Digital Processing and Management Tools for 2D and 3D Shape Repositories

Waqar Saleem

Max-Planck-Institut für Informatik
Saarbrücken, Germany

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing)
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Datum des Kolloquiums — Date of Defense
18. Juni 2010 — June 18th, 2010

Waqar Saleem
Max-Planck-Institut für Informatik
Campus E1 4
66123 Saarbrücken, Germany
`wsaleem@mpi-inf.mpg.de`

Abstract

This thesis presents work on several aspects of 3D shape processing. We develop a learning based surface reconstruction algorithm that is robust to typical input artifacts and alleviates the restrictions imposed by previous such methods. Using the human shape perception motivated paradigm of representing a 3D shape by its 2D views obtained from its view sphere, we compute the shape's "best views", extend these views to obtain the more dynamic "best fly" of the shape and also compute shape complexity which is used to compare the shape with others so as to obtain an ordering. Our example based method to "correctly" reorient a 2D shape in an image is also presented as well as a strategy to approximate shape descriptor values on the view sphere using just a few samples. This allows to bypass the often time consuming requirement of evaluating the descriptor on a dense sampling of the view sphere to obtain an accurate representation. We also present our work on accelerating shape similarity retrieval by using techniques from text retrieval. Lastly, we present some of the guiding principles behind the maintenance and development of a large scale, publicly accessible shape repository.

Kurzfassung

Diese Arbeit präsentiert verschiedene Aspekte der Bearbeitung von 3D Objekten. Wir entwickeln einen Lern-basierten "Surface Reconstruction"-Algorithmus, der robust gegenüber üblichen Messartefakten ist und die Einschränkung solcher Methoden verbessert. Aufgrundlage der menschlichen Wahrnehmung von 3D Objekten als 2D Sichten aus verschiedenen Perspektiven des "View Spheres" berechnen wir den "Best View". Wir erweitern die Best Views zu einem dynamischen "Best Fly" um das Objekt herum. Weiterhin bestimmen wir die Komplexität eines Objekts und verwenden dies um Objekte mit anderen zu vergleichen. Daraus gewinnen wir eine Sortierung der Objekte anhand ihrer Komplexität. Ebenso stellen wir eine Exemplar-basierte Methode zur "richtigen" Orientierung eines 2D Objekts in einem Bild vor, sowie eine Strategie um die Deskriptor-Werte der View Sphere durch wenige Samples zu approximieren. Somit konnen wir die oft langsame Auswertung der Deskriptoren für eine feine Abtastung der View Sphere umgehen um eine genaue Darstellung des Objekts zu erhalten. Wir stellen weiterhin unsere Arbeit zur Beschleunigung zur Abfrage ähnlicher Objekte unter Verwendung von Retrieval-Techniken vor. Schließlich legen wir einige Grundprinzipien der Entwicklung und Wartung einer umfangreichen, öffentlich-zugänglichen Objekt-Sammlung dar.

Summary

Corresponding to the increase in use and popularity of digital 3D shape models in a variety of applications, methods to acquire, store and analyze these shapes are becoming more important. We start off by showing how these methods fit into a *digital shape pipeline* and then present our work in various stages of this pipeline.

A common way to digitize a real world object is to capture points on the shape's surface using a laser scanner. The resulting point cloud suffers from usual measurement artefacts like noise and outliers. Further, it may have holes or missing data. For subsequent processing, a cleaner, more convenient representation is required. We present a novel, learning based method that robustly processes such a point cloud to output a triangle mesh, which is the de facto standard for surface representation in computer graphics.

While shapes exist as three dimensional objects in the real world and, digitally, in the computer, they are ultimately presented on 2D media, e.g. monitor or paper. Presenting a 3D shape on 2D requires a projection giving rise to a *view* of the shape. As infinitely many 2D views of a 3D shape are possible and one can typically present just a few views, a strategy to rank views and select the "best" ones becomes necessary. We present our work on selecting *best views* of 2D and 3D shapes. We also extend the problem of finding best view to that of computing *best fly*–a dynamic representation of the shape–and show how our computed best views can easily be extended to solve the best fly problem.

The view finding methods above make use of the *view sphere* of a shape, specifically of computing certain descriptor values at discrete samples on the surface of the view sphere. The denser the view sphere is sampled, the more accurate are the results obtained. This introduces a tradeoff between accuracy and efficiency. We show how it is possible to obtain an accurate, continuous representation of descriptor values by sampling the view sphere only at a few discrete samples and thus maintaining efficiency.

Our techniques mentioned till now mostly work with and analyze a single 3D shape in isolation. The increase in popularity and ease of availability of 3D shapes gives rise to the need for a new kind of methods that can reason about shapes in context of a larger collection. It is desirable to know how a shape relates to others in a collection. Does the collection contain similar shapes? Are the other shapes more or less complex than my current shape? Are there shapes in the collection that were acquired under identical conditions? The list goes on. We have addressed the first two issues in our work. We present an indexing mechanism to significantly accelerate current shape similarity methods allowing us to rapidly search through a large collection for shapes similar to a query shape. Second, we leverage insights from human vision research to obtain a measure of the complexity of a shape and use it to sort a collection of shapes according to shape complexity.

Nearing the end, we present how many of the ideas presented above can be and are showcased in a publicly accessible, voluminous collection of shapes, whose upkeep, development and maintenance consumed a significant portion of the research time allotted for this thesis and in turn inspired new ideas.

Zusamenfassung

Aufgrund größer werdender Bedeutung von 3D Modelle werden Werkzeuge zum Erzeugen, Verteilen und Analysieren zunehmend wichtiger. In dieser Arbeit wird zuerst gezeigt, wie diese Methoden in eine *Digitale Shape Pipeline* hinein passen. Anschließend zeigen wir wie sich diese Arbeit in die verschiedenen Teile der Pipeline einfügt.

Der übliche Weg reale Objekten zu digitalisieren erfolgt über das Abtasten der Oberfläche mittels eines Laser-Scanners. Die so gewonnene Punktwolke beinhaltet Messfehler aufgrund von Rauschen und Ausreißern. Weiterhin kann die Punktwolke unvollständig erfasst worden sein. Für die anschließenden Schritte benötigt man eine saubere und einfach zu verwendendene Darstellung. Wir stellen eine neue Lern-basierte Methode vor, die eine robuste Umwandlung der Punktwolken in Triangle Mesh ermöglicht. Das Triangle Mesh ist der de facto Standard für die Darstellung von Oberflächen in der Computergrafik.

Obwohl drei dimensionale Objekte nur in der Realität und digital im Computer vorhanden sind, werden sie doch meist in 2D dargestellt z.B. auf dem Monitor oder auf Papier. Die Darstellung von 3D Objekten in 2D erfordert eine Projektion und damit eine *Sichtweise* auf das Objekt. Da es unendlich viele mögliche 2D Sichten eines 3D Objekts gibt und man normalerweise nur wenige darstellen kann, ist es notwendig die Sichten zu ordnen und die “Beste” auszuwählen. Wir präsentieren unsere Arbeiten zum auswählen der besten Sicht auf ein 3D Objekt. Wir erweitern dabei das “Best View”-Problem zum “Best Fly”-Problem, wobei wir eine dynamische Darstellung des Objekts berechnen. Dabei zeigen wir wie mit unserer Berechnung des Best Views einfach das Best Fly-Problem gelöst werden kann.

Üblicherweise verwenden die Methoden für das Finden bester Sichten die *View Sphere* eines 3D Objekts, im Speziellen berechnen sie werte bestimmter Deskriptoren auf diskreten Punkten der Oberfläche des View Sphere. Je feiner die View Sphere abgetastet wird, desto besser ist der Best View. D.h. es gibt einen Tradeoff zwischen Qualität und Effizienz. Wir zeigen wie es möglich ist eine genaue und kontinuierliche Darstellung des Deskriptors zu erhalten indem nur wenige Punkte verwendet werden und so die Effizienz erhalten bleibt.

