasso Administration*

In shape modeling, the geometry is usually the main information conveyed by the model, along with some material or texture information. It gives a precise description of the shape but does not explicitly convey any semantic information. To our knowledge, Falcidieno and Spagnuolo [FS98] are the first to present a shape abstraction model based on semantics. They highlighted the importance of a shape representation which encapsulate the *"different levels of abstraction needed for describing object form, function and meaning"*.

We review here two different classes of shape abstractions based on their underlying primitives. The first set of methods is based on the use of planes to abstract the shape, while the second one is based on a network of curves. We do not review techniques that retrieve and use the structure relationship between parts in a collection of man-made objects, and we refer the interested reader to [MWZ+14] for a survey.

2.4.2 Planar Abstractions

Planes are one of the simplest geometric primitives, they are easy to store and manipulate, and are naturally present in many CAD or man-made objects. That is why many algorithms abstract shapes using planar representations. Among them, mesh simplification techniques try to minimize the distance between the original surface and the simplified one, while other methods rather work on the volume of the shape, and propose a perceptually similar abstraction.

Shape abstraction via distance minimization: The objective of shape simplification is to find the closest geometric approximation of a shape, given a budget of primitives and an error metric. Among the existing solutions, *clustering algorithms* find a k-partitioning of the shape, and replace each partition by a representative element. *Resampling* algorithms aim at computing a new distribution of points over the original surface. While *decimation techniques* iteratively remove mesh elements by collapsing them until a given threshold error or number of primitives is reached.

In all cases a distance between the original and simplified mesh is used to measure the geometric quality of the abstraction. The *Hausdorff distance* d_H is the most commonly used. It measures the maximum distance of a set to the nearest point in the other set. Formally, given two shapes M_1 and M_2, represented by two sets of vertices V_1 and V_2,

$$d_H(M_1,M_2) = \max(\max_{v\in V_1} d(v,M_2), \max_{v\in V_2} d(v,M_1)), \tag{2.8}$$

with $d(v,M) = \min_{w\in M} ||\mathbf{v}-\mathbf{w}||$ the distance between a point v and a shape M.

Among decimation methods, Garland and Heckbert [GH97] proposed an algorithm based on a *pair contraction* scheme $(v_1, v_2) \rightarrow \bar{v}$, where two input points (v_1, v_2) are collapsed into a single one $\bar{v}$. Each contraction reduces the number of surface elements and has an associated cost to measure the difference with the surface. A pair contraction algorithm first defines a set of possible pair contractions and places them into a queue according to their associated cost. Then, at each iteration, the cheapest pair is collapsed and the queue is updated accordingly. The process ends when an error threshold or a defined number of contractions is reached. The main difference among pair contraction algorithms is the definition of the contraction cost. Garland and Heckbert use a quadric error metric (QEM), that measures the distance between a point and a set of planes. It leads to a highly efficient and powerful algorithm where the cost of a contraction and the new position are simple to compute and are linked to the curvature of the original surface (see Chapter 4 in [Gar99]). The greedy scheme of decimation techniques leads to natural multi-resolution abstractions. It also implies that there is no guaranty for a level to be optimal.

On the contrary, *clustering* algorithms partition the shape in k clusters. Then, they replace each cluster by a planar proxy, i. e. a position in 3D and its associated normal, to approximate the model. Some algorithms (e. g. [RB93] [Lin00] [SW03]) divide the space in clusters using uniform grids or adaptive trees. Other methods, such as Variational Shape Approximation (VSA) [CSAD04], partition the model by aggregating its elements. In VSA, the algorithm iteratively proceeds in two steps: partitioning and fitting. First, a k-partition on the shape is generated by aggregating together faces minimizing a given error metric. Then, the proxy approximating best the region is computed, and the closest face to the proxy center is used as the initial seed for the partitioning of the next iteration. The authors introduce the $L_{2,1}$ metric which computes distances between normals. It gives interesting visual results as normals play an important role in the perception of surfaces. This clustering method gives better results than decimation but at a higher computational cost.

Finally, *resampling* techniques (see [Tur92, AdVDI03, YLL+09]) compute a new, potentially coarser, point distribution on the surface and establish a new connectivity. Among them, Turk [Tur92] uses particle simulation along with a repulsion strategy to spread uniformly k samples on the original model. Their distribution take the curvature into account in order to spread more particles in regions of high curvature. The samples are then triangulated to create the simplified mesh.

On point sets, simplification techniques allow to reduce the model size before further processing and they remove geometric and topological noise coming from capture. Pauly et al. [PGK02] modified several mesh simplification algorithms to work on points . More recently highly efficient algorithms have been developed. For example, a parallel clustering method based on a special geometric ordering [LB15] is able to process millions of points in few milliseconds.

In any case, all those shape simplification techniques remain on the surface of the shape, which is not the best solution when reaching very high levels of abstraction.

Shape abstraction via perceptual minimization: Billboard clouds [DDSD03] are an extreme shape simplification method used for fast rendering. Their goal is to preserve the visual appearance of a complex model with a minimum set of planar polygons having textures and transparency maps. The transparency maps play an essential role to convey the visual appearance of complex shapes and visual effects like parallax. The same idea of visual appearance preservation at a minimum cost is also present in crowd animation. Starting from an animated 3D character [KDC+08] stores the animation for a discrete set of view directions in a time and memory efficient, yet vi-

sually appealing way. The authors transform the input in a *polypostor*, a set of 2D polygons with textures, along with the positions of the polygons' vertices that represent best the rendered animation (from the given view point).

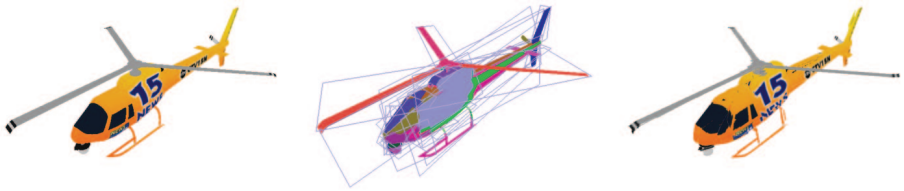


Figure 2.11 – Image from [DDSD03] which shows the original model with 5138 polygons, the set of 32 planar polygons used for the billboard cloud, and the final billboard cloud rendered with textured polygons.

Other planar abstraction techniques aim at representing a shape by a minimal set of cross planar sections that preserves its main features. We call a planar section the set of points lying on a plane and remaining inside the volume of the shape. It was shown that humans effortlessly recognize the original model from its abstract representation [DS10]. In this line of work the abstraction and the original shape have very different geometries, and cannot be compared using a usual distance measure. Thus the evaluation of a shape abstraction is more difficult to quantify and only depends on human perception.

Figure 2.12 – The left and center images comes from [MSM11] and respectively depict planar sections created by users (with thick lines representing strong agreement), and the planar sections automatically generated. The right image is a cardboard model, constructed by [HBA12].

In [MSM11] authors performed a user study to evaluate how humans position a set of planar cuts to depict a model. They show that i) a small number of sections is sufficient to represent a shape, ii) humans design consistent planar sections, iii) most of these sections are linked to the main geometric features of the shape. Based on these observations, the authors designed an algorithm to iteratively find a set of planar sections that cover as best as possible the main geometric features of the shape (see Figure 2.12). More recently Hildebrand et al. [HBA12] designed a method to automatically create cardboards models. Cardboards models are physical objects made of planar sections assembled by slicing (see Figure 2.12). Their algorithm assures their constructibility. Finally, in [MMS13] the authors evaluate the human perception of different planar section abstractions. Their goal is to understand how well different planar sections convey the shape information, that is how well a humans are able to retrieve the underlying surface geometry by looking at its abstract representation.

2.4.3 Skeleton Abstractions

Apart from planar representations, another common primitive for shape abstractions are skeletons. In the following we review several types of representations based on skeletons. *animation skeletons* mimics bones and joints to animate articulated models, *Medial Axis* are centered structures which represent the volumetric extend of a 3D shape, *curve skeletons* are made of curves embedded in the model, while *Exoskeletons* are curves that remain on the surface to depict a model.

Animation skeletons: Animation skeletons are heavily used to create the motions of virtual characters. The main idea is to use the metaphor of a physical skeleton to drive the surface deformation. An animation skeleton is a graph of nodes linked by segments, that represents the joints and bones of the character. Thus it can be seen as a shape abstraction that preserves the functional information of the model for animation. The surface is attached to the skeleton using a skinning process (overview of existing techniques in [JDKL14]).

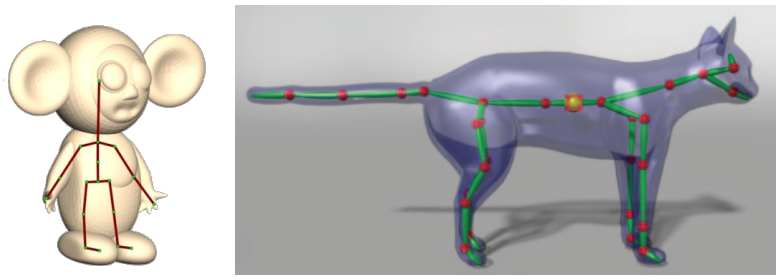


Figure 2.13 – On the left fitted skeleton on a static mesh [BP07], on the right fitted skeleton from an animated sequence [LD14a].

While many rigging techniques have been developed to compute the skinning, only a few methods aim at extracting an animation skeleton from a static mesh. Among them Popović et al. [BP07] adapt a generic skeleton so that it fits inside a static mesh (see Figure 2.13). Indeed, retrieving an animation skeleton from a 3D shape is a ill-posed problem, as it implies to infer the function of the object (here its motion range) from its geometry. Other methods rather used an animated character to retrieve the skeleton and associated set of skinning weights that best fit the animation ([dATTS08], [LD14a]). They create a compact and faithful abstraction of the animation.

Medial Axis: The medial axis is the set of all the points inside the 3D object that are at equal distance of at least two surface points. Working on 2D objects, the medial axis is composed of a network of 1D curves and points, while on 3D objects it is made of 2-dimensional sheets, 1D curves and points. The medial axis transform (or MAT) [Blu67] is the most common medial structure, it is made of the medial axis along with a radius function representing the size of the maximal enclosed sphere at each medial point. From the MAT, ones can easily reconstruct the original surface. However, it is really sensitive to noise and requires a long computation time. Thus it is usually approximated or simplified before being used (see e. g. [ACK01, DZ04, CL05, SFM07, MGP10]). Among existing simplification techniques, Faraj et al. [FTB13] propose a multi-resolution simplification of the MAT using an edge collapse strategy (see Figure 2.14 left). The cost of a collapse is linked to the scale at which the largest sphere of an edge absorbs the other one. Working on point sets, Calderon and Boubekeur [CB14] propose a piecewise smooth approximation of the medial surface, computed thanks to morphological operations (see Figure 2.14 right).

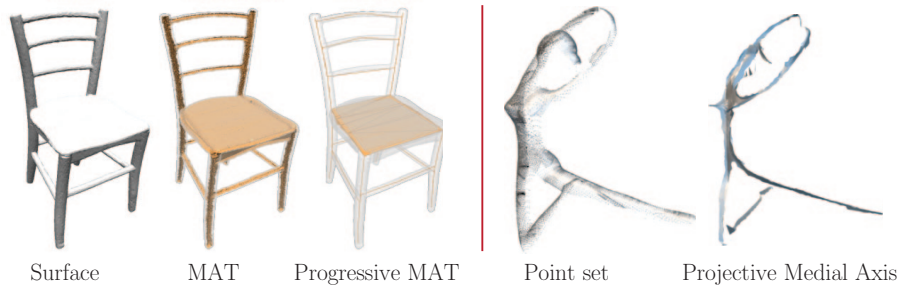


Figure 2.14 – On the left Progressive Medial Transform, image from [FTB13], on the right Projected Medial Axis, image from [CB14].

Curve Skeletons: A curve skeleton is made of a set of 1-manifold curves and aims at capturing the topology and geometry of an object. It can be seen as a graph, with curved edges connecting 3D nodes and a spatial embedding mimicking the input shape. The curve skeleton is usually required to be centered, robust, smooth and have a similar topology than the input model (see [CSM07] for a full list of properties).

To construct a curve skeleton several techniques contract the original shape until they reach a 1-dimensional curve skeleton ([ATC+08], [TAOZ12]). Toward this goal, they used the mean Curvature Flow Process, a motion that iteratively moves each surface point in the anti-normal direction, with a speed proportional to the local mean curvature H.

Others methods use *Reeb Graphs* and the *Morse theory* to compute a curve skeleton. A Reeb Graph is essentially a data structure which describes the topology of smooth scalar functions defined on a manifold. In a skeletonization process, the goal is usually to find a function which leads to an adequate Reeb Graph, that is then used as skeleton. The most straightforward choice is the z-function which represents the elevation along the surface, but better solutions have been developed. For example some methods are based on harmonic [NGH04] or geodesic [TVD09] functions.

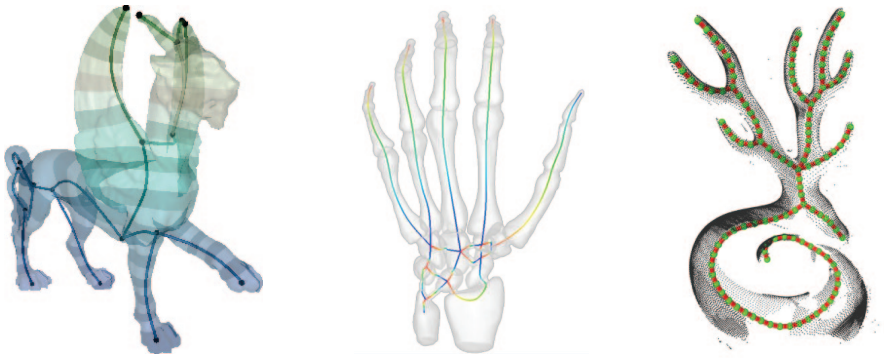


Figure 2.15 – From left to right, Reeb Graph using an harmonic function [TBTB12], Analytic Curve Skeleton [TBTB12], L_1 Medial Skeleton [HWCO+13].

Finally some techniques used a decomposition of the surface to construct a curve skeleton. In [TBTB12], the surface is decomposed into a set of topological cylinders and disks. The position of the skeleton is then computed by integrating the surface geometry along harmonic maps. In [BAS14], the authors presented a polar-annular mesh (PAM) that is a mesh-skeleton

co-representation. It partitions a manifold mesh into cylinders and disks regions, and uniquely defines the skeleton of the mesh via the connectivity of the PAM.

Techniques to find skeletons on point sets have also been developed. Tagliasacchi et al. [TZCO09] built a skeleton based on the idea of a generalized rotational symmetry axis of a shape; their method is mainly suited to work on "cylindrical-like" shapes. Huang et al.proposed the L_1 medial skeleton [HWCO+13]. It is based on the L_1-median, a statistical tool which represents the unique global center of a given set of points and is robust to outliers and noise. The authors apply the L_1-median in a local way and use it in concert with a regularization procedure to construct the L_1-medial skeleton. The abstraction has the good properties of being centered, robust to noise, to missing data and to outliers.

Exoskeletons : An exoskeleton is a set of connected curves lying close to the original object surface. Similarly to cross planar sections, the objective of exoskeleton techniques is to create a faithful representation of the shape with a minimal set of curves. Once again, the quality of the abstraction is evaluated based on human perception. In [DGGDV11] the objective is to build an exoskeleton which is a faithful representation of the structural parts of the shape and which follows important geometric features (see Figure 2.16 left). Mehra et al. [MZL+09] developed a method working on man-made models, not necessary manifold or connected. They built a two steps procedure which first find a smooth manifold surface as close as possible to the original model, and then extract a set of representative curves from this manifold (Figure 2.16 right).

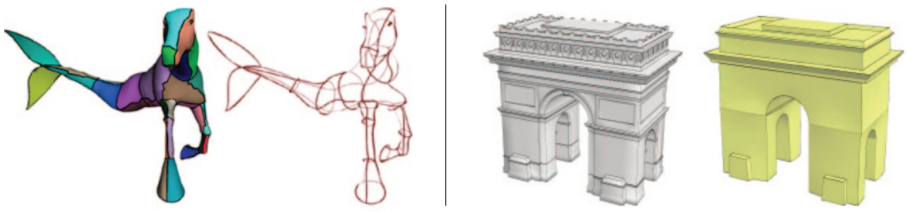


Figure 2.16 – A shape and its associated exoskeleton [DGGDV11] (left), abstraction of a monument at a given resolution by [MZL+09] (right).

Part I

Shape Abstraction for Interactive Modeling

In this first part we study how a shape abstraction can be used to build interactive tools. As we just saw, abstracting a shape consists of expressing it in a more powerful way, removing unnecessary information to keep only its essential characteristics. We present here two different use cases which both share the same pipeline. The abstraction of the shape is first computed in a fast preprocessing step, and then used as a meaningful structure to build an interactive system.

In Chapter 3 we see how we can use the abstraction of the user body in a navigation tool. We abstract the body of the user thanks to a skeletonization process and we use her motions to create a simple and efficient navigation system.

In Chapter 4 we present a new shape abstraction, called *Sphere-Mesh*, and use it as a deformation tool. We describe a simplification algorithm which creates a hierarchy of *Sphere-Meshes* having different resolutions. We then use the hierarchy as a new multi-resolution structure useful for interactive deformations of a shape.

CHAPTER 3

BODY ABSTRACTION FOR NAVIGATION

3.1 Motivations

3D scanners capture the geometry of real objects and enable the creation of virtual models representing them. Thanks to recent advances, they are available to everyone. In the same way that we use cameras to record videos, we can now take advantage of RGB-D sensors to capture the geometry of our surroundings. They output a stream of depth and color images in real time, which have to be analyzed on the flight before being usable. This analysis usually consists in retrieving simple abstraction models to understand the raw data coming out from RGB-D sensors.

Real time digitization and analysis allow not only to scan fixed items, but also to capture and analyze moving objects. We can at present detect a human moving in front of an RGD-D sensor. To do so, a segmentation step first extracts the user from the background. Then, an abstraction step analyzes her body to build a simple abstraction structure, namely a skeleton. Finally, the motions or other positional information can be retrieved from this abstraction.

This new high level input has many potential applications. For instance, the analysis of user motions can be used to create *mid-air interactions*. They seem to be particularly well adapted to 3D applications as they provide a new tri-dimensional way to interact with the virtual content [BCF+08], in contrast with classical 2D mapping.

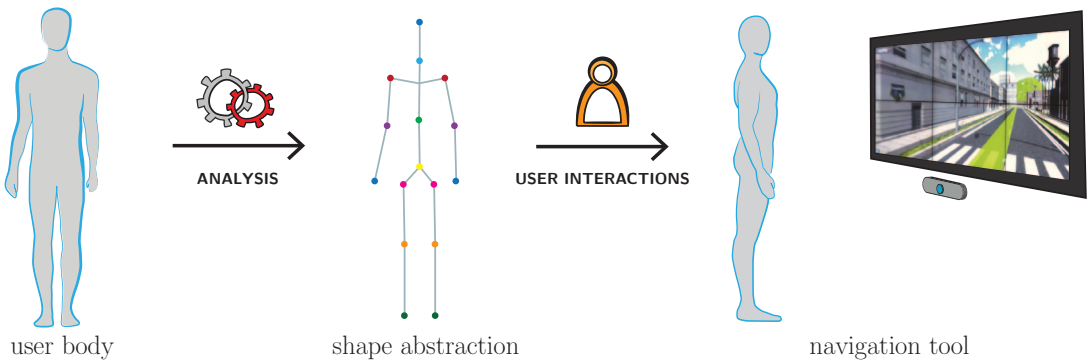


Figure 3.1 – *LazyNav* general pipeline. From a stream of depth images, we retrieve a skeleton and use it to create mid-air interactions for a ground navigation system.

For instance *Natural User Interfaces* (NUI) can be greatly improved by mid-air interactions. These interfaces aim at remaining invisible to users. Thus, the famous quote *"less is more"* depicts well their philosophy. Their objective is to simplify interactions until they feel natural for a wide range of users, from novices to experts.

Here, our objective is to develop a system to navigate in a 3D virtual scene using mid-air interactions (see Figure 3.1). We first notice that, while 2D devices have their own standards for ground-navigation (e.g., flying and orbiting using the keyboard and the mouse), mid-air techniques still lack natural metaphors for traveling in a scene. We observe that ground navigation requires only two degrees of freedom: one for walking (forward or backward) and one for turning the view (rotating to the left or right). Moreover, in many 3D applications, the user must interact with the scene. The ground navigation is therefore not the primary action she has to perform. Consequently, the system should keep the user hands, eyes or local head orientation completely free and available for interacting with the virtual content (selecting or editing virtual objects) or performing social interactions while playing (i.e. showing content to others).

Thus we propose *LazyNav*, a ground navigation control mechanism which is based on non-critical body parts. Starting with the user in front of the depth sensor, we exploit a skeleton abstraction to retrieve her motions and use them to create a natural navigation tool (see Figure 3.1). We adopt a *"lazy"* approach i. e. , the user easily controls the navigation with non-tiring motions, which are easy to discover, have a fast learning curve and are socially acceptable.

In the following we first explain how we develop a set of several body motions that follows the above guidelines (Section 3.3). In particular, we explain the motions computation pipeline (Section 3.3.2) and we highlight several assumptions that appears mandatory for developing mid-air ground navigation techniques (Section 3.3.3). Then, we describe the system architecture of *LazyNav* (Section 3.4) and we perform informal and formal user studies to analyze both our motions and our ground navigation assumptions (Section 3.5). Finally, we highlight general advice for ground navigation design using mid-air interactions (Section 3.7).

3.2 Related Work

Navigating in a virtual environment (VR) is a very common scenario and therefore has led to numerous interactions metaphors. In desktop configuration the most popular navigation is the First Person View (FPV) paradigm where the keyboard and the mouse are used concurrently to navigate. A good review of existing 2D inputs techniques for navigation was made by Jankowski et al. [JHH14]. Here, we focus on related works which use the 3D user motions to navigate.

Walking-In-Place techniques (WIP)

The most straightforward solution for ground navigation might be to have a one-to-one mapping between the user gestures and the virtual motions, i. e. to ask the user to actually perform the motions in the real world. However, this solution is restricted by a limited workspace. For example, long or infinite walks are impossible. Several walking-in-place (WIP) techniques have been developed to solve this issue. They are usually implemented in immersive environments, like Head Mounted Displays or Cave [CNSD93], and require sophisticated tracking equipments. For instance, the *Cyberith Virtualizer* [CH14] is a platform where the user can walk, run and jump in

place but also turn or squat down. Ikeda et al. [ISKY04] present an immersive telepresence system to navigate in photo-realistic scenes by walking on a treadmill. One-to-one immersive mapping requires non-trivial and costly equipments. For example, low-friction base platform [CH14] or belt systems ensure the player safety, high precision sensors capture her motions and head-mounted displays are used to visualize the virtual world.

On the contrary to our work, WIP techniques try to mimic as much as possible the user locomotion. This certainly gives a better immersive feeling, however it also results in more tiring and sub-efficient interactions. We rather focus on full-body interactions that are as natural as possible, but also effective and which allow the user to perform a secondary action while navigating.

Desktop Configurations

A number of methods exploit hand gestures to navigate in a 3D scene. Using a leap motion sensor, Adhikarla et al. [AWB+14] mimic well-known touchscreen gestures (rotate, pan and zoom), whereas Nabiyouni et al. [NLB14] evaluate several traveling metaphors (air plane and camera-in-hand) and multiple ways to control the speed (discrete and continuous). Simeone et al. [SVAG14] have a different approach, more related to our work. Their key concept is to use only the lower body part of the user for ground navigation. They work in a desktop environment where the user is seated in front of a computer and uses her foot to control the navigation. We address different scenarios and target public environment where the user is standing in front of a screen rather than seating at a desk.

General Body Motions

Several techniques use general body motions to navigate in a 3D virtual environment. On the contrary to WIP techniques that preserve the user proprioception, Pettré et al. [PSM+11] propose to maintain the user equilibrioception by performing leaning motions to navigate. They use a simple articulated platform on which the user is standing to help her leaning in the desired direction. Although freehand interactions could be achieved by this immersive device, the user cannot perform a secondary action while traveling in the virtual scene. Ren et al. [RLOW13] use a freehand (no hands-on device) gestural technique, with a broom metaphor to travel in a 3D scene. The user hands control the walk and her shoulders control the view rotation. In one of their experiment they give a real physical device (a broomstick) to help users understanding and performing gestures. In the work of Roupé et al. [RBSJ14], users lean the bust forward or backward to walk, rotate the shoulders to turn, and raise their arm to stop the motions.

In all these approaches, leaning the bust or rotating the shoulders seem to be natural interaction choices, however no common interaction is especially defined for the walk. We believe the set of motions used for ground navigation can be deeper analyzed. Therefore, we evaluate several body motions to understand why some of them are easier to perform, understand or remember. Keeping the user hands and head orientation free is our key design element as it preserves the ability to interact with either the virtual or the real world.

3.3 Motions

3.3.1 Design

We start by defining as many body motions as possible that follow our two principal criterion. First, the user should not need her hand, eyes nor head rotation to perform the navigation interaction. Second, the motion should be easy to perform, understand and not tiring.

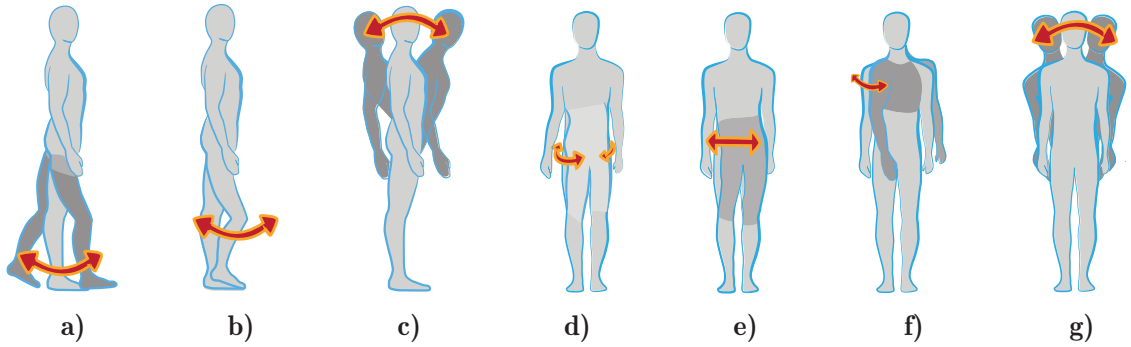


Figure 3.2 – **Designed user motions.** a) do a step (the user just puts one foot forward or backward) b) bend knees c) bend bust d) rotate hips e) translate hips f) rotate shoulders g) lean bust.

We end up with seven different motions illustrated in Figure 3.2, and we classified them into two groups according to their positions in the space of body motions (Figure 3.6).

Some motions behave in the sagittal plane, i. e. bend the bust and perform a step forward/backward (Figure 3.2 b) and a) respectively). While others behave in the coronal plane or around the vertical body axis i. e. lean the bust, translate the hips, bend the knees and rotate the shoulders or hips (Figure 3.2 g), e), b), f) and d) respectively). The step motion is part of both groups as it behaves in the coronal plane and makes a distinction between right and left.

3.3.2 Live analysis of captured body geometry

User motions are computed in three main stages depicted in Figure 3.3. First, an RGB-D sensor constructs a depth image. Then the user is extracted from the depth image and her skeleton is retrieved. Finally the user motions are computed from the skeletal joint positions.

Figure 3.3 – User motions computation.

Depth image: We use an RGB-D sensor to compute our depth image. It uses a infrared (IR) laser and camera to find the depth of each pixel. The process share similar principles with common stereo vision algorithms. In a nutshell, given two aligned cameras that output two side views of the same scene, we can retrieve the depth of a point identified in the two images by simple trigonometric computations. This technique fails when the detection of similar points on the two images fails too (for instance when there is repeated patterns on the scene, or when there is no texture on a large zone).

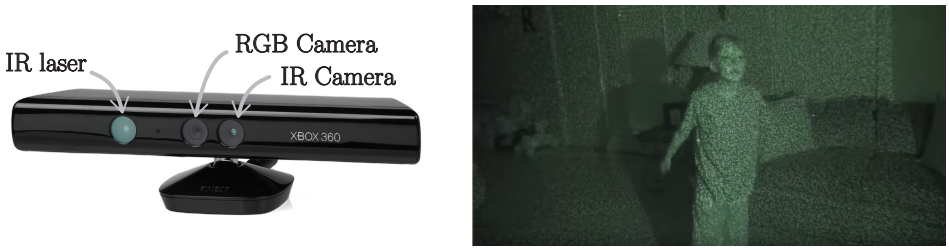


Figure 3.4 – RGB-D sensor (here, a *Kinect*) and an image of speckle patterns on a scene.

To avoid this drawback, common RGB-D sensors do not use two cameras but rather an IR projector and an IR camera (see Figure 3.4). The laser projects a known speckle pattern and the camera registers the projection of the pattern on the surfaces of the scene (Figure 3.4). Finally the device computes the depth in the image by comparing the original and deformed speckle images [ZSMG]. This technique is called *structured light* and works on any scene with any lighting condition as long as the IR speckles are visible to the IR camera. Thus, outdoor scenes or specular surfaces are more challenging for structured light methods.

Skeleton extraction: Once a depth image has been computed, we can extract the user body from the background and then retrieve the user skeleton. Actually, the skeleton has a known fixed topology and is composed of 30 predefined joints. The objective then becomes to retrieve the best 3D positions of each joint.

In [SFC+11] the skeletonization is performed independently for each frame in only 5 milliseconds. To this end, the authors use an intermediate representation where the user body is decomposed into a set of predefined body parts. Working on each pixel independently, they use a classifier to find its corresponding part. They train a deep random forest on a large data set with great variability in both poses and shapes. They also demonstrate the importance of the training of the classifier to achieve their results. The more variability on the training set, the better the classifier. Once all the pixels have been classified, a mean-shift based algorithm [CM02] is used to find local modes in joint positions and to generate proposals for the final 3D position of each joint, along with corresponding confidence scores. Figure 3.5 depicts the general pipeline of the method. This technique is robust enough to work independently on each frame, and still gives accurate and stable skeleton positions with little jitter from frame to frame.

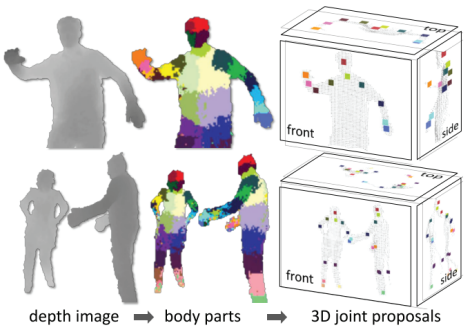


Figure 3.5 – Image from [SFC+11]

Angles computation: Our algorithm starts from this robust abstraction to define all our motions by measuring angles on the skeleton. More precisely, we measure the angles made by the body components at the current position with respect to a reference pose captured once by the RGB-D sensor at the beginning of the session. This makes our system adaptive to the initial pose and robust to the user morphology (e. g. height). Moreover, we couple the motion amplitude and the virtual velocity to ensure a proper speed control.

In practice, we use a set of vectors defined over the tracked skeleton joints (see Figure 3.6) to compute our angles. For the shoulder rotation (resp. hips rotation), the vector spans the two shoulders (resp. hips) positions **LS** and **RS** (resp. **LH** and **RH**). For the lean bust motion, the neck **N** and waist position **W** are used instead. For the bend bust motion, the vector goes from the user position **P** to the user head **H**, and we compute the angle in the z axis. For the hips translation, the angle from the user position **P** to the left hip **LH** is compared with the one from the user position **P** to the right hip **RH** in the x axis. For the bend knee motion, we use a vector that goes from the left knee **LK** to the right knee **RK**. Finally, to compute the step motion, we use the same angle than for the bend knee motion and we also compare the sum of the knee positions between the rest and current poses in the z axis to determine if the user goes forward or backward. We choose knees over feet because they are more likely to be inside the sensor frustum, and thus are more accurately detected.

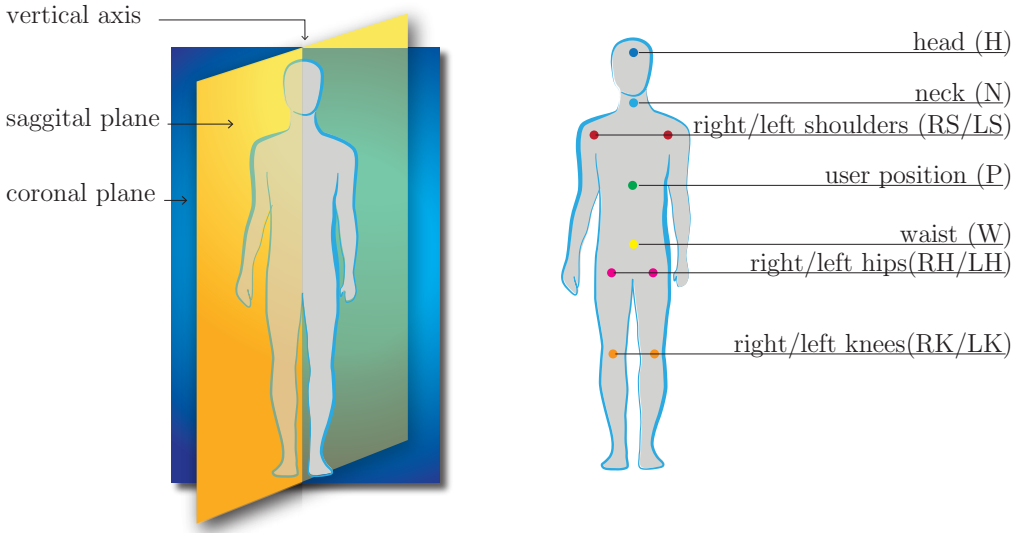


Figure 3.6 – plane metaphor (left) - tracked skeleton joints on the user (right)

3.3.3 Assumptions

Our objective is to find the best pair of motions to navigate in a virtual scene i. e. for walking and for turning the view. Before evaluating the possible pairs of motions, we define several assumptions they shall verify to be easy to use.

Uncorrelated body parts: some body parts are easier to move in an uncorrelated way than others. For instance, it is difficult to dissociate the shoulders rotation from the hips one. Thus, the

two distinct actions of the ground navigation (walking and rotating the view) should use dissociated motions and body parts.

Correlation between virtual and real motions: Having a good correlation between the motions performed in the real world and their effects in the virtual world helps the user to understand, remember and perform the interaction. Therefore, we consider that motions in the sagittal plane are better suited to walk, whereas motions in the coronal plane or around the vertical body axis are better suited to rotate the view.

Lazy navigation: we link the motion amplitude to the virtual speed in order to have an accurate and lazy interaction. We believe the user needs a comfortable rest pose where no interactions are happening, and at the same time *"lazy motions"* to navigate in the scene. Thus a good trade-off has to be found between motions amplitude, virtual speed and rest pose.

Secondary action: ground navigation is a basic interaction, but a fully operational system may require the user to do other things and we aim at preserving the ability to perform *"secondary actions"* while navigating. Such actions can be either *virtual*, having an impact in the virtual environment (e.g., selecting, grabbing or moving 3D objects) or *real*, having an effect in the real world (e.g., pointing something to someone or carrying a real object).

3.4 System Architecture

We provide a flexible design for our system by dividing our implementation into three main blocks: *motion receptors*, *transfer functions* and *actuators* (see Figure 3.7). This allows to easily try, plug, configure or disconnect user motions from the virtual camera. We also expose several parameters that are easy to understand and adjust, all of them being readily edited in a specific configuration file.

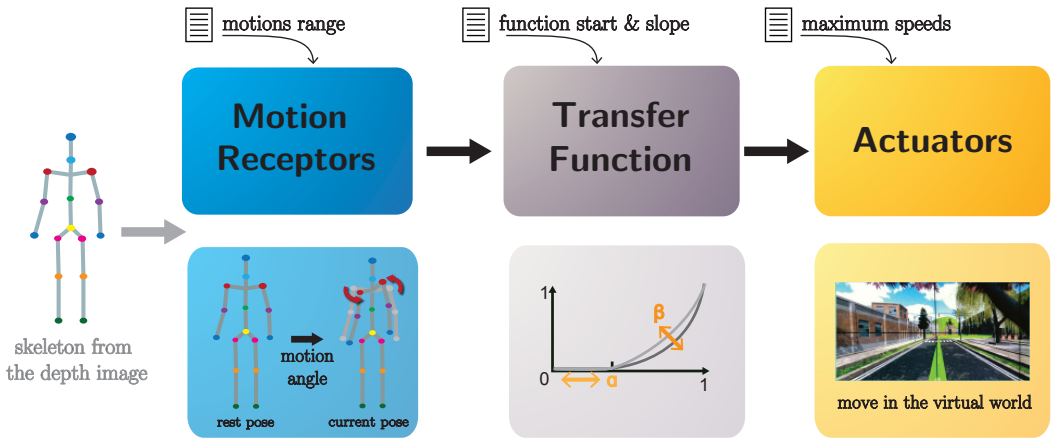


Figure 3.7 – Architecture design.

3.4.1 Motion receptors

The motion receptor block captures the motions made by the user. Using the 3D joints of the skeleton captured by the RGB-D camera, we compute our angles between the reference and current poses (as explained Section 3.3.2) before normalizing them using a specific range. For each motion, the range defines the largest possible angle. Finally, we output a value between 0 and 1.

3.4.2 Transfer function

Based on the motion receptor angle value x, we apply a transfer function f to connect the user interaction with the virtual camera motion. This intermediate remapping procedure allows to easily plug different user movements while keeping a uniform camera motion. The output of this block is a value between 0 and 1, computed as follow:

$$f(x) = \begin{cases} 0 & \text{if} \quad x \in [0, \alpha[\\ (\frac{x-\alpha}{1-\alpha})^{\beta} & \text{if} \quad x \in [\alpha, 1] \end{cases}$$

where α and β are parameters which respectively control the beginning of the motion effect (i.e., negligible values are ignored) and the slope of the function, as depicted in Figure 3.7. The parameter α has a great impact on the rest pose position and allows trading-off a comfortable rest pose for small enough motions to control the navigation.

3.4.3 Actuator

The resulting normalized modulated value is used to control the walk (resp. rotation) speed in the virtual scene. The transfer function output is multiplied with the application-dependent maximum speed (resp. maximum rotation speed) parameter and directly used in the 3D engine camera primitives (called `Move()` and `Rotate()` functionalities).

3.5 Motion Analysis

With our system architecture in hand, we analyze the resulting navigation to validate or discard our assumptions, to find out what are the best pairs of motions for ground navigation and finally to outline general advices for ground navigation using mid-air 3D interactions.

3.5.1 System Setting

We perform our user studies using an immersive widescreen display (4.0:1.15 m) made of 8 high-resolution screens (7680×2160 pixels). We use a *Microsoft Kinect* as the RGB-D sensor and the *Zigfu SDK* [HSZ+] to capture and process the tracked skeleton joints quickly and accurately. Our test scenario was generated using the *Unity3D* game engine. We perform a pilot and a user study. In all our experiments, users navigate in a realistic 3D scene representing a virtual city, depicted in Figure 3.8.

The user initially stands far from the display (about 3.0 m) to have a better field of view. A menu allows selecting the navigation motions inside the application at runtime. During the initialization

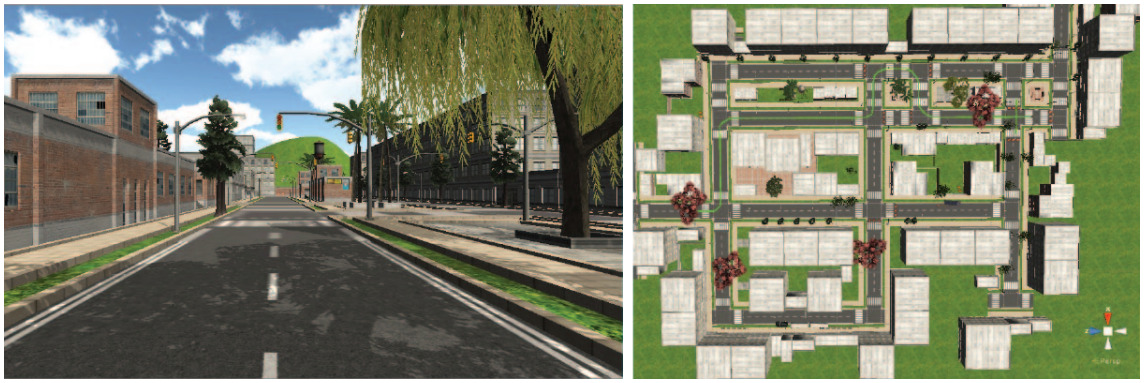


Figure 3.8 – **Virtual Scene used in our experiments**. Live user view (left), top view of the scene (right).

stage, we display the two motions currently available to the user (see Figure 3.9) while the system is capturing her rest pose (no T-pose required) so that the system is as self-discoverable as possible.

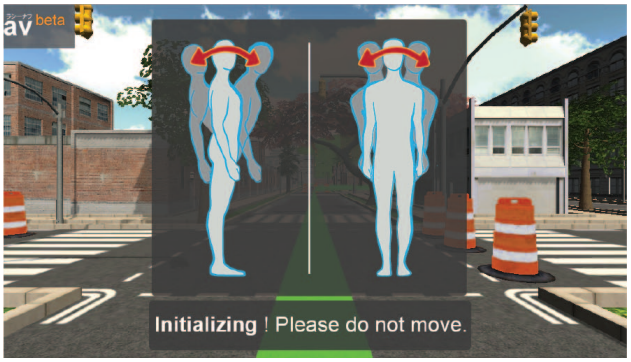


Figure 3.9 – **Initialization stage**. Available motions are displayed to the user while we capture her rest pose.

In Section 3.5.2, we explain how we realize our first pilot user study to evaluate our assumptions and determine a first ranking of our selected motions. Then, in Section 3.5.3, we keep only the best pairs of interactions and perform a formal user study.

3.5.2 Pilot User Study

Procedure

In the following, [V: rotate shoulders - W: bend the knee] denotes a pair of interaction where V stands for the *rotating the view* action and W stands for the *walking* action.

We have 7 available motions that can be used either for walking or rotating the view, as described in Section 3.3. The same motion cannot be used to perform two distinct actions, thus 7 pairs are discarded and we end up with 42 sets of possible interactions. From this, we also discard 6 sets of motions (see Figure 3.10) that were judged too correlated to be doable:

[V: rotate the hips - W: translate the hips],

[V: lean the bust - W: bend the bust],

[V: bend the knee - W: step],

and their inverses,

[V: translate the hips - W: rotate the hips],

[V: bend the bust - W: lean the bust],

[V: steps - W: bend the knee].

Since the number of potential interactions is significant, we conducted a qualitative pilot study with 30 users (20 males and 10 females). Among them, only 5 individuals had already used mid-air devices to navigate in a virtual environment before. We divided the participants in 6 groups, with all the users inside one group performing the 6 same interactions. Each user performed 6 sets of motions: 3 pairs of interactions and their opposites, for example [V: rotate shoulders - W: bend the knee] and its opposite set [V: bend the knee - W: rotate shoulders].

Tasks

We designed two different tasks. First, the user discovers the motions and their effects and she can freely navigate in the virtual city. Then, when the user feels comfortable with the pair of motions, we ask her to follow a virtual path displayed in the scene. As long as she is close enough, the path is green; if she goes too far, it becomes red. We repeat these two actions for the 6 different sets. We ask the user to think-aloud, and let her skip the path actions if she does not feel comfortable enough with the current interaction pair. Finally the user has to fill a questionnaire to give us general feedbacks on the interactions tried.

Results

Assumptions validation: The pilot user study allows us to validate some of our assumptions (detailed in Section 3.3.3). First, using correlated body parts to perform different actions is clearly difficult for the user. As shown in Figure 3.10, users did not manage to finish the path when the walk and view interactions used too correlated body parts (e. g. rotate the shoulders / rotate the hips or translate hips / lean the bust). Moreover people were usually able to better synchronize their actions, i. e. turning the view while walking, when the two motions were only thinly correlated. Second, having a similar correlation between real and virtual movements appeared to be easier. The set of motions with opposite correlation, i. e. a motion in the sagittal plane to rotate the view and a motion in the coronal plane to walk, were more difficult to perform. Users report being *confused*, and feel *unnatural*, some talked about *a coordination game* where it is tough to remember the good interaction, and they tend to forget the virtual impact of the interaction quickly. On the contrary, users generally need less time to understand and remember a good coordination interaction, they report them as *natural* and *easy to remember*.

Favorite interactions: We ask users to rate each motions they did for the six interactions on a 5-point-Likert scale (e.g. from 1: very poor motion to 5: very good motion). We compute the Kruskal-Wallis one-way ANOVA test on the obtained results. This statistical test is used to compare several

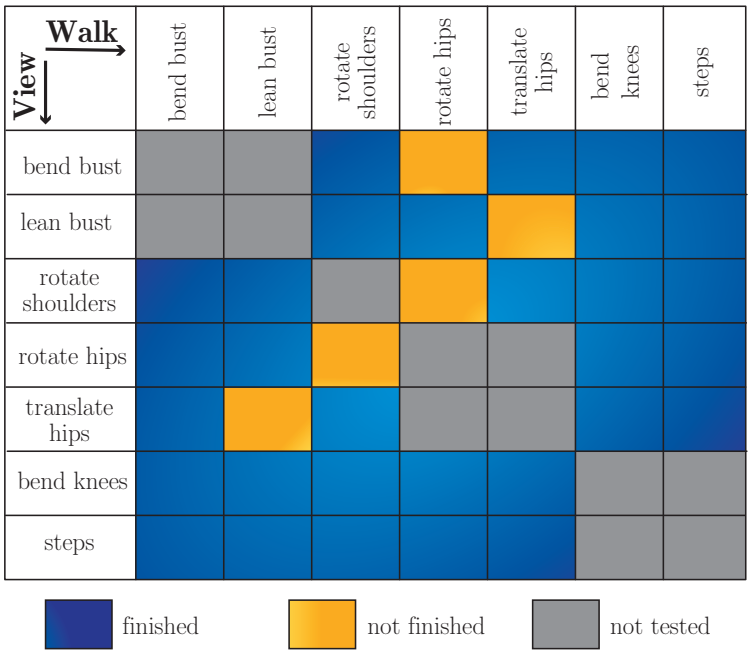


Figure 3.10 – **Pilot user study**: motions sets.

independent samples, and evaluate if at least one of them originate from a different distribution. We found out that there is no significant difference between the results of all the user groups for all the motions ($p > 0.05$), except for the walk interaction with shoulder rotation. This motion is significantly different among user groups ($p = 0.019$).

Finally, the favorite motions to turn the view are rotate the shoulders (3.24) and lean the bust (2.12), the other motions receiving bad scores such as, for example, 1.04 for bend the bust, 2.01 for bend the knees, and 1.35 for do a step. The favorite motions to walk are bend the bust (3.00), bend the knees (2.75), and do a step (3.42), while the others received bad scores with, for instance, 1.2 for rotate the hips, and 1.50 for lean the bust.

Social acceptance: We ask users to tell if each motion is socially acceptable or not, with the same 5-point-Likert rating scale (results depicted in Figure 3.11). We compute the Kruskal-Wallis one-way ANOVA test and find out that there is no significant difference between the user groups ($p > 0.05$). The result goes from an average of 3.5 for the translate hips motion to an average of 4.42 and 4.52 for respectively the bend knee and the rotate shoulders motions. Therefore, we can state that all our motions are socially acceptable by most users i. e. they will feel comfortable performing them in public.

3.5.3 User Study

Procedure

Based on our pilot user study, we restricted the number of available interactions to perform a quantitative study on the best sets of actions. Following the users answers, we decided to keep only

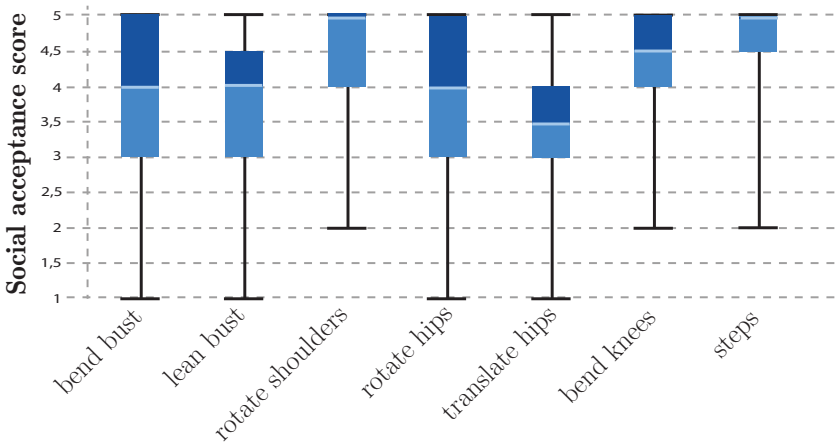


Figure 3.11 – **Pilot user study** users rating the motions on a 5-point-Likert scale for the question *"Would you feel comfortable performing this action in public?"*

the three favorite motions to rotate the view and to walk. Moreover, these motions are consistent with our assumption on the *correlation between virtual and real motions* and are considered as socially acceptable.

For our user study, we gathered 10 users (5 men, 5 women) aged from 20 to 28 (mean 24.5 years old) that did not participate in the pilot user study and were new to ground-navigation using mid-air devices. All the users tried the following 9 interaction possibilities in random order:

a) [V: rotate the shoulder - W: bend the bust],
b) [V: rotate the shoulder - W: bend the knee],
c) [V: rotate the shoulder - W: do a step],
d) [V: rotate the hip - W: bend the bust],
e) [V: rotate the hip - W: bend the knee],
f) [V: rotate the hip - W: do a step],
g) [V: lean the bust - W: bend the bust],
h) [V: lean the bust - W: bend the knee],
i) [V: lean the bust - W: do a step].

Tasks

We asked each user to first get used to the current interaction pair by following a short path. During this preliminary task, the user had only to move forward, backward and to turn on the left or on the right. In the second task, we required the participant to follow a path in the city as naturally as possible. We told her to be as accurate, fast, and lazy as possible. Again, the path remained green as long as the user was close enough, red otherwise.

For both tasks, the user was doing a secondary action while navigating in the scene. It could be either holding a coffee mug in her hand, carrying a grocery bag or a backpack. After each interaction, we asked the user to rate it from $(1 : \text{very poor})$ to $(7 : \text{very good})$ for the following properties: understandable, comfortable, easy-to-use, not tiring, accurate, secondary action doable, synchronization between walking and turning, not error prone.

Results

Movement: For each interaction, the 3D positions of the tracked user points were recorded to analyze the user motion and to get an idea of the *laziness* of each interaction. A given motion should be less tiring if the user only performs small movements.

Our application runs at 30Hz which means we have 30 values of all the user tracked joints per second. For each user u and each interaction i, we compute the movement $m_j(u,i)$ made by each tracked joint p^j (with $j \in [\![1..10]\!]$ as mentioned in Section 3.3.2) as the sum of the differences between its positions at runtime t and $t+1$ for the all user sequences:

$$m_j(u,i) = \sum_{t=t_0}^{t_n} (p^j_{t+1} - p^j_t)^2,$$

where t_0 and t_n are respectively the task starting and ending timesteps. Figure 3.12 shows a box plot of the total movement performed for each interaction and for each tracked joint ($\sum_u m_j(u,i)$). The distance is measured in the normalized scale of the Microsoft Kinect sensor coordinate. The step motion (Figure 3.12 third column) generates slightly more shoulders and head movements than others. We assume this is due to the fact that the whole user body is moving while she is doing a step forward or backward. The rotate shoulders and rotate hips motions (first and second rows) have a very similar pattern. This concurs with the observation that users do not really dissociate the hips and shoulders rotations (i.e., they rotate the all bust).

To evaluate the amount of effort required by each interaction, we sum the movement of all the tracked joints for one interaction ($motion\ quantity = \sum_{j=0}^{10} m_j(u,i)$). Then, we perform a one-way ANOVA Kruskal-Wallis test and do not find any significant difference between the interactions ($p = 0.48$). Consequently, we cannot conclude that some motions require globally more movement than others. Therefore, to establish if some motions are more lazy than others, we rely on the user questionnaire only.

Time: The time spent to follow the path is recorded for each user and each interaction. The average time for each pair of motions (with the previous order) is respectively 3.42, 2.80, 2.89, 3.68, 3.95, 2.82, 3.90, 2.87, and 2.52 minutes. To analyze the difference among the interactions, we apply once again the one-way ANOVA Kruskal-Wallis test and do not detect any significant difference ($p = 0.16$).

We also apply the Wilcoxon-Signed-Ranks paired t-test to detect if their is significant differences ($p < 0.05$) between pairs of interactions. We find out that *a / b* ($p = 0.013$), *a / i* ($p = 0.037$), *b / d* ($p = 0.01$), *b / e* ($p = 0.01$), *c / e* ($p = 0.02$), *d / f* ($p = 0.01$), *d / i* ($p = 0.02$), *e / d* ($p = 0.03$), *e / i* ($p = 0.04$) and *d / i* ($p = 0.04$) are not significantly different. The median value of the time spent on the "path following" task is shown in Figure 3.13 with the quartiles (25%:75%). The time variation among users is different depending on the interactions. For instance, the interactions involving the step motion (*c, f* and *i*) seems to have a smaller variance among users than others, whereas the interaction [V: rotate the hip - W: bend the knee] (*e*) results in an important time variation among users.

Accuracy: For each trial, we collected in-the-scene user positions, i. e. the position of the user avatar in the scene coordinate system for each timestep, and we measured the accuracy of the

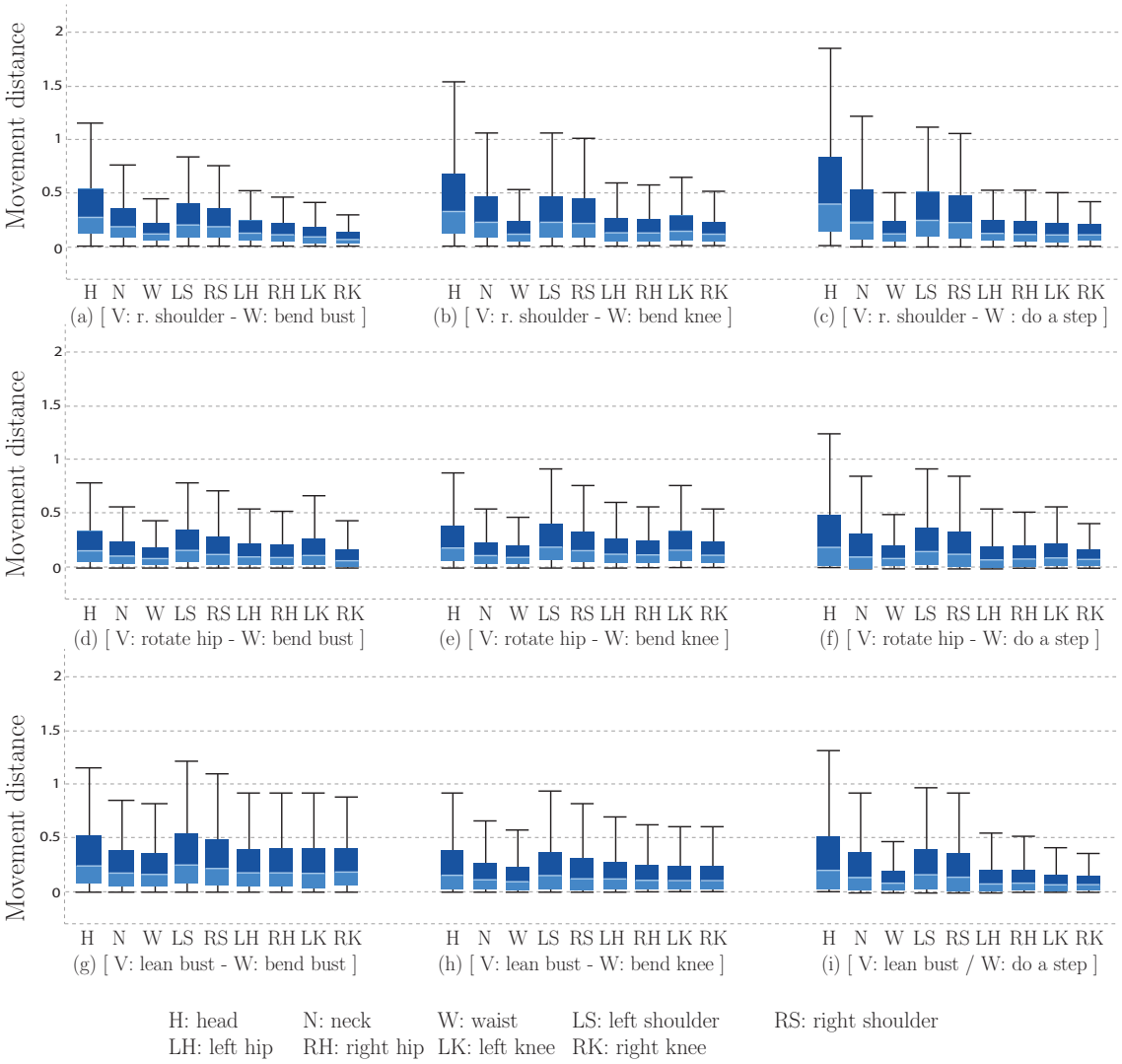


Figure 3.12 – Average user motion for each tracked joint and each interaction.

participant to follow the path. To do so, we compute the average of the squared distance from the avatar to the path and we express this value as a percentage of the full track length. Figure 3.14 depicts the results through their quartiles. The more accurate motions are [V: rotate hips - W: bend bust] (*d*) and [V: lean bust - W: bend knees] (*h*) with respectively 1.51% and 1.77% of error in average, whereas the less accurate motions are [V: rotate shoulders - W: bend bust] (*a*) with 5.72% and [V: rotate hips - W: bend knees] (*e*) with 5.02% of error.

Subjective Questionnaire: Figure 3.15 shows the results of the user preferences (rated on a 7-point-Likert scale) obtained for the 9 interactions and for each criteria. We confirm that our subjective questionnaire has a good reliability using the Cronbash's alpha test ($\alpha = 0.724$).

Overall the interactions were better ranked for the properties "understandable" (average 6.56), and "secondary action doable" (average 6.06) and they were moderately ranked (average 4.61) for the "synchronization between walking and turning" property.

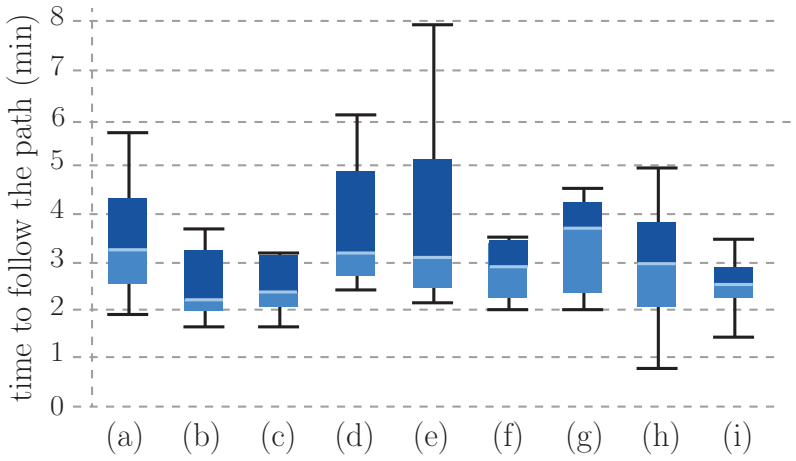


Figure 3.13 – Boxplot representing the quartiles of the time spent to follow the path (second task) for each interaction.

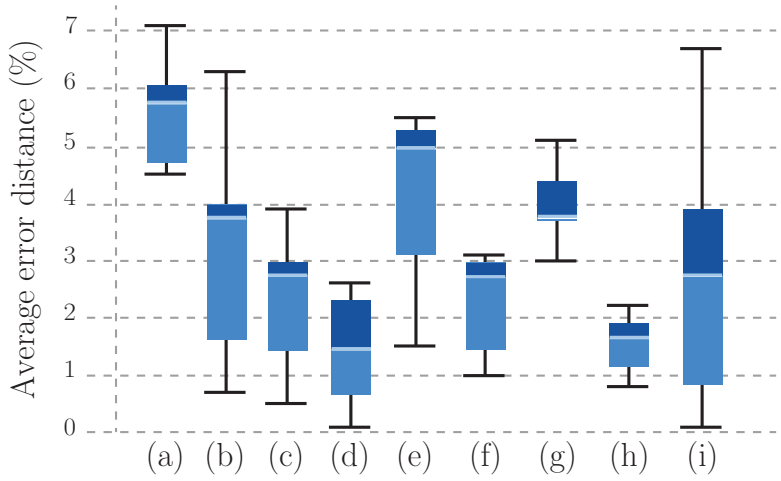


Figure 3.14 – Percentage of average error distance of in-the-scene user position in the path following task.

We analyze the user preferences in each criterion using the two-way ANOVA Friedman-Test, finding a significant difference ($p < 0.05$) between the 9 interaction sets for the following criterion: comfortable ($p = 0.38$), ease-of-use ($p = 0.12$), not tiring ($p = 0.52$), secondary action doable ($p = 0.81$), and synchronization between walking and turning ($p = 0.09$).

As we can see on the left side of Figure 3.16, the interactions *b*, *c*, *e* and *f* give similar results. This supports the idea that users have a tendency to perform the same gesture to rotate the hips and the shoulders. In all cases they rotate the whole bust. However the angles we capture are different, especially the motion range for the rotate hips motion is smaller than the motion range for the rotate shoulders motion. This may explain why users rate the interactions with rotate hips slightly higher than the ones with rotate shoulders for the not tiring and second action doable criteria. The right part of Figure 3.16 shows a comparison of the other motions. We can see that the *i* and *d* motions which are respectively [V: lean the bust - W: bend the knees] and [V: rotate hips - W: bend the bust] are overall better ranked than others, whereas the motion *i* [V: rotate shoulders - W: bend the bust] receives overall bad scores.

Figure 3.15 – Subjective preference for each interaction criterion.

Secondary action: During the study, we asked users to perform a secondary action while navigating. It could be either holding a coffee mug in their hand, carrying a grocery bag, or a backpack. During the user study, we observe that these actions could have some drawbacks on the interactions. First, when the user is carrying a grocery bag in her hand, the bag can interfere with the tracked body points, i. e. it moves in front of the knees and it is wrongly detected as a body point by the RGB-D sensor. This phenomenon appears more specifically when the user is rotating the shoulders or hips. And it has a negative impact especially when the rotation is coupled with the bend knees motion. Second, when the user is carrying a backpack she sometimes rotates her shoulders too much, the backpack then becomes visible by the RGB-D sensor and is wrongly understood as one of the shoulders. As a conclusion, bust rotations should be avoided or have a smaller range in order to avoid interferences between the navigation and some secondary actions.

3.6 Discussion

Based on our experiments, we now highlight several rules which appear as mandatory to develop good mid-air interactions for ground navigation. Starting from a set of 7 general motions that obey our initial conditions (no need for arm gestures or head/eye rotation), our pilot user study allowed us to restrict our approach to only 3 motions to rotate the view and 3 motions to walk. It also

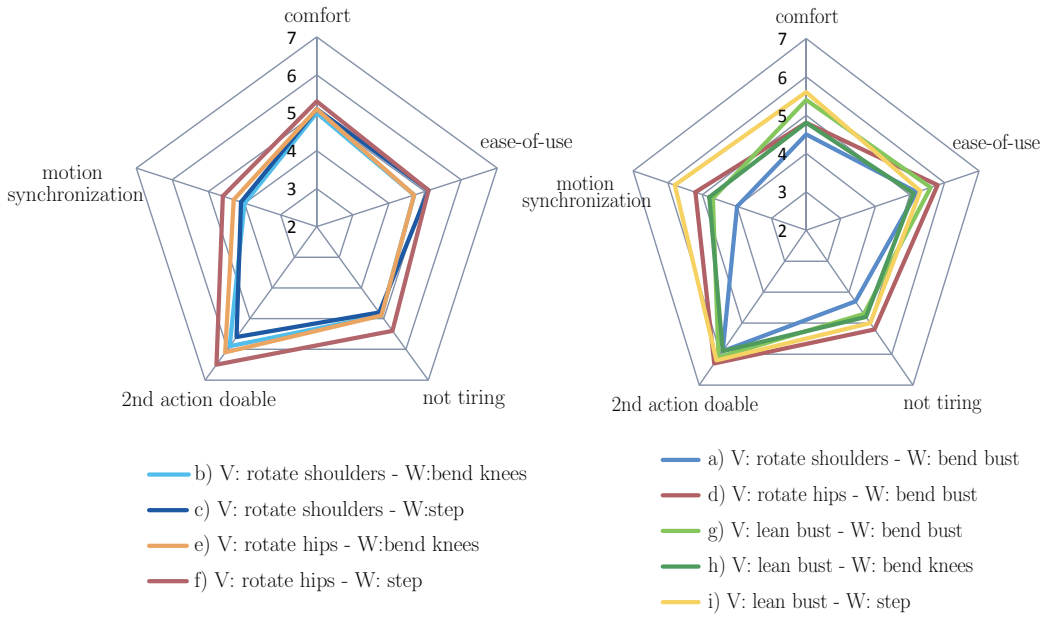


Figure 3.16 – User ranking for the criteria with significant differences among interactions.

validates some of our assumptions. First, to be more natural, the motions in the virtual and real worlds should behave in correlated human body planes. Second, motions for distinct interactions (eg. walking and turning) should use uncorrelated body parts. Finally, all the motions we kept are socially acceptable.

In the second study, we acquire a deeper understanding on the 9 best sets of motions, by measuring accuracy, time and amount of movements performed by the users. We also performed a quantitative analysis by asking the users to rate the motions on several properties. We did not clearly find a single interaction that would globally stand out from the others. However, several comments can be made from this study. First, users do not make a significant difference between rotating the hips or shoulders and only a single motion – "rotate the bust" – should be kept in a practical implementation. Rotating and leaning the bust seem to be the best motions to rotate the view. Rotating the bust appears less tiring than leaning it, but it may interfere with secondary actions. For the walk interaction, stepping, bending knees and bending the bust all have their advantages and drawbacks. While bending the bust is easy to control, it is more tiring than others. Stepping is a motion that is really easy to understand and remember but, as a discrete action, it makes more difficult to smoothly control the speed. Finally the bend knees motion is not tiring but slightly less natural as users have to remember which knee goes forward.

3.7 Conclusion

We have proposed a complete system for interactive mid-air ground navigation which is suitable for a large collection of applications. Our system can use several alternative body motions to control the virtual walk-through and lets critical body parts (hands, arms, head and eyes) free to perform other tasks. We support our method with a complete study that can help application developers to select a particular interaction modality for ground navigation. Although no ideal pair

of interaction seems to emerge, we report a list of advices for designing a particular application scenario.

So far, we used the pilot user study to set our parameters (e.g., transfer function slope and start, maximum speed and motion ranges) as efficiently as possible, performing the main user study with a fixed set of parameters. As future work, we plan to deeper analyze the effect of our parameters on the navigation. In particular, the motion range is user-dependent and we plan to make it user-adaptive, at the potential cost of a longer initialization step which may limit applicability to public spaces.

As the analysis of the amount of performed motions for each interaction does not highlight significant differences between the interactions, we rely only on the user questionnaire to evaluate the *"laziness"* of a motion. However, a deeper understanding of the interaction fatigue could be reached by building upon the work of Hincapié-Ramos et al. [HRGMI14].

As there is no general agreement on one best pair of motions, another direction for future work would be to design a data-driven system, learning from the user motions to adjust the interactions while she is navigating the scene. In this case, the interaction could be expressed as a weighting sum of canonical motions, optimizing the weights depending on the performed movements.

In this chapter we have seen how to use an abstraction of the user body, namely a skeleton, to create an efficient navigation tool. We designed our motions in order to allow other interactions apart from the navigation. As future work, we would like to use this ground navigation system in a novel modeling application where the user could create, deform or composed shapes interactively by simply using her body and voice as the manipulation primitive. Thus the user could be navigating in the world while creating it, in a new immersive and intuitive modeling system.

CHAPTER 4

EXTREME SHAPE APPROXIMATION FOR DEFORMATION CONTROL[1]

4.1 Introduction

In the previous chapter, we retrieved the abstraction of a user body from the live stream of an RGB-D sensor. From this abstraction we created new mid-air interactions and allowed the user to navigate easily in a virtual scene. In this chapter, we aim at analyzing the geometry of the virtual objects rather than the user motions coming from the real world. Following the same pipeline, we compute a new extreme shape approximation from an input shape, before using it as a powerful structure for deformation control (see Figure 4.1).

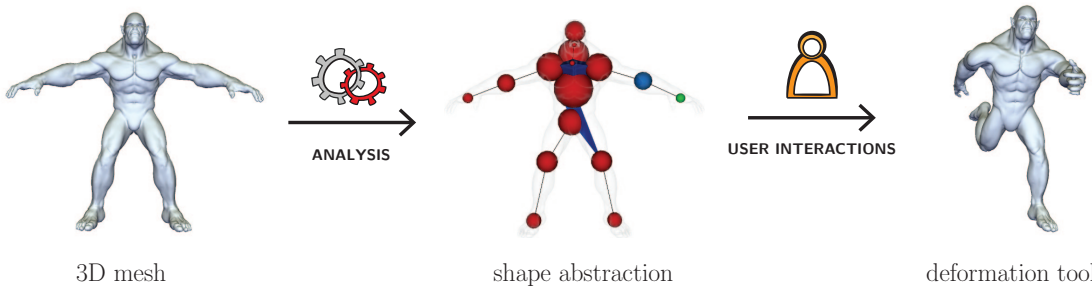


Figure 4.1 – The abstraction of the input shape is automatically created and then used as a control structure for the deformation of the original model.

The deformation of a virtual model is a long and tedious task. The representation of a shape is usually complex and made of millions of polygons. Thus, it is not worth considering a realistic deformation without the help of a shape abstraction to guide it. In general, the user applies a simple deformation on the elements of a *spatial control structure* which is tied to the model. When the user moves the control structure, the deformation is automatically propagated to the original shape. These kind of methods follow three main stages: the computation of a control metaphor for the deformation, the creation of a link between this structure and the original model, and finally the deformation of the shape through the manipulation of the control primitives by the user. Apart from few techniques which automatically compute control structures, this first step usually re-

[1] Common research work with Jean-Marc Thiery.

mains a manual task. In this work we propose a new extreme and multi-resolution approximation of a shape that is automatically computed. We bind the original model to our approximation and use it as a new high-level multi-resolution deformation structure for shape modeling. Our shape approximation, called *Sphere-Mesh*, comes from both simplification methods and volumetric abstractions.

Simplification techniques represent the surface of a shape at different levels of detail by either decimating, clustering or resampling the mesh vertices (see Section 2.4). They are very efficient when enough primitives are allowed for the simplified model. However, when coarse approximations are reached, they fail at capturing even simple structures. This poor behavior comes from the point interpolation scheme. Indeed, simplification methods intrinsically remain on the surface of their input model. We argue that volume primitives are more efficient than points for coarse approximations of shapes. Thus, our algorithm is based on the efficient scheme of decimation algorithms, but we replace the point interpolation by a sphere one.

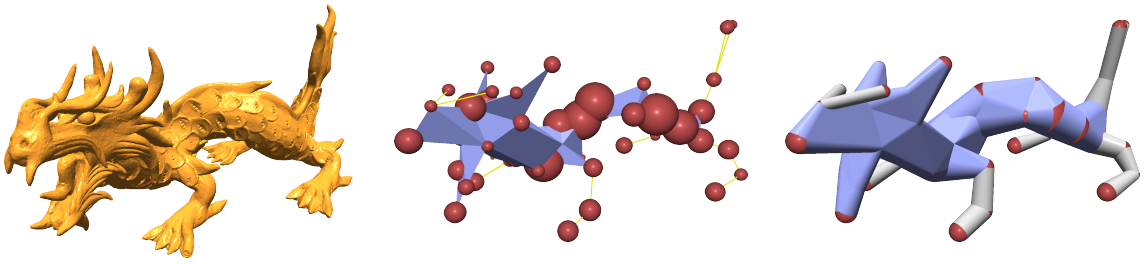


Figure 4.2 – A mesh (left, 200k triangles), a sphere-mesh approximation (middle, 50 spheres in red, edges in yellow, triangles in blue) and its interpolation (right). The spheres interpolated along edges are depicted in grey, while the ones interpolated along triangles are in blue.

A *sphere-mesh* is a mesh of which vertices are augmented with a radius representing the local thickness of the shape. Indeed, the sphere-mesh vertices are spheres instead of usual 3D points, hence its name. To represent the shape, the spheres are linearly interpolated along the *sphere-mesh* simplices (edges or triangles). Figure 4.2 depicts an input mesh and its sphere-mesh approximation at a given resolution. The sphere-mesh structure is displayed in the middle, while its interpolated geometry is shown on the right. Even at this coarse level, the sphere mesh is a good abstraction of the input shape.

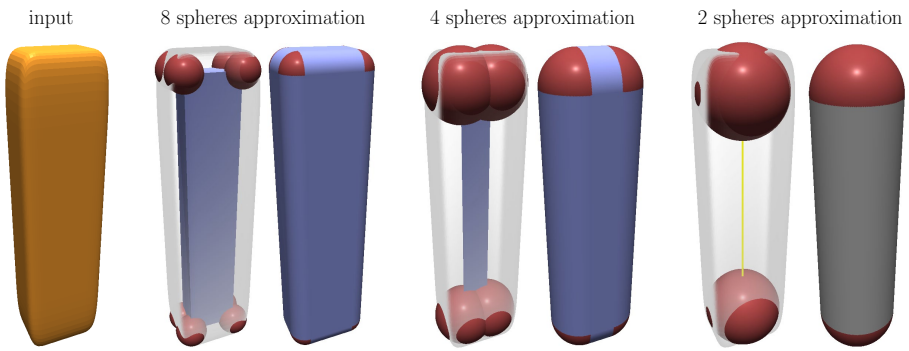


Figure 4.3 – Transition from surface to volumetric representation

Interestingly, a classical polygonal mesh is a special case of a sphere-mesh where vertices are spheres with zero radius. Our abstraction remains on the surface when enough primitives are pro-

vided, but it evolves progressively to a volumetric object while coarse levels are reached. Thus, our shape representation also share similarity with volumetric and skeleton abstractions (Section 2.4). Going from fine to coarse levels, our abstraction does not conserve the topological structure of the input mesh. In Figure 4.3 the sphere-mesh is homeomorphic to the input surface with 8 spheres, degenerates to a single sheet of triangles with 4 spheres and finally becomes a single edge with 2 spheres.

In the following we review existing control structures for the deformation and related works in simplification and volumetric abstractions (Section 4.2). We define our sphere-mesh representation (Section 4.3) and give an overview of our decimation technique (Section 4.4). Then we present the *Spherical Quadric Error Metric*, a key component of the shape approximation (Section 4.5) and we explicitly detail the shape approximation algorithm (Section 4.6). Finally, we use the sphere-mesh as a new control structure for shape deformation (Section 4.7) and we present approximation results (Section 4.8).

4.2 Background & Related Work

4.2.1 Common control structures for deformation

Spatial abstraction structures have been developed to help artists perform deformations. The wide range of possible deformations, from local details to global modifications, has led to a large amount of control structures, that all have their advantages and drawbacks. They can be classified according to their dimensionality: points and frames are the simplest 0-dimension structures, animation skeletons are 1-dimensional, 2D structures (such as patches) remain on the surface of the model and finally 3D cages deform the space they enclosed. We review here the main categories of existing deformation techniques (see Figure 4.4), and we refer the interested reader to the recent course of Jacobson et al. [JDKL14] for more details.

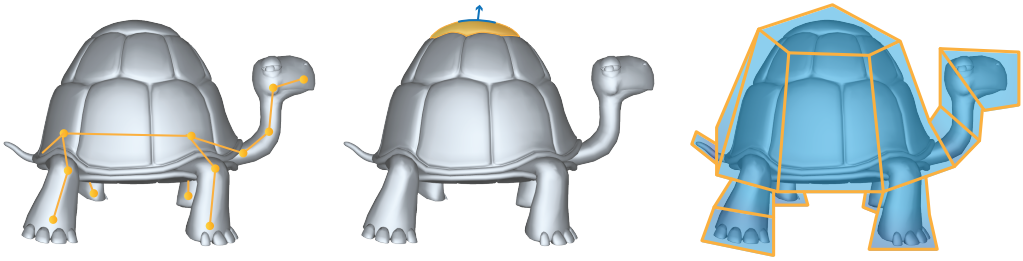


Figure 4.4 – Common control structures for shape deformation.