Die bisher erwähnten Techniken bearbeiten und analysieren einzelne 3D Objekte meist unabhängig von einander. Die zunehmende Bedeutung und einfache Verfügbarkeit von 3D Objekten führt zu eimem Bedarf neuerer Methoden die mit Objekten im Kontext großer Sammlungen umgehen können. Es ist wünschenswert zu wissen, wie ein Objekt im Verhältnis zu anderen Objekten einer

Sammlung steht. Enthält die Sammlung ähnliche Objekte? Sind andere Objekte weniger komplex als ein gegebenes? Gibt es Objekte in der Sammlung, die unter ähnlichen Bedingungen digitalisiert wurden? Die Liste ließe sich fortführen. Wir behandeln in dieser Arbeit die ersten zwei Fragen. Wir präsentieren dazu eine Indizierung, die die gegenwärtigen Methoden zur Bestimmung der Ähnlichkeit von Objekten beschleunigt und die es erlaubt sehr schnell zu einem Anfrage-Objekt aehnliche Objekte in großen Sammlungen zu identifizieren. Bezüglich der zweiten Frage, verwenden wir Einsichten über die menschliche Wahrnehmung um ein Maßfür die Komplexität eines Objekts zu entwickeln und benutzen es um Objekte anhand ihrer Komplexität zu sortieren.

Schließlich zeigen wir wie viele der vorgestellten Ideen umgesetzt werden könnten und exemplarisch in einer umfangreichen und öffentlich-zugänglichen Sammlung von Objekten umgesetzt sind. Die Sammlung, für deren Entwicklung und Wartung ein großer Teil der Forschungszeit dieser Arbeit aufgewendet wurde, inspirierte neue Ideen.

Acknowledgements

I was lucky enough to conduct my post-graduate studies at one of the top destinations for computer science research world wide, the MPII in Saarbrücken. Here, I observed first-hand how cutting edge research is conducted and experienced the dynamic and intellectually stimulating environment out of which such work blossoms.

During my years in Saarbrücken, I came to know a wide variety of amazing people from close and afar who enriched my experience beyond words. They are Robert Bargmann, Irena Galić, Syed Ibne Hasan, Kanela Kaligosi, Abdul Qadar Kara, Mehmet Kiraz, Sadia Masood, Dimitrios Michail, Josiane Xavier Parreira, Imran Rauf, Saurabh Ray, Jeff Schoner, Hans Raj Tiwary, Amir Wasim, Ingmar Weber and Melanie Windl.

Any academic achievement of mine is due to the educators who have left indelible impressions on me through their excellence in teaching, enthusiasm towards their subjects, ability to inspire and keen interest in their students' development. Going backwards in time, these have been: at MAJU, Dr. Shabbir Ahsan, Dr. Zafar Ansari, Dr. Afaq Hyder (late), Mr. Masood Mir, Dr. Mehmood Naqvi, Dr. Ansaruddin Syed, and Dr. Abbas Zaidi; and from my school years, Mr. Salman Ahmad, Mr. Ranjit Bulathsinghala, Mr. Errol Fernando, Mrs. George, Mr. Lambert, Mr. Abdur Rehman, Mr. Saqlain and Mr. Vaz. At Saarland University, it was always a pleasure to attend lectures by Dr. Alexander G. Belyaev, Prof. Dr. Wolfgang J. Paul, and Prof. Dr.-Ing. Philipp Slusallek.

My studies at the MPII were funded by the IMPRS program and the AIM@SHAPE project. Living in a foreign land with a new language and oft confusing bureaucracy to deal with, things could easily have gotten messy for me. Kerstin Kathy Meyer-Ross of IMPRS and Sabine Budde and Conny Liegl of AG4 took it upon themselves to not only make all such processes invisible to me, but to ensure that my sojourn was as smooth and comfortable as it could be.

I am thankful to members of my PhD committee–Dr. Alexander Belyaev, Prof. Dr. Hans-Peter Seidel, Prof. Dr.-Ing. Philipp Slusallek, and Dr. Michael Wand–for taking time out of their busy schedules to attend my defense. In particular, Dr. Belyaev helped me through numerous revisions of this document.

Sans the resolute support of my family, my PhD may never have reached completion. I am grateful to them for affording me the space and freedom to pursue my studies over several long years, and to Mifrah, my wife, for her kind and patient understanding when those years dragged on longer than they should have.

The 2D and 3D shape models used in this dissertation are due to the AIM@SHAPE Shape Repository, the Stanford Data Archives, Farfield Technology, Cyberware and the MPEG7 dataset. Development of some of the ideas presented in the following pages is due to fruitful discussions with Boris Ajdin, Dr. Longin J. Latecki, Prof. Dr. Remco Veltkamp and Dr. Shin Yoshizawa.

Lastly, I want to extend my heartfelt gratitude to the numerous kebab shops strewn all over Saarbrücken. You nourished me in ways far more delectable than anything else I could otherwise have found.

Contents

List of Figures

Chapter 1

Introduction

After text, images and video, the next wave of digital media is expected to comprise 3D shape content. Indeed, an increasing number of shape repositories are being established either online or as in house catalogs. This creates a need for tools to analyze and manage contained shapes.

This thesis presents such techniques. While each of the techniques can be used outside the context of a shape library, almost all of them can be showcased in a repository and would contribute towards automation of common management tasks. Here, we provide a brief overview of each of our techniques and mention how they can be incorporated in a shape library.

There are several ways to create 3D content – it can be designed (from scratch) using specialist tools and softwares, synthesized from 2D images, obtained by editing existing content, or acquired from real world objects. Once created, the content can be cataloged for later reference, placed in virtual or augmented environments, animated, shared, re-edited and/or analyzed.

This gives rise to the notion of a *digital shape pipeline*. An object, whether real,

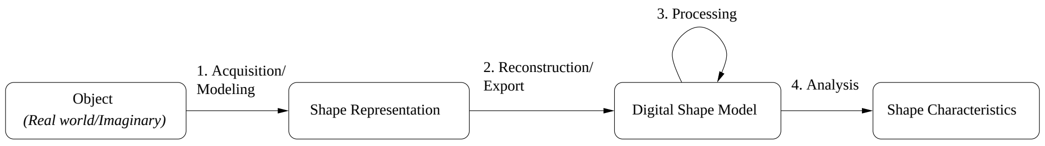


Figure 1.1: The digital shape pipeline

existing as a part specification or in an artist's imagination, is first given a digital representation. This could be the internal representation of the modeling software being used, or *range data* obtained from a 3D scanner. In this intial form, the shape is not useful for general use and needs to be converted to a standard format (e.g. VRML) or representation (e.g. triangle mesh). The resulting digital shape model might still not be ready for dissemination or analysis, thus requiring further processing like remeshing, simplification or smoothing. At this point, the digital shape can be used for any desired application.

Digital geometry processing refers to tools spanning the entire pipeline. Shape acquisition is the process of acquiring a digital representation of a real world object. Depending on how accurately the digital shape should represent its real counterpart, different methods can be used for this step. The most accurate representations are obtained using contact scanners or range scanners. The former move a mounted arm over the surface of the object and obtain coordinates of surface points using the position of the arm in relation to a fixed point. This method is unsuitable for shapes that are sensitive to touch or hard to reach physically. Laser scanners obtain point coordinates by projecting a laser beam on the surface and catching the reflected beam at a sensor. As projection and sensor positions and angles are known, surface points can be computed by triangulation. To maximize coverage of the shape, it has to be scanned multiple times from several directions. The obtained data is referred to as *range data* and hence laser scanners are also called range scanners. Due to relatively low prices of range scanners, range scanning is the method of choice for accurate 3D shape digitization.

Range data is subject to common measurement errors like noise and outliers, and shape specific properties like holes. Holes occur in the point cloud when the corresponding region of the shape's surface was not scanned, either due to incomplete coverage by the scanner or, as is more common, due to occlusion whereby some portion of the shape is hidden from the scanner's view by another portion. *Surface Reconstruction* techniques that reconstruct a surface representation from point data therefore have to be robust to such artifacts.

Direct reconstructions of range data are highly dense and detailed, leading also to large file sizes. This is especially prohibitive in context of shape repositories that have to store many shapes and be able to provide them to users over a network connection. Such fine detail is also not important to most users of shape repositories and they would readily trade representation detail with size. *Shape simplification* or *fairing* methods obtain a simplified representation of a given shape according to some criteria, e.g. preservation of curvature. The reverse process, i.e. obtaining a denser representation of a given shape, is achieved through *subdivision* techniques.