On surface techniques. The most common on-surface deformation methods are generally called *linear variational techniques*. They find the deformed surface as the result of an optimization process. As input, the user defines a region of interest (ROI) which will be deformed, while the rest of the surface remains fixed. In this region, the user characterizes a prescribed deformation for a set of *handles*. The objective is to find the best transformation of the ROI, i. e. a transformation that looks natural and physically plausible. To do so, an optimization problem is solved, usually minimizing *bending* and *stretching* energies, with the positions of the handles and the fixed part of

the surface used as constraints. For the sake of speed, all the linear variational techniques reduce the problem to a linear approximation. Among them, multi-resolution methods first deform a base mesh and then retrieve the positions of fine scale details using local intrinsic frames. Other methods directly use differential representations of the surface (such as Laplacian coordinates) and solve the problem in an intrinsic manner (we refer to [BS08] for a survey).

Apart from few exceptions, the definition of the ROI and handles is usually made by hand and has received little attention. Nevertheless, Nealen et al. [NSACO05] define handles from silhouettes or sketched lines on the surface. The user do not move the lines, but rather sketch new ones, that are used as constraints in the variational system. Schemali et al. [STB12] automatically generate handles from a larger set of feature lines. The user then moves handle lines with a transformation widget and the underlying region is deformed accordingly. The ROI is also automatically computed and can be adjusted interactively to enlarge or reduce the region.

Skeletons. The skeleton are the oldest and the most common control structures [MTLT88]. They are particularly well suited for the deformation of articulated characters. A skeleton is a set of rigid segments, called *bones* that are connected by *joints*. The *skinning* weights bind the surface to its skeleton, i. e. each weight $w_j(i)$ represents the amount of influence of the j^{th} joint on the i^{th} vertex. Thus, when the user moves a joint, the surface is updated accordingly. The linear blend skinning (LBS) is the simplest method to compute the positions of the deformed vertices $\mathbf{v}'_\mathbf{i}$. It is defined as

$$\mathbf{v}'_\mathbf{i} = \sum_{j=1}^{m} w_j(i)\mathbf{T_j}(\mathbf{v_i}),$$

with $\mathbf{T_j}(\mathbf{v_i})$ the joints transformations applied on the mesh vertex $\mathbf{v_i}$. More advanced techniques, such as dual quaternions, removes some artifacts of the LBS (see [Kv05], [KCvO08]). However, all these methods are only based on the mesh geometry and cannot handle volumetric effects such as contact (to avoid self intersections of the mesh), muscle bulge or skin elasticity. To solve these issues recent works combine the skeletal abstraction with a volume representation based on implicit surfaces (see [VBG+13], [VGB+14]).

The construction of the skeleton and the computation of skinning weights are two difficult tasks. Animation skeletons (Section 2.4) are usually constructed by hand, apart from few techniques which take as input a skeleton with a known topology and embed it into the shape [BP07]. The skinning weights should fulfill several properties: partition of unity, smoothness, locality, positivity... Some techniques have been developed to compute them automatically. Baran and Popović [BP07] compute weights based on equilibrium heat equations. The bounded biharmonic weights (BBW) [JBPS11] are based on the minimization of a Laplacian energy. They are smooth and shape-aware but sensitive to the quality of the input mesh. Recently, Dionne and De Lasa [DDL14] propose to use a sparse voxelization of the input shape to compute weights from geodesic distances between the bones and the surface.

Cages. Sederberg and Parry [SP86] were the first to use a 3D structure to control a deformation. Their technique is called free form deformation (FFD) and embed an object into a parallelepiped to deform it. It has led to cage-based methods that deform the entire 3D space embedded in a polygonal control structure, called a *cage*. Most of these methods express a point p inside a cage C as

$$\mathbf{p} = \sum_{j=1}^{m} w_j(\mathbf{p})\mathbf{c_j},$$

with c_j is the original position of the j^{th} vertex of the cage. The weights $w_j(.)$ are referred as *coordinates*. The primitives of the cage are used as *handles*. Moving a vertex of the cage to a new position $\mathbf{c}'$ induces a deformation of the enclosed space ($\mathbf{p}' = \sum_{j=1}^{m} w_j(\mathbf{p})\mathbf{c}'_\mathbf{j}$). Mean Value Coordinates (MVC) [JSW05] and Harmonic Coordinates (HC) [JMD+07] are based on the above formulation. Whereas, Green Coordinates (GC) [LLCO08] enrich it by adding a normal component ($\mathbf{p} = \sum_{j=1}^{m} w_j(\mathbf{p})\mathbf{c_j} + \sum_{j=1}^{m} \phi_j(\mathbf{p})\mathbf{n_j}$). It allows to induce rotations from the translation of the cage vertices and to produce quasi-conformal deformations in 3D.

The construction of the cage is often seen as an orthogonal problem of the coordinates evaluation. It is a difficult task usually performed by hand. The control structure should fulfill many different and conflicting properties. More specifically a cage should be very coarse and composed of planar facets. It should also fully enclose the original shape, remain tight to the surface and mirror its symmetries. Finally, the cage should have enough degree of freedom to perform the desired transformations. Deng et al. [DLM11] use a QEM simplification of the mesh to create a coarse enclosing cage, then they remove its self-intersections. Their technique only works for mesh inputs which limits its usefulness. Xian et al. [XLG12] use a tree of oriented bounding boxes to construct a cage, however their merging procedure do not guarantee a tight and coarse cage. The few automatic solutions that have been proposed are time consuming and gives poor results compared to man-made cages.

Similarities and open problems. Despite their large disparity, all these techniques share the same goal: reducing the burden of deforming a shape. A tremendous amount of work have been done to compute "good" deformations, i. e. to be able to interactively deform a complex shape from few user interactions while having plausible and realistic deformations. However the automatic creation of control structures is most of the time left to the user. It remains a long and tedious task, that requires good animation skills and an advanced understanding of the deformation pipeline. Moreover, several structures might be needed to perform different deformations on a single shape. The sphere-mesh abstraction is a new, high-level structure. It can be used as a control structure for the deformation and it shares similarities with both skeleton and cages. Its hierarchy is automatically constructed and can be used to interactively deform a shape at different scales.

4.2.2 Related work

Mesh simplification In the same way as mesh simplification algorithms, our representation is designed to approximate the shape of an object with a given budget of primitives (see Section 2.4 for more details). Our technique share two main components with decimation methods, especially with the work of Garland and Heckbert [GH97]. First, we follow a similar pipeline, i. e. we iteratively contract pairs of primitives to reduce the size of the abstraction. Second, and more importantly, our method is based on a similar core component, a *quadric error metric*. This metric is defined to measure and optimize the difference between the original shape and its simplification. Since the primitives of classical simplification algorithms are points, Garland and Heckbert [GH97] designed a quadric error metric that measures the distance from a point to a set of planes.

We argue that *extreme* mesh simplification requires defining volumetric elements instead of points, while providing a simple topological structure between them. Unlike classical simplification meth-

ods, our abstraction is made of spheres representing the shape volume. Thus, we design a new quadric error metric which measures the distance from a sphere to a set of planes.

Shapes from volumetric primitives Approximating shapes with a set of simple geometric primitives can be performed in numerous ways in digital shape modeling, with applications including for example shape recognition, multi-resolution visualization or collision detection. The medial axis transform [Blu67] (MAT) of a 3D surface mesh is a well-known volumetric abstraction (see Section 2.4). Alternatively, constructive solid geometry (CSG) methods model the shape of an object as a tree carrying simple geometric primitives on its leaves and boolean operations on its internal nodes. Extremely efficient at representing certain classes of manufactured objects, these models are usually defined from scratch and do not cope easily with automatic shape approximation.

Beyond MAT and CSG, the representation of volumes as the union of primitives has also been recently studied for spheres [WZS+06], ellipsoids [LCWK07], as well as axis-aligned or oriented bounding boxes [LCWK07]. Some approaches compute a sphere tree from an approximation of the medial axis [BO02, BO04, SKS12]. In a different context, spatial hierarchies have been extensively used in real time multi-resolution visualization [RL00] and physics [JP04]. Often based on simple bounding primitives (e.g., spheres or boxes) organized in a spatial binary tree, such structures aim at quickly culling empty space but only provide poor quality shape approximation at their coarser levels.

In contrast, we aim at representing the input 3D object with few fitting primitives while we also consider their *interpolation* over the simplices of a mesh sub-structure.

Sphere Skeletons Shape representations based on sphere interpolations have recently gained interest in shape modeling with the ZBrush tool [Pix01], popular in the SFX industry. With this tool, a skeleton of spheres – called *ZSpheres* – is manually constructed to define a shape at coarse grain, before refining its surface with displacements. With *B-Meshes*, Ji et al. [JLW10] improved this class of representations, in particular with a better mesh extraction.

In contrast to these methods, we propose to approximate *automatically* an existing, possibly dense mesh, with an interpolation of spheres. Additionally, our sphere-mesh representation extends the interpolation beyond skeletons, using polygons in addition to edges to model non-tubular regions.

4.3 Sphere-Mesh representation

We aim at approximating a 3D shape as a *Sphere-Mesh* composed of a set of spheres S indexed by edges E and triangles T. Each sphere S_i has a center $q_i \in \mathbb{R}^3$ and a radius r_i.

To approximate the original shape the radii of the spheres are linearly interpolated along the mesh edges and faces. Intuitively, a segment $[S_i, S_j]$ models the union of the interpolated spheres between S_i and S_j:

$$[S_i, S_j] = \cup_{u\in[0,1]}\{\, S(u\mathbf{q_i} + (1-u)\mathbf{q_j},\ ur_i + (1-u)r_j)\},$$

and corresponds to the convex hull of $S_i \cup S_j$ (see Figure 4.5). The same property holds for higher dimensional primitives such as triangles.

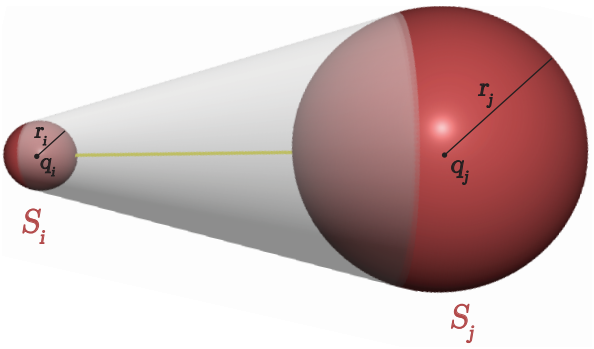


Figure 4.5 – Interpolation of spheres along an edge.

From a morphological point of view, a *sphere-mesh* $\{S,E,T\}$ corresponds to a *Minkowski sum* of the polygon mesh defined by the sphere centers with a sphere having a spatially-varying radius.

Surface extraction A surface approximating the input mesh can be extracted from a sphere-mesh as a convolution surface [BS91] between the base mesh and the spheres. In our work, spheres are interpolated along the edges and faces of the base mesh. An interpolated edge corresponds therefore to a cone cut by orthogonal planes at each edge extremity, and an interpolated triangle corresponds to a triangular prism with 3 faces from the edges extrusion and two triangular faces representing the lower and upper crust. For the sake of simplicity, although convolution-surface extraction algorithms exist [MS98, ZBQC13], we mesh a sphere-mesh (if needed) by contouring the signed distance to the sphere-mesh using a marching cube algorithm.

Rendering The sphere-mesh primitives are rendered efficiently on the GPU using the geometry shader [dTL08], without any surface extraction or conversion to triangle meshes.

4.4 Algorithm Overview

We now describe how to approximate, at a desired level of detail, an input surface mesh with a sphere-mesh, using our spherical quadric error metric to tailor a bottom-up decimation algorithm. Since a sphere-mesh models a surface as the outer boundary of the interpolation of its spheres, an ideal input to our approximation algorithm is a closed orientable surface. Nonetheless, as shown in Section 4.8, our approach can deal with flawed input including holes and non-manifold edges.

Similar to [GH97], we reduce the input mesh by collapsing its edges iteratively in a greedy fashion. We order the reduction operations by their cost $\mathbf{Q}_I(\mathbf{s})$, with I being the connected set of vertices collapsed altogether, and $\mathbf{s}$ being the sphere approximating the region.

When considering the collapse of an edge $[u,v]$ of the mesh, we create a quadric $\mathbf{Q}_{uv} = \mathbf{Q}_u + \mathbf{Q}_v$, find the sphere that best approximates the constructed region $\mathbf{s}_{uv} = argmin_{\mathbf{s}}\{\mathbf{Q}_{uv}(\mathbf{s})\}$, and set the corresponding collapse cost c_{uv} of $[u,v]$ to $\mathbf{Q}_{uv}(\mathbf{s}_{uv})$. The suggested edge-collapse $[u,v] \rightarrow \mathbf{s}_{uv}$ is then put into a priority queue $\mathcal{Q}$ with its associated cost c_{uv}.

At first, the priority queue $\mathcal{Q}$ is initialized with all possible edge-collapses. When pruning the best element $[u,v] \rightarrow \mathbf{s}_{uv}$ from $\mathcal{Q}$, the edge $[u,v]$ is collapsed and a new vertex $\mathbf{s}_{uv}$ is created with

the corresponding quadric $\mathbf{Q}_{uv}$. All possible edge-collapses with the neighbors of the new vertex are put into the queue and former neighboring ones are removed. The algorithm stops when the number of vertices to delete is reached.

Additionally, we prevent edge-collapses that result in the inversion of the orientation of the triangles that are involved in the operation similarly to Garland and Heckert [GH97].

4.5 Spherical quadric error metrics

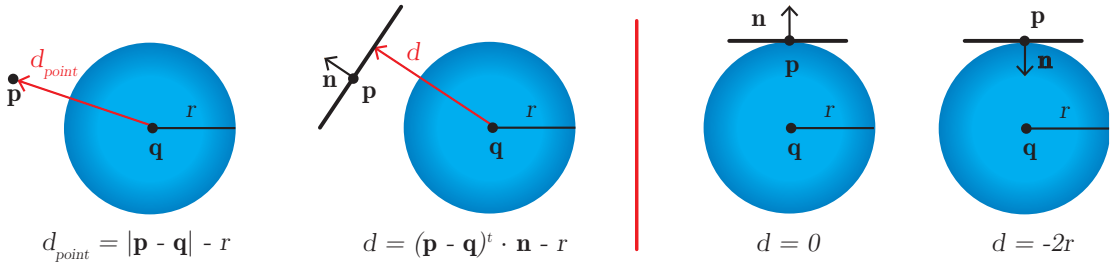
A key component of our approach is the definition of its underlying geometric metric, the *spherical quadric error metric* (SQEM). We denote $S(\mathbf{q},r)$ the sphere S of center $\mathbf{q}$ and radius r, and $\{\mathbf{p},\mathbf{n}\}^{\perp}$ the plane that is orthogonal to $\mathbf{n}$ and intersects $\mathbf{p}$:

$$\{\mathbf{p},\mathbf{n}\}^{\perp} \equiv \{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{n}^T \cdot (\mathbf{p}-\mathbf{x}) = 0\}.$$

The SQEM is based on the signed distance $d(S(\mathbf{q},r),\{\mathbf{p},\mathbf{n}\}^{\perp})$ from the sphere $S(\mathbf{q},r)$ to the oriented plane $\{\mathbf{p},\mathbf{n}\}^{\perp}$:

$$d(S(\mathbf{q},r),\{\mathbf{p},\mathbf{n}\}^{\perp}) = (\mathbf{p}-\mathbf{q})^T \cdot \mathbf{n} - r \tag{4.1}$$

This distance differs from the classical distance between an *unoriented* point $\mathbf{p}$ and a sphere, which is equal to $|\mathbf{p}-\mathbf{q}|-r$. It also takes into account the orientation of the plane. Figure 4.6 depicts the difference between the classical distance from a sphere to a point (a) or from a sphere to an oriented plane (b) and shows the importance of the orientation of the plane (c).



a) distance from a sphere to a point b) distance from a sphere to a plane c) importance of the orientation of the plane

Figure 4.6 – Differences between the distance from a sphere to an unoriented point and to a plane (left). The signed distance from a sphere to a plane depends on the plane orientation.

To ease the reading, we represent a sphere $S(\mathbf{q},r)$ as a vector in $\mathbb{R}^4$ by writing $\mathbf{s} = (\mathbf{q},r)$, and we note $\bar{\mathbf{p}} = (\mathbf{p},0)$ and $\bar{\mathbf{n}} = (\mathbf{n};1)$ two vectors in $\mathbb{R}^4$. Using this notation, the distance defines in Equation (4.1) becomes:

$$d(S(\mathbf{q},r),\{\mathbf{p},\mathbf{n}\}^{\perp}) \equiv d(\mathbf{s},\{p,n\}^{\perp}) = \bar{\mathbf{n}}^T \cdot (\bar{\mathbf{p}}-\mathbf{s}).$$

The spherical quadric error metric $SQEM_{p,n}(\mathbf{s})$ is the *squared* distance from the plane $\{\mathbf{p},\mathbf{n}\}^{\perp}$ to a variable sphere $\mathbf{s}$:

$$SQEM_{p,n}(\mathbf{s}) = d(\mathbf{s},\{\mathbf{p},\mathbf{n}\}^{\perp})^2.$$

Thus, it is a quadric with respect to $\mathbf{s}$,

$$\begin{aligned} SQEM_{p,n}(\mathbf{s}) &= d(\mathbf{s}, \{\mathbf{p}, \mathbf{n}\}^{\perp})^2 \\ &= (\bar{\mathbf{n}}^T \cdot \bar{\mathbf{p}} - \bar{\mathbf{n}}^T \cdot \mathbf{s})^2 \\ &= (\bar{\mathbf{n}}^T \cdot \mathbf{s})^2 - 2(\bar{\mathbf{n}}^T \cdot \bar{\mathbf{p}})(\bar{\mathbf{n}}^T \cdot \mathbf{s}) + (\bar{\mathbf{n}}^T \cdot \bar{\mathbf{p}})^2. \end{aligned}$$

Note, that the squared value of the distance from a point to a sphere (i. e. $(|\mathbf{p}-\mathbf{q}|-r)^2$) cannot be expressed as a quadric with respect to $\mathbf{s}$.

Finally the SQEM boils down to:

$$SQEM_{p,n}(\mathbf{s}) = \mathbf{Q}(\mathbf{s}) = \frac{1}{2}\mathbf{s}^t\ \mathbf{A}\ \mathbf{s} - \mathbf{b}^T \cdot \mathbf{s} + c. \tag{4.2}$$

with $\mathbf{A} = 2\bar{\mathbf{n}} \cdot \bar{\mathbf{n}}^{\mathbf{t}} = 2 \left[\begin{array}{c|c} \mathbf{n} \cdot \mathbf{n}^t & \mathbf{n} \\ \hline \mathbf{n}^t & 1 \end{array}\right] \in \mathscr{S}^4$, $\mathbf{b} = 2(\mathbf{n}^T \cdot \mathbf{p}) \left[\begin{array}{c} \mathbf{n} \\ \hline 1 \end{array}\right] \in \mathbb{R}^4$ and $c = (\mathbf{n}^T \cdot \mathbf{p})^2 \in \mathbb{R}$.

We refer to such a quadric $\mathbf{Q}$ by writing its components $\mathbf{Q} \equiv (\mathbf{A}, \mathbf{b}, c) \in \mathscr{S}^4 \times \mathbb{R}^4 \times \mathbb{R}$ explicitly. The sum of two quadrics $\mathbf{Q}_1 \equiv (A_1, b_1, c_1)$ and $\mathbf{Q}_2 \equiv (A_2, b_2, c_2)$ is naturally defined by summing up their different components: $\mathbf{Q}_1 + \mathbf{Q}_2 \equiv (\mathbf{A_1} + \mathbf{A_2},\ \mathbf{b_1} + \mathbf{b_2},\ c_1 + c_2)$. Thus, to compute the squared distance from a sphere to a set of planes we only need to sum the quadrics defined by each of the individual planes of the set. Similarly, the multiplication of a quadric by a scalar is computed by multiplying each component: $\lambda(\mathbf{A}, \mathbf{b}, c) \equiv (\lambda\mathbf{A}, \lambda\mathbf{b}, \lambda c)$. It follows that:

$$\begin{aligned} (\mathbf{Q}_1 + \mathbf{Q}_2)(\mathbf{s}) &= \mathbf{Q}_1(\mathbf{s}) + \mathbf{Q}_2(\mathbf{s}) \\ (\lambda\mathbf{Q})(\mathbf{s}) &= \lambda\mathbf{Q}(\mathbf{s}). \end{aligned}$$

4.6 Approximation Algorithm

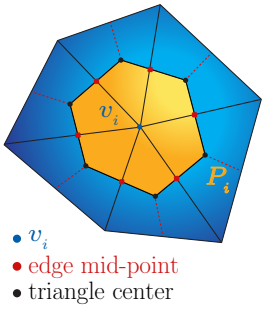
4.6.1 Shape approximation

Our goal is to partition the input mesh into regions I_k (sets of vertices), such that each region geometry P_{I_k} is approximated by a sphere $\mathbf{s}_k$ and the integral of the squared distance from the mesh to its approximation is minimized. The cost of such a partition is

$$\mathfrak{C}(\{I_k, \mathbf{s}_k\}_k) = \sum_k \int_{\xi \in P_{I_k}} d(\mathbf{s}_k, \{\mathbf{p}_\xi, \mathbf{n}_\xi\}^{\perp})^2 d\sigma_\xi. \tag{4.3}$$

Note that since we target shape approximation, this energy does not enforce the spheres to remain strictly inside the shape.

We equip each vertex v_i of the input mesh with its so-called barycentric cell P_i, depicted in the opposite figure. P_i is given by one third of the adjacent triangles $T_1(v_i)$ of the vertex.



We define the squared $\mathscr{L}_2$ distance from a sphere $\mathbf{s}$ to P_i as the integral over P_i of the squared distance to $\mathbf{s}$:

$$d(\mathbf{s}, P_i)^2_{\mathscr{L}_2} = \int_{\xi \in P_i} d(\mathbf{s}, \{p_\xi, n_\xi\}^{\perp})^2 d\sigma_\xi$$

The squared distance to the sphere is constant on a triangle, since all its oriented points describe the same plane, and on an adjacent triangle t_j of v_i it is equal to $\mathbf{Q}_{t_j}(\mathbf{s})$ (Equation (4.2)). Thus, the squared distance from the barycentric cell P_i to a sphere $\mathbf{s}$ is simply given by a weighted sum of the spherical quadrics $\mathbf{Q}_{t_j}$ of the triangles that are adjacent to the vertex v_i:

$$d(\mathbf{s},P_i)^2_{\mathscr{L}_2} = \sum_{t_j \in T_1(v_i)} \frac{area(t_j)}{3} \mathbf{Q}_{t_j}(\mathbf{s}) \triangleq \mathbf{Q}_i(\mathbf{s}).$$

Each region P_{I_k} being defined as the union of the barycentric cells of its vertices I_k, the squared distance from a sphere $\mathbf{s}$ to P_{I_k} is given by summing up the different squared distances:

$$d(\mathbf{s},P_{I_k})^2_{\mathscr{L}_2} = \mathbf{Q}_{I_k}(\mathbf{s}) = \sum_{i \in I_k} \mathbf{Q}_i = \sum_{i \in I_k} d(\mathbf{s},P_i)^2.$$

Finally, $\mathbf{Q}_{I_k}$ is the sum of the spherical quadrics of the barycentric cells of the vertices in the set I_k.

Using this notation, the cost of a partition $\{I_k,\mathbf{s}_k\}_k$, defined as the squared $\mathscr{L}_2$ distance from the input surface to the set of spheres, is

$$\mathfrak{C}(\{I_k,\mathbf{s}_k\}_k) = \sum_k \mathbf{Q}_{I_k}(\mathbf{s}_k).$$

Thus, during the decimation process, we obtain at each step a partition of the input mesh into k regions, with k being the number of vertices of the current sphere-mesh level. At the beginning, each vertex is equipped with its barycentric cell, then the successive collapses merge pairs of regions.

4.6.2 Quadric Minimization

To find the sphere that best approximates a given region we need to minimize the cost of collapsing the region into a sphere. This cost is represented by a spherical quadric $\mathbf{Q} \equiv (\mathbf{A},\mathbf{b},c)$ which has a global minimum, since $\mathbf{A}$ is a symmetric positive semi-definite matrix by construction. However, this minimum may not be unique.

We limit ourselves to the set of spheres that have a positive radius, since spheres with a negative radius describe concavities, and are located outside the object as depicted in Figure 4.7. Moreover, we also bound the maximum radius size, since the SQEM of a single plane can be minimized by a sphere of arbitrary large radius (our radius bound strategy is detailed later in Section 4.6.3). Consequently, the minimization of the quadric is performed in the subspace $\mathbb{R}^3 \times [0;R]$.

When looking for the minimizer of the cost $\mathbf{Q}_{uv} \equiv (\mathbf{A},\mathbf{b},c)$ of collapsing an edge $[u,v]$, several cases need to be considered.

$\mathbf{A}$ is invertible. Since $\mathbf{Q}(\mathbf{s})$ is a quadratic form, it has a global minimum which occurs when $\partial\mathbf{Q}/\partial x = \partial\mathbf{Q}/\partial y = \partial\mathbf{Q}/\partial z = \partial\mathbf{Q}/\partial r = 0$. Since the gradient of $\mathbf{Q}$ is

$$\nabla\mathbf{Q}(\mathbf{s}) = \mathbf{A}\mathbf{s} - \mathbf{b},$$

the global minimizer of the quadric is given by $\bar{\mathbf{s}} = \mathbf{A}^{-1}\ \mathbf{b}$. When the global minimizer $\bar{\mathbf{s}} \in \mathbb{R}^4$ is located outside the restricted subspace (i. e. when $\bar{r} < 0$ or $\bar{r} > R$), the minimizer is found in the corresponding hyper-plane (either $r = 0$ if the radius is negative, or $r = R$ if it is bigger than R).

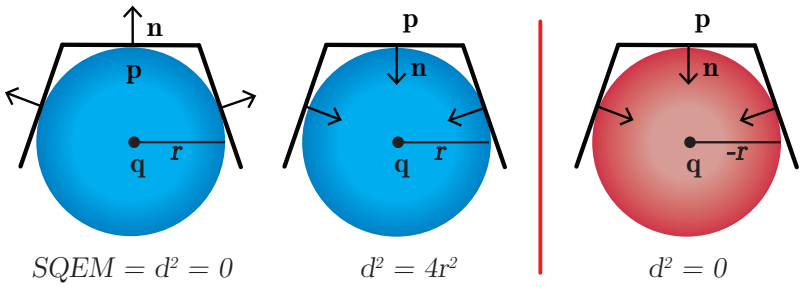


Figure 4.7 – Spheres minimizing the SQEM for a concavity (left) and a convexity (middle) while restricting our spheres to have positive radius. With no restriction the sphere minimizing the SQEM for a convexity is outside the object and has a negative radius (right).

Otherwise this minimizer would have a local neighborhood entirely contained in $r \in [0,R]$, and would be therefore a local minimum, which is impossible since a non-degenerate quadric has exactly one local, hence global, minimum.

A is not invertible. The SQEM can be degenerated, for instance if it represents a set of planes parallel to a line or a planar region. In this case, we approximate the region with a sphere which center lies along the edge $[u,v]$, still restricting the radius to be in $[0;R]$. That is, the minimizer is found in the domain $[\mathbf{q_u};\mathbf{q_v}] \times [0;R]$.

We denote $\mathbf{w}$ the vector $\mathbf{q_u}\vec{\mathbf{q}}_\mathbf{v}$; the center of the minimizer sphere can be expressed as $\mathbf{q} = \mathbf{q_u} + \lambda \mathbf{w}$. Thus, our minimization problem is reduced to a smaller one, namely:

$$\mathbf{Q}(\mathbf{q},r) = \frac{1}{2}(\lambda,r)^t \ \tilde{\mathbf{A}} \ (\lambda,r) - \tilde{\mathbf{b}}^t \cdot (\lambda,r) + \tilde{c},$$

$$\text{with } \tilde{\mathbf{A}} = \left[\begin{array}{c|c} \mathbf{w}^t \ \mathbf{A}_{22}^{00} \ \mathbf{w} & \mathbf{A}_{32}^{30\ t} \cdot \mathbf{w} \\ \hline \mathbf{A}_{32}^{30\ t} \cdot \mathbf{w} & \mathbf{A}_{33}^{33} \end{array}\right] \in \mathscr{S}^2, \quad \tilde{\mathbf{b}} = \left[\begin{array}{c} \mathbf{b}_2^{0\ t} \cdot \mathbf{w} - \mathbf{w}^t \ A_{22}^{00} \ \mathbf{q_u} \\ \hline \mathbf{b}_3^3 - \mathbf{A}_{32}^{30\ t} \ \mathbf{q_u} \end{array}\right] \in \mathbb{R}^2$$

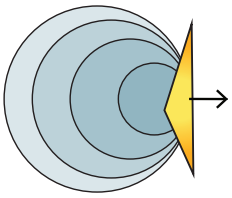
$$\text{and } \tilde{c} = (c - \mathbf{b}_2^{0t} \cdot \mathbf{q_u} + \frac{1}{2}\mathbf{q_u}^t \ \mathbf{A}_{22}^{00} \ \mathbf{q_u}) \in \mathbb{R}.$$

This energy is a 2-dimensional quadric w.r.t. (λ,r) that we want to minimize on the domain $[0;1] \times [0;R]$. Following the same reasoning, its global minimizer in $\mathbb{R}^2$ is $(\lambda,r) = \tilde{A}^{-1} \cdot \tilde{b}$. If this minimizer does not belong to the square $[0;1] \times [0;R]$, its restriction is once again located on its boundary (i. e. $\{\lambda = 0; r \in [0;R]\}$, $\{\lambda = 1; r \in [0;R]\}$, $\{r = 0; \lambda \in [0;1]\}$ or $\{r = R; \lambda \in [0;1]\}$). It results in a simple second order polynomial minimization in dimension 1.

If, for some reason (e. g. the region is planar), $\tilde{\mathbf{A}}$ is not invertible, we collapse the edge to its mid-point ($\lambda = 1/2$), and find the optimal value for the radius in $[0;R]$, which is a problem that is, this time, always properly conditioned.

4.6.3 Radius bound

Approximating a surface using a large sphere can be cumbersome if the region is not large enough itself. In this case, the resulting sphere can cover a large part of the outside of the object.



For example, an infinite number of spheres can fit a single plane (see side figure), since the only requirement is that this plane touches the sphere.

To solve this problem, we bound the diameter of the sphere by the directional width $\mathscr{W}$ of the region [GH01] i. e. , by the smallest extent of the region, when considering all possible directions.

To do so efficiently, we pre-sample the unit sphere uniformly with a fixed number of directions $\mathbf{k_j}$ (33 in our implementation including the 3 canonical axes). Each region P_u stores its interval $\mathfrak{I}_j(P_u)$ along all directions $\mathbf{k_j}$,

$$\mathfrak{I}_j(P_u) = [m_u^j; M_u^j] \text{ with } m_u^j = \min_{\mathbf{x}\in P_u}(\mathbf{x}^T \cdot \mathbf{k_j}) \text{ and } M_u^j = \max_{\mathbf{x}\in P_u}(\mathbf{x}^T \cdot \mathbf{k_j}).$$

The intervals of the union of two regions P_u and P_v can be obtained by iterating over all directions $\vec{k}_j$,

$$\mathfrak{I}_j(P_u \cup P_v) = [\min(m_u^j, m_v^j);\ \max(M_u^j, M_v^j)].$$

Since, at first, each region P_i is composed of only the barycentric cell of one vertex v_i, the radius bounds are initialized with

$$\mathfrak{I}_j(P_i) = [\min_{\mathbf{x}\in P_i}(\mathbf{x}^T \cdot \mathbf{k_j});\ \max_{\mathbf{x}\in P_i}(\mathbf{x}^T \cdot \mathbf{k_j})].$$

The directional width $\mathscr{W}(P)$ of the region P is approximated by $\tilde{\mathscr{W}}(P) = \min_j |\mathfrak{I}_j(P)|$. To allow coarser approximations at the first stages of the reduction, we set the maximum diameter of the sphere representing a region P_u to be slightly larger ($2R(P_u) = \frac{3}{2}\tilde{\mathscr{W}}(P_u)$ in practice).

This bounding heuristic shows several benefits. First, a sphere representing a planar region is constrained to be a point, as the directional width of a plane is zero. Second for convex shapes, a sphere is *likely* (although not guaranteed) to remain inside the shape when it is tangent to the geometry (as the directional width of a set equals that of its convex hull [GH01]). Finally, this heuristic proved empirically to be more efficient than other alternatives such as bounding boxes, either axis-aligned or aligned according to the principal component analysis of the region.

4.6.4 Mesh data structure

At each step of the decimation, the location and radii of the collapsed and created spheres are recorded, as well as the set of edges and/or triangles that are deleted or created. Thus, the connectivity of the resulting mesh is directly induced by the input connectivity and the set of successive edge-collapses. Notice than even when starting from a manifold input mesh the edge collapse process is likely to generate non-manifold edges.

All the structural events are stored using the data structure proposed by De Floriani et al. [DFMPS04]. Their structure allows to encode a hierarchy of meshes with a minimum memory overhead while maintaining high performance when degenerating to non-manifold meshes. After the decimation process, the user can navigate in real time in the sphere-meshes hierarchy to find the desired level of details.

4.6.5 Neighborhood enrichment

As for all decimation-based simplification methods, collapsing two vertices of the mesh $\mathbf{v_a}$ and $\mathbf{v_b}$ which do not share an edge, is often useful if they are close enough in the original mesh. Following Garland and Heckbert [GH97], we allow the contraction of a pair of vertices if $|\mathbf{v_a} - \mathbf{v_b}| < \varepsilon$ in addition to edges contraction. This heuristic induces a topological change as two disconnect regions become connected. However this topological simplification is often needed when reaching coarse levels of simplification. This strategy is especially effective if the shape contains large opposite regions that should be collapsed together. For example, large shell-like parts such as the chair model or the wings of the Pegaso in Figure 4.12. The parameter ε is intuitive and part of common decimation-based simplification frameworks.

4.6.6 Importance-driven distribution

In the context of shape abstraction, it is common to allow the user to control the *relative importance* of the features during the simplification process. We parameterize our approximation process by introducing a weighting kernel K_σ in the definition of the cost of a partition (Equation (4.3)):

$$\mathfrak{C}_\sigma(\{I_k, \mathbf{s}_k\}_k) = \sum_k \int_{\xi \in P_{I_k}} K_\sigma(\xi) d^2(\mathbf{s}_k, \{p_\xi, n_\xi\}^\perp) d\sigma_\xi \tag{4.4}$$

In this equation, K_σ describes the respective importance of all points on the manifold. When $K_\sigma = 1 \; \forall \xi$, the formulation of the cost of a partition reduces to the original one. Now, to compute the quadric $\mathbf{Q}_i$ of the vertex v_i we take the integral of the *kernel* into account:

$$\mathbf{Q}_i = \sum_{t_j \in T_1(v_i)} \left(\int_{\xi \in P_i \cap t_j} K_\sigma(\xi) d\sigma_\xi \right) \mathbf{Q}_{t_j}.$$

In Figure 4.8, we present results obtained when setting importance kernels based on the total curvature ${\kappa_1}^2 + {\kappa_2}^2$, which naturally favors highly-protruding geometry ($K_\sigma(\xi) = 1 + \sigma \; BBD^2 \; (\kappa_1(\xi)^2 + \kappa_2(\xi)^2)$,with BBD the bounding box diagonal of the model).

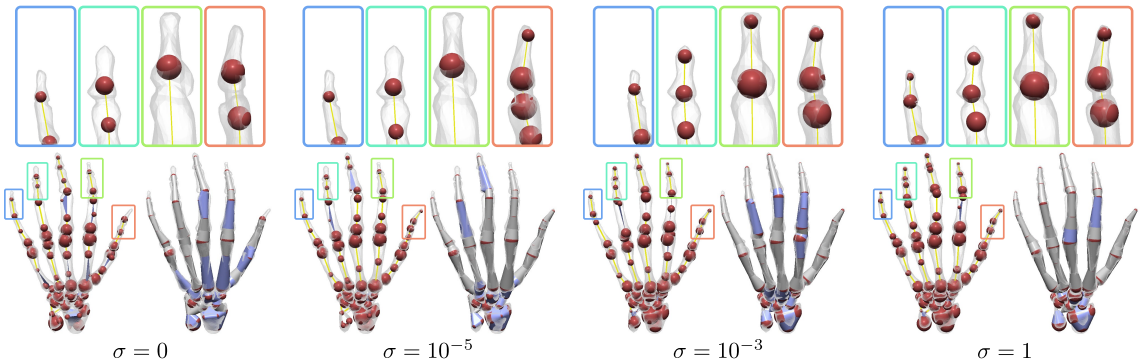


Figure 4.8 – **Influence of the *importance* parameter σ:** Fine details are better preserved as we increase the influence of the total curvature kernel. When σ is null (left), the results correspond to the standard remeshing introduced in Section 4.6.1. All sphere-meshes have 77 spheres.

Other properties, e. g. local feature size [AB99] or conformal factor [BCG08], could be used as importance kernels.

4.7 Deformation Control

Beyond shape approximation, a sphere-mesh degenerates naturally into an internal structure – eventually into a skeleton in tubular regions – which is a convenient metaphor for a number of interactive shape modeling applications. Alternatively to skeletons and cages, a sphere-mesh can for instance be used as an automatic high-level structure for controlling a shape.

In this section, we explain how to tie a mesh to its sphere-mesh and use the latter to interactively control the deformation of the former in a multi-resolution fashion.

4.7.1 Overview

Given a mesh, its sphere-mesh and a skinning machinery, we compute a skinning of the mesh to establish a relationship between the mesh and its sphere-mesh. Then, we provide the user with interactive *control primitives* in the form of spheres, edges and triangles from the sphere-mesh to deform the original mesh. The mesh geometry is updated in real time according to the skinning and the current sphere-mesh layout i.e., translations, rotation and scaling prescribed by the user on the primitives. Moreover, the user can instantly change the editing scale by navigating the sphere-mesh hierarchy (see Figure 4.11). After each deformation, the sphere-mesh is updated to fit the deformed geometry.

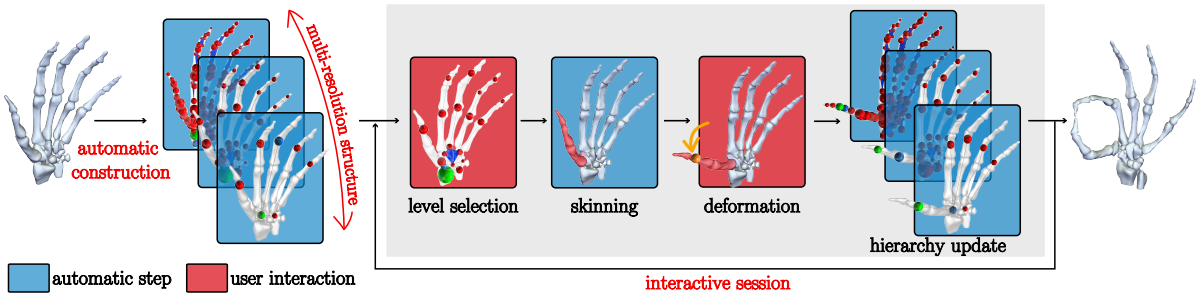


Figure 4.9 – Overview of a deformation session

Figure 4.9 depicts a deformation session. First, the multi-resolution structure is automatically constructed. Then, the user browses the hierarchy and interactively performs deformations at prescribed level-of-details. The skinning weights are automatically computed and the hierarchy is updated to fit the deformed model. This framework, as well as providing an *automatic* multi-resolution control structure (while most deformation cages are constructed manually for instance), combines interesting properties of several alternative methods:

- as with skeleton-based systems, elongated parts (e.g., arms, legs) can be bent by simply setting rotations on selected spheres,
- as with cages [LLCO08], the volume of a region can be smoothly controlled, using spheres radii,
- as with multi-resolution mesh editing [ZSS97], the user can quickly go from large scale deformation to tuning fine details.

The good behavior of sphere-meshes at extreme simplification levels is a key feature here.

4.7.2 Skinning

The skinning machinery defines the relationship between the mesh vertices and the rig structure, i. e. the sphere-mesh. As for skeletons, the skinning weights represent the impact that each sphere-mesh primitive has on a particular vertex. Several properties are required for these weights, especially locality, smoothness and partition of unity.

To illustrate the use of sphere-meshes as control structures, we implemented a deformation machinery based on the work of Baran and Popović [BP07]. More precisely, we define the weight w_i^j of the vertex i w.r.t. the control primitive j as the solution to the following system:

$$(\Delta + H) \cdot w^j = H \cdot p^j$$

where Δ is the discrete surface Laplacian with cotangent weights, w^j is the n-dimensional vector of weights w_i^j for all vertices i. $H = \lambda I_n$ is a diagonal matrix with λ being a value describing the diffusion of the weights over the mesh (the smaller λ, the bigger the diffusion) and p^j is a vector such that

$$p_i^j = \begin{cases} 1 & \text{j is the closest control primitive to vertex i,} \\ 0 & \text{otherwise.} \end{cases}$$

The approximation algorithm gives us a clustering of the original mesh where each cluster is approximated by a sphere. Thus to compute p^j, we limit the search of the closest primitive to the ones adjacent to the sphere to which the vertex corresponds in this clustering.

The result of this linear system is a set of weights w_i^j that sum up to 1 for all vertices ($\sum_{j \in n} w_i^j = 1$) [BP07]. The left hand-side matrix of the linear system $\Delta + H$ is sparse and independent of j. Therefore, we can factorize the system once and solve it iteratively over each primitive j to obtain the weights of the whole input mesh with respect to all the primitives.

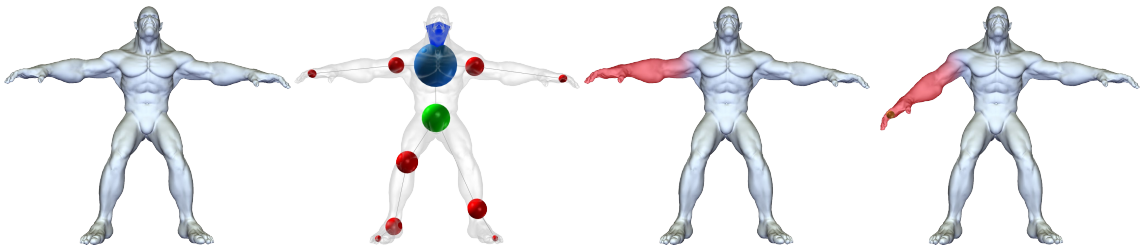


Figure 4.10 – **Sphere-mesh-based deformation:** starting from a surface mesh (left), a sphere-mesh is automatically computed (middle left) and the input mesh is skinned to its elements (middle right). The user can then manipulate the sphere-mesh to smoothly deform the input (right).

As a result, the user can easily define smooth deformations on the mesh by applying rigid transformations on the primitives of its sphere-mesh approximation (see Figure 4.10). Although the skinning method of Baran and Popović works well in our experiments, any alternative skinning scheme may be used with our structure, thus providing a multi-resolution control interface for all linear blend skinning (LBS) techniques.

4.7.3 Multi-resolution free-form deformation

To turn the previous framework into a multi-resolution one, we record, at each edge-collapse of the initial sphere-mesh approximation, the positions and radii of the two collapsed vertices and of the resulting one, as well as the set of edges and triangles that were created or deleted (see Section 4.6.4). We structure these events in a binary tree so that the user can navigate through the so-defined hierarchy to gather the desired control structure intuitively, i. e. one level-of-detail sphere-mesh.

At deformation time, the user simply applies rigid transformations to the individual primitives of the sphere-mesh at a chosen level of detail and gets a real-time feedback of the induced smooth deformation on the original high-resolution mesh (see Figure 4.11). Since the left hand side of the linear system $(\Delta + H)$ does not depend on the control structure, it does not need to be re-factored, and the update of the whole structure as well as the computation of new weights can be performed in real-time. Each time the user switches the current editing level of detail, the spheres in the hierarchy are updated bottom-up, computing the new SQEM $\mathbf{Q}_i$ at each level of the hierarchy. This update step is necessary to ensure that the entire sphere-mesh hierarchy fits the deformed geometry of the mesh, and not the initial one.

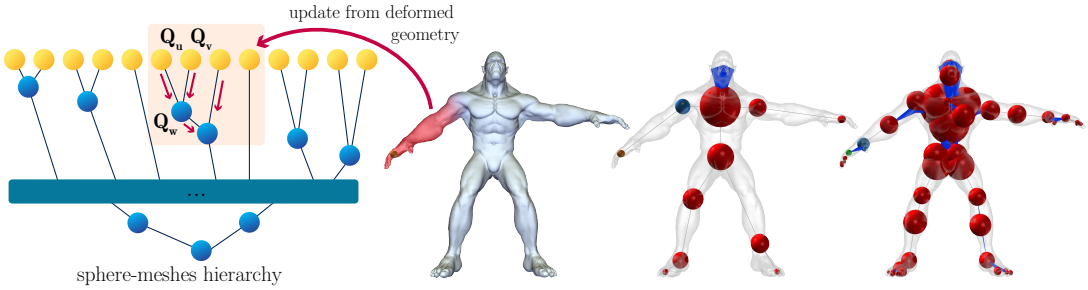


Figure 4.11 – **Sphere-mesh update**: after a given deformation, the hierarchy is updated in a bottom-up fashion to find the new positions of the spheres according to the given deformation. In the sphere-mesh hierarchy the yellow dots represent the input vertices, while the blue dots represent the sphere-mesh vertices.

In practice, we start by computing the new SQEM for each leaf (i. e. , mesh vertex) of the tree based on the current geometry of the mesh. For all recorded collapses (i. e. , internal tree node) $[u,v] \rightarrow w$, the sphere $\mathbf{s}_w$ of the parent w is obtained by minimizing children quadrics, i. e. $\mathbf{Q}_w = \mathbf{Q}_u + \mathbf{Q}_v$, and $\mathbf{s}_w = argmin_{\mathbf{s}} \mathbf{Q}_w(\mathbf{s})$ (see Figure 4.11). This tree update is linear in the number of input vertices and is performed in a few milliseconds on models made of several hundreds of thousands vertices, as only the geometry of the spheres requires an update, while the topology of the initial sphere-mesh remains unchanged.

4.8 Results

We implemented our shape approximation method in C++ and report performances on an Intel Core2 Duo running at 2.5 GHz with 4GB of main memory. The entire algorithm is controlled by one parameter, the target number of spheres and σ, that is set to 1 unless otherwise mentioned.

Figure 4.12 – **Sphere-mesh approximation results** on various meshes at different levels of simplification. Results obtained with $\sigma = 1$. The number of spheres of the approximation is shown between the parentheses.

4.8.1 Performances

In Figure 4.12, we present the results of our automatic shape approximation method for different input models, covering a wide range of geometric features, topology and quality. For all examples but the last, we show the input shape, the resulting sphere-mesh structure and the shape approximation emerging from the linear interpolation of the spheres along the structure. Even under a drastic approximation restricted to tens of spheres, each sphere-mesh succeeds at capturing, in an adaptive manner, the essence of the shape. This becomes even clearer when displaying the interpolated sphere-mesh geometry: spherical, conic and cylindrical components are quickly captured with the sphere-mesh, even in the presence of numerous fine scale features. We can also observe a number of cases where a tubular structure would not properly capture the shape at coarse scales (e. g. *Chair*, *Elk* and *Vase* models), highlighting the usefulness of polygons on top of edges in the sphere-mesh representation.

Our approach robustly handles fine components (e. g. *Flamingo* model), complex topologies (e. g. *Moebius* model) and non-uniformly distributed geometric structures with rapidly varying sizes (e.g., *Sea horse* model). Indeed, one desirable property of extreme approximation methods is the ability to ignore small structures to quickly abstract a complete shape component with a single primitive. In this context, we can observe that near spherical components are promptly captured with a single sphere (e. g. *Elk* and *Elephant* models), while near tubular structures are modeled with edge chains (e. g. , arms and legs). The last row of Figure 4.12 (*Pegaso* model) illustrates the natural multi-resolution structure that comes with our approach, with smaller components emerging progressively while reaching finer level-of-details. We also performed experiments on pathological cases. The poor quality *Camel grid* is smoothly approximated at coarse scale with a sphere-mesh. The *Half bear* model illustrates how the algorithm behaves for incomplete data sets (right part of the model). The shape approximation quality is not damaged in the regions where the input is complete.

Additionally, we analyze the influence of noise in Figure 4.13. Although the global structure of the approximation is preserved when adding more and more noise, it is the local volume approximation which suffers the most from the input quality degradation. Indeed, when the input is very noisy, the SQEM minimization leads naturally to zero radius spheres (i. e. , points), and thus becomes equivalent to QEM minimization in that case.

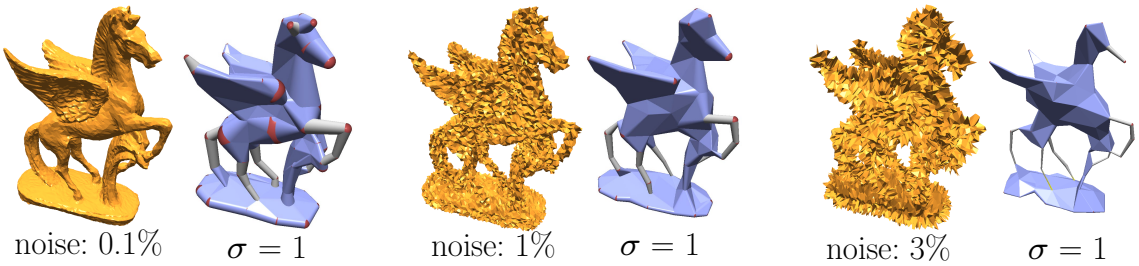


Figure 4.13 – **Noise sensitivity:** evolution of the sphere-mesh approximation with an increasing amount of noise (random per-vertex displacement expressed in percentage of the bounding box diagonal).

Table 4.1 shows the performances of the sphere-mesh algorithm. The initialization time comprises the mesh structure construction from a file and the initialization of the priority queue with all possible edge-collapses. The decimation is performed until no edge remains (computation of the whole multi-resolution structure).

Input Model (#V / #T)		Init. (ms)	Dec. (ms)	Sphere-mesh (#S / #E / #T)	H	M12	M21	QEM Simplification (#V / #T)	H	M12	M21
Boz.	(22774 / 45564)	1701	1844	100 / 22 / 146	**3.020**	**0.472**	**0.438**	100 / 194	11.703	1.519	0.591
Camel	(2020 / 4040)	122	113	50 / 19 / 34	**7.201**	**0.465**	**0.384**	50 / 96	12.477	1.071	0.576
Neptune	(28052 / 56112)	1714	2358	60 / 15 / 72	**2.650**	**0.384**	**0.388**	60 / 124	10.186	0.943	0.518
Centaur	(3401 / 6796)	209	212	50 / 17 / 42	**3.557**	**0.373**	**0.401**	50 / 94	15.626	1.629	0.523
Chair	(9935 / 19894)	568	745	35 / 16 / 22	**1.517**	**0.313**	**0.345**	35 / 72	14.974	1.404	0.364
Dancer	(7942 / 15884)	472	551	54 / 11 / 67	**1.352**	**0.166**	**0.182**	54 / 102	5.970	0.517	0.319
Gorilla	(1917 / 3830)	121	112	36 / 5 / 37	**6.109**	**0.530**	**0.446**	36 / 68	9.353	1.610	1.122
Elk	(5194 / 10388)	309	370	45 / 2 / 69	**2.560**	**0.273**	**0.289**	45 / 90	20.802	2.199	0.936
Hand	(11724 / 23464)	696	884	70 / 53 / 16	**1.980**	**0.324**	**0.352**	70 / 130	14.074	1.140	0.561
Flamengo	(26394 / 52839)	1403	1941	40 / 12 / 32	**2.837**	**0.417**	**0.479**	40 / 74	35.135	5.738	0.507
Sea monster	(39485 / 78966)	2210	3169	150 / 60 / 161	**6.081**	**0.367**	**0.364**	150 / 292	11.445	0.947	0.402
Elephant	(24955 / 49918)	1263	1818	45 / 6 / 68	**3.357**	**0.564**	**0.620**	45 / 90	6.371	1.038	0.684
Wolf	(3401 / 6796)	220	212	55 / 12 / 75	**3.618**	**0.380**	**0.405**	55 / 106	8.144	1.079	0.441
Fish	(7376 / 14748)	532	521	12 / 3 / 6	**4.272**	**0.762**	**0.735**	12 / 20	14.661	1.395	1.036
Sea horse	(162248 / 324524)	18859	16271	200 / 18 / 344	**1.324**	**0.257**	0.271	200 / 400	4.668	0.324	**0.231**
Moebius	(21126 / 42523)	1275	1615	700 / 316 / 366	**0.762**	**0.078**	**0.074**	700 / 1510	1.675	0.242	0.123
Raptor	(12908 / 25852)	1423	879	45 / 23 / 30	**4.310**	**0.428**	**0.425**	45 / 84	18.069	1.555	0.577
Vase	(51801 / 103598)	6442	5203	120 / 8 / 197	**2.093**	**0.321**	0.395	120 / 230	4.698	0.539	**0.265**
Camel grid	(5752 / 11508)	296	503	50 / 5 / 72	**4.199**	**0.900**	**0.629**	50 / 92	17.104	1.853	0.730
Half bear	(9202 / 17969)	530	642	60 / 5 / 66	14.784	**0.400**	2.559	60 / 80	**14.296**	0.939	**0.362**
Pegaso	(15319 / 30658)	899	1153	100 / 11 / 170	**2.554**	**0.381**	**0.393**	100 / 204	6.113	0.578	0.424
Pegaso	(15319 / 30658)	–	–	75 / 12 / 119	**4.676**	**0.485**	**0.479**	75 / 148	6.835	0.790	0.496
Pegaso	(15319 / 30658)	–	–	50 / 12 / 71	**4.973**	**0.621**	**0.600**	50 / 94	7.554	1.179	0.634
Pegaso	(15319 / 30658)	–	–	25 / 7 / 34	**6.128**	**1.237**	1.081	25 / 46	21.096	2.478	**1.056**

Table 4.1 – Performance and timings for our sphere-mesh approximation algorithm (models of Figure 4.12). #S / #E / #T: number of spheres, wire edges, and triangles. H: Hausdorff distance. M21: mean distance from the approximation to the original model. M12: mean distance from the original model to the approximation. All distances are expressed in percentages of the input model bounding box diagonal.

Worst case scenarios. Worst case scenarios for our approach are thin-shell models that contains large concave parts and disconnected components encompassing each other. At the final stages of the simplification, spheres fitting a large region with low curvature can bulge out from the shape (red boxes) due to the locality of the surface optimization process.

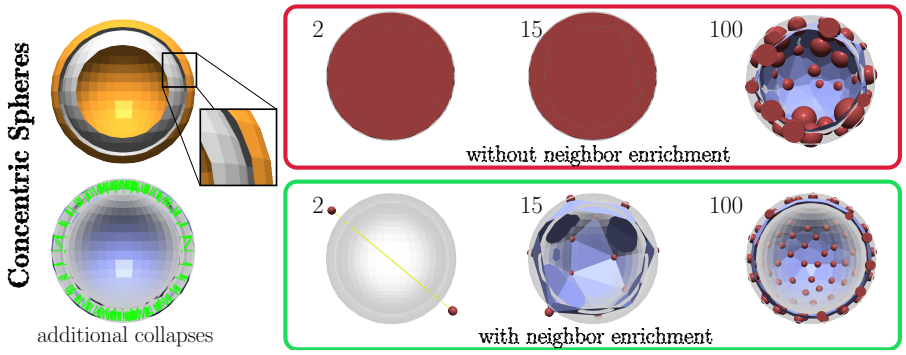


Figure 4.14 – **Results on thin shell models:** cross sections with back faces are shown in grey and additional potential collapses in green.

Figure 4.14 represents a thin-shell sphere with the outer sphere approximated by a single large sphere, which covers its inside entirely and ignores the inner concave sphere. A better sphere-mesh approximation of such a thin shell can again be obtained using the *neighborhood enrichment* strategy (green boxes). Collapses of vertices that are "facing" each other are unlikely to happen, since the SQEM accounts intrinsically for the normal orientation; we also prevent linking them

during the *neighborhood enrichment* based on their normals. Finally, a large number of spheres are required to approximate such shapes (15-100) because the sphere-mesh representation captures shapes with few connected convex components better.

4.8.2 Comparisons

Mesh decimation. We compare our method to QSlim [GH97] which implements a mesh simplification algorithm based on the *quadric error metric*. When a sufficient number of primitives is allowed, traditional triangle meshes simplifications approximate faithfully enough the input 3D object. However, as the number of primitives diminishes, meaningful parts completely vanish. Our sphere-mesh approximation captures similarly well the objects with a high number of primitives, but degrades gracefully to volumetric structures even when drastically decimated, preserving the main parts of the objects (as depicted in Figure 4.15).

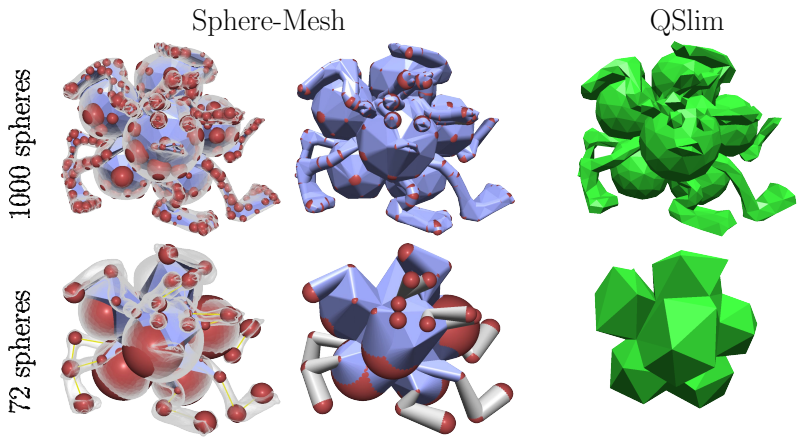


Figure 4.15 – **Comparison to polygonal simplification (QSlim [GH97]):** evolution of the approximation with a decreasing number of spheres (resp. vertices) for the sphere-mesh (resp. simplified mesh).

In Table 4.1, we report the approximation error of QEM simplifications, for the same number of primitives as with our SQEM method. Beyond the visual assessment, the approximation quality of our sphere-meshes clearly outperforms the one of decimated polygon meshes, both for Hausdorff and mean distances, for most examples. However, in the case of incomplete inputs (*Half bear* model), the mean distance from the approximation to the input surface (M21) is significantly higher using a sphere-mesh. Indeed, the sphere-mesh geometry interpolation tends to "fill" holed regions which, depending on the application, may be a benefit or a drawback.

Medial axis transform The medial axis transform (MAT) is not designed to represent shapes with the same number of primitives as sphere-meshes, which become volumetric structures only at very coarse simplification levels (see Figure 4.16).

Nevertheless, sphere-meshes and the MAT share geometric and structural similarities. First, the MAT is defined as well in terms of sphere interpolation over a non-manifold structure composed of triangles and edges. Second, the MAT can offer as well a multi-resolution description of the input shape, through a filtering process.

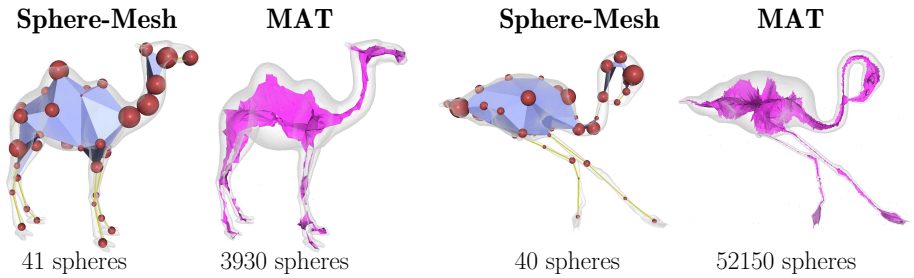


Figure 4.16 – **Visual comparison to the medial axis transform.** Sphere-meshes are obtained with $\sigma = 0$. Medial axes are extracted with Powercrust [ACK01].

Beside these similarities, there are at least two main differences between our approach and MAT (see Figure 4.16). First, sphere-meshes degenerate from surface to volumetric structures progressively and parts of the sphere-mesh geometry can remain homeomorphic to the input surface (see the body of *Flamengo*). Moreover, this transition from surface to volumetric structures arises in an adaptive manner (body of the *Flamengo* sphere-mesh compared to its legs). In contrast, the MAT is a purely volumetric structure. Second, although a filtering process can also provide a MAT-based multi-resolution description of the input shape, this comes as a natural side effect of our approximation technique. In our case, we simplify the input mesh directly, instead of simplifying a precomputed medial structure The MAT is not designed to reach the simplification levels we obtain and existing filtering techniques focus on denoising the medial axis rather than coarsening it.

Sphere skeletons In contrast to sphere-meshes, which target the approximation of an existing high resolution mesh (e.g., scanned object), sphere-skeletons such as Z-Spheres [Pix01] or B-Meshes [JLW10] are designed for interactive shape creation and facilitate the creation of coarse shapes from scratch. However, both representation models can be compared in terms of flexibility and expressiveness. To some extent, the sphere-mesh representation extends Z-Sphere and B-Meshes beyond tubular structures. The presence of polygons (in addition to edges) in the sphere-mesh topological structure helps modeling large flat regions which cannot be captured by tubular components (see the *chair* example in Figure 4.12). Using a sphere skeleton, flat regions would either be more complex to model or less accurate in terms of approximation.

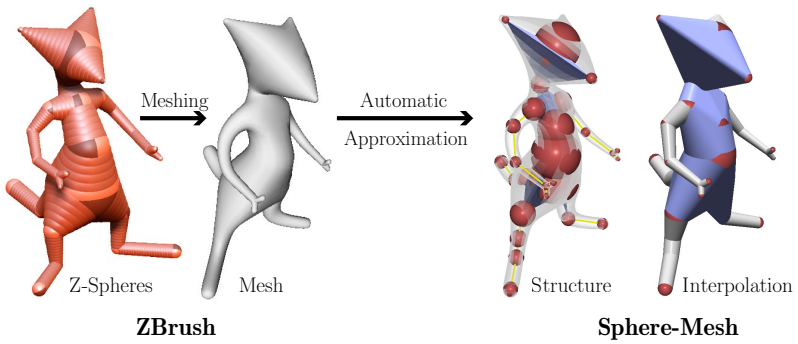


Figure 4.17 – **Comparison with sphere skeletons:** On the left, a sphere skeleton constructed with ZBrush and its associated mesh surface. On the right, a sphere-mesh (30 spheres) automatically computed from this surface.

Sphere skeletons methods often come with a specific surface extraction method, to pursue the modeling session with displacement painting for instance. Sphere-meshes allow the reversal of the pipeline, by recovering automatically a sphere-based structure from a detailed input surface (see Figure 4.17).

4.8.3 Control Structure for the deformation

Figure 4.18 shows few steps of an interactive modeling session, where the user navigate in the hierarchy to perform deformations at very different scales, from large deformation of the shoulders, arms, and head orientation, to small deformation of the fingers. The sphere-mesh is used as

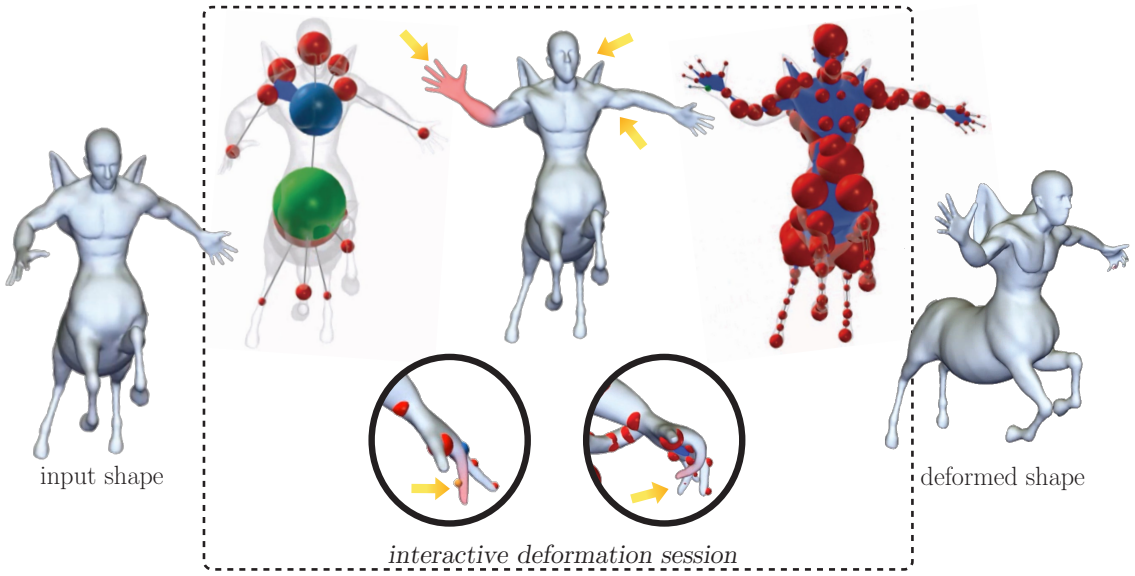


Figure 4.18 – **A modeling session** demonstrates the potential of the sphere-mesh as a control structure in an interactive deformation session.

a multi-resolution structure, and the user can easily navigate its hierarchy to find the prescribed levels of details. The automatic recomputation of the sphere-mesh levels to adapt the deformed geometry allows a smooth process where the user seamlessly navigates in the hierarchy and perform deformations.

4.9 Discussion

Representation

Currently, a sphere-mesh represents the surface as a base complex (vertices, triangles, ...) and a spatially-varying thickness (spheres radius), but the actual surface geometry is expressed as the linear interpolation of the spheres over the complex and is not directly transformable into e. g. a triangle mesh. “Closing the loop” and extracting a high quality mesh from the sphere-mesh boundary is an interesting direction for future work.

Incomplete data and open surfaces

As shown in the result section, incomplete surfaces with large holes often see their holes filled by the sphere-mesh geometry. Indeed, there is no explicit modeling of boundaries in the sphere-mesh representation and defining a restriction of the interpolated geometry to an open 2-manifold is left as future work.

Shape approximation

Currently, our definition of sphere-meshes does not allow to enforce, in a simple manner, that spheres remain strictly inside the shape or that the resulting mesh preserves the input's topology i.e., topological errors can occur at extreme simplification levels. Furthermore, sphere-meshes represent shapes with few connected convex components better, and models featuring large concave parts still require a large number of spheres to be modeled properly.

Optimization

Our approximation algorithm worked well on all the examples we tried but does not, unfortunately, provide the global minimizer. Indeed, our approximation strategy is limited by the fact that we fit spheres to regions, but do not optimize for the interpolation between the spheres, which is our major research direction for future work. In some cases, the interpolated sphere-mesh geometry can be a poor approximation of the input surface even if the spheres fit their respective regions correctly.

Bottom-up algorithms (e. g. [GH97]) typically do not optimize for the connectivity at each step, which is not optimal and could benefit, for instance, from intermediate try-and-test steps. However, this would require efficiently computing the input/sphere-mesh distance for each try, dramatically increasing the computational complexity of the algorithm. The partitioning is also computed by minimizing the one way $\mathscr{L}_2$ distance from the input mesh to its simplified geometry, and the results could be further improved by minimizing the Hausdorff distance instead. Lastly, we use a bottom-up reduction of the input mesh connectivity as the backbone of our approximation algorithm. Alternative mechanisms, such as variational or statistical sphere distributions may be derived to exploit our SQEM differently.

Deformation structure

As we have shown in Section 4.7, the sphere-mesh is an interesting metaphor for the deformation of an input shape. Its multi-resolution scheme is particularly useful and its volumetric behavior at extreme simplification levels is a key feature. The sphere-mesh can be used like a skeleton metaphor by translating the center of the spheres. While the modification of their radius allows to control the volume of a shape, similarly to cage-based deformations. However, when the volume of the original model is drastically changed, the sphere-meshes hierarchy does not approximate the geometry faithfully anymore, since we only update the position of the spheres after a deformation. A simple solution would be to regularly update their radius too. For now the user navigates in the hierarchy and the skinning weights are automatically recomputed for each level. Finding the weights in a multi-resolution fashion is an interesting direction for future work. Lastly, at fine

scales the sphere-mesh remains on the surface, and we believe that it could be a useful structure to compute variational deformations on the surface.

The sphere-mesh is currently built in an automatic fashion from a single input mesh. Thus, the deformation structure is simply inferred from the geometric properties of the mesh. This can lead to a poor abstraction if the geometry does no reflect accurately the motion range of the shape. To solve this issue, we could try to build a sphere-mesh from an animated input. In this case, the sphere-mesh would be well adapted to the motion range depicted in the animation (see [LD12] and [LD14b] for a similar line of work).

Another direction of research is to include the user in the automatic simplification loop. The user would then be able to add constraints and enforce the positions of some key spheres when needed, performing a trial-and-error process inside the optimization loop. This solution would allow a new tradeoff between the tedious construction of (multi-resolution) deformation structures and the user freedom to enforce a desired deformation range (not present in the original geometry alone).

4.10 Conclusion

In this chapter, we proposed a shape approximation algorithm based on a sphere interpolation over a mesh structure, for capturing complex shapes with a reduced set of connected spheres. The main technical contribution is the spherical quadric error metric (SQEM), whose minimization finds spheres representing best the tangent spaces of a set of polygons. We also proposed an algorithm for efficiently computing a sphere-mesh approximating a triangle mesh, with a natural multi-resolution structure and an explicit control on its feature sensitivity. We analyzed its performance on a diverse set of inputs, showing that a sphere-mesh approximation performs in general better than mesh decimation. Finally, we show that the sphere-mesh abstraction is a new spatial abstraction for shape modeling. Indeed, we use the sphere-meshes hierarchy as an automatic multi-resolution control structure for interactive free-form deformations and we believe that this metaphor could be enrich by further improvements.

Beyond the possible technical improvements discussed earlier, we believe that sphere-meshes can be further developed for shape processing and analysis methods, including reconstruction from point clouds, progressive compression, shape recognition and visualization scenarios.

Part II

Similarity-based Modeling

In the first part of this manuscript, we propose to use shape abstractions as useful primitives for interactive modeling tools. The two first chapters, while technically different, share the same pipeline. The spatial shape abstraction is computed as a preprocess and then used as control structure in interactive tools, either to navigate in a scene or to deform a shape. In this part, we focus on a different shape analysis topic: the detection of similarities. We demonstrate how efficient similarity detection methods can be interleaved with other processes to create new modeling applications.

In Chapter 5 we propose an editing framework based on similarity detection. We show how shape analysis tools can alleviate the user work in both selection and editing tasks, by proposing a framework where similar selections are automatically discovered from a reference one. In this chapter the shape analysis is performed as a joint work with the user interactions.

In Chapter 6 we propose a new shape abstraction with an embedded notion of similarities. To this end, we interleave a shape decimation and a similarity detection algorithms.

CHAPTER 5

SIMILARITY-BASED SELECTION & EDITING

5.1 Motivations

In the previous chapters, we introduce the notion of shape abstraction and use it as a useful pre-process in modeling applications. In this chapter, we aim at combining computations and interactions together, so that they are of mutual benefit. While most shape analysis methods consist of long offline processes, we rather use computational power and shape analysis tools to interactively improve the user interactions. We propose to study the interactive detection of similarities to alleviate the editing process for the user. The principal objective is to select a region on a shape, detect similar ones, and finally use this pool of similar regions to perform non local editing on all of them at the same time.

We divide the process in two different stages: the selection of reference and similar regions, and the non local editing. Figure 5.1 represents the pipeline performed to select a reference region and automatically detect similar ones.

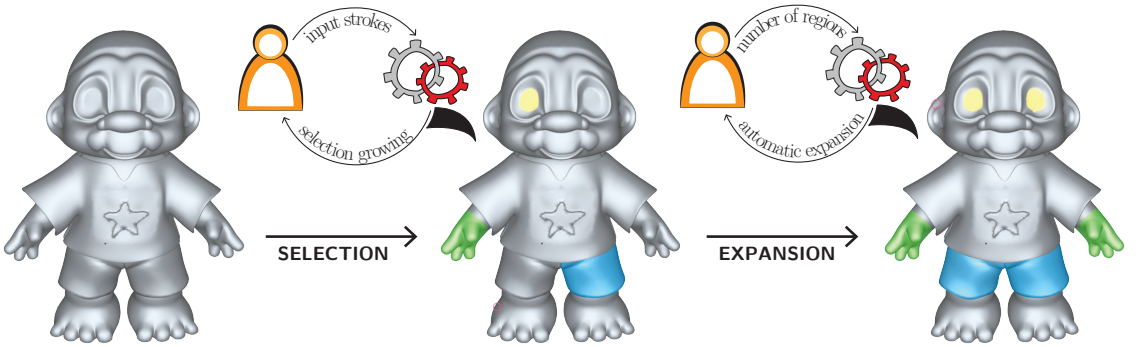


Figure 5.1 – ***SimSelect* general pipeline.** the user interactively selects and retrieves similar regions.

The selection process is one the basic, fundamental operations in interactive 3D modeling, but it remains a difficult and time-consuming task for any CG user, from casual graphics enthusiasts to highly skilled SFX designers.

Models are getting more and more complex, especially with the ability to capture real-world objects with high resolution geometric details. Current interactive modeling software aims at manip-

ulating these models. In particular, high-end 3D packages – such as Maya, 3DSMax or ZBrush – are widely used for post-processing acquired meshes or creating and deforming new ones. These packages have many digital surface tools, such as *detail control* (e. g. normal mapping, procedural displacement), *repairing* (e. g. hole filling, cleaning), *uv-control* (e. g. constraints positioning, alignment), *geometric signal modulation* (e. g. smoothing, sharpening) or *copy-pasting* (e. g. region cloning, component merging). All these tools can either be globally applied or, more often, used on selected portions of the input. While their automation has motivated a large portion of interactive geometry research over the last few years, the computational aspect of the selection process itself has emerged only recently as a key topic and it remains a long and tedious task.

More specifically, there are at least two main challenges when interactively defining a selection on a surface. The *accuracy* required to hand-track the boundary of a particular region and the *repetitiveness* one endures when it comes to the selection of multiple similar regions on the surface. Both problems become critical for captured geometry missing high-level structures.

For 2D image editing, the accuracy issue has been addressed by introducing new interaction metaphors, less demanding for the user and requiring only few approximate gestures to control an automatic content-aware selection process running on-the-fly. At any time, the so-defined selection can be refined *progressively*, as in the *Paint Selection* system [LSS09] (see Figure 5.2). Most of these tools are based on heuristics which assume, for instance, that the selection should be bounded by a coherent high magnitude gradient structure or that the selected region should exhibit particular statistics in its texture. They often exploit popular combinatorial or variational methods for the optimization step such as the graph cuts or the k-means algorithms. The repetitiveness issue has been addressed only recently. Yücer et al. [YJHS12] propose the editing of an entire image collection thanks to the automatic transfer of edits, performed using non-rigid alignment. More recently, Guerrero et al. [GAWJ15] detect similar regions on 2D strokes inputs along with their mapping, which allow them to transport edits such as texturing.

Figure 5.2 – Paint Selection

When moving to 3D geometry, the problem becomes harder. Firstly, depending on the scale, the notion of *"coherent"* selection may drastically change, from isolated components (e. g. a ring on a finger) which require to take into account the surface topology, to local bumpy patterns (e. g. a stamp on wax) which involve an analysis of the geometric signal. Secondly, as opposed to images, which decorrelate signal (color) from parameterization (pixels coordinates), 3D surfaces embed both notions in their positional field. Although intuitive semi-automatic methods have been introduced to separate a shape into several components, we notice in practice that most scenarios still require the user to spend a significant amount of time selecting each and every region she wants to edit, ultimately selecting all polygons to be processed with a brush, a lasso [SS10] or simple combinatorial criteria like a polygon strip selection [Aut]. However, we empirically observed that CG users often interact with a shape at three different levels by editing either:

i) a single piece in a large multi-components model,

ii) a large part where only the low frequency structures of the shape matter,

iii) a local region presenting a strong on-surface structure.

We base our approach on such a classification, and propose different tools to perform and automatically retrieve similar selections, depending on their type. The editing of a set of similar regions is also a complicated task, and we present possible solutions to transport edits as well as remaining open questions.

In the sections that follow, we first propose a new system for interactive selection on 3D surfaces, called *SimSelect*. Our basic idea is to classify selections according to the above classification, into three different types: *connected components*, *parts* and *patches*. We provide the user with a specific interaction metaphor and a specific automatic selection optimization for each of them. The type-aware selection process (Section 5.3) improves the accuracy of the selection. While the *expansion* procedure (Section 5.4) reduces the repetitiveness of the task, thanks to an automatic retrieval of selections similar to the current one (see Figure 5.1). Finally, we show that the similarity based selection is a first step toward non local editing and we present potential solutions to transpose edits to similar regions (Section 5.6).