Once a shape has been obtained at a desired level of detail, it can be placed in a virtual environment or in a simulation, it can be interacted with, animated or deformed, it can be cataloged for later look up, inspected to reveal interesting properties of the real shape it represents or showcased in virtual museum applications. The shape can be used in simulations to observe its behavior under different conditions before it is sent to manufacture. For advertising purposes, a shape can be rendered with appealing visual effects. A shape can also be used to retrieve other shapes from a collection that are similar to it, or it could be decomposed into parts and assembled into a new shape with parts from other shapes. Recently, printers have been introduced that even make it possible to print 3D shapes. The power and flexibility afforded by having a digital representation opens the doors to limitless possibilities to interact with the shape.

However, in order to guide a user to a desired shape in a repository, visual cues to the shape have to be presented in the form of shape thumbnails. Each thumbnail shows the shape from a particular view and together, several thumbnails provide the user with a good idea of the shape without their having downloaded it. *Best view* methods aim to compute the viewpoints most suitable for obtaining such views of a shape. Depending on the targeted application, e.g. object recognition and maximizing visible area, the goodness of a particular view may vary and a view that is deemed good for one application might not be so for another.

In maintaining a large shape repository, it also becomes important to present users with different ways to browse contained shapes. Basic categorizations of shapes can be with respect to

- representation type, e.g. mesh, implicit surface, boundary representation,
- represented object, e.g. cultural artifact, furniture, human, or
- applicable terms, e.g. unrestricted use, share alike, attribution, or even
- user who uploaded the shape.

Ranking shapes based on their properties can provide another means to efficiently guide users to desired shapes. These properties could be of several types, i.e.

- directly accessible from shape data, e.g. number of geometric primitives, file size,
- relative to the repository, e.g. time of addition to the repository, popularity among downloaded shapes, or
- derived from the shape's geometry, e.g. topology, shape complexity.

The quality of a shape repository is measured in terms of the value it provides to its users. In addition to obvious steps like populating the repository with a large variety of shapes, keeping the collection up to date by continually adding new shapes and minimizing the search effort between a user and their desired shape, there are other factors that go towards making the repository useful. In order to provide the user a variety of sortings of shapes as discussed above, the repository needs to store a large amount of information on each shape. Entering this data manually is cumbersome for a user. Thus, it is helpful to have automatic tools in the repository to extract maximum information from a shape and thus reduce user load. Digital shapes can easily take up hundreds of megabytes in file size. Adding multi-resolution shape viewers to the repository allows users to inspect a shape satisfactorily before downloading it. Similarly, integrating simplification tools allows users to download simplified versions of highly detailed stored shapes, if desired. For publicly accessible repositories, suitable access policies need to be put into place. Dissemination of shapes also needs to be controlled through licensing options.

1.1 Contributions and Outline

Most of the work presented in this thesis deals with Steps 2 and 4 of the digital shape pipeline shown in Figure 1.1. In particular, we make contributions in the following areas of digital shape processing. For each of these, we mention related work in Chapter 2. For brevity, existing tools and methods that we commonly use in our techniques below are outlined in Chapter 3. Most of the work has already been published or is currently under review for publication. Towards the end of this section, we also mention some of our work that we believe deserves further attention.

1.1.1 Learning based Surface Reconstruction

As range data is typically noisy and incomplete, stochastic methods are natural candidates to process them. Such methods are increasingly found to model natural phenomena better than strictly logical methods [Mumford99]. In particular, statistical learning [Hastie01] is hoped to provide a key to the nature of human intelligence [Poggio03].

In Chapter 4, we present our work on *neural meshes* [Ivrissimtzis03, Jeong03], a learning based surface reconstruction algorithm that uses a neural network to learn shape and topology from an unorganized point set. Earlier methods restrict the topology of the shape to be reconstructed and in training the surface, apply global changes to it when, in fact, they mean to increase representation density in only specific "active" regions of the surface. In contrast, our method is able to learn the topology of the target shape thus freeing it of restrictions on the input point set. Also, by associating an "activity counter" to each vertex in the reconstructed triangle mesh, changes are localized to intended regions only. Our reconstructed surface can be adapted to different shape properties and we demonstrate reconstructions sensitive to points density in the range data and shape curvature. As expected, our method exhibits robustness to input artifacts. This work has been published previously as [Isgro05, Saleem04, Saleem07a] after which very similar work appeared in [do Rego07, do Rego09].

Surface reconstruction methods are interesting in context of shape repositories in that raw range data can be made available to the research community through a repository. Different researchers can try out their respective algorithms and add their reconstructed surfaces back to the repository.

1.1.2 Computing best view

3D shapes are typically represented by one or more thumbnails, views of the shape from different viewpoints. Selecting these viewpoints automatically is known as the "best view" problem and is of interest in various contexts, e.g. shape recognition and visualization. Depending on the desired application, the goodness of a viewpoint may vary and a view that is good for one application may not be so for another.

We note that visualization methods which intend to compute results for a human audience fail to incorporate the existing, rich body of work on human shape perception [Blanz99, Tarr01, Todd04] (see also references therein). In our opinion, this omission is a fundamental oversight and hence our approach to the problem, presented in Section 5.1 is based exactly on the principles suggested in these ignored works–we represent a shape by its silhouettes from different viewpoints, we use similarities between these silhouettes to compute the shape's best views as those that are both stable [Weinshall97] and salient [Lee05]. Note that while some earlier methods happen to use some of these steps as well, none of them is perceptually motivated and thus none uses all steps together.

This work was published as [Yamauchi06] and since then has been cited as a standard reference on best view in papers on multi-view methods in robotics [Welke07, Welke08], view selection [Feixas09, Mortara09], shape retrieval [Laga07], shape orientation [Fu08] and saliency applications [Liu07, Kim08].

Most, if not all, shape repositories require users to manually add thumbnails when adding a shape to the repository. This is added burden on the user and, as such thumbnails can be computed automatically, also unnecessary. Incorporating a best view method like ours to compute thumbnails of shapes as they are added to the repository can make the user's task of adding a shape to the repository less cumbersome.

1.1.3 Computing "best fly"

Best view methods yield static views of the shape. In the case where more than one view is computed, information on how these views relate to each other is absent. This calls for a dynamic representation of the shape in the form of an animation recorded by a camera flying around and pointing towards the shape.

In Section 5.3, we formally state and compile the restrictions imposed by previous work on the camera path in order to compute it. When presenting previous work, we show how none of them meets all requirements. Our dynamic view method builds on our perception based best view method and we show how it meets all of the said requirements. We are the first to consider altering speed and zoom of the camera along its path around the shape to better meet these requirements. Also, we coin the term "best fly" for the dynamic view selection problem. This work was published as [Saleem07b].

Just like static views, dynamic views of a shape can be made available on a shape repository web page in popular animation formats, e.g. Flash or animated GIF. It can be argued that a dynamic view provides better value as it takes the screen space of a single static view while delivering more information to the viewer.

1.1.4 2D Shape Orientation

Just as it is important to present a 3D shape from a good viewpoint, it is important to orient a 2D shape to its natural orientation. Humans have well defined notions of correct orientations for most shapes, e.g. given two pictures of a horse from the same viewpoint but rotated such that its feet are at the bottom in one picture and its head in the other, we would immediately pick the first one as being correctly oriented. This is relevant in 3D contexts as well because their views are eventually also 2D shapes and almost all view selection methods are invariant to rotation, i.e. they are equally likely to output either one of the two mentioned horse images as they cannot distinguish between them.

A method to reorient a 2D shape in an image is therefore needed. It turned out that this problem had simply not been addressed before. We found some work on image orientation which we show in Chapter 2 to be a different problem. In Section 5.2, we present our example based solution to 2D shape orientation. We show an existing, generic dataset can be pruned semi-automatically for the purpose and how a query shape is matched against it and reoriented. As the notion of "correct" orientation depends on human interaction with objects of that shape, it cannot be computed automatically and hence, we need a human user to identify correctly oriented shapes when setting up our database.

Our method was published as [Saleem07c] and was extended the next year by [Fu08] to 3D shapes where user input on correctly oriented 3D shapes is used to train a classifier which then assigns a class to unseen query shapes and reorients them accordingly.

1.1.5 Computing Complexity of 3D Shapes

Like correct orientation, complexity of a shape is also a human dependent notion. More so, it is possible that people disagree on which shape is the more complex among two given shapes. Previous work on automatic computation of shape complexity has taken cues from the shape's geometry. That, however, makes it sensitive to small, commonly occurring irregularities like surface noise.

We draw inspiration once again from human shape perception [Koenderink79, Cutzu97] and present a method that achieves robustness to common shape artifacts by representing shapes by their view silhouette and measuring their complexity in terms of the similarities among their views. This is illustrated in side by side comparisons of rankings of a small set of shapes using our method and some previous methods. The method and obtained results are presented in Chapter 7 and have been published earlier as [Wang08a, Wang08b].

As large unordered collections tend to get confusing, managers of shape repositories like to impose some ordering on contained shapes so that users can browse them according to different criteria and depending on their needs at the time, gain access to their desired shape(s) in the least amount of time. Shape complexity is a natural candidate for such a criterion as users often want to try out their algorithms on shapes of increasing complexity.

1.1.6 Efficient Shape Retrieval

The Internet provides access to millions of documents but at any given time, a user only wants to see a few specific ones. That is what makes search engines indispensable. Though shapes have not yet reached comparable volumes, the same argument applies to shape retrieval.

Almost all previous work on shape retrieval has focused on the quality of retrieved results with little regard to efficiency. In Chapter 8, we present our shape retrieval technique inspired from text/document retrieval. Using local sampling, we use a “bag of words” approach to convert a shape into a text document. Thus, all shapes in a collection are added to an inverted index structure which can then be queried efficiently. We simulate a large collection of shapes by applying parametrized deformations to shapes in a small base collection. Our method is able to perform queries on the resulting collection of one million shapes in under a second.

We believe one of the factors of the usefulness of a shape repository is the ease with which desired shapes can be found. While general browsing criteria quickly present a class of desired shapes to the user, making the repository searchable makes it easier for the user to get to specific shapes. This may be beneficial if the desired shapes span several criteria and thus would not be grouped together when stored shapes are sorted according to general shape properties. Indeed, many online shape repositories offer keyword based searches. Offering a shape similarity based retrieval provides a more geometric means to find the desired shape.

1.1.7 Design and Maintenance of a Shape Repository

Many of our ideas presented so far have either been implemented in or inspired by the AIM@SHAPE Shape Repository which we developed and maintained over the course of the research time allotted for this thesis. The repository manages and provides a seamless interface to over one thousand shapes by putting them in a knowledge management framework. New shapes of many types, representing a broad range of real world objects are routinely added to the repository. To enhance user experience, various means to search and browse are provided along with online viewing and simplification tools and automatic metadata extraction tools. To give users control over use of the shapes they add to the repository, legally binding licences are incorporated that can be specified for a shape at the time of upload. Since its inception almost five years ago, the repository has been visited more than two million times and shapes have been downloaded from it almost a hundred thousand times. These shapes have so far featured in all major digital geometry research conferences. We present some of the guiding principles behind the Shape Repository in Chapter 9.

Technical documents on the repository have been submitted as various in-project reports available through the website [AIM@SHAPE] of the EU Project FP6 IST NoE 506766 AIM@SHAPE and we also contributed to the technical submission [Danovaro07] describing the multi-resolution mesh viewer incorporated in the repository. Note that the repository is the outcome of efforts of many project partners from the AIM@SHAPE project.

1.1.8 Minor Contributions

Continuously approximating descriptor values on a view sphere. Digital shape processing techniques that either output views of a shape or use shape views in an intermediate step to compute some other property, like complexity, make use of "view descriptor" values on a shape's view sphere. These values are evaluated at discrete points on the view sphere and are thus dependent on the discretization. We investigated how these discrete values can be used to construct a continuous function that can then be evaluated at any point on the view sphere independent of its discretization. The approximation along with error values is presented in Chapter 6. We also demonstrate the use of our approximated values in performing common shape operations like computing its representative and equivalent views, and computing view likelihood.

The View Transfer operation. We introduce in Chapter 6 a new shape operation called "view transfer" whereby a chosen view of a shape is transferred to another, similar shape, and show how descriptor values sampled from our continuous representation can be used to perform this operation. The premise of view transfer is that similar shapes have similar best view parameters. Using this operation, computed best views of a stored shape can be transferred to other similar shapes in the repository without having to explicitly compute best views of the latter shapes.

In the same chapter, we also present our derivation of an optimal radius for a shape's view sphere. While view spheres are commonly used, there existed no formal conditions relating their size to that of the viewed shape and while the center of the view sphere is fixed at the shape's center, its radius was heuristically chosen to be some multiple of the shape's bounding box diagonal. A preliminary version of this chapter has been published as [Saleem08].

Chapter 2

Previous Work

This thesis consists of work on various areas in digital shape processing. We give a short introduction to and present related work on each of these areas in this chapter.

2.1 Neural Network based Surface Reconstruction

The goal of Surface Reconstruction is to reconstruct the shape represented by a set of points, or *point cloud*, sampled on its surface, typically obtained through laser scanning of the shape. Depending on the scanning technology, each point may also be equipped with information about the surface normal at that point. In the latter case, the points are said to be *organized* and points without normal information are termed *unorganized*. In this thesis, we focus on the more general case of unorganized points.

The first general purpose method for surface reconstruction from unorganized points appeared in [Hoppe92]. Since then, there has been considerable activity in the field and methods drawing from various backgrounds have appeared in the literature, from differential geometry and level set methods to implicit functions. A short survey is presented in [Schall05]. Our focus is on surface reconstruction methods based on statistical methods. Except for a few, [Jenke06, Patané06], the vast majority of these methods rely on training a Neural Network [Bishop95]. Roughly speaking, a neural network consists of interconnected nodes that carry some associated information. Connections between nodes may also have some information associated with them. After some initialization, the information in the neural network is made to adapt to inputs and, optionally, corresponding desired outputs that are progressively introduced to it. This is how the neural network "learns" from the input. "Training" the neural network consists of presenting it with inputs in order to make it learn. It stops when the neural network meets some desired condition, e.g. computed outputs fall below an error threshold.

In [Gu95, Mostafa99, Yang00], neural networks have been trained to reconstruct parametric and freeform surfaces representing the input point cloud. The neural network learns a function $f(x, y)$ such that $\|f(x_i, y_i) - z_i)\| < \epsilon$ for all points (x_i, y_i, z_i) in the point cloud and for some acceptable error level ϵ. Neural networks have been extended to *functional networks* in [Iglesias04] to reconstruct the point cloud with B-spline and Bezier surfaces (see also references therein).

The above methods fall under the *supervised learning* category, i.e. they assume a relationship among the input variables (x_i, y_i, z_i) and train the neural network to learn this relationship. *Unsupervised learning* methods make no such assumption. Instead, they train the neural network to learn a surface. Nodes of the neural network represent points in 3D space and they move towards the learned surface as the neural network learns. When learning stops, the position and connectivity of the nodes form the learned surface, which either directly or indirectly represents the desired surface. The methods described in [Barhak01a, Barhak01b, Hoffmann98, Várady99] train a neural network to learn control grids for reconstruction of the surface or parametric grids for subsequent parameterization of the surface. In [Knopf04, Yu99], the neural network is trained to interpolate or approximate the point cloud itself, thus directly learning the desired surface.

In the unsupervised learning methods described above, the topology of the learned surface and its number of vertices remain unchanged since initialization. As they initialize the neural network as a 2D grid, they can accurately represent the desired surface only if it represents a surface patch. Also, the learned surface may under-represent detailed features of the desired surface. As a solution to the latter problem, subdivision is suggested in [Yu99]. In [Barhak01b, Várady99], where the surface has the topology of a quad grid, the authors suggest tracking the activity of each vertex with an associated counter, which increases each time the vertex participates in learning. They can then spot active vertices by their high counter values, and add entire rows/columns of vertices in their neighborhoods. Both these solutions are global in nature and end up adding new vertices in unwanted regions of the surface as well.

The unsupervised learning method we present in Chapter 4 adapts its node distribution and topology according to the input point cloud. That allows it to learn fine features as well as complex topologies.

2.2 View Selection

Digital shapes represent and contain 3D information. However, these shapes are ultimately presented on 2D media – paper, monitor etc. This necessitates a projection from the 3D space of the shape to the 2D space of the display medium. Having chosen a projection method, a given 3D shape can have infinitely many 2D projections corresponding to infinitely many projection directions. Each projection is a *view* of the shape.