5.2 Related Work

Interactive mesh segmentation: With classical interactive surface segmentation systems, the user typically draws rough strokes over the surface to initialize an automatic segmentation process. Based on the mandatory interaction, we can classify these techniques into two main families: *boundary-based* and *region-based* methods.

Boundary-based techniques ask the user to draw a coarse boundary of the selection by either tracing strokes along the desired boundary [MFL11, LLS+05, FKS+04] or across it [ZT10]. These strokes induce a small set of inside and outside constraints which are combined with a regularization prior – typically based on the surface curvature – to initialize an optimization procedure. In the optimization, all these relations can, for instance, be expressed as a linear system to solve or as weights on the edges of a graph to cut. Region-based techniques mainly differ in the user input, which is instead defined by brushing directly on the selection [FLL11], or by specifying explicitly inside or outside areas [JLCW06].

We refer the interested reader to [FML12] for a survey on interactive segmentation techniques, and to [TPT15] for a more general overview of segmentation methods (automatic or not).

Similarity detection for Interactive Editing: In Chapter 2 (Section 2.3), we give an overview of similarity detection techniques on 3D shapes. In this chapter we focus on interactive shape analysis for our selection, expansion and editing processes. Thus, we focus on techniques that are fast enough for interactive modeling processes.

Leifman and Tal [LT12, LT13] propose an approach to perform the colorization of a mesh with repetitive patterns based on few user strokes. In a nutshell, the user defines a small set of colored "scribbles" on pattern instances of the surface. The mesh is then smoothed out to preserve as best as possible the boundaries of the pattern while the smallest details are removed. On this smooth version of the mesh, all the vertices are classified using a supervised learning algorithm to determine the pattern they belong to. Then a few subset of classified vertices are used as the inputs of a global colorization algorithm on the original mesh.

Zelinka and Garland [ZG04] perform non local edits on a per-vertex basis. They compute a similarity map on the mesh based on the evaluation of a signal along geodesic fans. To be rotationally

invariant they align the descriptors using a brute force approach, i. e. finding the discrete rotation that minimizes the distance between descriptors. Several edits are proposed, especially the smoothing and the enhancing of vertices or their displacement along their normals. However, the mis-alignment in the local frames prevents the authors from performing more involved deformations, i. e. transformations with an important tangential displacement for instance. The similarity map is used to weight the amount of editing to perform. Given a reference vertex, the most similar vertices have a weight close to one and are fully edited, while the most dissimilar ones have a null weight and are not edited at all. A transfer function can easily be used to modify the weighting effect.

Maximo et al. [MPVF11] propose a system called SAMPLE to detect similar disk regions. They express a height-map on the region using Zernike moments to be rotationally invariant. Then, they compute a similarity map using a per-vertex descriptor. The edit is propagated to regions that have not already being edited and have a similar center. Given a reference and similar disk regions, a local frame $\{u, v, n\}$ is found for each patch and used to propagate the editing. Only two examples of non local edits are demonstrated in the paper, the first one replicates a parametrization domain on each similar regions, while the second one transports an editing mask. The main drawback of the technique is the fixed region size. To edit disk regions with different sizes the all process has to be recomputed. Also, the potential mis-alignment of the similar $\{u, v\}$ frames or the impact of local geometric differences on the editing are not addressed in the paper. Finally the technique is rotationally invariant, but not robust to changes in scale or isometric deformations.

Beyond particular applications, the performance of such systems are linked to their underlying *descriptors*, which capture the essence of a shape or a region, and come with a distance modeling the notion of similarity. Ideally, this descriptor should be compact, fast to extract, translation/rotation-invariant, robust to small scale variations (e.g., noise) and to partial matching. This list of desirable properties often leads to statistical models (e.g., multi-dimensional histograms) which offer a good compromise between accuracy and scalability.

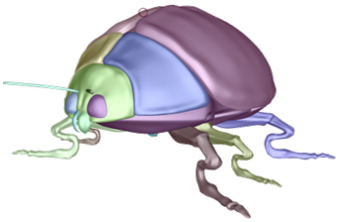
5.3 Reference Selection

The reference selection is a subset of the input polygons, in our implementation a selection is simply a connected set of triangles of the input mesh. We classify the selections in three categories. The user selects the tool associated to the desired selection type and performs the appropriate interaction, i. e. *clicking* components, *cutting* parts and *brushing* patches. The design of the interactions is inherently linked to the properties of each selection type in order to be intuitive to discover, fast to perform and to give enough freedom to the user when needed. In the following, we describe the algorithm we run for each case to define the subset of polygons belonging to the reference selection. All these algorithms are fast enough on dense polygon meshes to be run on-the-fly during the interaction, allowing an interactive trial-and-error process.

5.3.1 Connected component selection

At loading time, we build the adjacency graph of the mesh based on the polygons edges. Each triangle of the input mesh stores an integer representing the identity number of its connected component. At interaction time, the user clicks on a connected component to select it.

We gather all faces belonging to the component and add them to the reference selection. Although trivial, this selection type turned out to be quite important, since many meshes are composed of numerous components in our early experiments. For instance, in the side figure each color represents a different component on a ladybug model.



5.3.2 Part selection

Our part selection algorithm is largely inspired from the one defined by Zheng and Tai for their part-brush tool [ZT10]. The isolines of a harmonic field, guided and progressively refined by user strokes, are used as cutting boundaries for the selection of a part. Using an harmonic field to retrieve the boundary of a part is an interesting choice since each isoline of the field is a smooth connected loop dividing the mesh into two distinct parts.

Figure 5.3 – **Part Selection:** the user stroke induces a set of constraints (close up) and a harmonic field (left); the selection boundary is defined as the best harmonic field isoline, while the conformal factor (middle) helps to decide which side of the cut the selection is on (right).

The harmonic field is obtained by solving, in the least-squares sense, a Poisson equation $\Delta\Phi = 0$ with boundary constraints depending on the user strokes. Thus, the problem can be expressed as:

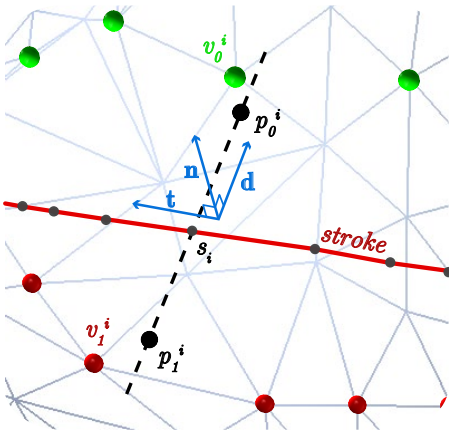
$$\begin{bmatrix} \mathbf{L} \\ \hline \mathbf{W_0 P_0} \\ \mathbf{W_1 P_1} \end{bmatrix} \Phi = \begin{bmatrix} \mathbf{0} \\ \hline \mathbf{W_0 B_0} \\ \mathbf{W_1 B_1} \end{bmatrix},$$

where $\mathbf{L}$ is a n-by-n matrix representing the mesh Laplacian with the cotan scheme (detailed in Section 2.2). $\mathbf{P_0}$ and $\mathbf{P_1}$ are positional constraints matrices of respective size c_0-by-n and c_1-by-n with c_0 and c_1 the number of 1- and 0-constraints defined by the user. $\mathbf{W_0}$ and $\mathbf{W_1}$ are diagonal matrices representing the weighting of the positional constraints. Finally, $\mathbf{B_1} = (1..1)^T$ and $\mathbf{B_0} = (0..0)^T$ are vectors of size c_1 and c_0 respectively. Similarly to [ZT10], we choose the best isoline of the harmonic field to be the cutting boundary of the part.

The system is factorized only once when the mesh is loaded. Then, we update it from the few dynamic constraints stemming from the user strokes during interaction. To do so, we use the factorization downdating/updating technique [XZCOX09], that allows us to solve the system interac-

tively, while the user is drawing the cut. We handle meshes with multiple connected components by setting 0-constraints for all the components which are not touched by the user strokes.

On the contrary to Zheng and Tai [ZT10], who request the user to draw strokes *across* the desired cut, we argue that locating them *along* the cut is a more natural metaphor because it mimics the *"slicing"* of the shape better. To build the harmonic field, we generate a set of inside and outside constraints for few vertices based on the user strokes, i. e. we set the lower part of the system $\mathbf{P_0}$ and $\mathbf{P_1}$. First, for each stroke point $\mathbf{s}_i$, we construct a right handed frame based on the normal $\mathbf{n}$ at $\mathbf{s}_i$ and on the stroke direction vector $\mathbf{t}$. Following the third vector of the frame $\mathbf{d}$, we define $\mathbf{p}_0^i = \mathbf{s}_i + \varepsilon\mathbf{d}$ (resp. $\mathbf{p}_1^i = \mathbf{s}_i - \varepsilon\mathbf{d}$)[1]. Finally, we set a constraint in the system for $\mathbf{v}_0^i$ (resp. $\mathbf{v}_1^i$), the closest vertex to $\mathbf{p}_0^i$ (resp. $\mathbf{p}_1^i$). The embedded figure illustrates this constraint positioning technique. Last, to define on which side we set positive constraints, we assume that the user is more likely to select a protuberant part rather than the rest of the shape. We use the conformal factor (see Section 5.4.2) to decide which side is the more protuberant. As a result, the constraints distribution remains both independent of the stroke direction and consistent along different strokes (see Figure 5.3).



5.3.3 Patch selection

Inspired by 2D systems such as [Pho] or [LSS09], we let the user define patches by brushing them, while optimizing on-the-fly the selection boundary (see Figure 5.4).

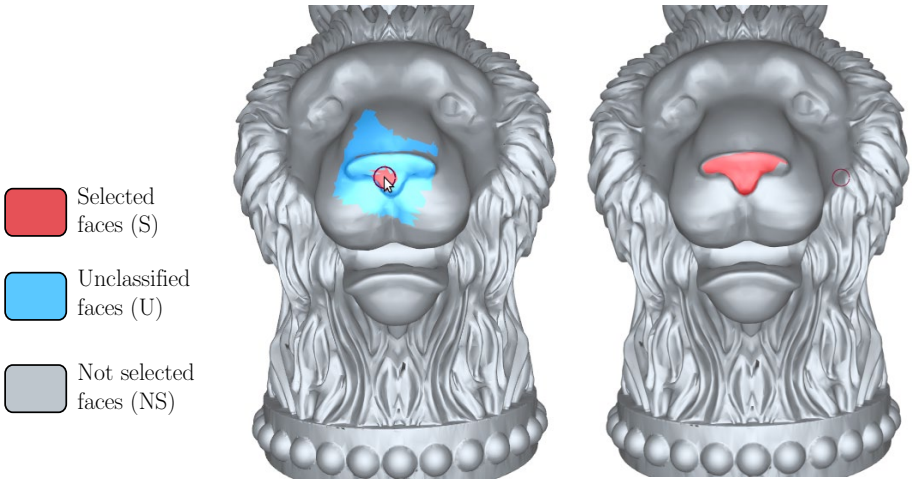


Figure 5.4 – **Patch algorithm:** given a set of unclassified polygons (left) the final patch selection is delimited using a geometry-aware flood-filling (right).

[1] $\varepsilon = 10^{-2}\%$ of the bounding box in our experiments

First, every vertex touched by the user brush is marked as *selected* (S). Starting from this set, we flood-fill the surface and mark every encountered vertex as *unclassifed* (U) until we reach vertices located further than a threshold distance f_c from the centroid of S. We mark these vertices as *non-selected* (NS). The threshold distance is relative to the widget size, that can be interactively adjusted during the interaction. Last, we perform two competing flood-fillings on U, one starting from S, the other from NS. They compete based on a cost function to gather the elements of U. When they terminate, the patch selection is the set of triangles belonging to S. In practice, we found that using the normal-based $L^{2,1}$ metric [CSAD04] as a cost function gives the best results.

The algorithm allows a smooth trial-and-error process since it runs interactively while the user is brushing over the mesh. The user can modify the selection size by changing the widget radius (depicted by a red circle in Figure 5.4). If the algorithm does not give satisfactory result in some regions, e. g. if the user want to segment a region with no differences in its normals, she can simply perform a *background brush*.

5.4 Expansion

Beyond the component, part and patch selection schemes, we propose an *expansion* process which gathers additional similar regions over the surface to enrich the reference selection. In this second stage, the user can navigate in the potential selections space by simply sliding a value k to augment the current selection with the k most-similar regions. This expansion functionality, which significantly diminishes the repetitiveness burden of selection, can be expressed as a local shape matching problem. For each selection type, we use a specific descriptor class combined with fast retrieval algorithms to detect candidate regions efficiently.

5.4.1 Connected component expansion

If the reference selection is an entire component, the expansion process may propose to the user similar selections among the set of connected components. To sort them and select the k most similar ones, we describe each component using an approximation of the Shape Context descriptor [MBM05], that we called ASC.

An ASC is a 2D histogram measuring the distribution of normalized distances and normal deviations with respect to the center of the component (see Algorithm 1). We start by computing the center $\mathbf{c}$ and a normalizing factor r_{max} using the MiniBall algorithm [Gö99]. Then, we fill the histogram by accumulating, for each vertex $\mathbf{v}$ of the component with normal $\mathbf{n}$, the area of the dual face of $\mathbf{v}$ in the bin (d, θ) of the ASC histogram. d is the normalized distance between $\mathbf{v}$ and $\mathbf{c}$ and θ is the angle formed by $\mathbf{n}$ and by the vector going from $\mathbf{v}$ to $\mathbf{c}$ (see Figure 5.5). Taking the area of the dual face for each vertex is a weighting scheme which ensures robustness against sampling variations. The ASC is rotation- and translation-invariant. It has a linear computational complexity and, thanks to its typically low number of bins, is robust to small scale variations.

In practice, at loading time, we compute the ASC of each component of the model. Then, during an interactive selection expansion, we measure the similarity between the reference component and all the others using the L^2 distance between their ASC (see Figure 5.5) and we return the k closest components.

Algorithm 1: ASC Computation

for each connected component of the mesh **do**
  initialize its descriptor: ASC ($10 * 10$ bins)
  compute its Miniball (center: $\mathbf{c}$, maximum radius: r_{max})
  for all $\mathbf{v}$ of the connected component **do**
    $d = \frac{\|\mathbf{v}-\mathbf{c}\|}{r_{max}}$
    $\theta = acos((\mathbf{v}-\mathbf{c})^T \cdot \mathbf{n})$
    $ASC[\mathbf{d}, \theta] += \sum_{t_i \in T_1(v)} \frac{area(t_i)}{3}$
  end for
end for

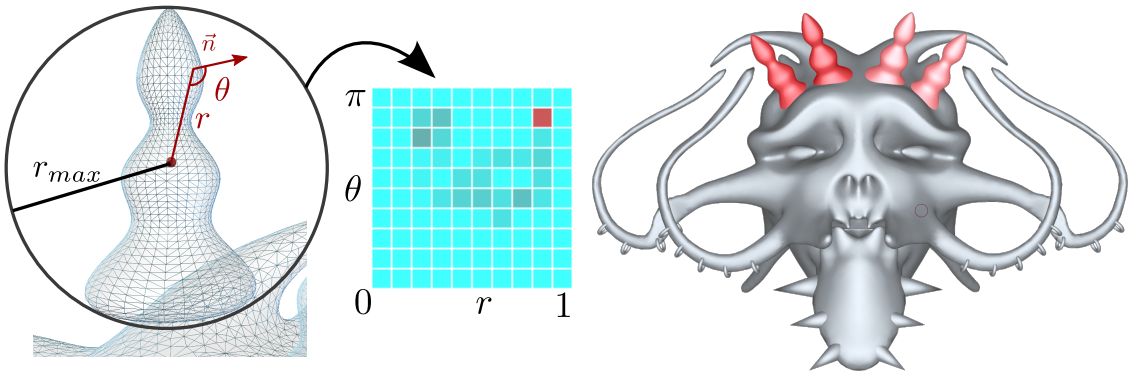


Figure 5.5 – **Connected component expansion:** using the center and radius of the miniball (left), we compute the ASC descriptor (2D histogram in blue) and find the most similar components (right)

5.4.2 Part expansion

Finding regions similar to a reference part selection is more challenging. Since the input shape does not come with a segmentation, we have to compute one before measuring the similarity between resulting regions. To do so, we use the *conformal factor* of the shape and we define a pool of candidate part regions to compare with.

The *uniformization theorem* demonstrates that any 2-manifold can be conformally mapped to a surface with the same topology having a constant Gaussian curvature. The conformal factor [BCG08] is a scalar function on the surface which describes this mapping and which is invariant to isometric transformations. We compute it once when loading the mesh, by solving the following linear system:

$$L\Phi = K^T - K^{origin},$$

where K^{origin} is the mesh Gaussian curvature and K^T is the average Gaussian curvature. Intuitively, the conformal factor represents the local stretch required to transform the model into a surface with constant Gaussian curvature. For instance, protuberant parts such as arms, legs or fingers have a large conformal factor value.

We make the assumption that the user traces cuts roughly along isolines of the conformal factor. This allows us to populate our pool of potential part regions. Given the reference part selection, we compute the average conformal factor c_s of its related stroke points, and find all the isolines of the similarity map with the same conformal value on the shape. These isolines are similar to

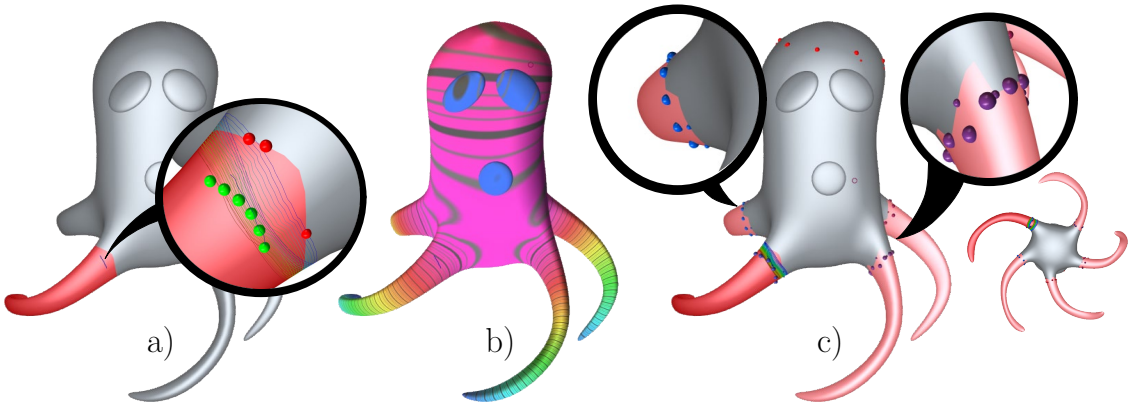


Figure 5.6 – **Similar part detection:** a) the reference part selection (close up: user constraints and isolines of the field) b) similarity map: conformal factor c) detected similar regions

boundaries of potential similar selections and we use them as virtual strokes. For each isoline we repeat the part selection process (explained in Section 5.3.2) and add the resulting potential part region to our pool of candidates (see Figure 5.6). With this pool in hand, we can now search for similarity to the reference by computing a descriptor for each of its elements and returning the k closest parts. Rather than using the ASC descriptor, we exploit the previously computed conformal factor and use a 1D histogram representing the conformal factor distribution of each candidate part region. To sort the candidate parts with respect to the reference, we use the χ^2 distance between their histograms (see Algorithm 2).

One strength of this algorithm is that it uses the locality of the part selection and the globality of the similarity map to find the best similar selections. The algorithm is also invariant to isometric variations.

Algorithm 2: Part expansion

hypothesis: user strokes are roughly along a conformal isoline.
compute the average of the user strokes conformal factor : a_s
compute the descriptor of the reference selection d_s
find all the isolines with the same conformal factor value a_s
for each isoline i **do**
  set harmonic constraints on each side of the isoline
  find the potential part region i
  compute its descriptor d_i
end for
sort the potential part regions by their distance to d_s
select the k first part regions

5.4.3 Patch expansion

As in the case of parts, we need to generate a pool of candidate patch regions to expand a reference patch selection. We propose to compute a similarity map to find centers of potential similar patches (see Figure 5.9-*b*). We treat each vertex of the shape as a potential patch center, and we compute one descriptor per vertex. In the case of patches, we need to define a descriptor which is discrimi-

native enough on the small scale high frequency signal which makes patches singular structures of interest on a surface (e.g., the relief of an ear). The similarity map captures the distance between the descriptor of the reference and the descriptors of all the other vertices (see Algorithm 3). Once this map is built, we select the k most similar vertices to be the center of similar regions. However, we still need to retrieve the exact boundaries of their patches. To do so, we learn a model of the reference patch selection w.r.t. its neighborhood using a support vector machine (SVM) to classify all faces in its vicinity.

Similarity map computation. As we cannot easily retrieve the boundaries of the patch, we built a descriptor on a disk region which embeds the selection. In the following, S_i is the set of all the vertices inside the sphere of center $\mathbf{v}_i$ and radius r, with r being the reference patch bounding sphere radius. We define a patch-based variant of the shape context $PSC(i,r)$ as a 2D histogram which first dimension captures the normalized distance d between a vertex $\mathbf{v}$ of S_i and its center $\mathbf{v_i}$, and the other dimension captures the angle θ between the normal of $\mathbf{v}$ and the normal of the center $\mathbf{n}_i$. The similarity map is built by computing the χ^2 distance between the reference descriptor and all the others.

Algorithm 3: Patch similarity map

find the reference patch center v_{ref} and the patch radius r
compute the reference descriptor $PSC(v_{ref}, r)$
for every vertex i of the mesh **do**
  compute $PSC(i,r)$
  $similarityMap(i) = \|PSC(i,r) - PSC(v_{ref},r)\|$
end for

---

function: compute $PSC(i,r)$
  find all vertices in S_i
  for every $\mathbf{v_j}$ in S_i **do**
    $d = \frac{\|\mathbf{v_j} - \mathbf{v_i}\|}{r}$
    $\alpha = acos({\mathbf{n_{v_j}}}^T \cdot \mathbf{n_{v_i}})$
    $PSC_{i,r}(d,\alpha) += \sum_{t_k \in T_1(v_j)} \frac{area(t_k)}{3}$
  end for

Once the patch similarity map is computed, we know where the potential patch regions are located on the surface (i.e., the minimum of the map) but we still need to compute their exact extent. To do so, we *learn* a model of the reference patch selection with respect to its neighborhood using a support vector machine.

Patch learning. From a local similar region S_i centered on vertex i, we want to know for each vertex in S_i if it is selected or not. This is a classification problem, and we use a Support Vector Machine method (or SVM) to solve it.

We recall the fundamental principles behind support vector classifiers. Given a set of l training examples $(\mathbf{x_i}, y_i)$ where $\mathbf{x_i}$ is a vector in $\mathbb{R}^d$ and $y_i \in \{-1;1\}$ is a class label, an SVM aims at finding the hyperplane (with equation $\mathbf{x}^T \cdot \mathbf{w} + b = 0$) that best separates the data. Once the best solution is found, it can be used to classify new (unclassified) data. The best hyperplane maximizes

the distance from the hyperplane to the data, also called *margin*. In fact, we can show that only the closest examples from the hyperplane are actually necessary to define it. They are called *support vectors*, hence the name of the technique. Using the Lagrange multipliers, the problem can be recast as a *Quadratic Programming Problem* (QP) defined as:

$$\begin{aligned} \text{minimize:} \quad & W(\alpha) = -\alpha^T \cdot \mathbf{1} + \frac{1}{2}\alpha^t \mathbf{H}\alpha \\ \text{subject to:} \quad & \alpha^T \cdot \mathbf{y} = 0 \\ & \mathbf{0} \leq \alpha \leq C\mathbf{1} \end{aligned} \tag{5.1}$$

where α is the vector of the l non-negative Lagrange multipliers to find, $\mathbf{H}$ is an l-by-l matrix with $\mathbf{H}_{ij} = y_i y_j (\mathbf{x_i}^T \cdot \mathbf{x_j})$ and C is a weighting constant (see [CV95] for more details). If C is infinite the problem admits a solution only for separable data. Setting C to a finite (possibly small) value, we find a *soft-margin* classifier, i. e. we allow some misclassified data.

A SVM can be used for a larger range of problems. If the data is not linearly separable, we can transport it into another (usually higher-dimensional) space where it will be, and then solve the classical SVM problem in this space. In other words, we can find a function Φ that transforms the input d-dimensional vectors $\mathbf{x}$ into d'-dimensional vectors $\mathbf{z}$ such that $\mathbf{z} = \Phi(\mathbf{x})$. By rewriting the above problem (Equation (5.1)), we see that we do not have to actually perform the mapping. The algorithm only requires to compute the dot product between two d'-dimensional vectors. That is, we only need to know the *kernel* of the higher dimensional feature space, i. e.

$$K(\mathbf{x_i}, \mathbf{x_j}) = \Phi(\mathbf{x_i})^t \cdot \Phi(\mathbf{x_j}).$$

Solving the problem in this way, the hyperplane separating the data lives in some unknown feature space. Going back to the original space, the data is separated by some non-planar contours, such as curves for example.

Several classic kernels have been developed. Among them, we use the radial basis function (RBF) kernel, defined as:

$$K(\mathbf{x_i}, \mathbf{x_j}) = e^{(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)}.$$

The γ parameter of the RBF represents how much it fits the training set (i.e., the reference selection). To have a reasonable value of γ for all selections, we set it to the median of the median distances between points and their closest neighbors (computed in the feature space). Ideally, going in the direction of example-based selection from large data sets, computing cross validation process for each selection could help to find the best parameters. For now, as we want our approach to work for all models and all selections, this is left as future work. Before using the RBF kernel, we express our data in a 3-dimensional space, that we call the *feature space* in the following.

Feature space. We observe experimentally that patch regions typically represent on-surface structures where the geometry is mostly altered in the normal direction (i.e., displacement). Therefore, we express the elements of S_i in a more suitable feature coordinate system, reflecting this characteristic in the learning and classification stages. For each region (reference patch selection and candidate patch regions) we define a local frame

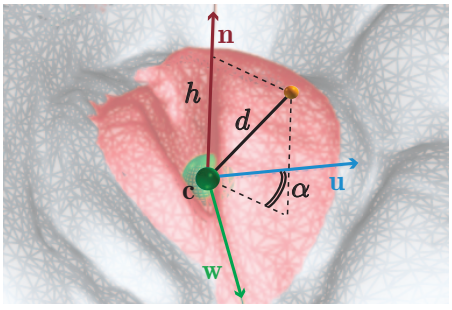


Figure 5.7 – Coordinates in feature space

$F_i = (\mathbf{c}, \mathbf{u}, \mathbf{w}, \mathbf{n})$ where $\mathbf{c}$ is the center of the patch, $\mathbf{n}$ is the average normal of the patch, $\mathbf{u}$ and $\mathbf{w}$ are the two principal directions of curvature (detailed below). We use F_i to compute the feature coordinates (d, h, α) of every vertex $\mathbf{v}_j$ in S_i, with d the (rescaled) length of the vector $\overrightarrow{\mathbf{cv}}$, h the length of the projection of $\overrightarrow{\mathbf{cv}}$ on $\mathbf{n}$ and α the angle between the projection of $\overrightarrow{\mathbf{cv}}$ in the $\{\mathbf{u}, \mathbf{w}\}$ plane and $\mathbf{u}$ (see Figure 5.7). The SVM is trained with all the vertices of S_{ref} and used to classify all the vertices of all S_i.

Registration. To guarantee that the whole approach is rotation invariant and to properly reproduce the boundary of the patch in the expansion, the local orientation of F_i and the position of its center c are critical. This is indeed a *registration* problem, where we seek a rigid transformation from F_{ref} to F_i which aligns their neighborhoods as well as possible. This problem, classical in reconstruction from scans, can be efficiently addressed in two steps: a first estimation of a global transformation from F_{ref} to F_i followed by a local adjustment to re-align the two regions.

For the global registration, we initialize F_i with a normalized principal component analysis (PCA) performed on the *normal field* of S_i. With similar geometric structures in S_{ref} and S_i (as detected by our similarity map) the position of the centers and the direction of $\mathbf{u_{ref}}$ and $\mathbf{u_i}$ are usually consistent. However, we also need to define the orientation of $\mathbf{u}$ in a consistent manner across both frames. To do so, we use differences between their first and second moments. More precisely, we flip the $\mathbf{u}$ axis of F_i, if

$$\|(m^1_{loc}, m^2_{loc})^t - (m^1_{ref}, m^2_{ref})^t\| \leq \|-(m^1_{loc}, m^2_{loc})^t - (m^1_{ref}, m^2_{ref})^t\|,$$

with $m^1 = \sum_{v \in S_i} area_v h_v < \mathbf{v} - \mathbf{c}, \mathbf{u} >$ and $m^2 = \sum_{v \in S_i} area_v h_v < \mathbf{v} - \mathbf{c}, \mathbf{u} >^2$. This technique allows to have consistent frames, even when the patch have reflective symmetries, as depicted in Figure 5.8.

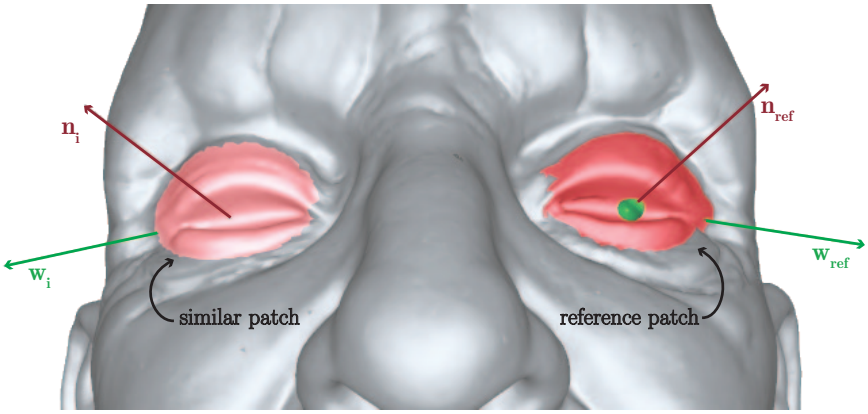


Figure 5.8 – **Registration:** the frame of the similar patch (left) is automatically retrieved with well oriented axis even in the presence of reflective symmetry. The *u* axes are not visible on the figure as they are inside the model.

When F_i is found, we use the *Iterative Closest Point* (ICP) algorithm [BM92] to better adjust it. As our SVM feature space depends heavily on F_i, improving its orientation and center using the ICP significantly improves the results of the SVM classification (see Figure 5.9).

Robustness. Taken independently, SVM classification and ICP registration have their own weaknesses (see Figure 5.9). For instance, when using ICP only, one can define virtual strokes (by

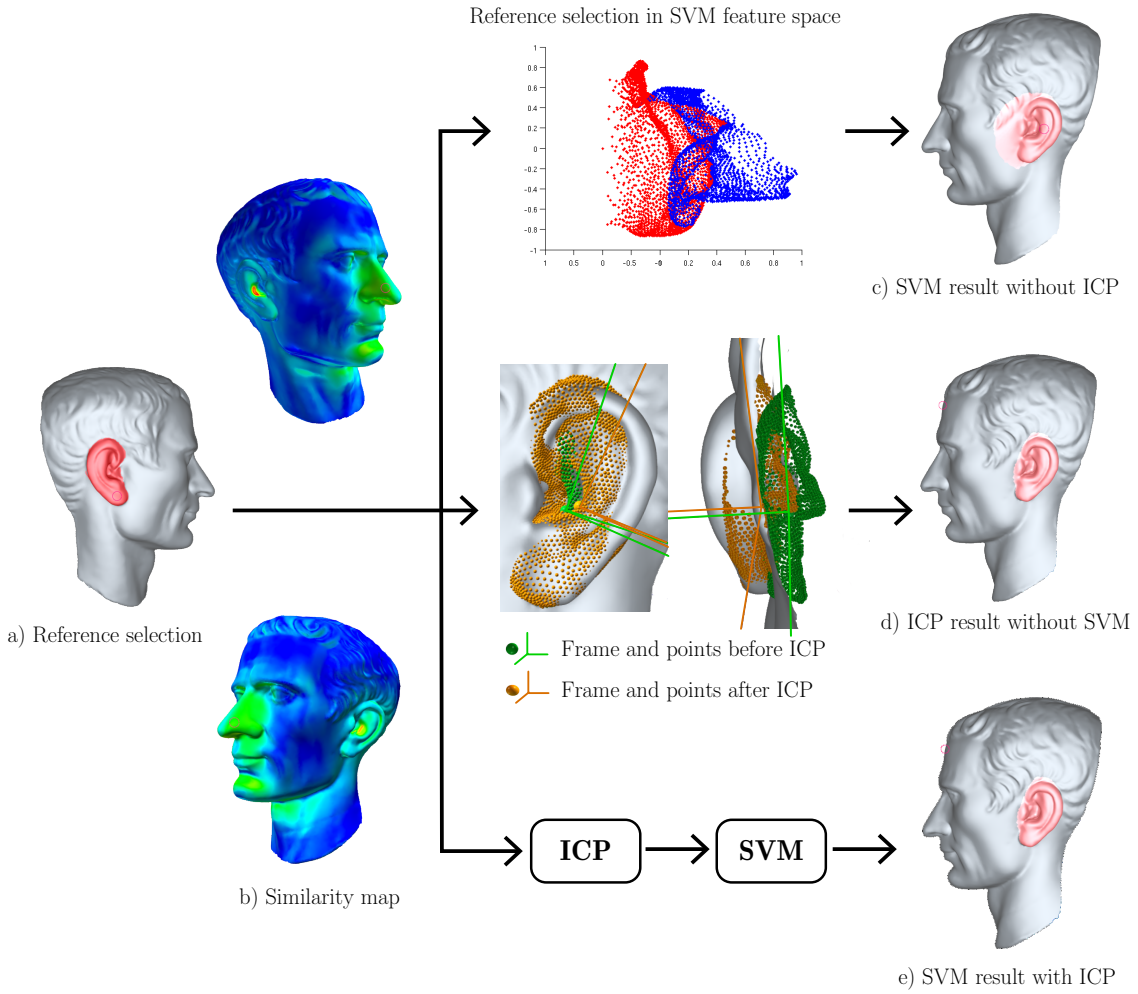


Figure 5.9 – **Patch expansion:** as shown on the right, both the SVM classification and the ICP registration are instrumental and complementary when expanding patch selections.

expressing the original stroke of the reference in the local frame F_i) and run the patch selection algorithm. Unfortunately, this simple "template" strategy is sensitive to small variations in the local geometry and results in inadequate boundaries and holey selections (see Figure 5.9-*d*). A better result is achieved when running our patch learning technique after the ICP (see Figure 5.9-*e*), since the SVM classification accounts for the target geometry when retrieving the patch extent. Similarly, as the SVM depends on F_i, the classification is usually too rough without the local optimization performed by the ICP (see Figure 5.9-*c*).

5.5 Selection & Expansion Results

Implementation and Performance. We implemented our SimSelect tool in C++ and report performances on an Intel Core 2 Quad/2.83GHz/8GB. We use the factorization downdating/updating techniques [XZCOX09] implemented in the CHOLMOD package [DH09] to solve the linear systems. We use libSVM [CL11] and libICP [GLU12] for our patch learning and patch registration techniques. The interface is implemented in OpenGL with the Qt SDK.

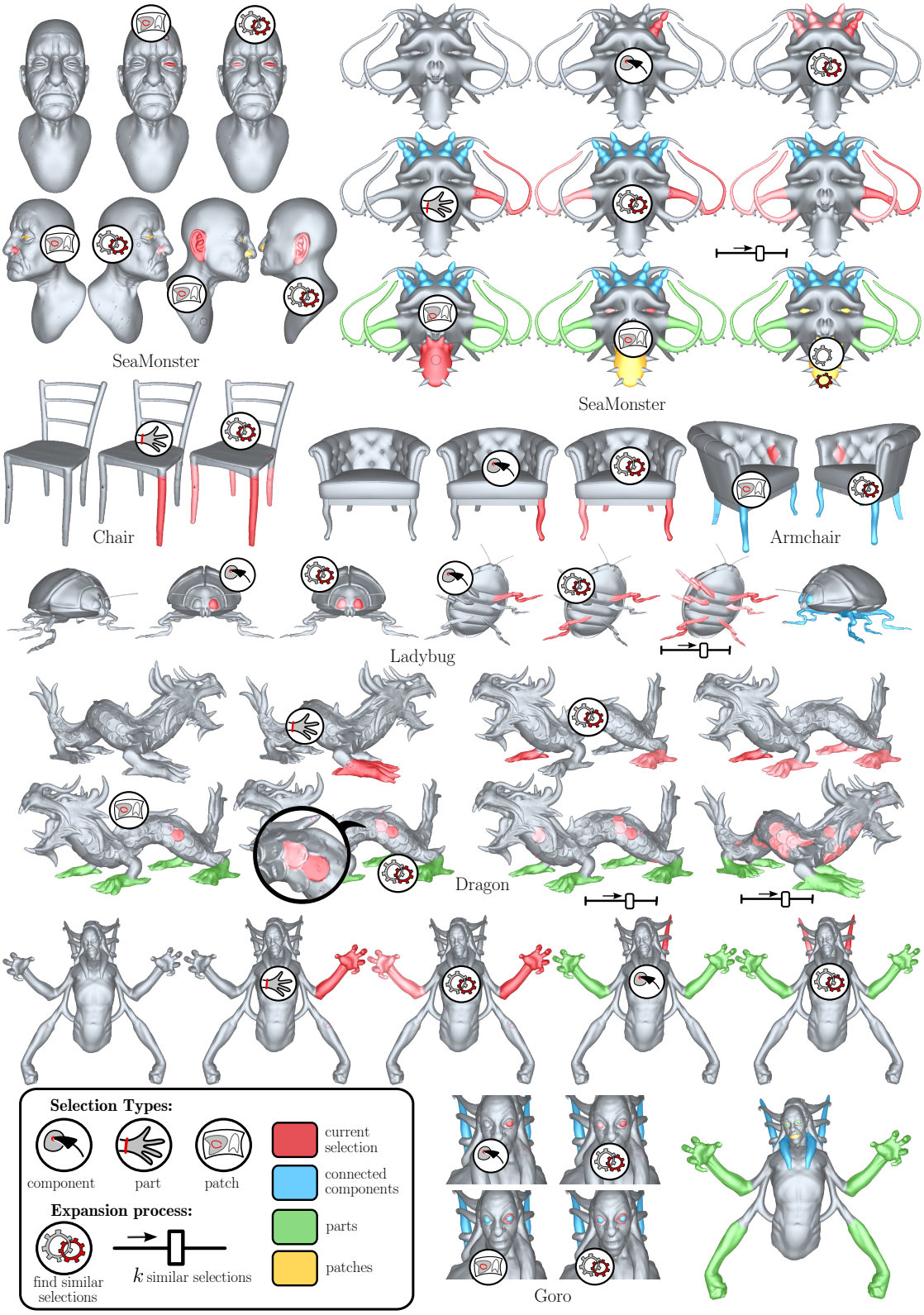


Figure 5.10 – **SimSelect Results:** the different models show results of the selection for patches (yellow), parts (green), and connected components (blue). The current selection is always in red, while automatic similar selections are displayed with color saturation proportional to similarity. User interactions are depicted by the circled logos and the slider.