Views of a digital shape convey a preliminary idea of the shape. This is useful in many scenarios. Complex, highly detailed shapes typically take up hundreds of megabytes or more in file sizes, leading to lengthy download times. Some repositories even require payment for accessing stored shape models. In such cases, the user would like to have an idea of the shape before committing to a download. For digital museum applications, it might be desirable to show some views of a digitized cultural heritage object without giving the user access to the full 3D model. Managers of a digital shape repository might want to display a few "catalog views" of some showcase models to promote their repository. For machine vision applications, certain views of a shape are required to recognize it and retrieve similar shapes.

This necessitates a strategy to choose from the potentially infinite views of a shape, a few "interesting" or informative ones. This is also known as the "best view" problem. As a single view cannot convey information on the full shape, some methods find instead the N best views, possibly with a ranking.

The static views found by the kinds of methods discussed above can still be ambiguous as they do not convey information on how the views relate to each other. Another class of methods therefore aims to compute a camera path, travelling along which a virtual camera pointing at the center of the shape captures an animation of the shape. This animation then provides a dynamic view of the shape.

The predominant approach in view selection is to define a measure or *view descriptor* that assigns some goodness value to each view. Representing a shape by its views, the view that maximizes the value of a chosen view descriptor is chosen as the shape's best view. In general, methods differ in the view descriptor they maximize.

2.2.1 Static View Selection

An early method [Kamada88] characterizes a view by the angle between the viewing direction and normals of visible faces of the shape. A good view is one that views most faces directly, i.e. the viewing direction and face normals are parallel. An angle of 90° corresponds to a degeneracy. Another angle based approach is given in [Podolak06] which computes a shape's symmetry planes and then assigns a score to a view based on the angle between the viewing direction and the plane normals. Here, the best view is the one that contains the least symmetries, i.e. the viewing direction is parallel to most plane normals.

The method given in [Colin88] subdivides the shape into an octree and characterizes a view based on the number of visible octree cells. For this descriptor, the more the number of visible cells, the better the view. The descriptor proposed in [Plemenos96] evaluates views based on the number of visible triangles and the visible projected surface area. This is enhanced in [Sokolov05] to additionally include curvature information at visible vertices, and generalized in [Vázquez03a] to *viewpoint entropy*, a probability measure that measures the information content of a view. The best view is then the one with the maximum information content. A similar approach is proposed in [Bordoloi05, Takahashi05].

Another curvature based measure is proposed in [Lee05]. Each vertex of the shape is assigned a saliency value based on its multi-scale curvature properties. *View saliency* is then the sum of saliency values of visible vertices. The assumption here is that a shape's best view is the one which contains its most salient features, where salient features are those that are distinctly different from their neighborhood. The view selection techniques in [Gumhold02, Shacked01] also maximize a custom view descriptor. A comparative study of several view descriptors is presented in [Polonsky05].

In the computer vision and pattern recognition communities, a few object views are selected for later recognition [Denton04], reconstruction [Lee04] or similarity retrieval [Mokhtarian00] of the object. Views most similar to other views up to a threshold are taken as representative views of the object in [Lee04, Mokhtarian00]. Similar approaches are presented in [Arbel99, Hall05].

The best view problem is also studied in robot motion and graph drawing. A good survey of techniques from these areas can be found in Chapter 2 of [Vázquez03b].

2.2.2 Dynamic View Selection

There has been surprisingly little work on finding dynamic representations of shapes. Some online repositories [Demand, Repositorya] offer 3D viewers for interactive exploration of stored shapes, but this approach becomes impractical when the shape model is large, as large file transfers are then required to explore the full shape. This problem could be circumvented by offering a simplified version of the shape for exploration but that defeats the original purpose of exploring shape details. An interesting solution is proposed in [Danovaro07] whereby a simplified version of the shape is first displayed for exploration, and the user can then choose regions of the shape for which they want to view more details.

Few methods [Sokolov06a, Sokolov06b] compute an animation of the shape as we proposed earlier. While best view methods sample all possible views of the shape from its viewsphere to choose the best among them, taking such an approach to the best fly problem is not feasible as the size of the search space increases exponentially with the number of viewpoints to be visited in the fly. To guide the computation of the fly, some heuristics have previously been used which we formalize below.

1. *Brevity* – the animation should not be long,
2. *Information* – the animation must be maximally informative,
3. *Exploration* – the camera path should avoid fast returns to already visited viewpoints
4. *Smoothness* – the path should be smooth.

Depending on how the path computation is performed, shape exploration methods are classified as either *offline* or *online*. Offline methods analyse the object once and compute the fly in advance. Online methods compute a path in real time each time they visit the object. Another classification is made on the nature of exploration conducted. *Global* methods aim to give a general understanding of the shape model. The camera stays outside the shape model, restricted to its viewsphere. In *local* methods, the camera enters the model and becomes part of it. Our method presented in Section 5.3 is a global, offline method that satisfies the above heuristics.

Previous methods either fail the Smoothness condition [Sokolov06b] or deal with it artificially by putting in a damping factor where sharp turns occur [Sokolov06a]. The path is often computed incrementally [Sokolov06a]; at each viewpoint in the path, the next viewpoint is selected after considering each candidate viewpoint's view goodness, distance from the starting point of the path and the fraction of the model uncovered from that viewpoint. Such methods suffer from the drawback that they cannot guarantee that the computed path will pass through a given set of points, without violating any of the heuristics or complicating the computation.

2.3 View Sphere Model

In Chapter 6, we present our work on approximating view descriptors on a shape's view sphere. This is in contrast to the traditional approach of evaluating descriptor values at discrete samples on the view sphere.

The little work we came across on approximating view descriptors comes from the best view literature, where the goal is to find the viewpoint that maximizes the value of a chosen descriptor. In [Plemenos96], the view sphere is divided into eight spherical triangles corresponding to the three axes. The view descriptor is evaluated at the triangle vertices, and a "best" triangle is chosen. This triangle is then recursively subdivided, choosing the best among the new triangles each time. A similar strategy is employed in [Vázquez03c].

Our approximation in Chapter 6 differs from these in that we are interested not in evaluating the descriptor at one "best" viewpoint, but at any given point on the view sphere. We therefore build a continuous function that can be trivially evaluated at any 3D point.

2.4 Shape Orientation

View selection methods, like the ones discussed above, select the best view of a shape by identifying a best viewpoint on the shape's view sphere. However, this alone is not sufficient to select a view uniquely as it does not address another degree of freedom, the up vector. Views of a shape from the same viewpoint but with different up vectors are rotations of each other, i.e. the shape is oriented differently in each of the views. As no strategy is employed to fix an up vector for a given shape, automatic view selection methods often yield arbitrarily oriented views which are later fixed by hand.

Humans have well defined notions of the "correct" orientation of a shape. Given, for example, two side views of a horse with the horse standing on its feet in one and on its tail in the other, we will immediately pick the former view as the correctly oriented one. Incorrectly oriented views seem implausible and unrealistic, even if they have been determined to be the best according to some descriptor. It is therefore important to fix a strategy to choose a correct up vector for a given shape when acquiring its views, or to reorient the shape in a view once it has been acquired. We take the latter approach in Section 5.2.

We could not find previous work on automatic shape orientation. There has recently been some work on *image* orientation (cf. [Luo05, Vailaya00, Wang04]), whereby the correct orientation of a natural image ($0°$, $90°$, $180°$ or $270°$) is decided by performing statistical analyses of image features. However, the shape orientation problem is different. Given a view of a shape, we want to automatically determine the correct orientation of the shape for the view.

2.5 Shape Repository

Digital shapes are of interest in several communities and application areas – computer games, digital museums, virtual environments, movies, and digital geometry research. While it is possible to design shapes from scratch, the process is often tedious and cumbersome. It is therefore desirable to have one or more ready sources of shapes that can meet the needs of the above communities. These exist in the form of shape repositories that can be accessed via the Internet.

To assist users in quickly finding a shape model of their choice, these repositories typically annotate some information to the stored models. Almost all repositories assign each model a suggestive name and present one or more views of it as accompanying thumbnails. In addition, models are classified according to the types of object they represent (animal, plane etc.) [Benchmark, Cacheforce, Databasea, Databaseb, Flash Fire Designs, Modelsa, modelsb], by the type of data (static, animated or motion capture) [Demand], or by their origin (how they were acquired) [Databaseb]. Along with origin information, it is also common to show with each model some meta-information like geometry and texture description [at Georgia Tech, Browser, Flash Fire Designs, Modelsa, Repositoryb, Turbo Squid], ownership, notes on applicable tools [Databasea, Repositoryb] and file formats [Databaseb], references to research papers containing the model [at Georgia Tech], and/or similar shapes in the repository [Databasea].

In Chapter 9, we present an information theoretic approach to a Shape Repository that automatically subsumes all the above classification criteria and provides users a rich set of features to interact with a chosen shape before committing to download it.

2.6 Word Based Approaches to Shape Retrieval

The larger a collection, the more difficult it becomes to find things in it. This applies to collections of shapes as well. In this thesis, we are interested in shape retrieval based on similarity of fetched shapes to a query shape. Unsurprisingly, shape retrieval generates interest from many communities – computer vision, pattern recognition, computer-aided design, engineering, shape modeling, computer graphics, virtual reality, multi-media, databases, and even machine learning and human-computer interaction. A good survey is presented in [Tangelder04]. The general approach is to use a shape descriptor to quantify stored and query shapes into feature vectors. Query execution entails matching the query feature vector with stored ones.

In Chapter 8, we aim to improve upon the above “query-against-all” model as it is inefficient for large shape collections. We employ for the purpose first a “bag of words” approach to convert shapes into textual words and second an inverted index, popularized in document retrieval, for efficient shape retrieval.

Lately, considerable attention [Fraundorfer07, Nistér06, Philbin07, Philbin08, Schindler07] has been given to “bag of features” techniques for image retrieval and to their optimization. The basic idea is to treat extracted features, whatever they may be, as an unordered collection or a bag. Objects, images in this case, match when their bags contain one or more identical features and the goodness of the match is determined by the extent of overlap. Bag of words is a specialization of bag of features in that features are expressed as words.

We found two methods similar to our approach. The first [Biswas07] uses a bag of words approach for 3D shape retrieval but does not use text retrieval methods for efficient retrieval like us. All shapes in the collection are sampled, multiple pairs of samples from each shape are selected and analyzed to extract features, the features are quantized and then binned. A query shape is led through the same process and its matches are determined based on the bins its features fall into.

The other, Video Google [Sivic03, Sivic06], uses text retrieval techniques to find those key frames in a video sequence that contain a shape in a query image. SIFT features [Lowe99] are extracted from key frames and the query image, and quantized into *visual words*. Using a technique similar to the one we use and explain in Chapter 8, an inverted index of visual words and key frames is built and key frames are then ranked by the normalized inner product of their word frequency vector and the word frequency vector computed for the query image.

2.7 Shape Complexity

Humans can easily judge a given shape to be complex or simple, and, given a set of shapes, perform a sorting of the shapes based on their geometric complexity. Automatically estimating shape complexity, however, has received surprisingly little attention.

In [Rossignac05], an attempt has been made to formalize the notion of shape complexity by defining a few measures that could lead to its estimation.

- *Algebraic complexity* is the degree of the polynomial used to represent the shape.
- *Morphological complexity* is an estimate of the amount of fine details in the shape, and is computed as the largest value of r for which the shape is r*-smooth* or r*-regular*.
- *Combinatorial complexity* is the number of vertices used in the shape representation.
- *Representational complexity* is a qualitative measure of the amount of redundancy in the shape representation.
- *Topological complexity* is also a qualitative measure comprising of the genus of and non-manifold elements in the shape.

While these measures may capture some aspects of how humans gauge shape complexity, they are limited in terms of how they can be applied for automatic complexity estimation. The first two measures are restricted to specialist shape representations and all three quantitative measures are indiscriminate – shapes of varying complexities can easily end up having the same values for these measures.

More discriminative approaches have involved the use of information theory. Page and colleagues [Page03, Sukumar06] note that the canonical simplest shape, the sphere, has the same curvature throughout its surface. Therefore they compute a shape's complexity as the entropy of its curvatures viewed as a probability distribution. The method proposed in [Rigau05] builds upon the observation that inside the sphere, each surface point is visible from every other surface point. A shape's *inner complexity* is then measured in terms of the *mutual information* between regions of the shape that are mutually visible to each other through the shape's interior. An *outer shape complexity* is also proposed that considers visibility between regions of the shape and a bounding sphere.

Our method, presented in Chapter 7, builds upon evidence from human vision [Cutzu97] and psychology [Koenderink79] research which claims that humans perceive 3D shapes as arrangements of 2D view images.

Chapter 3

Preliminaries

Throughout this thesis, we use certain tools and concepts one or more times. To simplify exposition later and to avoid repetition, we describe them in detail in this chapter. The methods presented here are not our contributions – we use them as black boxes in our algorithms and cite relevant sources. We will make explicit when we modify a certain method from the original for our use.

When analyzing a 3D shape, it is easier to work with views of the shape, which are 2D images. Many of our methods require these views to be compared to each other according to similarity, and most shape similarity techniques rely on the boundary contour of the shape in the view image. These methods typically yield a *similarity distance* between a pair of shapes, which is zero if the shapes are identical, and increases with dissimilarity between the shapes. For this reason, this distance is sometimes also referred to as the "dissimilarity distance".

In Chapter 7, we use Similarity Structure Analysis (SSA) to analyze the similarities between views of a shape in order to compute the shape's *complexity*.

3.1 Obtaining Shape Views

A view of a shape is simply a snapshot from a (virtual) camera of the shape against a background. To keep the view simple, the background is typically plain. Keeping shape texture and background constant, the necessary parameters for a view are

- distance of the camera from the shape

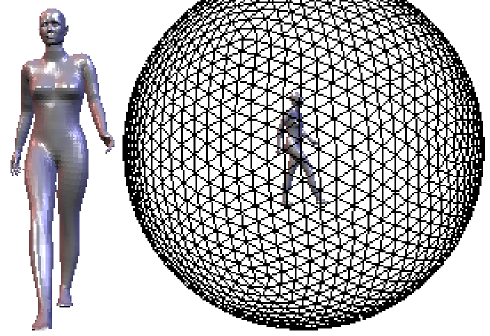
- viewing direction of the camera
- viewing frustum of the camera
- up-vector of the camera

As shape views are often taken to represent the 3D shape in some context, a single view is usually not enough. Then, the *number of views* and *position of the camera* for these views also become important considerations.

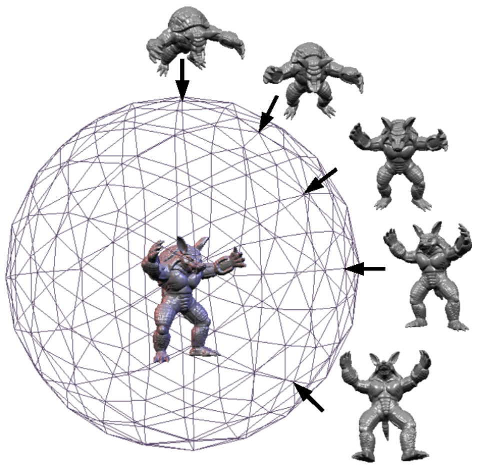
The usual solution is to place a *view sphere* around the shape. This is a sphere whose center lies at the geometric center of the shape, and whose radius is some multiple of the length of the diagonal of the bounding box of the shape. Cameras are placed on the surface of the view sphere and pointed towards the center of the sphere. This links the position of a camera to its viewing direction. The distance of the camera from the shape is simply the radius of the view sphere. A small radius corresponds to a zoom in of the shape, and a large radius to a zoom out. At one extreme, only part of the shape is visible in the corresponding view and at the other, the view consists almost entirely of just the background. A suitable value for the view sphere radius is generally chosen heuristically. In Chapter 6, we present a derivation for an "optimal" radius, which we found missing in the literature.

Covering the entire surface of the sphere with cameras will yield all views (up to a zoom factor) that look directly at the shape. However, this leads to infinitely many camera positions and is thus computationally impossible. A close approximation would lead to an infeasibly large number of views. Instead, a sufficient number of camera positions that are uniformly distributed over the surface are considered. For this reason, the view sphere is typically modeled by a platonic solid – either by a dodecahedron or its dual, an icosahedron – with the same center and radius. Cameras are then placed at the vertices of the polyhedron, or at the barycenters of its faces. In the latter case, the camera positions are projected to the surface of the sphere being approximated. As explained so far, the number of views obtained by this approach is limited to 12 (icosahedron) or 20 (dodecahedron). If more views are required, camera positions are computed in the same way for the polyhedron after application of a few subdivision steps to it. Throughout this thesis, we initially approximate the view sphere by an icosahedron and place cameras at its vertices. We prefer the icosahedron as its triangle mesh structure easily lends itself to common subdivision methods. After each subdivision step, all newly added vertices are projected back to the surface of the view sphere. Figure 3.1 shows view spheres at different levels of subdivision for two shapes.