The selection process is computed as the user is interacting with our tool and the three algorithms are efficient enough to run at an interactive rate on all models we tried (see Table 5.1 and 5.2).

The speed of our expansion step depends on the selection type:

- For connected components, most of the work is done at loading time (4sec for the *SeaMonster* mesh, with 29 connected components and 110k faces). Then, the detection of similar components is done in real time (less than 2ms for the *SeaMonster*).
- To detect similar parts, we update a linear system for each potential selection, so the computation time depends on the number of vertices and grows linearly in the number of potential selections (see Table 5.1). Changing the number k of selected similar selections (i. e. moving the slider) is interactive because all the potential candidates are computed at the beginning of the expansion process.
- The detection of similar patches depends on the number of vertices and on the patch size. The most expensive step is the similarity map creation, which requires the computation of a descriptor for each vertex of the mesh. Depending on the selection, the SVM training can also last few seconds. These two expensive steps depend only on the reference selection and are computed once, when the user starts the expansion process. Changing k is also interactive, as the only steps that remain are the ICP computation and the SVM prediction (see Table 5.2).

Model	(#V/#F)	PT (ms)	ST (ms)	PS	ET (ms)
Octopus	(12k/25k)	374	16	6	97
SeaMonster	(57k/114k)	1793	83	35	2385
Goro	(82k/165k)	2822	95	12	1158
Dragon	(100k/200k)	3588	162	47	6900
Chair	(14K/29K)	360	102	4	56
Hand	(55K/110K)	2870	102	8	1056

Table 5.1 – **Timing for parts selection.** PT: Precomputation time, ST: Selection time (average between each mouse motion), PS: number of potential strokes, ET: Expansion time.

Model	(#V/#F)	Selection (ms)	Expansion (ms)			
			r	Map	SVMt	ICP - SVMp
SeaMonster	(57k/114k)	8	0.07	3776	718	4
Armchair	(102k/201k)	27	0.16	14304	3829	142
Julius	(43k/85k)	11	0.2	11623	6470	24
Old Man	(84k/168k)	18	0.06	8119	1712	18
			0.05	7165	1731	13
			0.02	36689	5768	131
Dragon	(100k/200k)	25	0.06	10591	1175	10
Goro	(82k/165k)	7	0.02	5685	1471	56

Table 5.2 – **Timing for patches selection.** The selection time is the average computation time between each mouse motion. For the expansion time, *Map* is the similarity map computation time, *SVMt* is the SVM training time, *(*ICP-SVMp) is the total time for both ICP and SVM prediction computation.

Discussion. To the best of our knowledge, our system is the first which intends to compute interactive selections of different kinds and to expand them automatically. Our selection step is mostly inspired by existing interactive segmentation tools. We aim at finding a good tradeoff between user freedom and interactive feedback. The selection is inferred from few, rough interactions that the user can easily refine to explore the space of possible selections, both for local selection and global expansion. The expansion process is unified for the user who only has to control the number of similar selections she wants. The detection of similar selections greatly reduces the repetitiveness and the time consumed for selecting regions on surfaces in a select-and-edit workflow. We show some examples of results obtained using our tool in Figure 5.10. Defining different expansion processes, depending on the reference selection type, is a key element in our approach. For instance, we illustrate in Figure 5.11-*d* how expanding the selection of a part using the patch technique provides a less accurate result (not isometry-invariant) in a longer time due to the large radius of the selection. Similarly, defining this reference part selection using the patch tool is quite a tedious task for the user, requiring the navigation of the 3D camera to access hidden regions.

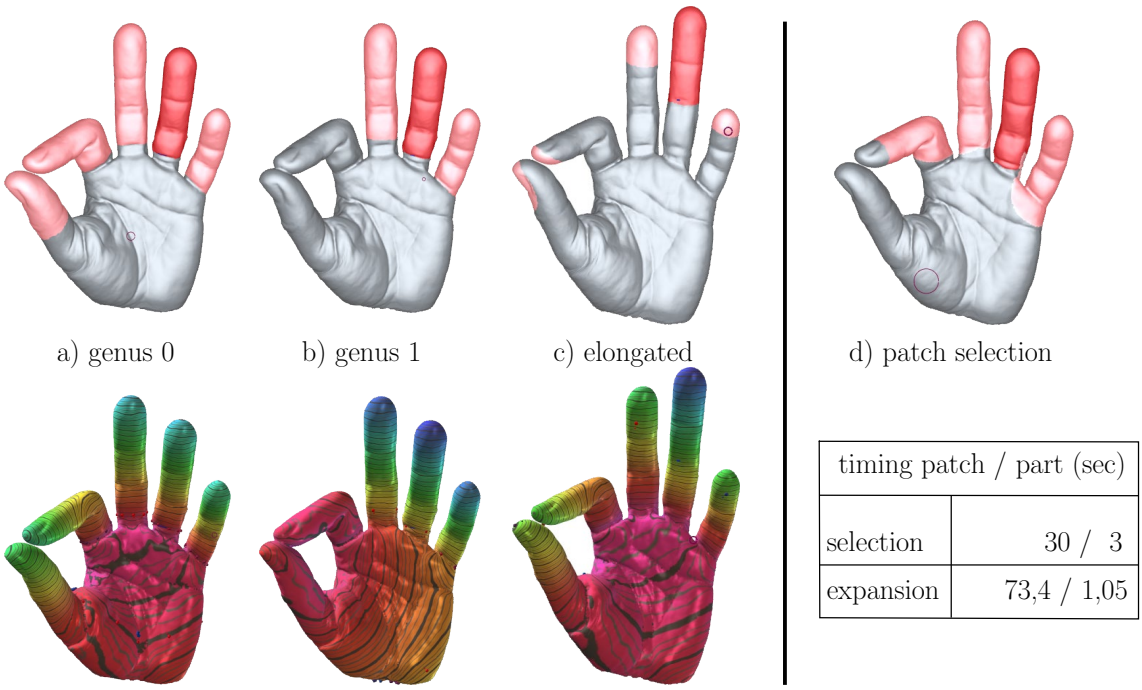
The detection of similar connected components and parts is largely inspired from shape retrieval algorithms and partial shape matching techniques. However such algorithms aim at detecting similar models in huge databases and require long preprocessing, while we have developed specific solutions to retrieve surface portions on a single mesh *instantly*.

User feedback. Contrary to Chapter 3, our objective here is not to perform a formal user study. However, we gathered initial user feedback when using our tool to perform several selections on a variety of models. The details of the tasks as well as the users results and comments are provided in Annex B. Over 16 participants, ranging from novice users to experienced CG designers, we gathered the following statistics:

- 44% of the users found the 3 selections equally important,
- 81% found the expansion very useful,
- 56% would always use the tool if it were available in their favorite CG software (38% would often).

We also notice that after an exploratory stage, most people naturally used the adequate tool to perform a selection. Thus, our classification seems to be relevant. We believe that these first results and user feedback are essential for future improvements of the project, as well as for calibrating a formal user study.

Limitations of the expansion process. Our system has several limitations which could motivate future work. Firstly, the expansion could be accelerated. Using level-of-details meshes would speed-up the part expansion, and the patch similarity map could be computed on the GPU. Secondly, the similar parts detection greatly relies on the conformal factor which depends on the mesh topology. Moreover, although the conformal factor is invariant to isometry, it depends on the shape elongation. If a part is much more elongated than another one, the cutting boundaries will be poorly positioned (see Figure 5.11). Thirdly, the detection of similar patches is not scale-invariant, as we use the radius of the patch to locate potential patch centers for the expansion. Extending our patch descriptor to make it invariant to affine and isometric deformations, would be an interesting direction for future work. One solution could be to express the geometry of the patch on a local normalized exponential map [TSS+11] and to use an affine-invariant descriptor in this space, for



timing patch / part (sec)	
selection	30 / 3
expansion	73,4 / 1,05

Figure 5.11 – **Pathological cases:** (a-b) The part expansion depends on the mesh topology. Here, the artificial tunnel between the two fingers prevents the detection of similar parts. c) the elongated finger results in inaccurate boundaries for similar parts d) use of the patch tool to select a part.

instance a geometric version of the ASIFT descriptor [MY09]. However, this raises the question of the computational cost of such a solution.

5.6 Toward similarity-based editing

5.6.1 Input and requirement

After the expansion process we end up with a set of similar regions. Depending on the selection type, we also have different invariance among the regions, and other pieces of information. More specifically,

- the connected components are invariant to scale, rotation, and translation. As the expansion process is very light, we do not have other data that could be used for the editing.
- The parts are invariant to isometry, and we have some understanding about their boundaries. Parts often represent elongated regions on the shape, and we know the values of the conformal factor on each vertex.
- Finally patches are rotation and translation invariant and they form disk-like regions. From the expansion process we have a precise corresponding mapping of a center and a frame $\{u,v,n\}$ on each similar region. We could also gain some insight from the SVM result, for example we could know the probability for each surrounding vertex to be inside or outside the selection.

Using the knowledge already acquired during the expansion process to compute an edit on all similar regions is an intuitive choice. It would allow to reduce the computation necessary to transport the editing among the regions. On the other hand, it might also mean that different inputs are required from the user to perform a given edit depending on the selection type. Therefore, one may prefer to have a unified editing process for all selection types.

5.6.2 Transporting the editing to similar regions

Many different types of editing can be applied on a mesh. In the previous chapter we review shape modeling techniques which use spatial control structures to help the deformation (Sections 4.2.1 and 4.7). Other editing applications can be performed such as colorization, texturing, smoothing, enhancement, remeshing or copy-pasting to name a few. We decide to classify all these edits in two categories, namely *vertex-based* and *region-based* edits, that do not require the same inputs to be duplicate on several similar regions.

Vertex-based Editing. An edit can be performed on a per-vertex basis. In this case the processing applied on a vertex is not related to the one applied on its neighbors. Smoothing, coloring or displacement along the vertex normal are examples of such vertex-based techniques (see [ZG04]). As the editing does not depend on the local context, it can easily be performed on similar regions. To do so, the only requisite is a similarity map which is used to weight the amount of editing performed on each vertex. As long as the map is smooth enough, it is a sufficient requirement for non local per-vertex editing. Figures 5.12 and 5.13 demonstrate respectively displacement along the normal and smoothing based on three different similarity maps computed from the Heat Kernel Signature, Wave Kernel Signature and Growing Least Square descriptor (defined in Section 2.3.5). Once the map is computed the editing can be performed in real time on a vertex $\mathbf{q}$ and transport

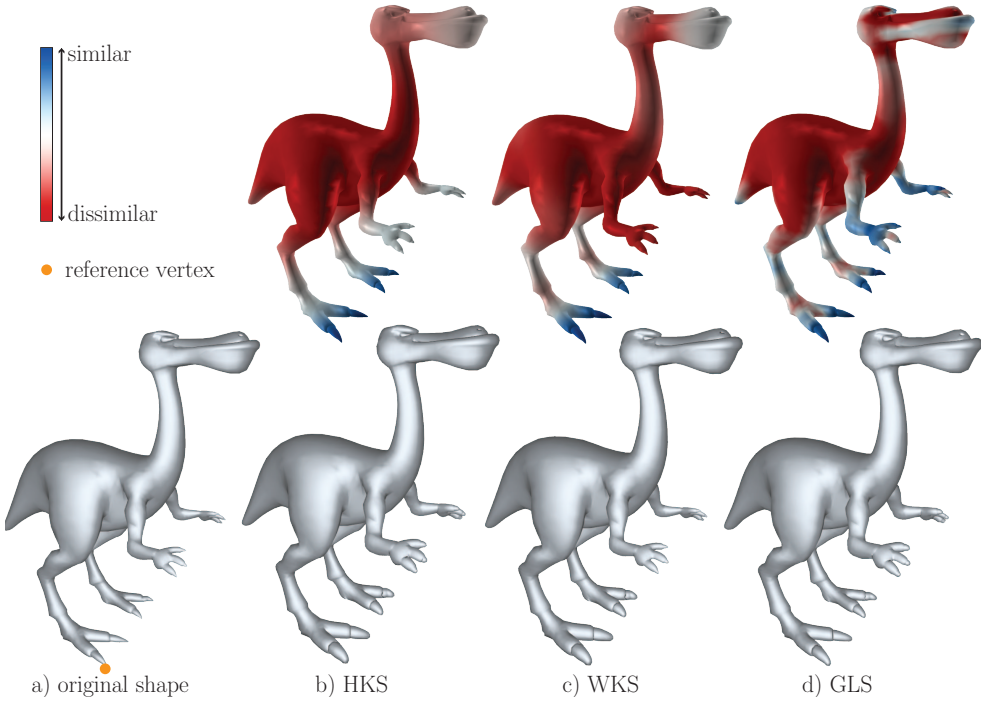


Figure 5.12 – **Displacement along normals:** similarity maps are based on HKS, WKS and GLS descriptors.

to all the similar vertices $\mathbf{p}$ by applying a simple weighting scheme. For instance the displacement along the normal is defined by,

$$\mathbf{p} = \mathbf{p} + sim(\mathbf{p},\mathbf{q}) \cdot d \cdot \mathbf{n},$$

where $\mathbf{n}$ is the normal of the vector, d is a quantity of motion computed from the user input, and $sim(\mathbf{p},\mathbf{q})$ is the similarity measured between $\mathbf{p}$ and $\mathbf{q}$ (varying between 1 for very similar vertices and 0 for dissimilar ones). Following the same idea, the Laplacian smoothing is defined by,

$$\mathbf{p} = \mathbf{p} + sim(p,q) \cdot \mathbf{L}(\mathbf{p}),$$

with $\mathbf{L}(\mathbf{p})$ the Laplacian operator. Finally, we can easily apply a transfer function to change the similarity based weighting scheme, for instance to prevent the editing on vertices which similarity is under a given threshold.

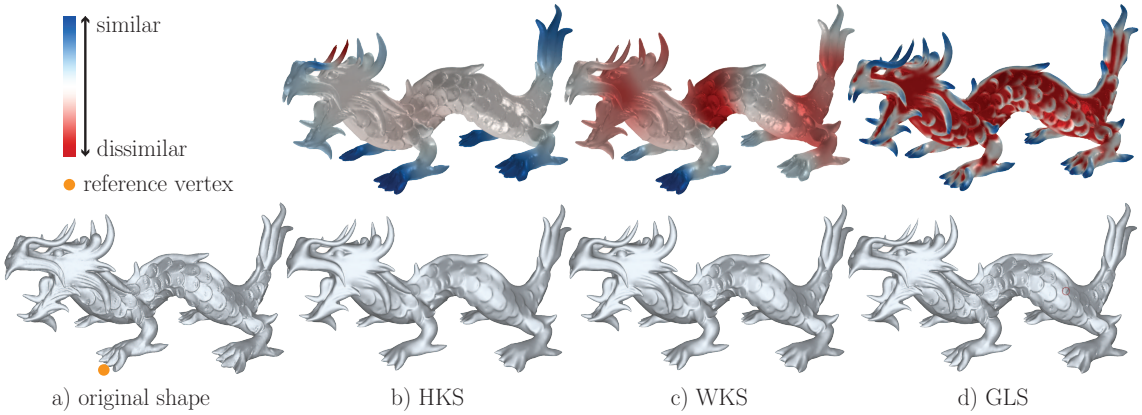


Figure 5.13 – **Smoothing:** similarity maps are based on HKS, WKS and GLS descriptors.

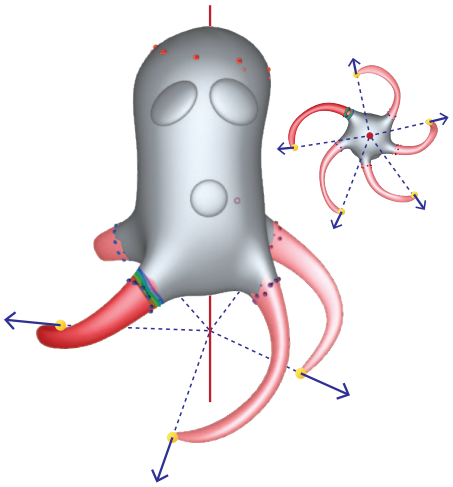
Per-vertex based edits could also be applied on our selected regions. In this case we would have to define a similarity map from the set of selected vertices, for example setting the similarity to one for vertices closed to the region center, and smoothly decreasing it when getting closer to the region boundaries. Nevertheless, the per-vertex editing do not use the local neighborhood defined by the selections which limits the possible applications.

Region-based editing Being able to edit simultaneously similar regions is a more challenging task since the notion of neighborhood has to be consistent over regions. However, the spectrum of region-based edits is large and contains complex applications such as deformations or copy-pasting. In the following we review possible strategies than we plan to study as future work.

The first straightforward idea is to find a one-to-one mapping among the vertices of the different regions to transport the result of an edit. This strategy would be efficient even if the regions undergo isometric deformations. However, being able to find a one-to-one mapping is a difficult problem, not performed interactively. In the case of 2D strokes, Guerrero et al. [GAWJ15] propose a two steps approach where they first detect similar vertices by comparing their local neighborhood, and then find a non-linear mapping between regions by searching smooth surfaces in a particular transformation space. However, extending their technique to 3D is not straightforward, and would require time-consuming computations. Another possible strategy would be to use functional maps (see Section 2.3.6). This solution is especially appealing if we are able to express the editing as a function over the vertices. In this case, we would not even need a one-to-one mapping. However,

to build a functional map we need to compute the eigenvalues of the Laplacian for each region independently and on-the-fly, which is a long and tedious process.

Another solution is to transport the editing primitives rather than their result. This would allow to better fit the local geometry, especially in the case of regions that are not perfectly similar. This strategy share common principles with our part expansion process (see Section 5.4.2) and with the colorization method explained in [LT13]. We could apply it easily to patch selections, since they are rotation and translation invariant only. Indeed, we already have a corresponding local frame for each region, and can use it to transport the editing primitives. For instance, we could apply an on-surface variational method to deform the region (see Section 4.2.1). In this case, the patch would be the region of interest, its center the handle and we could easily transport the deformation by expressing the motion of the user widget in each local frame. Applying this idea of transporting editing primitives is not trivial in the case of parts. As they undergo isometric deformations, a local frame per region is not enough to transport an edit. For connected components and parts the solution might rather be to analyze the global positional patterns. Indeed, a user might want to replicate an editing primitive according to a symmetry axis or plane (as depicted in the side figure).



5.6.3 Toward Non-local Editing

The vertex-based and region-based edits are built on the same workflow. The user works on a reference selection, and the editing is transported on similar regions. This scenario can fail if the similarity detection is not robust enough. We believe that more involved non-local edits could be perform by getting a general understanding of the similar regions as a whole, i. e. by analyzing all the similar regions together.

Following a process based on signal analysis we could for instance discover the differences and similarities among the regions. We could use this knowledge to create new editing tools enhancing the similarity among the regions, or on the opposite, enhancing their local differences.

An interesting example of the potential of non-local editing is the construction of control structures for the deformation. These structures are essential to perform complex deformations but they are most of the time built by hand. We believe that studying the signals of similar regions, and especially their variations, could help the interactive or automatic construction of such structures. For instance, given a human in a certain pose, with one arm extended and the other folded, we could use the geometric information contained in both arms along with some predefined rules to build a control structure. Indeed, knowing that the skeleton bones should be of the same size, we could find automatically the most relevant positions of both elbow joints using the geometric differences of the regions. We believe that there are many other potential applications of the non-local analysis of similar regions.

5.7 Conclusion

In this chapter we have proposed an interactive system to accurately define selections of different types on meshes and expand them on the whole surface based on automatic similarity detection. From a user perspective, only a small set of simple metaphors have to be carried out, namely *clicking* components, *cutting* parts, *brushing* patches and *expanding* the selection. From a technical point of view we have improved existing local segmentation algorithms to obtain instant feedback.

The expansion of a reference to detect similar regions is our main technical contribution. Based on our classification, we designed specific algorithms to retrieve similar selections. They share the same principle: a pool of candidate selections is built if necessary and particular descriptors are computed for each region to compare the candidates and select the most similar ones. In the case of patches, we use a learning algorithm to retrieve complex boundaries of patches. The classification of the selections and their different invariances allow to have an interactive system.

Performing interactive non local editing remains a difficult task and we only survey possible solutions. The computation time which is not the priority in most similarity detection techniques, is an important matter in our case. However, we believe that the use of similar selections can greatly alleviate the user work in editing tasks. Several alternatives are possible to transport an edit and the best one might be a global framework working for all selection and all editing types. Nevertheless, keeping interactive computation times would be difficult in this case and one might have to divide the problem. Finally, we believe that new editing applications can emerge from the analysis of non-local selections.

The notion of *computational interactions* has a great importance in this chapter. It enables higher level interactions through computation and is used at the three different stages of the system: selection, expansion, and editing.

CHAPTER 6

SIMILARITY-BASED SIMPLIFICATION

6.1 Motivations

In the previous chapters we have seen how to use shape abstractions and similarity detection independently to create new modeling applications. We believe that the two notions can be combined together and be mutually beneficial. Therefore, in this chapter we use a similarity-based measure to drive a simplification process.

Figure 6.1 shows the interactions between the simplification process and the similarity-based detection. This non-local simplification algorithm leads to a new shape approximation with an embedded notion of similarity. The resulting multi-resolution abstraction could then be used as a new non-local control structure for the deformation.

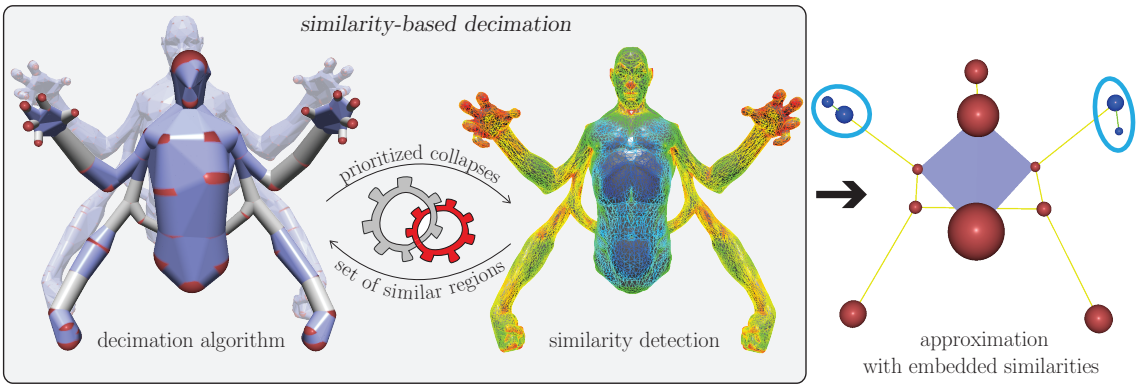


Figure 6.1 – **Non Local Simplification pipeline:** at each step of the simplification process, a measure of similarity is used to detect and simplify similar primitives in parallel. The resulting spatial abstraction, is a multi-resolution structure with an embedded notion of similarity.

Our method is based on two main components: a decimation algorithm and a measure of similarity. We could use any decimation method, yet we base our approach on the sphere-mesh algorithm since it faithfully approximates a shape even at extreme simplification levels.

The techniques presented in Chapter 4 and Chapter 5 both have their strengths and limitations. In Chapter 4 we build a multi-resolution shape approximation, while in Chapter 5 we use interactive

similarity detection techniques to select and edit similar regions. In the following, we review the different weaknesses of both methods, and explain how the present technique can alleviate them.

The *sphere-mesh* algorithm allows to build a multi-resolution control structure which is useful for the deformation. However, a sphere-mesh hierarchy contains as many levels as the number of collapses (for instance more than $82k$ levels for the Armadillo). Navigating in such a big structure can be troublesome. In fact, when using the sphere-mesh as a deformation instrument, we only let the user navigate in the smallest degrees of the hierarchy. A better solution would be to automatically gather the similar steps together in a single level, and propose a condensed hierarchy to the user rather than the full one.

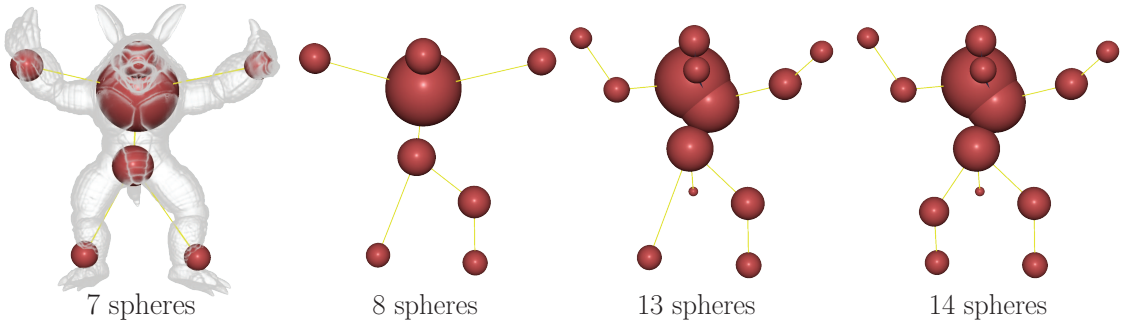


Figure 6.2 – **Decimating the Armadillo model.** The 7^{th} level the sphere-mesh is symmetric and captures faithfully the similarities of the original model. However, the 8^{th} level looses this property because of the sphere approximating the left knee region. While a user would expect the similar sphere to emerge on the right knee at the next step, it only appears 6 levels later.

The sphere-mesh construction is based on a greedy algorithm which is inherently local. The spherical quadric error metric computes the squared distance from a sphere to a set of planes and does not take into account high-level shape's properties such as similarities among regions. Nevertheless, many levels of the resulting abstraction hold the similarities of the input shape faithfully, as depicted in Figure 6.2 (left). In many other cases the sphere-mesh hierarchy is not perfect, and the next sphere to appear or disappear in the hierarchy is not the expected one. For instance, while navigating the Armadillo hierarchy, we observe that the knee regions are not collapsed at the the same time (see Figure 6.2), due to their different simplification cost. Sometimes, the sphere-mesh approximation even fails at capturing the similarities of the input model, as in the case of the cat decimation depicted in Figure 6.3. Finally, even if the similarities often exist in the sphere-mesh, they are neither detected nor used. In this chapter we aim at using a notion of similarity to gain a non-local insight on the shape and alleviate the aforementioned problems.

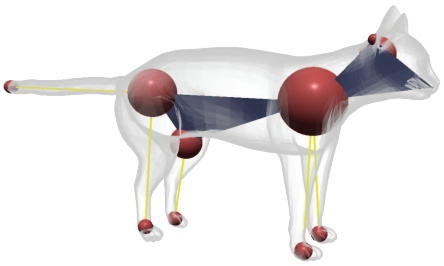


Figure 6.3 – Asymmetrical simplification

In Chapter 5, we focus on similarity detection and we propose a system to automatically retrieve similar regions on the shape. We also describe potential solutions to transfer an edit to similar regions. However, even with the similarity information, transferring an edit is a non trivial task, difficult to perform interactively. In the case of complex edits, like deformations, the use of a con-

trol structure might be mandatory. Potential solutions consist in transferring the control structure or the resulting deformations to other regions. Another direction of research is to compute a control structure that embeds the similarities directly. We believe that we can build such a structure by combining similarity detection and mesh simplification.

The main technical components of our method are the simplification algorithm and the similarity computation. Our technique is based on the decimation algorithm used in both QSLIM [GH97] and sphere-mesh methods. This algorithm performs greedy collapses to simplify the shape, creating at the same time an underlying clustering of the input shape with growing regions. While most descriptors are computed per-vertex (see Section 2.3.5) we aim at computing a descriptor per region (i. e. per cluster of the decimation). Especially, we want to define how to aggregate the descriptors while decimating the mesh. Our contributions are: the construction of a condensed hierarchy of sphere-meshes, with similar collapses performed together and the aggregation of similarity among regions at different resolutions.

In the following, we first review existing descriptors and structure-aware decimation techniques (Section 6.2). We then give an overview of our algorithm (Section 6.3) and explain how we aggregate descriptors to detect similarities among regions (Section 6.3.1). Finally, we present the first results (Section 6.4) of our approach and propose possible directions for future work (Section 6.5).

6.2 Background & Related Works

In the following we first highlight the underlying geometric properties of the QEM and SQEM to show their descriptive potential. We then focus on the GLS descriptors as they are simple, robust and multi-scale. Finally, we review a method related to our work, that uses high-level structural insights to drive a decimation process.

Quadric error metrics: Our simplification approach is based on the sphere-mesh algorithm (described in Chapter 4). The metric used to measure the error is one of the major components of decimation techniques. In the following, we give some geometric insights about the QEM and SQEM metrics to highlight their ability to describe a region.

In [Gar99], Garland studies the Quadric Error Metric (QEM) which is defined as the squared distance from a point $\mathbf{q}$ to a plane $\{\mathbf{p},\mathbf{n}\}^{\perp}$. Following a similar calculus derivations as the SQEM (see Section 4.5) we end up with:

$$Q(\mathbf{q}) = \mathbf{q}^t\ \mathbf{A}\ \mathbf{q} + 2\mathbf{b}^T \cdot \mathbf{q} + c.$$

with $\mathbf{A} = [\mathbf{n}\cdot\mathbf{n}^t] \in \mathscr{S}^3$, $\mathbf{b} = -(\mathbf{n}^T\cdot\mathbf{p})\mathbf{n} \in \mathbb{R}^3$ and $c = (\mathbf{n}^T\cdot\mathbf{p})^2 \in \mathbb{R}$.

The isosurfaces of the QEM ($Q(\mathbf{q}) = \varepsilon$) are quadric surfaces in $\mathbb{R}^3$ (possibly degenerated), that tend to follow the curvatures of the original model. They are more elongated along the regions of low curvature, and they get stretched in regions of high curvature, as depicted in Figure 6.4. Indeed, Garland shows that for a small region around a point $\mathbf{p}$ on a smooth surface, the two smallest eigenvalues of the QEM matrix $\mathbf{A}$ are proportional to the

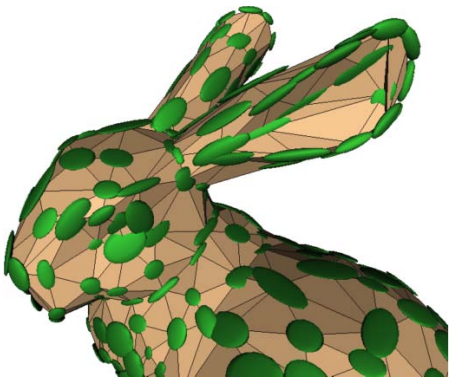


Figure 6.4 – quadric isosurfaces [Gar99]

square of the principal curvatures at $\mathbf{p}$ and the corresponding eigenvectors of $\mathbf{A}$ are the principal directions of curvature.

The Spherical Quadric Error Metric (SQEM) exploited in sphere-mesh is very similar to the QEM. However, we use spheres instead of points and our quadrics live in $\mathbb{R}^4$ rather than in $\mathbb{R}^3$. Thus it is more difficult to visualize the isosurfaces of our quadrics, that are 3-dimensional surfaces in $\mathbb{R}^4$. Nevertheless, the top-left sub-matrix of our quadric $\mathbf{A}_{\mathbf{SQEM}_{22}^{00}}$ is the same as the QEM one. Thus its eigenvalues and eigenvectors provide information about the principal curvatures of the underlying surface and could be used to describe a region.

GLS descriptor: The Growing Least Square method [MGB+12] aims at analyzing point clouds in scale-space. To do so, the authors fit a sphere on growing neighborhoods around each point, and use the fitted surface to compute some geometric properties of the underlying region. In a sense, the technique is closed to the SQEM, but it uses an algebraic sphere rather than a geometric one.

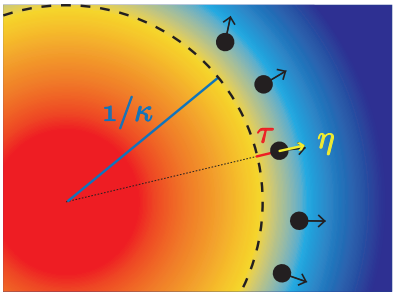
Starting from a set of points $\mathbf{p_i}$ with associated normals $\mathbf{n_i}$, the neighborhood of a point $\mathbf{p}$ at a given scale t is the set $\mathscr{P}_t$ of data points such that $\mathscr{P}_t = \{\mathbf{p_i}, ||\mathbf{p_i} - \mathbf{p}|| \leq t\}$. The algebraic sphere is defined as the 0-isosurface of a scalar field:

$$s_{\mathbf{u}}(\mathbf{p}) = [1 \quad \mathbf{p}^t \quad \mathbf{p}^T \cdot \mathbf{p}] \cdot \mathbf{u},$$

where $\mathbf{u} = [u_c \ \mathbf{u_l} \ u_q]^t \in \mathbb{R}^5$ is the vector of the (respectively constant, linear and quadratic) parameters of the sphere. For each data point, the authors fit the algebraic sphere onto the neighborhood $\mathscr{P}_t$ in the least square sense, following the technique from Guennebaud et al. [GGG08] but with a weighting scheme which is scale dependent $w_i(t) = (\frac{||\mathbf{q_i}-\mathbf{p}||^2}{t^2} - 1)^2$. This lead to the following closed-form formulas for the sphere parameters

$$\begin{aligned}
u_q &= \frac{1}{2}\frac{\sum w_i \mathbf{p_i}^T \cdot \mathbf{n_i} - \sum \tilde{w}_i \mathbf{p_i}^t \cdot \sum w_i \mathbf{n_i}}{\sum w_i \mathbf{p_i}^T \cdot \mathbf{p_i} - \sum \tilde{w}_i \mathbf{p_i}^t \cdot \sum w_i \mathbf{p_i}}, \\
\mathbf{u_l} &= \sum \tilde{w}_i \mathbf{n_i} - 2u_c \sum \tilde{w}_i \mathbf{p_i}, \\
u_c &= -\mathbf{u_l}^t \cdot \sum \tilde{w}_i \mathbf{p_i} - u_c \sum \tilde{w}_i \mathbf{p_i}^T \cdot \mathbf{p_i},
\end{aligned}$$

with $\tilde{w}_i = w_j / \sum w_j$ (see [GGG08] for more details). The scalar field is then normalized using the Pratt's normalization which enforces a unitary gradient vector on its 0-isosurface, yielding $\hat{\mathbf{u}} = \mathbf{u}/\sqrt{||\mathbf{u_l}||^2 - 4u_c u_q}$. Using the field in this form to compute a geometric descriptor is difficult since the parameters are interdependent and do not directly represent geometric properties. Thus, the authors define three intuitive parameters from the field (depicted in the side figure):



- $\tau = s_{\hat{\mathbf{u}}}(\mathbf{p})$ the algebraic distance between the evaluation point and the sphere,
- $\eta = \nabla s_{\hat{\mathbf{u}}}(\mathbf{p}) / ||\nabla s_{\hat{\mathbf{u}}}(\mathbf{p})||$ the unit normal of the field at $\mathbf{p}$,
- $\kappa = 2\hat{u_q}$ the signed curvature of the sphere.

One additional parameter is the fitness φ which represents how well the point neighborhood $\mathscr{P}_t$ is fitted by $s_{\hat{u}}$ ($\varphi =$

$||\mathbf{u_l}||^2 - 4u_c u_q$). One can easily compute a pairwise dissimilarity measure between two points $\mathbf{p_a}$ and $\mathbf{p_b}$ at a given scale t:

$$d(a,b,t) = w_\tau t^{-1}(\tau_a - \tau_b)^2 + w_\kappa t(\kappa_a - \kappa_b)^2 + w_\varphi(\varphi_a - \varphi_b)^2, \tag{6.1}$$

with w_τ, w_κ and w_φ three scalars used to weight the different parameters (usually set to 1). This measure is dimensionless and rotation invariant since it ignores the respective normals of the points. The method has several interesting properties: it is fast to compute, the descriptor is highly discriminative and multi-scale, and the derivatives of all the parameters can be computed analytically in both space and scale.

Structure-aware decimation: Salinas et al. [SLA15] propose a structure-aware decimation algorithm. Their goal is to reach extreme simplification levels to simplify correctly urban scenes with few primitives. The authors highlight that the lack of structural information in greedy decimation techniques leads to poor extreme approximation results. Thus, they guide a decimation algorithm using a high-level structural knowledge of the scene. In more details, they precompute a global set of planar proxies organized in a proximity graph. Then, they use the QSLIM algorithm [GH97] with a modified QEM, where the original error metric is augmented to take into account both the structural knowledge and the local geometry.

The technique is especially designed for urban scenes where an organized graph of robust planar proxies can be faithfully computed. Following Salinas et al. [SLA15] we believe that a high level knowledge of an object can improve the simplification. However, we aim at interleaving the analysis and the simplification processes, rather than using a precomputed analysis. We believe it can lead to new potential applications using both the multi-resolution structure from the decimation and the knowledge from the shape analysis.

6.3 Similarity-based decimation algorithm

We base our method on two main components: the decimation approach and the similarity computation. Our approach is closely related to the decimation one, and in Algorithm 4 we expose in blue the changes from the original decimation method. At the initialization stage we compute all the possible collapses, with their cost and a descriptor per-collapse, and we put them into a priority queue. Then when a collapse is performed, we also retrieve the most similar ones using their descriptors and we collapse them as well.

A collapse $(u,v)_i$ is inside the set $\mathscr{S}$ of the reference collapse $(u,v)_{ref}$ if and only if:

- its distance to the reference collapse is below a given threshold, i. e. $d((u,v)_i,(u,v)_{ref}) < \varepsilon$,
- it is not part of the neighborhood of any collapse already in $\mathscr{S}$, i. e. $u_i \notin N_1(u_j)$ and $u_i \notin N_1(v_j)$ and $v_i \notin N_1(u_j)$ and $v_i \notin N_1(v_j)$, $\forall (u,v)_j \in \mathscr{S}$.

Our algorithm is based on the aggregation of descriptors, however the type of descriptor used does not impact the main framework. Thus we try several descriptors, and especially the eigenvalues of the quadrics and the GLS explained in Section 6.2.