Most mesh viewing softwares model cameras with a default viewing frustum of 45° in the x and y directions. The software used for the experiments in this thesis



(a) A model and its view sphere.



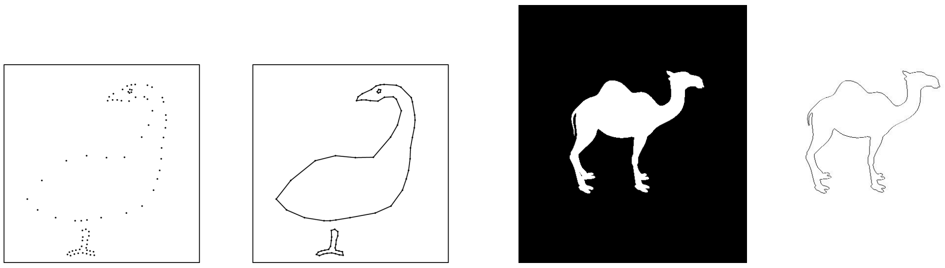
(b) Views from a view sphere.

Figure 3.1: Note that the model, radius of the view sphere relative to the bounding box of the model, and the number of subdivision steps applied to the view sphere differ in each sub-figure.

follows the same convention. Unless mentioned otherwise, we do not specify any up vector for our cameras, thus using our system's default up vector which corresponds to the positive y direction. This can lead to faulty orientation of the shape in obtained views. However, this is a typical problem in view-based methods, as computing the "correct" up vector for a 3D shape is a problem that has just recently received attention [Fu08]. In Section 5.2, we propose a method to correct orientation of shapes in their view images. In the literature up to the time of writing this thesis, shape orientation in views is generally fixed manually. The fact that we end up with arbitrarily oriented views does not hinder our methods as subsequent steps applied to the views are rotation invariant, i.e. given views which are identical in all parameters except the up vectors, the methods treat them identically.

3.2 Obtaining Shape Contours

To obtain the boundary of a 2D shape in an image as a closed contour, we first scan the image for boundary pixels and then employ the popular Crust method [Amenta98], illustrated in Figure 3.2. Given a set of points sampled closely enough on a line, the Crust method uses Delaunay triangulation to constuct the points' Crust, which is a reconstruction of the original line as a polyline with the input points as vertices. As our boundary points are just a pixel apart in image



The Crust of a point set, [Amenta98]. A shape and its boundary.

Figure 3.2: We use the Crust method to extract the boundary contour of a 2D shape.

space, they satisfy the closeness condition required by the Crust method.

The above strategy works well in general cases, i.e. we are able to extract shape boundary as a closed, connected contour. However, it fails when the shape boundary contains noise or fine features. In such cases, the computed contour contains a large number of disconnected edges. Luckily, detection of this case is easy (iteration over all obtained edges). When this occurs, we remove noise from the original shape by iteratively applying blurring and morphological opening to the original image. At each iteration, we compute the resulting shape's Crust and check if it is a closed contour. Iteration stops when a closed contour is obtained. While this admittedly modifies the shape, shape features lost in this way are too fine for subsequent methods to work with in the fist place. The resulting Crust still preserves the overall shape.

3.3 Computing 2D Shape Similarity

2D shape similarity methods analyze the shape to extract a *feature vector* from it. The similarity distance between two shapes is defined as the matching cost of their feature vectors. It is desired of feature vectors to be scale, rotation and translation invariant. Different methods accomplish this by either preprocessing the shape, or by incorporating these requirements in the computation and comparison of feature vectors.

There are two classes of shape 2D similarity methods to choose from. Methods from one class operate on the boundary contour of the shape in the image, while those from the other utilize pixel information of the entire shape. During the course of our research, we used a pixel method based on Zernike moments, and two boundary contour methods based on Curvature Scale Space (CSS)

Method	BEP reported	BEP reimp
Shape context	76.51	41
Image edge orientation histogram		41
Hausdorff region		56
Hausdorff contour		53
Grid descriptor		61
Distance set correspondence	78.38	
Fourier descriptor		46
Delaunay triangulation angles		47
Deformation effort	78.18	
Curvature scale space (CSS)	**84.12**	**52**
Convex parts correspondence	76.45	
Contour-to-centroid triangulation (CCT)	**84.33**	**79**
Contour edge orientation histogram		41
Chaincode nonlinear elastic matching		56
Angular radial transform	70.22	53

Table 3.1: Accuracy of similarity measures in Bull's Eye Percentage (BEP). All methods were reimplemented and BEP figures from the reimplementations (right) are shown alongside figures claimed by the methods' authors (center), reproduced from [Veltkamp06]. We use the CSS and CCT methods in this thesis.

[Mokhtarian03] and contour to centroid triangulation (CCT) [Attalla05]. Boundary methods generally outperform pixel methods [Latecki00]. The CSS method has in fact been incorporated into the MPEG-7 standard but was later outperformed by the CCT method [Veltkamp06], see also Table 3.1. Performance is judged in terms of retrieval accuracy from the MPEG-7 benchmark database [Latecki00, Veltkamp06]. Below, we present each of the three methods we used.

3.3.1 Zernike Moments

Zernike moments of a function $f(x,y)$ are projections of f on to a set of complex Zernike polynomials, $V_{nm}(x,y)$, that form an orthogonal basis over the unit circle.

$$V_{nm}(x,y) = V_{nm}(\rho,\theta) = R_{nm}(\rho)\mathbf{e}^{\mathbf{i}m\theta},$$

where

$$\begin{cases} m, (n-|m|) \text{ even}, |m| \leq n \\ R_{nm}(\rho) = \sum_{s=0}^{\frac{n-|m|}{2}} -1^s \cdot \frac{(n-s)!}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n-|m|}{2}-s\right)!} \cdot \rho^{n-2s}, \end{cases}$$

where n and m are integers and (ρ,θ) are the polar coordinates of (x,y). For a discrete image, $I(x,y)$, the Zernike moments of *order n* and *repetition m* are defined as

$$A_{nm} = \frac{n+1}{\pi} \sum_x \sum_y f(x,y) V^*_{nm}(x,y), \qquad x^2+y^2 \leq 1,$$

where $V^*_{nm}(x,y)$ is the complex conjugate of $V_{nm}(x,y)$.

When using Zernike moments as image features [Chong03, Khotanzad90, Mukundan98], the image is required to be square and the restriction, $x^2+y^2 \leq 1$, is fulfilled by normalizing pixel locations. The feature vector simply consists of the image's Zernike moments, and matching cost is the L2 distance between feature vectors.

As the method employs frequency analysis in polar coordinates, computed moments are automatically rotation invariant. Scale invariance is achieved by normalizing all moments with respect to the second order moments that represent area of the viewed shape. To achieve translation invariance, the shape is first translated such that its centroid lies on the center of the image.

3.3.2 Curvature Scale Space (CSS)

The CSS method [Mokhtarian03] is based on the observation that repeatedly smoothing any closed contour eventually leads to a contour that is fully convex. More specifically, the number of *curvature zero crossings* in the contour tends to zero with smoothing. The boundary contour for any non-trivial shape is a closed loop with an equal number of alternating curvature minima and maxima. Between each such pair of curvature extrema is a curvature zero crossing, i.e. a point on the boundary contour where the curvature is zero. As the shape is smoothed, the extrema vanish and the zero crossings merge with each other until none remain. A shape's *CSS image* is a plot of number of smoothing iterations against the positions of curvature zero crossings on the boundary contour. The CSS feature vector of the shape then consists simply of the positions and heights of the maxima in the CSS image. Figure 3.3 contains an illustration of the process. When comparing two feature vectors, one of them is shifted such that the highest maxima in both are aligned. Matching cost is then defined in terms of differences in corresponding maxima heights starting with the highest in each feature vector. Additional maxima in one of the feature vectors along with differences in position if corresponding maxima in both feature vectors are penalized.