Algorithm 4: Non local decimation

Initialization: Put all the collapses $(u,v) \rightarrow w$ and their associated cost c_{uv} in a priority queue. Compute a descriptor per vertex and aggregate the descriptors per collapse.
while the priority queue is not empty **do**
  select the cheapest collapse c_{cur}
  create the set $\mathscr{S}$ of valid similar collapses
  for all collapse c_{sim} in $\mathscr{S}$ **do**
    collapse the current pair and update all the neighboring ones.
    aggregate the descriptors of the neighboring collapses.
  end for
end while

6.3.1 Aggregating descriptors

Most descriptors are computed per-vertex, but in our approach we are rather looking for one per region. Thus, we need to aggregate the descriptors when we perform a collapse. Among possible solutions, we can recompute a new descriptor at each collapse, that is $[(u,d_u),(v,d_v)] \rightarrow (w,d_w)$. In this case d_w only depends on the current shape and not on the previous decimation steps. An other idea is to keep record of the descriptors during the model decimation, yielding $[(u,d_u),(v,d_v)] \rightarrow (w,\{d_u,d_v\})$. In this case, we need some compact structure to aggregate and compare groups of descriptors efficiently.

In the following, we explain the different techniques and descriptors that we have tried. All the descriptors need to be compatible with our main framework (see Algorithm 4), i. e. they should have the same three functionalities, defined as follow:

Per-vertex computation: builds the first descriptor for each vertex of the original shape (see Algorithm 4, initialization stage).

Aggregation: creates a new descriptor when performing a collapse, that represents the merged region resulting from the decimation step.

Similarity measure: compares a reference descriptor to all the others and decides which regions are similar or not.

6.3.2 Quadric recomputation

Our first idea is to use the knowledge already contained in the quadrics of the decimation algorithm. Indeed, during the simplification process, when a collapse is performed the quadric of the new region is formed by adding the two previous ones (see Section 4.5). The quadrics form a low-resolution representation of the surface and they could be used as cheap descriptors since they are already computed and stored.

The two smallest eigenvalues λ_0 and λ_1 of the quadric's submatrix $\mathbf{A}_{\mathbf{SQEM}}{}^{\mathbf{00}}_{\mathbf{22}}$ are related to the surface curvature and are rotation invariant (Section 6.2). Thus, we decide to use them to compare the regions. As the per-vertex computation and aggregation stages are already computed during

the decimation process, all we need is to extract the eigenvalues of the new quadric after each collapse. Then, given two possible collapses a and b, we compare them using the L_2 distance,

$$d(a,b) = \sqrt{(\lambda_0(a) - \lambda_0(b))^2 + (\lambda_1(a) - \lambda_1(b))^2}.$$

Two small regions having the same curvatures should have the same eigenvalues and a dissimilarity measure closed to 0. However, the information contained in the eigenvalues turn out to be insufficient for our need.

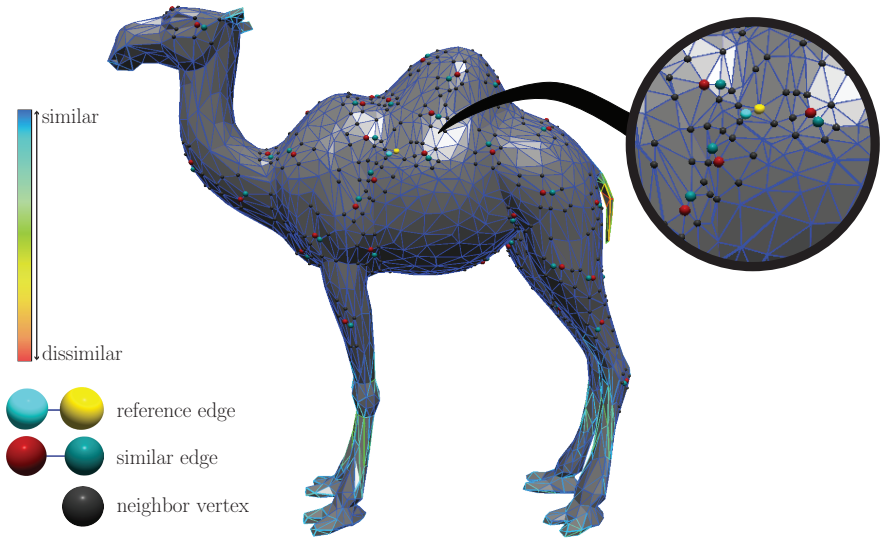


Figure 6.5 – **Quadric eigenvalues comparison:** the descriptors of the edges are compared to the reference one, using the eigenvalues of the quadrics.

In all the figures of this chapter, we represent the dissimilarities between a reference edge, that is automatically selected at each step of the decimation process, and all the others. For visualization purpose we scale the dissimilarity values between 0 (blue) and 1 (red). We also display the sphere-mesh topology (edges and triangles) rather than its interpolation. Finally, apart from the reference edge, similar edges, and neighbors, we do not show the spheres to better visualize the similarities between collapses (i. e. edges).

Figure 6.5 represents the similarity between the first reference collapse and all the others, going from blue for very similar edges to red for dissimilar ones. The eigenvalues of the quadrics are not discriminative enough, and finding an appropriate threshold for the dissimilarity measure is not straightforward. Also, the behavior of the eigenvalues at coarse scale is not known, as the link to the surface curvatures only holds for small regions. Thus, we decide to try other descriptors.

6.3.3 Averaging GLS

We use the growing least square descriptor since it is informative, multiscale and fast to compute. At the initialization stage we compute the GLS of every vertex for s different scales, that are sampled logarithmically between s_{min} and s_{max}, with s_{min} equals to k times the average edge size, and s_{max} a percentage of the original model bounding box.

To aggregate two descriptors into a single one, we simply compute their average, i. e. each parameter of the new GLS descriptor is the average of the two previous descriptors for each scale. Given a collapse $[u,v] \rightarrow w$, the new descriptor $d_w(\tau_w, \kappa_w, \varphi_w)$ is defined as:

$$\begin{aligned}\kappa(w,s_i) &= (\kappa(u,s_i) + \kappa(v,s_i))/2,\\ \tau(w,s_i) &= (\tau(u,s_i) + \tau(v,s_i))/2,\\ \varphi(w,s_i) &= (\varphi(u,s_i) + \varphi(v,s_i))/2.\end{aligned}$$

We use Equation (6.1) for all scales of the descriptors to measure the similarity between two collapses a and b, yielding:

$$d(a,b) = \sum_{s_i=s_{min}}^{s_{max}} d(a,b,s_i) = \sum_{s_i=s_{min}}^{s_{max}} w_\tau s_i^{-1}(\tau_a - \tau_b)^2 + w_\kappa s_i(\kappa_a - \kappa_b)^2 + w_\varphi(\varphi_a - \varphi_b)^2. \quad (6.2)$$

Figure 6.6 depicts the similarities between a reference collapse and all the others. At first, the descriptors give meaningful results, and the set of selected similar collapses is acceptable. However, as we decimate the model, the descriptors loose their discriminative power.

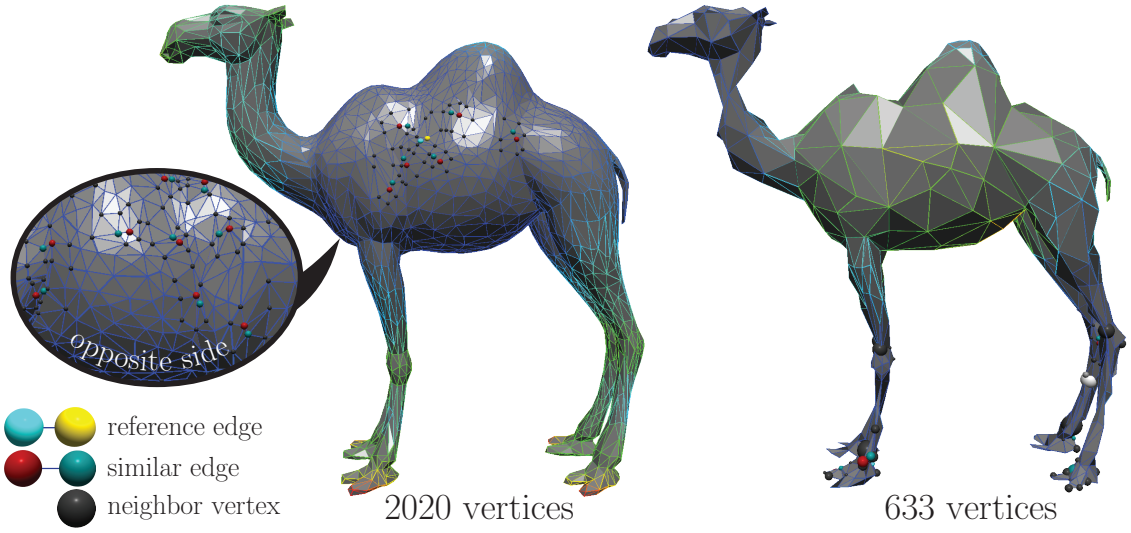


Figure 6.6 – **GLS descriptors comparison:** before the decimation process and after a few hundred collapses.

We believe that taking the average of the descriptors is not a good aggregation method in our case. Indeed, during the decimation process we aggregate regions with a lot of dissimilarities, especially in terms of small details. It is in fact an inherent feature of the decimation process, especially for extreme simplification levels. To improve the aggregation of GLS descriptors several solutions are possible. Firstly, one can recompute a new GLS descriptor from the union of the neighborhoods at each collapse, but at the expense of a high computational cost. Secondly, one can weight the two descriptors differently depending on the position of the new sphere (see Section 4.6.2 for details). Nevertheless, the center of the sphere does not remain on the edge and the weights computation requires the comparison of distances between the new sphere and the two old ones. Finally, the last possibility is to take into account the different scales of the GLS. Indeed, when the descriptor

is representing a vast region resulting from many collapses, the details inside the regions are not meant to be similar. Thus, we could weight the different scales in Equation (6.2) depending on the size of the reference region. However, we believe that the descriptor should depict the distribution of the features on the regions rather than being just an average.

6.3.4 Histograms from GLS

As we have seen in the last chapter, a common way to represent distributions and to aggregate data efficiently is to build histograms from them. To this end, we use the three (normalized) parameters of the GLS descriptor, namely $s^{-1}\ \tau$, $s\ \kappa$ and φ.

Histogram computation. As before we compute the GLS descriptor for each vertex and for s scales at the initialization stage. Then we build an histogram for each parameter. The three histograms have two dimensions: one for the parameter values and the other for the scales, with respectively x and s bins. Algorithm 5 details the histogram construction performed at the initialization stage. For each vertex, each scale, and each parameter, we set the appropriated bin equals to the area of the barycentric cell of the vertex P_v (see Section 4.6.1).

Algorithm 5: Construction of the GLS histograms

Require: the GLS descriptor with s scales and the barycentric cell P_v for each vertex v
  for all GLS parameters $p(i)$ **do**
    compute $p(i)_{min}$ and $p(i)_{max}$ the extrema of the current parameter
    for all vertex v of the original shape **do**
      initialize the 2D histogram with x-by-s bins
      for all scale s_j in s **do**
        compute the first dimension of the bin: $x_{cur} = \frac{p(i)-p(i)_{min}}{p(i)_{max}-p(i)_{min}}(x-1)$
        fill the corresponding bin: $hist_{p(i)}(v)[x_{cur}, s_j] += P_v$
      end for
    end for
  end for

Histogram aggregation. To aggregate two descriptors after a collapse $[u,v] \to w$, we simply set the new histograms to be the sum of the previous ones, yielding:

$$hist_{p(i)}(w)[x_{cur}, y_{cur}] = hist_{p(i)}(u)[x_{cur}, y_{cur}] + hist_{p(i)}(v)[x_{cur}, y_{cur}],$$

For all x_{cur} in $[\![0, x-1]\!]$, all y_{cur} in $[\![0, s-1]\!]$ and for all GLS parameter $p(i)$. Thus, while the decimation is performed, the sum of the values per column (i. e. at a constant scale) for each histogram is growing. Indeed, it represents the total area covered by the aggregated region.

Distance between descriptors. There are several distances to compare histograms. We choose one of them and define the similarity between two collapses a and b to be the sum of the distances between the different histograms. Namely,

$$d(a,b) = \alpha \cdot d(hist_\tau(a), hist_\tau(b)) + \beta \cdot d(hist_\kappa(a), hist_\kappa(b)) + \gamma \cdot d(hist_\varphi(a), hist_\varphi(b)).$$

In our current experiments, we give an equal importance to all parameters, setting $\alpha = \beta = \gamma = 1$. So far, we have tried to use the L_2 and χ_2 distances between histograms. Notice, that we set the number of y-bins to be equal to the number of scales. Thus, the computation of our 2D histograms for one parameter is actually equivalent to the computation of s 1-dimensional histograms (i. e. one per-scale).

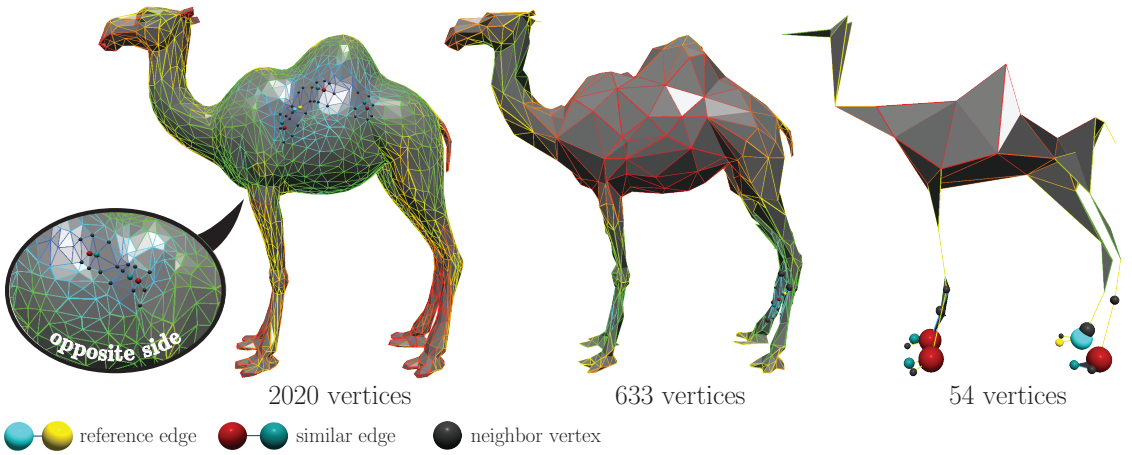


Figure 6.7 – **Histograms from GLS comparison:** for different stages of the decimation process.

In our experiments the χ_2 distance gives the best results and we use it in the following. Figure 6.7 depicts the similarities between edges. Similarly as the average of GLS (Section 6.3.3), the similarity map obtained before the decimation is informative. However, while we simplify the model, the histograms aggregate the information better than the average. Thus, we manage to keep a discriminative measure of similarity at the different decimation stages. For example, even with only 54 spheres to depict the model, the feet of the camel are detected as similar and are collapsed in parallel. We believe that knowing the statistical distribution at all scales actually matters, even when reaching extreme levels of simplification.

6.4 First Results

We implemented our similarity-based simplification algorithm in C++ and Cuda (for the GLS descriptor computation). The interface is built in OpenGL with the Qt SDK. The results in this section are all generated with the *"histograms of GLS"* aggregation. The GLS descriptor goes from s_{min} equals to 2 times the average edge size, to s_{max} equals to 25% of the original model bounding box, and the histograms are all of size 10-by-10. To define the set of similar collapses, we use a fixed threshold ε that is set to 0.2 (with a normalized dissimilarity measure).

Figure 6.8 shows that our technique has an improved similarity structure compared to the original sphere-mesh. However this behavior is not found in all the hierarchy levels, and for some of them our results are not better than the sphere-mesh ones.

On the frog model depicted in Figure 6.9, our technique retrieves the similarity patterns on the front legs better than the original sphere-mesh. During the decimation process, similar regions on the left and right legs are collapsed together. This behavior persists even at extreme simplification

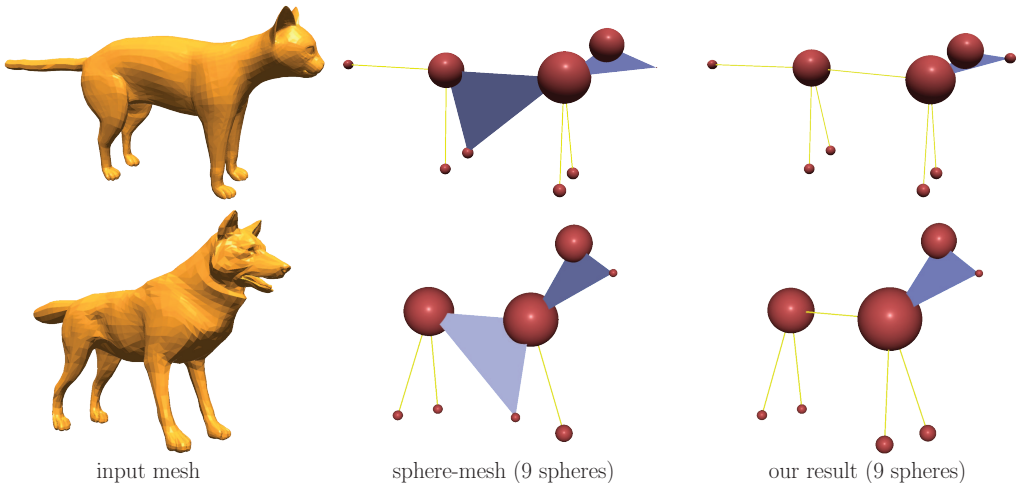


Figure 6.8 – **Similarity structure:** comparison between the sphere-mesh and our structure.

levels and after a large amount of collapses. However in other regions of the frog, like the head, our algorithm fails at retrieving the similarities.

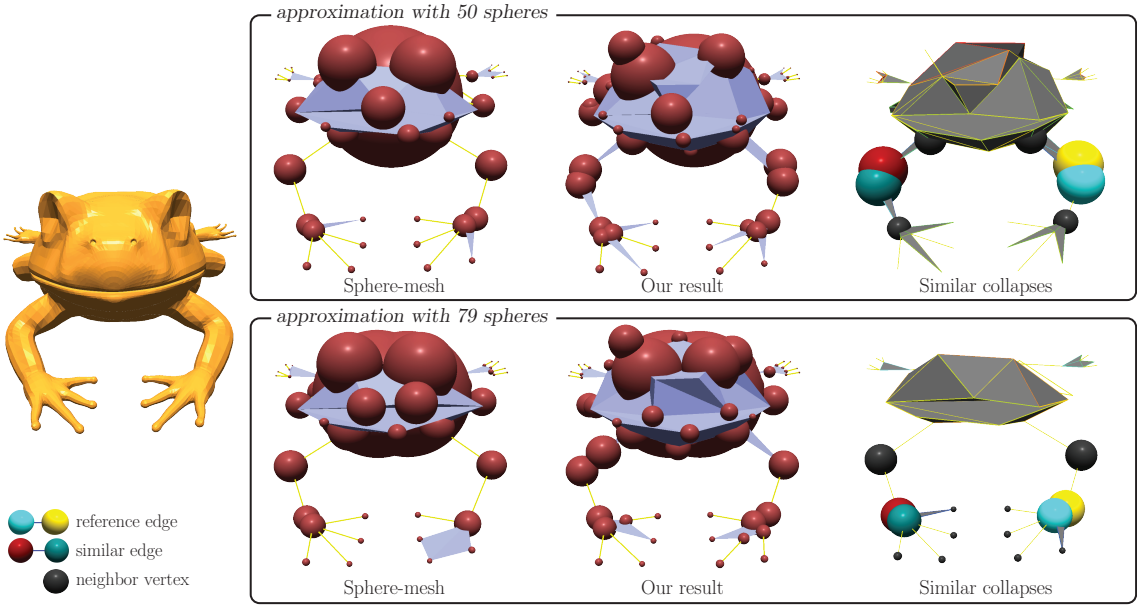


Figure 6.9 – **Similar collapses:** even at extreme simplification levels, similar regions are retrieved and collapsed together.

The detection of similar regions is useful even when the original sphere-mesh already preserves the similarities of the input shape. Indeed it allows to propose a condensed hierarchy to the user, easier to navigate. In practice, our structure is three to six times smaller than the original one which contains the full set of collapses. The number of levels is directly linked to the computation of similarities and to the threshold value ε. Setting the threshold to 0, our technique boils down to the original sphere-mesh algorithm.

Finally, our method allows to directly embed the similarity notion in the sphere-mesh hierarchy. In Figure 6.10 we show successive levels of the hierarchy for the *Hand* and *Flamengo* models. Similar regions, such as the fingers of the *Hand* model and the legs of the *Flamengo* are detected and collapsed together. These similarities are inherently embedded in our structure.

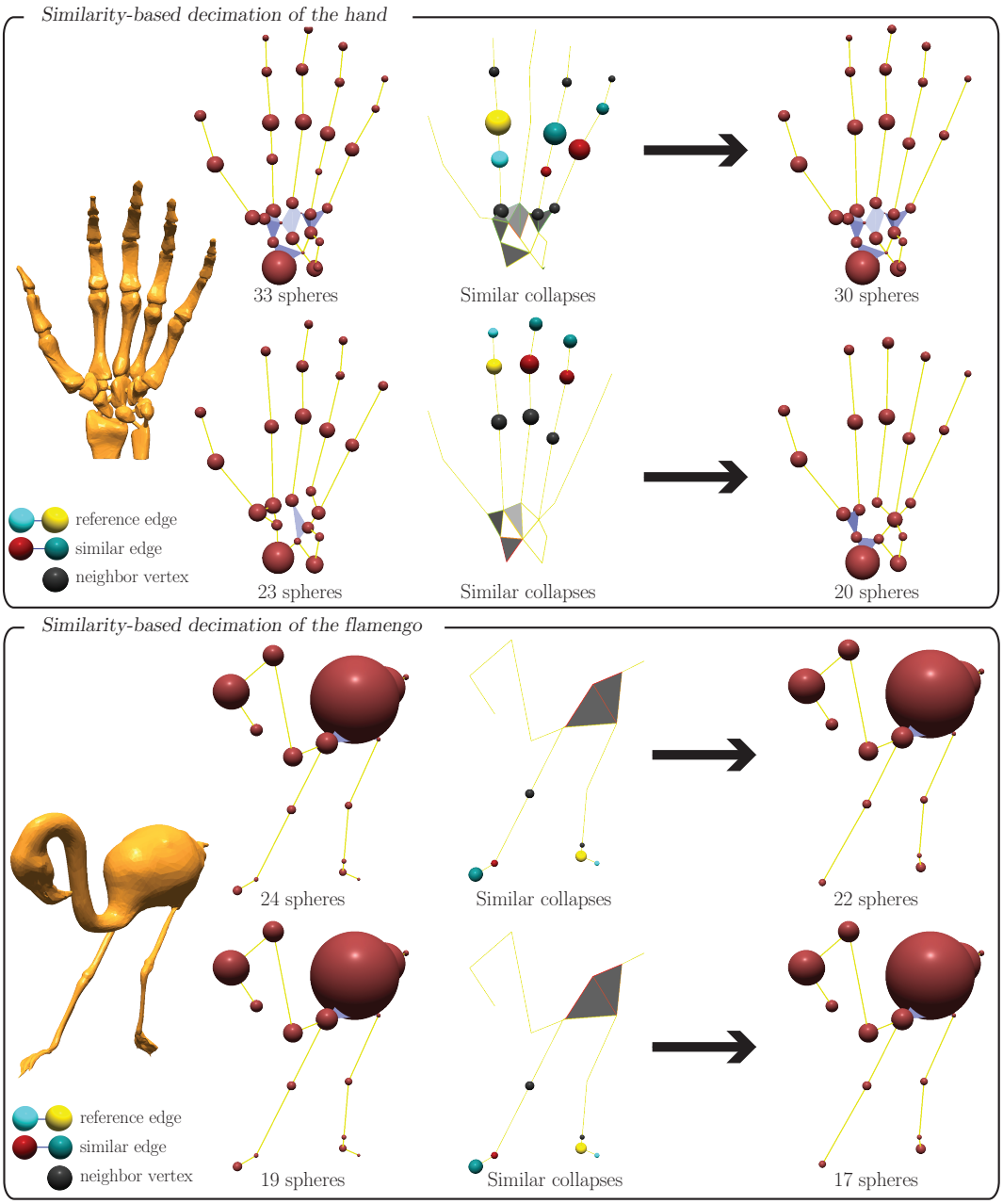


Figure 6.10 – **Similar collapses:** even at extreme levels of simplifications, similar regions are retrieved and collapsed together.

6.5 Conclusion & Future Work

In this chapter we presented a method which interleaves the similarity detection and the decimation process. This new algorithm has several advantages. First, it results in a condensed hierarchy structure, easy to navigate for the user and where similar regions are collapsed together. Second, it takes non-local information into account and improves the similarities in the decimation result. We believe that at extreme simplification levels, the similarities in the abstraction are as important as the local error to the original model to understand the shape and to use the abstraction in potential applications. Finally, our method provides a multi-resolution similarity measure, where similar regions of growing size are detected on the shape.

Aggregation of descriptors. Many improvements and directions of future work are possible from our technique. First of all, we investigate different solutions to aggregate descriptors per region. We believe that studying the distribution of the descriptors among the regions is a good aggregation scheme, since it takes into account the features at different scales to build the description. However, the solution we proposed, based on histograms of GLS, might not be the best one. We plan to study other approaches that compare distributions to improve the robustness of our results.

Similarity threshold. The main parameter of our algorithm is the similarity threshold used to decide which regions are similar or not. In our experiments we set it to a fixed value. However we aim at using a varying threshold, that would be automatically computed from the similarities. For each step, we could rank the current collapses in increasing order based on their similarity. Then we could detect the first gap in this space, and use it as an adaptive threshold. We believe that a deeper understanding of the similarity threshold can lead to an improved robustness and is an interesting direction for future works.

Acceleration structure. Finding the similar collapses is a time-consuming process, and the performance of our algorithm is significantly lower than the sphere-mesh one. A solution to alleviate this problem is to use an acceleration structure to retrieve the similar collapses more efficiently. For example, if we can express the descriptors as points in a high-dimensional space (for instance in $\mathbb{R}^{3s}$ for the GLS descriptors), we could use an acceleration structure to compare clusters of points in this space.

Other simplification algorithms. We perform our experiments with the sphere-mesh algorithm, but other simplification techniques, for example clustering or resampling the original model, could be adapted to take into account similarities between regions. Also, the GLS descriptor is not restricted to meshes and directly works on point sets, and we would like to adapt our similarity-based decimation process to set of points, following a similar line of work than [GAB12].

Non local deformation tool. Finally, we believe that our method can lead to new control structures for the deformation. As we have seen in Chapter 5, deforming similar regions automatically is a complicated task. Using our similarity-based deformation technique, we build a shape abstraction where the similarities are inherently embedded. It could be used to create new non-local applications, such as the extension of the deformation method presented in Chapter 4 to propagate interactions to similar parts of the control structure.

CHAPTER 7

CONCLUSION

In this thesis, we traveled among user interaction, shape modeling, and shape analysis worlds to propose new applications at their interface. In the following we review our different contributions and propose promising perspectives in the context of *computational interactions* for modeling.

7.1 Overview of the contributions

In the first part, we focused on spatial abstraction structures for modeling. These were constructed first in a preprocess step and subsequently operated on in interactive sessions by the user. While the two chapters of this part took different technical routes, they both used spatial abstractions to improve the user interactions in an interactive section.

In Chapter 3 we proposed a new navigation system, using the user body as a control structure. The editing of 3D objects displayed on a 2D screen usually requires the user to switch back and forth between viewing and modeling modes in order to visualize and adjust the edits. This repeated switching between modes is the consequence of the limited user interactions. In this chapter we focused on the viewing mode and we designed our system in a wide environment where the user was performing mid-air interactions in front of a screen. We postulated that the user's body could be a compelling instrument for traversing a 3D scene. Using a depth sensor, we extracted the user skeleton and built our set of interactions from this abstraction. Since the navigation is a smooth and simple process, we used non critical body parts to navigate in the scene, while the user hands and head are kept free for other tasks. We designed a framework to evaluate pair of motions for the navigation and highlighted several assumptions useful for building navigation tools. Specifically, the navigation should use uncorrelated body parts to be natural and easy to perform, the virtual and physical motions should behave in similar spaces, and the virtual speed should be tied to the motion amplitude. We performed a user study to evaluate different pairs of motions and validate our assumptions. In our study, we asked participants to execute simple secondary actions, like carrying a bag or a cup of coffee, while navigating in a scene. The integration of our navigation tool in more complex environments, such as modeling applications, is an interesting direction for future works.

In Chapter 4 we proposed a new shape approximation and used it as a control structure for the deformation. The key idea was to enrich the points representing the vertices by a radius, creating a structure called *sphere-mesh*. The multi-resolution approximation is the output of a decimation algorithm giving interesting results even at extreme levels of simplification. The interpolation of

the spheres along the sphere-mesh primitives approximates faithfully the original model. The main technical contribution of our decimation algorithm was the Spherical Quadric Error Metric, used to find the sphere approximating best a set of planes. This error drives the collapse of pairs of vertices in our decimation process. By using spheres instead of points, we can smoothly transition from a surface representation for high detail sphere-meshes to a volume representation for coarser ones. We used the resulting hierarchy of sphere-meshes as a new control structure for the deformation. To do so, we defined a skinning tying the model to its sphere-mesh, and we used the sphere-mesh primitives to control the deformation of the original shape. Once the deformation is performed, the sphere-mesh hierarchy is updated in real time to fit the deformed geometry. Contrary to most existing works, the structure is automatically computed in a multi-resolution fashion. The evolution from the shape surface to its interior was an essential feature for the deformation. It allowed us to perform deformations at different scales, mimicking handles on the surface to change details, skeleton structures to move tubular regions, and even volumetric primitives modifying the ambient space thanks to the sphere radius.

In the second part of this manuscript we focused on another aspect of shape analysis and used interactive similarities detection algorithms to create new modeling tools. Though different, both chapters interleaved the similarity detection with another process, either manual or automatic, to create new applications or abstractions.

In Chapter 5 we proposed a similarity-based selection system. Our system alleviated the two main issues of the selection process, i. e. the accuracy needed to select a precise area and the repetitiveness arising from the selection of multiple similar regions. Our framework was based on computational interactions to reduce both the complexity of the automatic similarity detection and the burden faced by the user to perform manual selections. To this end, we classified the selections in three specific classes: *connected components* in a multi-component model, elongated *parts* characterized by their closed-loop boundary, and *patches* representing local regions with strong on surface structures. The user interactions as well as the underlying selection and expansion processes were designed to fit as best as possible the characteristics of each selection type. For all selection types the expansion process was rotation and translation invariant. It was also scale-invariant for connected components, and isometry-invariant for parts which are likely to undergo quasi-isometric deformations. Adapting the expansion algorithms to each selection type significantly reduced the difficulty of the task, allowing an interactive process where the user only specified the number of similar selections. Our system enabled new high-level interactions through computation. Following this idea, we proposed to perform non local editing of vertices, where the amount of edits was driven by the similarity measure.

Finally, in Chapter 6 we created a bridge between shape abstraction and similarity detection. We modified and enriched a simplification pipeline to take into account similar regions during the process. Our objective was to create a multi-resolution structure approximating the original model and inherently embedding its similarities at different scales. To this end, we used the sphere-mesh algorithm described in Chapter 4 and several similarity detection methods studied in Chapter 5. The main technical contribution of our algorithm was the aggregation of descriptors for comparing regions instead of points while decimating the model. We showed that the distribution of descriptors was a good aggregation scheme for evaluating the similarities among regions, resulting in a condensed while informative structure describing a region. In practice we used histograms of the parameters defined in the Growing Least Square descriptors. Our technique led to significantly condensed sphere-mesh hierarchies, making navigation easier for the user. Our technique also im-

proved the similarities of sphere-meshes, leading to more symmetric structures for some levels. Finally, the notion of similar regions was detected and inherently embedded at the different resolutions of the sphere-mesh structure. We believe that such a high-level multi-resolution structure with embedded similarity information can lead to new applications for the analysis, comprehension and modification of 3D shapes.

7.2 Perspectives

Non local editing. In Chapter 5 we showed that the automatic expansion of selections to similar regions significantly reduced the repetitiveness of the selection process. Following this concept, we would like to build a non local editing framework where the similarities among regions will be used to alleviate repetitive editing tasks. We propose a classification of non-local edits in three main types with different objectives and inputs. *Per-vertex edits* are exclusively based on similar vertices and only require a similarity map. *Per-region edits* are more complex and rely on the local context. They are based on the local geometry of a region, not on a single vertex. Finally, *collaborative edits* take the whole set of similar regions into account to create new editing frameworks, rather than simply transporting the editing from a reference region to the other ones. While we already demonstrate the effectiveness of per-vertex editing, we would like to investigate the two remaining categories. To perform a per-region editing we can either transport the result of an edit or the editing primitives themselves. In any case, the process requires a precise mapping between a reference region and all the others. Performing this mapping interactively is an open question but we believe that mathematical frameworks, such as the one presented in [OBCS+12], can greatly reduce the complexity of the task. Also, some region-based edits might depend on other high-level information such as spatial contexts, global direction fields or symmetry patterns. Creating a framework robust to small dissimilarities and adapting the editing to the local geometry and context are interesting directions for future work. Finally we believe that collaborative editing can lead to a new set of applications, using the pool of similar regions as a whole to better understand it. Borrowing some concepts of signal analysis for example, we could learn the similarities and differences among the set of regions to enforce or reduce them. In a sense, we believe that collaborative editing is linked to the recent trend of analyzing and editing shape collections, but at the scale of the shape itself.

Control structures for the deformation. Most of the existing deformation methods aim at defining good deformations from simple interactions applied on control structures. Creating these structures automatically is a ill-posed problem, as it requires to infer unknown deformations from the shape geometry. Thus this tedious process is mainly left to the user. However, in Chapter 4, we showed that sphere-meshes can lead to interesting high-level structures for the deformation, automatically computed from the mesh geometry. The ability of sphere-meshes to faithfully abstract the shape at extreme simplification levels has led to powerful multi-resolution control structures. An alternative solution would be to create interactive systems, where the user provides a few high-level inputs to remove ambiguities and help the automatic construction of the deformation structure. Performing such semi-automatic approaches could improve existing control structures and lead to more powerful deformation tools. For instance embedding more knowledge on the original model, such as the shape similarities at different scales, could result in new non-local deformations. We are not aware of any control structure of this kind but we believe in their potential.

For example animating characters with similar parts could be resumed to the definition of a set of motions on one part and its replication, with possible phases, on similar regions.

Computational interactions. Users are more and more prone to be part of interactive modeling processes. The development of 3D sensors, 3D printers and virtual reality bring 3D worlds in the daily life of many individuals. Thus modeling tools should be made accessible to anyone from novices to experts. To this end, they have to be easy to discover, understand and efficient while still providing a great flexibility for expert users. We believe that the notion of computational interactions, interleaving user interactions and complex shape analysis, is essential for these new kind of applications. In fact, this concept has already been applied in few computer graphics fields. In 3D printing, some algorithms compute the weak sections of a model and highlight them, so that the user can interactively modify the shape until it becomes printable [US13]. Other methods help the user designing stable objects in a similar trial-and-error process where the stability is recomputed on-the-fly [PWLSH13]. Developing new human-computer interactions can also lead to interesting modeling applications. For instance, leaving the classical 2D inputs (mouse and keyboard) to provide new 3D interactions might improve many difficult modeling tasks. Some methods use a physical 3D modular structure as a skeleton to deform a virtual shape [JPG+14]. In conclusion, we believe exploration of novel 3D input methods, user perceptions of shapes and tasks, and automation of complex geometric processes to be interesting future directions of research for interactive modeling systems.

Learning. Most of our work requires to understand both the user abilities and the characteristics of shape analysis methods. Learning algorithms can greatly improve our results. In Chapter 3 we highlighted that there is not a clear agreement among users on a best pair of motions for traveling in a scene. Thus, our *LazyNav* system could use a learning process to automatically deduce the best combination of motions for every individual. Working on *SimSelect*, we already proved that learning algorithms can help to retrieve similar selections by using support vector machines. Going further, the selections of the user could also be learned to enhance the expansion algorithm. For instance, the local adjustments performed by the user to improve similar selection boundaries could be used to enhance the classification model for the next regions. We believe that, in many cases, analyzing the trial-and-error processes performed by the user will lead to a better understanding of the tasks and to an improvement of interactive modeling tools. This analysis could be either explicit or implicit. In the former case the user will deliberately points out an improvement to the system, while in the latter the user won't be aware of the analysis and the system will automatically classify her actions. Finally, although most shape representations are only based on visual and geometric properties of objects, we believe that many modeling systems would take advantage of a higher knowledge on the shape structure and functions as well. For instance, the classification of thousands of shapes could be used to learn the similarities and dissimilarities among categories or sub-categories, leading to more involved modeling systems, closer to the human analysis and perception of a shape.

APPENDIX A

PUBLICATIONS

Animated Mesh Approximation With Sphere-Meshes
Jean-Marc Thiery, Emilie Guy, Tamy Boubekeur, and Elmar Eisemann
Submitted to ACM Transactions on Graphics

LazyNav: 3D Ground Navigation with Non-Critical Body Parts
Emilie Guy, Parinya Punpongsanon, Daisuke Iwai, Kosuke Sato and Tamy Boubekeur
IEEE 3DUI 2015 - Best Paper Award

Ground Navigation in 3D Scenes using Simple Body Motions
Parinya Punpongsanon, Emilie Guy, Tamy Boubekeur, Daisuke Iwai and Kosuke Sato
International Conference on Artificial Reality and Telexistence 2014 - Demo Program

SimSelect: Similarity-based selection for 3D surfaces
Emilie Guy, Jean-Marc Thiery and Tamy Boubekeur
EUROGRAPHICS 2014 - Computer Graphics Forum.

Sphere-Meshes: Shape Approximation using Spherical Quadric Error Metrics
Jean-Marc Thiery, Emilie Guy and Tamy Boubekeur
ACM SIGGRAPH Asia 2013 - ACM Transactions on Graphics.

Click & Draw Selection
Emilie Guy, Jean-Marc Thiery and Tamy Boubekeur
ACM SIGGRAPH 2013 - Poster Program.

APPENDIX B

SIMILARITY BASED SELECTION - USER FEEDBACKS

In this annex, we report initial user feedback gathered after an interactive session. We provide to users a short demonstration video, a simple user guide (Section B.1) and a description of tasks to perform (Section B.2). After the interactive session we ask them to fill a form. Finally, we report users selection results (Section B.3) and statistics (Section B.4).

B.1 Simple User Guide

This guide describes the interface of the application (see Figure B.1) and the different interactions the user can perform.

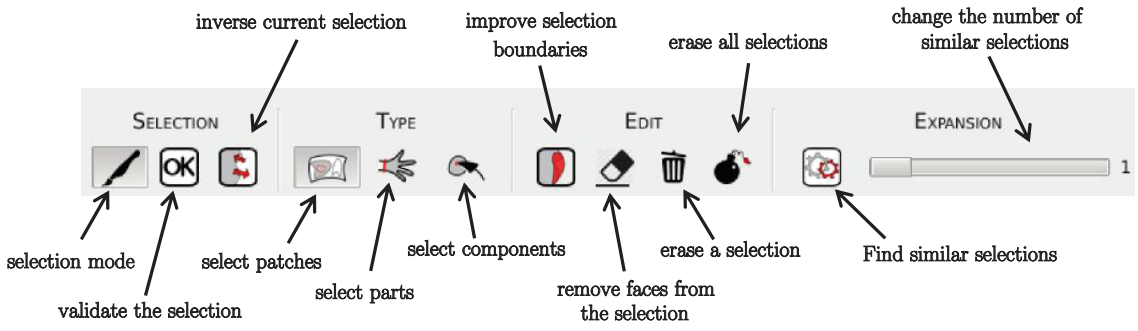


Figure B.1 – **SimSelect interface:** and explanation of the different available options.

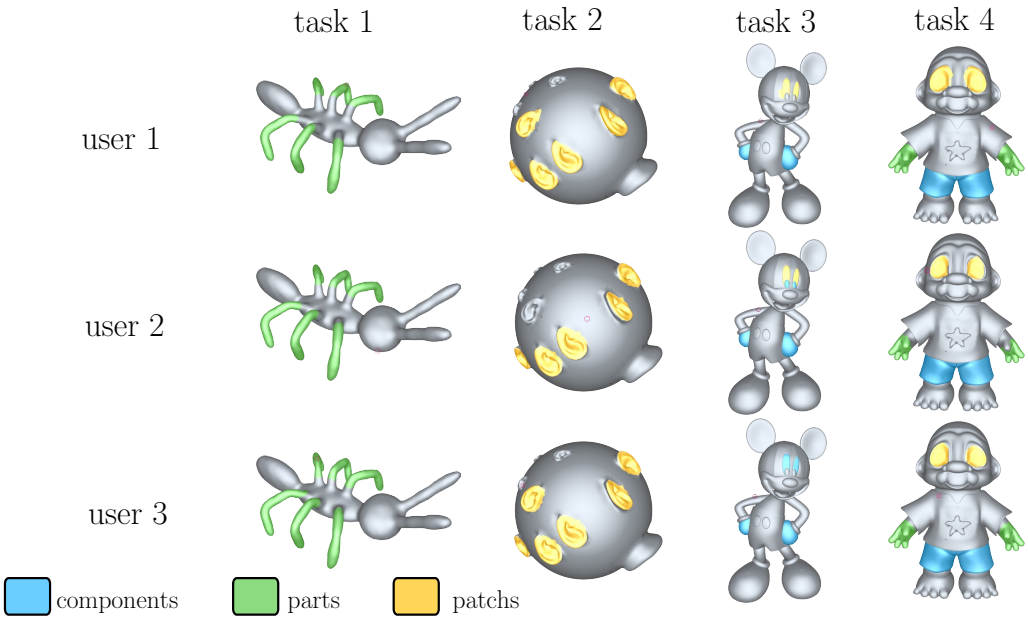
- Move around the object: use the mouse on the background.
  - wheel : zoom
  - LMB : rotate
  - RMB : translate
- Change the tool size : use the mouse wheel on the object
- User interaction:
  - Click on connected components
  - Brush over patches
  - Cut along parts boundaries

B.2 Training instructions

Thank you for participating to this training session. During the following, you will be able test our *SimSelect* tool that allows to perform selections on models and to find similar selections.

1. Please, watch the small video above (A. Demonstration Video) to learn more about our tool.
2. Perform the following selections on these different models. When you are satisfied with your selection, validate it by clicking the *"ok"* button, or by pressing *"Enter"*:
   - on the ant model
     - select the six legs
   - on the sphere with many ears model
     - select all the ears of average size
   - on the mickey model
     - select the two eyes
     - select the two hands
   - on the troll model
     - select the two eyes
     - select the pants
     - select the two hands
3. Thanks you for your participation, please answer the following form to give us your personal feedback

B.3 Resulting selections



user 4

user 5

user 6

user 7

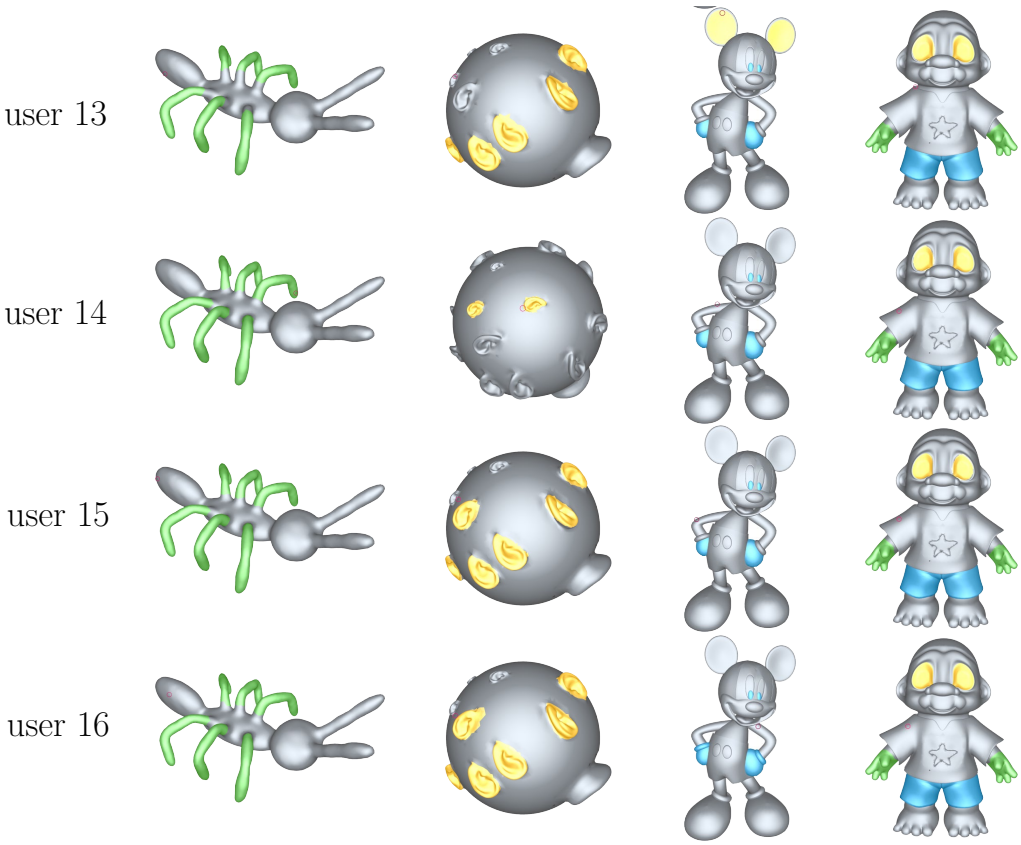
user 8

user 9

user 10

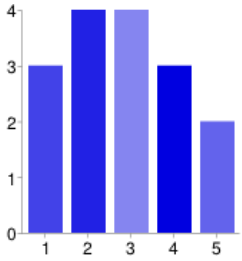
user 11

user 12



B.4 User Feedback Statistics

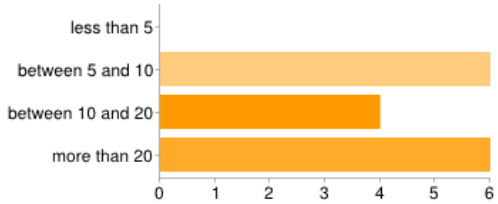
How often do you use graphics softwares ?



1: never	3	19%
2: occasionally	4	25%
3: regularly	4	25%
4: often	3	19%
5: always	2	13%

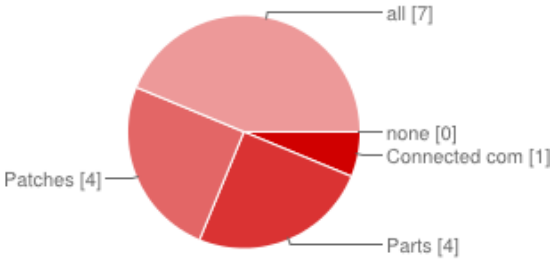
Your experience with the SimSelect tool

How many selections have you performed?



less than 5	0	0%
between 5 and 10	6	38%
between 10 and 20	4	25%
more than 20	6	38%

Which type of selection is the most useful for you?

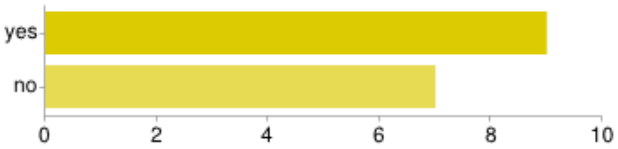


Connected components	1	6%
Parts	4	25%
Patches	4	25%
all	7	44%
none	0	0%

Why?

because it allows to select a component by indicating its limit (part) / Each type has its flaws and strengths (all)/ They are useful for different types of features (all)/Because it's a tool I understand (part)/they complete each others weaknesses (all)/ easier to use compare to the parts Very handy and fast (patch) / Even if it doesn't work every where, it is the fastest tool(part) Because selection is often tedious, even on simple objects (all)

Have you been annoyed by something during your selection?

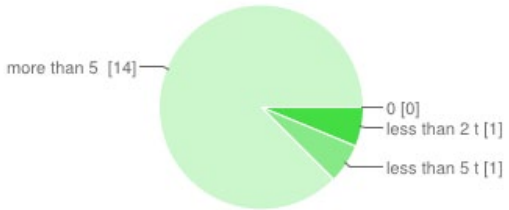


yes	9	56%
no	7	44%

If yes, please give us more details.

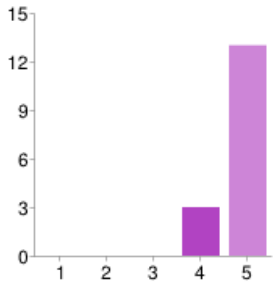
boundary improvement / Unexpected app crashes / sometimes difficult to understand the similar selections done automatically / While rotating the view, it is not handy to have to focus on clicking outside the object / he slider does not reset to 0 / the navigation mode is not handy (you can modify the selection by mistake) / the expansion slider!! / Need to test if the part is connected or not / Usual camera motion control are used to perform the selection

How many times have you used the expansion tool?



0	0	0%
less than 2 times	1	6%
less than 5 times	1	6%
more than 5 times	14	88%

How useful is the expansion tool?



1: never	0	0%
2: occasionally	0	0%
3: regularly	0	0%
4: often	3	19%
5: always	13	81%

Do you have any additional comments on the selection tool?

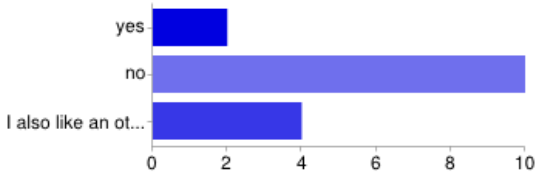
no /Nice user experience. / Good tool

Do you have any additional comments on the expansion tool?

no /Great tool /not always intuitive

Comparison with existing softwares

Do you prefer an other selection tool available in an existing software?

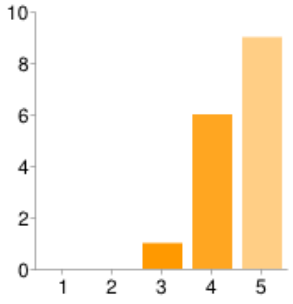


yes	2	13%
no	10	63%
I also like an other tool	4	25%

If you didn't say no, please explain why.

to be more accurate, i think other selection tools are complementary of these ones (eg lasso for vertex' selection) / I don't know many selection tools / I also like the lasso /I don't know any selection tools / I don't use other selection tools / auto boundary improvement / No need to switch to wireframe mode to adjust the selection. / A selection looking more like the one in Photoshop / Because i don't use any other software.

Would you use such a selection system if it was available in your favorite CG software?



1: never	0	0%
2: occasionally	0	0%
3: regularly	1	6%
4: often	6	38%
5: always	9	56%

Other general comments ?

Hope to see that in Blender soon ! / no / I am not really a user of CG softwares / Would be nice to separate selection from view control (with Alt-key for example)

BIBLIOGRAPHY

[AB99] Nina Amenta and Marshall Bern. Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999.

[ACK01] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *ACM symposium on Solid modeling and applications*, pages 249–266. ACM, 2001.

[AdVDI03] Pierre Alliez, Éric Colin de Verdière, Olivier Devillers, and Martin Isenburg. Isotropic surface remeshing. In *Shape Modeling International, 2003*, pages 49–58. IEEE, 2003.

[ASC11] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1626–1633. IEEE, 2011.

[ATC+08] O.K.C. Au, C.L. Tai, H.K. Chu, D. Cohen-Or, and T.Y. Lee. Skeleton extraction by mesh contraction. In *ACM Transactions on Graphics (TOG)*, volume 27, page 44. ACM, 2008.

[Aut] Autodesk. 3D Studio Max.

[AWB+14] Vamsi Kiran Adhikarla, Pawel Wozniak, Attila Barsi, Dave Singhal, Péter Tamás Kovács, and Tibor Balogh. Freehand interaction with large-scale 3d map data. In *3DTV-Conference(3DTV-CON), 2014*, pages 1–4. IEEE, 2014.

[BAS14] J. Andreas Bærentzen, Rinat Abdrashitov, and Karan Singh. Interactive shape modeling using a skeleton-mesh co-representation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 33(4), 2014.

[BBGO11] Alexander M. Bronstein, Michael M. Bronstein, Leonidas J. Guibas, and Maks Ovsjanikov. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Transactions on Graphics*, 30(1):1:1–1:20, 2011.

[BCBB14] Silvia Biasotti, Andrea Cerri, Alex Bronstein, and Michael Bronstein. Quantifying 3d shape similarity using maps: Recent trends, applications and perspectives. In *Eurographics - State of the Art Reports*, pages 135–159. The Eurographics Association, 2014.

[BCF+08] Doug A. Bowman, Sabine Coquillart, Bernd Froehlich, Michitaka Hirose, Yoshifumi Kitamura, Kiyoshi Kiyokawa, and Wolfgang Stuerzlinger. 3d user interfaces: New directions and perspectives. *Computer Graphics and Applications, IEEE*, 28(6):20–36, 2008.

[BCG08] Mirela Ben-Chen and Craig Gotsman. Characterizing shape using conformal factors. In *Proceedings of the 1st Eurographics Conference on 3D Object Retrieval*, 3DOR '08, pages 1–8. Eurographics Association, 2008.

[BEKB15] Davide Boscaini, Davide Eynard, Drosos Kourounis, and Michael M. Bronstein. Shape-from-operator: recovering shapes from intrinsic operators. *Computer Graphics Forum*, 34(2):265–274, 2015.

[BJP00] Serge Belongie, Malik Jitendra, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *NIPS*, volume 2, page 3, 2000.

[BK10] Michael M Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1704–1711. IEEE, 2010.

[Blu67] Harry Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, volume 32, pages 362–380. MIT Press, 1967.

[BM92] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256, 1992.

[BO02] Gareth Bradshaw and Carol O'Sullivan. Sphere-tree construction using dynamic medial axis approximation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 33–40. ACM, 2002.

[BO04] Gareth Bradshaw and Carol O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics (TOG)*, 23(1):1–26, 2004.

[BP07] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. In *ACM Transactions on Graphics (TOG)*, volume 26, page 72. ACM, 2007.

[BPK+07] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Lévy, Stephan Bischoff, and Christian Röossl. Geometric modeling based on polygonal meshes, 2007.

[BR07] Benedict J. Brown and Szymon Rusinkiewicz. Global non-rigid alignment of 3-d scans. *ACM Transactions Graphics*, 26(3), July 2007.

[BS91] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 251–256. ACM, 1991.

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

[CB14] Stéphane Calderon and Tamy Boubekeur. Point morphology. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2014)*, 2014.

[CH14] Tuncay Cakmak and Holger Hager. Cyberith virtualizer: a locomotion device for virtual reality. In *ACM SIGGRAPH 2014 Emerging Technologies*, page 6. ACM, 2014.

[CL05] Frédéric Chazal and André Lieutier. The λ-medial axis. *Graphical Models*, 67(4):304–331, 2005.

[CL06] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[CL11] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM TIST*, 2(3):27, 2011.

[CM02] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[CNSD93] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the cave. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 135–142, New York, NY, USA, 1993. ACM.

[Cra12] Keenan Crane. Cs 177: Discrete differential geometry, 2012. http://brickisland.net/cs177fa12/?p=104, last visit 2015-09-23.

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

[CSM07] Nicu D Cornea, Deborah Silver, and Patrick Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization & Computer Graphics*, (3):530–548, 2007.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[dATTS08] Edilson de Aguiar, Christian Theobalt, Sebastian Thrun, and Hans-Peter Seidel. Automatic Conversion of Mesh Animations into Skeleton-based Animations. volume 27, pages xx–xx, Hersonissos, Crete, Greece, 4 2008.

[DDL14] Olivier Dionne and Martin De Lasa. Geodesic binding for degenerate character geometry using sparse voxelization. *Visualization and Computer Graphics, IEEE Transactions on*, 20(10):1367–1378, 2014.

[DDSD03] Xavier Décoret, Frédo Durand, François X. Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. *ACM Transactions on Graphics*, 22(3), 2003.

[DFMPS04] Leila De Floriani, Paola Magillo, Enrico Puppo, and Davide Sobrero. A multi-resolution topological representation for non-manifold meshes. *Computer-Aided Design*, 36(2):141–159, 2004.

[DGGDV11] Fernando De Goes, Siome Goldenstein, Mathieu Desbrun, and Luiz Velho. Exoskeleton: Curve network abstraction for 3d shapes. *Computers & Graphics*, 35(1), 2011. Extended Papers from Non-Photorealistic Animation and Rendering (NPAR) 2010.

[DH09] Timothy A. Davis and William W. Hager. Dynamic supernodes in sparse cholesky update/downdate and triangular solves. *ACM Transactions on Mathematical Software (TOMS)*, 35(4):27:1–27:23, 2009.

[DK12] Tal Darom and Yosi Keller. Scale-invariant features for 3-d mesh models. *Image Processing, IEEE Transactions on*, 21(5):2758–2769, 2012.

[DLM11] Zheng-Jie Deng, Xiao-Nan Luo, and Xiao-Ping Miao. Automatic cage building with quadric error metrics. *Journal of Computer Science and Technology*, 26(3):538–547, 2011.

[DS10] Doug DeCarlo and Matthew Stone. Visual explanations. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, pages 173–178. ACM, 2010.

[dTL08] Rodrigo de Toledo and Bruno Lévy. Visualization of industrial structures with implicit gpu primitives. In *ISVC, International Symposium on Visual Computing*, 2008.

[DZ04] Tamal K. Dey and Wulue Zhao. Approximate medial axis as a voronoi subcomplex. *Computer-Aided Design*, 36(2):195–202, 2004.

[FKS+04] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 23:652–663, 2004.

[FLL11] Lubin Fan, Ligang Liu, and Kun Liu. Paint mesh cutting. In *Computer Graphic Forum (Proceedings of Eurographics)*, volume 30, pages 603–611, 2011.

[FML12] Lubin Fan, Min Meng, and Ligang Liu. Sketch-based mesh cutting: A comparative study. *Graphical Models*, 74(6):292 – 301, 2012.

[FS98] Bianca Falcidieno and Michela Spagnuolo. A shape abstraction paradigm for modeling geometry and semantics. In *Computer Graphics International, 1998. Proceedings*, pages 646–656. IEEE, 1998.

[FTB13] Noura Faraj, Jean-Marc Thiery, and Tamy Boubekeur. Progressive medial axis filtration. In *ACM SIGGRAPH 2013, Technical Briefs*, 2013.

[GÖ99] Bernd Gärtner. Fast and robust smallest enclosing balls. In *Proceedings of the 7th Annual European Symposium on Algorithms*, ESA '99, pages 325–338, 1999.

[GAB12] Thierry Guillemot, Andrès Almansa, and Tamy Boubekeur. Non local point set surfaces. In *Proceedings of the International Conference on 3D Imaging, Moldeing, Processing, Visualization and Transmission (3DIMPVT)*, 2012.

[Gar99] Michael Garland. *Quadric-based polygonal surface simplification*. PhD thesis, Georgia Institute of Technology, 1999.

[GAWJ15] Paul Guerrero, Thomas Auzinger, Michael Wimmer, and Stefan Jeschke. Partial shape matching using transformation parameter similarity. *Computer Graphics Forum*, 34(1):239–252, 2015.

[GCO06] Ran Gal and Daniel Cohen-Or. Salient geometric features for partial shape matching and similarity. *ACM Transactions on Graphics*, 25(1):130–150, 2006.

[GG07] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. *ACM Transaction on Graphics*, 26(3), 2007.

[GGG08] Gaël Guennebaud, Marcel Germann, and Markus Gross. Dynamic sampling and rendering of algebraic point set surfaces. *Computer Graphics Forum*, 27(2):653–662, 2008.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[GH01] Bernd Gärtner and Thomas Herrmann. Computing the width of a point set in 3-space, 2001.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[HBA12] Kristian Hildebrand, Bernd Bickel, and Marc Alexa. crdbrd: Shape fabrication by sliding planar slices. In *Computer Graphics Forum*, volume 31, pages 583–592, 2012.

[HRGMI14] Juan David Hincapié-Ramos, Xiang Guo, Paymahn Moghadasian, and Pourang Irani. Consumed endurance: A metric to quantify arm fatigue of mid-air interactions. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 1063–1072. ACM, 2014.

[HSZ$^+$] A. Hirsch, R. Shenberg, S. Zippel, T. Blackman, and B. Tucker. Zigfu development kit for unity3d.

[HWCO$^+$13] Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. L1-medial skeleton of point cloud. *ACM Transactions on Graphics*, 32:65:1–65:8, 2013.

[ISKY04] Sei Ikeda, Tomokazu Sato, Masayuki. Kanbara, and Naokazu Yokoya. An immersive telepresence system with a locomotion interface using high-resolution omnidirectional movies. In *Pattern Recognition (ICPR)*, volume 4, pages 396–399, 2004.

[JBPS11] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 30(4):78:1–78:8, 2011.

[JDKL14] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*, 2014.

[JH99] Andrew E. Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(5):433–449, 1999.

[JHH14] Jacek Jankowski, Thomas Hulin, and Martin Hachet. A study of street-level navigation techniques in 3d digital cities on mobile touch devices. In *3D User Interfaces (3DUI), 2014 IEEE*, pages 35–38, 2014.

[JLCW06] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. Easy mesh cutting. volume 25, pages 283–291, 2006.

[JLW10] Zhongping Ji, Ligang Liu, and Yigang Wang. B-mesh: A modeling system for base meshes of 3d articulated shapes. In *Computer Graphics Forum (Proceedings of Pacific Graphics)*, volume 29, pages 2169–2177, 2010.

[JMD+07] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Transactions on Graphics*, 26(3), 2007.

[JP04] Doug L. James and Dinesh K. Pai. Bd-tree: output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3):393–398, 2004.

[JPG+14] Alec Jacobson, Daniele Panozzo, Oliver Glauser, Cédric Pradalier, Otmar Hilliges, and Olga Sorkine-Hornung. Tangible and modular input device for character articulation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 33(4):82:1–82:12, 2014.

[JSW05] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics*, 24(3):561–566, 2005.

[KBLB12] Iasonas Kokkinos, Michael M Bronstein, Roee Litman, and Alex M Bronstein. Intrinsic shape context descriptors for deformable shapes. In *Computer Vision and Pattern Recognition (CVPR)*, pages 159–166. IEEE, 2012.

[KCvO08] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics*, 27(4):105:1–105:23, 2008.

[KDC+08] Ladislav Kavan, Simon Dobbyn, Steven Collins, Jiří Žára, and Carol O'Sullivan. Polypostors: 2d polygonal impostors for 3d crowds. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, I3D '08, pages 149–155, 2008.

[KPNK03] Marcel Körtgen, Gil-Joo Park, Marcin Novotni, and Reinhard Klein. 3d shape matching with 3d shape contexts. In *The 7th central European seminar on computer graphics*, volume 3, pages 5–17, 2003.

[Kv05] Ladislav Kavan and Jiří Žára. Spherical blend skinning: A real-time deformation of articulated models. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 9–16, 2005.

[Lav12] Guillaume Lavoué. Combination of bag-of-words descriptors for robust partial shape retrieval. *The Visual Computer*, 28(9):931–942, 2012.

[LB15] H'el'ene Legrand and Tamy Boubekeur. Morton integrals for high speed geometry simplification. In *Proceedings of the ACM SIGGRAPH/ EUROGRAPHICS High Performance Graphics Conference (HPG) 2015*, 2015.

[LCWK07] Lin Lu, Yi-King Choi, Wenping Wang, and Myung-Soo Kim. Variational 3d shape segmentation for bounding volume computation. In *Computer Graphics Forum*, volume 26, pages 329–338, 2007.

[LD12] Binh Le and Zhigang Deng. Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics*, 31(6), 2012.

[LD14a] Binh Huy Le and Zhigang Deng. Robust and accurate skeletal rigging from mesh sequences. *ACM Trans. Graph.*, 33(4):84:1–84:10, July 2014.

[LD14b] Binh Huy Le and Zhigang Deng. Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics*, 33(4):84:1–84:10, 2014.

[LF09] Yaron Lipman and Thomas Funkhouser. Möbius voting for surface correspondence. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 72:1–72:12. ACM, 2009.

[Lin00] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *ACM SIGGRAPH*, pages 259–262, 2000.

[LLCO08] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Transactions on Graphics (TOG)*, 27(3):78:1–78:10, 2008.

[LLS+05] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and Hans-Peter Seidel. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design*, 22(5):444–465, 2005.

[LSS09] Jiangyu Liu, Jian Sun, and Heung-Yeung Shum. Paint selection. In *ACM Transactions on Graphics*, volume 28, page 69, 2009.

[LT12] George Leifman and Ayellet Tal. Mesh colorization. In *Computer Graphics Forum*, volume 31, pages 421–430, 2012.

[LT13] George Leifman and Ayellet Tal. Pattern-driven colorization of 3d surfaces. In *Computer Vision and Pattern Recognition (CVPR)*, pages 241–248, 2013.

[LZ09] Bruno Lévy and Hao Richard Zhang. Spectral mesh processing. In *ACM SIGGRAPH 2009 Courses*, page 8. ACM, 2009.

[MBM05] G. Mori, S. Belongie, and J. Malik. Efficient shape matching using shape contexts. *Pattern Analysis and Machine Intelligence (PAMI)*, 27(11):1832–1837, 2005.

[MDSB02] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*, pages 35–57. 2002.

[Mel12] Nicolas Mellado. Analysis of 3d objects at multiple scales: application to shape matching, 2012.

[MFL11] Min Meng, Lubin Fan, and Ligang Liu. icutter: A direct cut out tool for 3d shapes. *Computer Animation & Virtual World*, 22(4):335–342, 2011.

[MGB+12] Nicolas Mellado, Gaël Guennebaud, Pascal Barla, Patrick Reuter, and Christophe Schlick. Growing least squares for the analysis of manifolds in scale-space. In *Computer Graphics Forum*, volume 31, pages 1691–1701, 2012.

[MGP06] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM Transactions on Graphics*, 25(3):560–568, 2006.

[MGP10] B. Miklos, J. Giesen, and M. Pauly. Discrete scale axis representations for 3d geometry. *ACM Transactions on Graphics (TOG)*, 29(4):101, 2010.

[MH12] Waleed Mohamed and A Ben Hamza. Reeb graph path dissimilarity for 3d object matching and retrieval. *The Visual Computer*, 28(3):305–318, 2012.

[MMS13] James Mccrae, Niloy J. Mitra, and Karan Singh. Surface perception of planar abstractions. *ACM Transactions on Applied Perception (TAP)*, 10(3), 2013.

[MPVF11] A. Maximo, R. Patro, A. Varshney, and R. Farias. A robust and rotationally invariant local surface descriptor with applications to non-local mesh processing. *Graphical Models*, 73(5):231 – 242, 2011.

[MPWC13] Niloy J Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3d geometry: Extraction and applications. In *Computer Graphics Forum*, volume 32, pages 1–23, 2013.

[MS98] Jon McCormack and Andrei Sherstyuk. Creating and rendering convolution surfaces. In *Computer Graphics Forum*, volume 17, pages 113–120, 1998.

[MSM11] James McCrae, Karan Singh, and Niloy J. Mitra. Slices: A shape-proxy based on planar sections. *ACM Transactions on Graphics*, 30(6):168:1–168:12, 2011.

[MTLT88] Nadia Magnenat-Thalmann, Richard Laperrière, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88*, pages 26–33, 1988.

[MWZ+14] Niloy J Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, Vladimir Kim, and Qi-Xing Huang. Structure-aware shape processing. In *ACM SIGGRAPH 2014 Courses*, page 13. ACM, 2014.

[MY09] Jean-Michel Morel and Guoshen Yu. A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2(2):438–469, 2009.

[MZL+09] Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. Abstraction of man-made shapes. *ACM Transactions on Graphics*, 28(5), 2009.

[NGH04] Xinlai Ni, Michael Garland, and John C Hart. Fair morse functions for extracting the topological structure of a surface mesh. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 613–622. ACM, 2004.

[NLB14] Mahdi Nabiyouni, Bireswar Laha, and Doug A Bowman. Poster: Designing effective travel techniques with bare-hand interaction. In *3D User Interfaces (3DUI)*, pages 139–140, 2014.

[NSACO05] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics*, 24(3):1142–1147, 2005.

[OBCS+12] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: a flexible representation of maps between shapes. *ACM Transactions on Graphics (TOG)*, 31(4):30, 2012.

[OMPG13] Maks Ovsjanikov, Quentin Mérigot, Viorica Pătrăucean, and Leonidas Guibas. Shape matching via quotient spaces. In *Computer Graphics Forum*, volume 32, pages 1–11, 2013.

[PGK02] Mark Pauly, Markus Gross, and Leif P Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of the conference on Visualization'02*, pages 163–170. IEEE Computer Society, 2002.

[Pho] Adobe Photoshop. http:/www.adobe.com/support/photoshop/.

[Pix01] Pixologic. Zbrush, 2001.

[PKG03] Mark Pauly, Richard Keiser, and Markus Gross. Multi-scale feature extraction on point-sampled surfaces. In *Computer Graphics Forum*, volume 22, pages 281–289, 2003.

[PMW+08] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J. Guibas. Discovering structural regularity in 3d geometry. *ACM Transactions on Graphics*, 27(3):43:1–43:11, 2008.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics*, 2(1):15–36, 1993.

[PSM+11] Julien Pettré, Orianne Siret, Maud Marchal, Jean-Baptiste de la Rivire, and Anatole Lécuyer. Joyman: An immersive and entertaining interface for virtual locomotion. In *SIGGRAPH Asia 2011 Emerging Technologies*, SA '11, pages 22:1–22:1, New York, NY, USA, 2011. ACM.

[PWLSH13] Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. Make It Stand: Balancing shapes for 3D fabrication. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 32(4):81:1–81:10, 2013.

[RB93] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximation for rendering complex scenes. *Modeling in Computer Graphics*, pages 455–465, 1993.

[RBG+09] Martin Reuter, Silvia Biasotti, Daniela Giorgi, Giuseppe Patanè, and Michela Spagnuolo. Discrete laplace–beltrami operators for shape analysis and segmentation. *Computers & Graphics*, 33(3):381–390, 2009.

[RBSJ14] Mattias Roupé, Petra Bosch-Sijtsema, and Mikael Johansson. Interactive navigation interface for virtual reality using the human body. *Computers, Environment and Urban Systems*, 43(0):42 – 50, 2014.

[RL00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH*, pages 343–352, 2000.

[RLOW13] Gang Ren, Chuan Li, Eamonn O'Neill, and Philip Willis. 3d freehand gestural navigation for interactive public displays. *Computer Graphics and Applications*, 33(2):47–55, 2013.

[ROA+13] Raif M Rustamov, Maks Ovsjanikov, Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Leonidas Guibas. Map-based exploration of intrinsic shape differences and variability. *ACM Transactions on Graphics (TOG)*, 32(4):72, 2013.

[Rus04] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *3D Data Processing, Visualization and Transmission (3DPVT)*, pages 486–493. IEEE, 2004.

[SFC+11] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from a single depth image. In *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011.

[SFM07] Avneesh Sud, Mark Foskey, and Dinesh Manocha. Homotopy-preserving medial axis simplification. *International Journal of Computational Geometry & Applications*, 17(05):423–451, 2007.

[SKS12] Svetlana Stolpner, Paul Kry, and Kaleem Siddiqi. Medial spheres for shape approximation. *Pattern Analysis and Machine Intelligence*, 34(6):1234–1240, 2012.

[SLA15] David Salinas, Florent Lafarge, and Pierre Alliez. Structure-Aware Mesh Decimation. *Computer Graphics Forum*, page 20, January 2015.

[SNB+12] Justin Solomon, Andy Nguyen, Adrian Butscher, Mirela Ben-Chen, and Leonidas Guibas. Soft maps between surfaces. In *Computer Graphics Forum*, volume 31, pages 1617–1626, 2012.

[SOG09] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, volume 28, pages 1383–1392, 2009.

[SP86] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Transactions on Graphics*, 20(4):151–160, 1986.

[SS10] Ryan Schmidt and Karan Singh. Meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, page 6, 2010.

[SSGD03] Hari Sundar, Deborah Silver, Nikhil Gagvani, and Sven Dickinson. Skeleton based shape matching and retrieval. In *Shape Modeling International, 2003*, pages 130–139. IEEE, 2003.

[STB12] Leila Schemali, Jean-Marc Thiery, and Tamy Boubekeur. Automatic line handles for freeform deformation. In *Eurographics 2012 (Short)*, 2012.

[SVAG14] AL. Simeone, E. Velloso, J. Alexander, and H. Gellersen. Feet movement in desktop 3d interaction. In *3D User Interfaces (3DUI)*, pages 71–74, 2014.

[SW03] Scott Schaefer and Joe Warren. Adaptive vertex clustering using octrees. In *Proceedings of SIAM Geometric Design and Computing*, pages 491–500, 2003.

[TAOZ12] Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. Mean curvature skeletons. *Computer Graphics Forum*, 31(5):1735–1744, 2012.

[TBTB12] Jean-Marc Thiery, Bert Buchholz, Julien Tierny, and Tamy Boubekeur. Analytic curve skeletons for 3d surface modeling and processing. *Computer Graphics Forum (Proc. Pacific Graphics 2012)*, 2012.

[TPT15] Panagiotis Theologou, Ioannis Pratikakis, and Theoharis Theoharis. A comprehensive overview of methodologies and performance evaluation frameworks in 3d mesh segmentation. *Computer Vision and Image Understanding*, 135:49 – 82, 2015.

[TSS+11] Kenshi Takayama, Ryan Schmidt, Karan Singh, Takeo Igarashi, Tamy Boubekeur, and Olga Sorkine. Geobrush: Interactive mesh geometry cloning. In *Computer Graphics Forum*, volume 30, pages 613–622, 2011.

[Tur92] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH 92)*, 26(2):55–64, July 1992.

[TVD09] Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi. Partial 3d shape retrieval by reeb pattern unfolding. In *Computer Graphics Forum*, volume 28, pages 41–55, 2009.

[TZCO09] Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or. Curve skeleton extraction from incomplete point cloud. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 71:1–71:9. ACM, 2009.

[US13] Nobuyuki Umetani and Ryan Schmidt. Cross-sectional structural analysis for 3d printing optimization. In *SIGGRAPH Asia Technical Brief*, 2013.

[VBG+13] Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. Implicit skinning: Real-time skin deformation with contact modeling. *ACM Transactions on Graphics*, 32(4):125:1–125:12, 2013.

[VGB+14] Rodolphe Vaillant, Gäel Guennebaud, Loïc Barthe, Brian Wyvill, and Marie-Paule Cani. Robust iso-surface tracking for interactive character skinning. *ACM Transactions on Graphics*, 33(6):189:1–189:11, 2014.

[VKZHCO11] Oliver Van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. A survey on shape correspondence. In *Computer Graphics Forum*, volume 30, pages 1681–1707, 2011.

[WMKG07] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete laplace operators: no free lunch. In *Symposium on Geometry processing*, pages 33–37, 2007.

[WZS+06] Rui Wang, Kun Zhou, John Snyder, Xinguo Liu, Hujun Bao, Qunsheng Peng, and Baining Guo. Variational sphere set approximation for solid objects. *The Visual Computer*, 22(9):612–621, 2006.

[XLG12] Chuhua Xian, Hongwei Lin, and Shuming Gao. Automatic cage generation by improved obbs for mesh deformation. *Vis. Comput.*, 28(1):21–33, 2012.

[XZCOX09] Kai Xu, Hao Zhang, Daniel Cohen-Or, and Yueshan Xiong. Dynamic harmonic fields for surface processing. *Computers & Graphics*, 33(3):391–398, 2009.

[YJHS12] Kaan Yücer, Alec Jacobson, Alexander Hornung, and Olga Sorkine. Transfusive image manipulation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA)*, 31(6):176:1–176:9, 2012.

[YLL+09] Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. Isotropic remeshing with fast and exact computation of restricted voronoi diagram. In *Symposium on Geometry Processing*, pages 1445–1454, 2009.

[ZBQC13] Cédric Zanni, Adrien Bernhardt, Maxime Quiblier, and Marie-Paule Cani. Scale-invariant integral surfaces. In *Computer Graphics Forum*, volume 32, pages 219–232, 2013.

[ZG04] Steve Zelinka and Michael Garland. Similarity-based surface modelling using geodesic fans. In *ACM SIGGRAPH Symposium on Geometry Processing*, pages 204–213. ACM, 2004.

[ZSMG] Zeev ZALEVSKY, Alexander SHPUNT, Aviad MAIZELS, and Javier GARCIA. Method and system for object reconstruction. patent WO2007043036.

[ZSS97] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *ACM SIGGRAPH '97*, pages 259–268, 1997.

[ZT10] Youyi Zheng and Chiew-Lan Tai. Mesh Decomposition with Cross-Boundary Brushes. In *Computer Graphics Forum*, volume 29, pages 527–535, 2010.