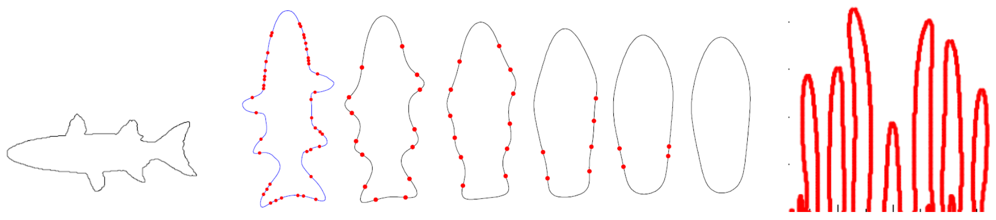


Figure 3.3: A shape (left), its smoothed versions (right) with highlighed zero crossings (red) and its CSS image (right), from [Mokhtarian96]. The festure vector is simply the positions and heights of maxima in the CSS image.

As the method relies on positions on the boundary contour, and not on positions in the image, it is intrinsically translation invariant. Scale invariance is achieved by initially normalizing each boundary contours to a total length of one. Alignment of highest maxima when comparing feature vectors leads to rotation invariance. In Section 5.2.2, when a rotation sensitive method is required, we assign an additional matching cost to the initial alignment of maxima. This penalizes shapes whose CSS images require large amounts of shifting for maxima alignment.

3.3.3 Contour to Centroid Triangulation (CCT)

The CCT method [Attalla05] connects the boundary contour to the center of the shape, dividing the contour into n equal length arcs. The spokes radiating from the centroid to the contour are ordered in clockwise direction with the longest one being first. Neighboring spokes define a triangle consisting of a chord corresponding to the arc of the shape. The feature vector stores for each spoke, its length, the angle it forms with its chord, and the ratio of chord to arc length. Each component is normalized to be between zero and one. Comparison cost is the L1 distance between components of feature vectors. As the method relies on the shape's centroid and spokes from the centroid to the contour, it is intrinsically rotation and translation invariant. Normalizing feature vectors also makes it scale invariant.

3.4 Similarity Structure Analysis (SSA)

SSA, or Multidimensional Scaling (MDS) [Borg05, Borg87] is a dimensionality reduction tool that projects high dimensional points to a lower dimension such that pairwise distances are preserved. We use SSA in Chapter 7 to obtain a 2D plot from an $N \times N$ similarity matrix, $\mathbf{S}$, where each entry $\mathbf{s}_{i,j}$ is the similarity distance between images i and j and . Note that by construction, $\mathbf{S}$ is symmetric and $\mathbf{s}_{i,i} = 0$.

N points, $\mathbf{P}_0 = \{\mathbf{p}_{i,0} | i \in \{1, \ldots, N\}\}$, are chosen at random in the Cartesian plane correspoding to the N compared images. The distance matrix, $D(\mathbf{P}_m)$, of the set, $\mathbf{P}_m$, $m \geq 0$, is computed such that $d_{i,j}$ is the Cartesian distance between $\mathbf{p}_{i,m}$ and $\mathbf{p}_{j,m}$. It follows that $d_{i,i} = 0$ for all i.

In order to compute the SSA plot, the *starting configuration* is set as $\mathbf{C}_0 = D(\mathbf{P}_0)$. An iterative process then starts whereby, in each iteration k, $k \geq 1$, the configuration matrix, $\mathbf{C}_{k-1}$, is checked to be the *SSA solution* of $\mathbf{S}$. If the solution has been reached then iteration stops. Otherwise, the positions of the points in $\mathbf{P}_{k-1}$ in the SSA plot are updated to $\mathbf{P}_k$, the new configuration matrix is computed as $\mathbf{C}_k = D(\mathbf{P}_k)$ and iteration continues.

3.4.1 Checking for an SSA Solution

To check whether a given $\mathbf{C}_m$, $m \geq 0$ is an SSA solution of $\mathbf{S}$, we need to construct the *ranking number matrix* of $\mathbf{C}_k$ and of $\mathbf{S}$. For a matrix, $\mathbf{A}$, to construct its ranking number matrix, $R(\mathbf{A})$, all entries $\mathbf{a}_{i,j}$ are sorted in descending order and given consecutive ranks. Thus, the largest entry gets a rank of one, the second largest a rank of two, and so on. Equal entries are assigned consecutive ranks. If the entries in $\mathbf{A}$ are then replaced by their ranks, we obtain $R(\mathbf{A})$.

$\mathbf{C}_m$ is an SSA solution of $\mathbf{S}$ when the condition, $R(\mathbf{C}_m) = R(\mathbf{S})$, is satisfied.

3.4.2 Updating Point Positions

Positions of points in $\mathbf{P}_m$, $m \geq 0$, are updated according to the *rank image matrix* of $\mathbf{C}_m$ with respect to $\mathbf{S}$. We denote the rank image matrix of a matrix, $\mathbf{A}$, with respect to another matrix, $\mathbf{B}$, as $R_{\mathbf{B}}(\mathbf{A})$. It contains the entries of $\mathbf{A}$ permuted such that the ranking number matrices of $R_{\mathbf{B}}(\mathbf{A})$ and $\mathbf{B}$ match, i.e. $R(R_{\mathbf{B}}(\mathbf{A})) = R(\mathbf{B})$.

For a given $\mathbf{C}_m$, $R_{\mathbf{S}}(\mathbf{C}_m)$ denotes the *intended* point configuration, i.e. it is desired that the Cartesian distances between the points in $\mathbf{P}_m$ follow a similar pattern as the similarity distances in $\mathbf{S}$, and that their distance matrix, which will be the next configuration matrix, be an SSA solution to $\mathbf{S}$. To achieve this, a *correction factor* is computed for each point pair $\mathbf{p}_{i,m}, \mathbf{p}_{j,m}$ as

$$f_{\{i,j\},m} = \frac{c'_{\{i,j\},m} - c_{\{i,j\},m}}{2c_{\{i,j\},m}},$$

where $c'_{\{i,j\},m}$ and $c_{\{i,j\},m}$ are entries in $R_{\mathbf{S}}(\mathbf{C}_m)$ and $\mathbf{C}_m$ respectively. The correction factor for a point pair can be thought of as the force between them; the 2 in the denominator denotes how the points exert equal forces on each other. A positive value of $f_{\{i,j\},m}$ indicates that the current distance between the point pair is an underestimate and should be increased, whereas a negative value implies a shortening of the distance.

The displacement of $\mathbf{p}_{i,m}$ with respect to $\mathbf{p}_{j,m}$ is then given as

$$\vec{\mathbf{d}}_{\{i,j\},m} = f_{\{i,j\},m} \cdot (\mathbf{p}_{i,m} - \mathbf{p}_{j,m}).$$

The total displacement for $\mathbf{p}_{i,m}$ with respect to all other points is then given as

$$\vec{\mathbf{d}}_{i,m} = \frac{1}{N-1} \sum_{j=1, j \neq i}^{N} \vec{\mathbf{d}}_{\{i,j\},m}.$$

The averaging above prevents displaced points from overshooting. Finally, the new point position is given by

$$\mathbf{p}_{i,m+1} = \mathbf{p}_{i,m} + \vec{\mathbf{d}}_{i,m}.$$

3.4.3 Stopping Condition

Ideally, iteration stops when the current configuration matrix, $\mathbf{C}_m$, $m \geq 0$, is an SSA solution of $\mathbf{S}$. Indeed the update of point positions explained above aims to achieve just that. However, as each point is acted upon by all other points, the distance matrix of the new point positions is typically still not a solution to $\mathbf{S}$. Thus, the points are moved again and again until a stopping condition is reached. With each iteration, the distance matrix of the point positions comes closer to the SSA solution of $\mathbf{S}$. This is reflected by progressively smaller values of $|f_{\{i,j\},m}|$ and $|\vec{\mathbf{d}}_{i,m}|$. Note that when the solution is reached, $f_{\{i,j\},m}$ and consequently $\vec{\mathbf{d}}_{i,m}$ will both be zero. In fact, after a certain number of iterations, $|\vec{\mathbf{d}}_{i,m}|$ becomes negligible. Therefore, iteration is stopped when the values of all $|f_{\{i,j\},m}|$ fall below a certain threshold. The point positions when iteration stops form the final configuration of the SSA plot.

3.5 View Descriptors

Many of our methods rely on views of shapes, such as those obtained in Section 3.1. A shape view is typically quantified using a *view descriptor*. Zernike moments presented in Section 3.3.1 are one such view descriptor. Two other descriptors we use are viewpoint entropy and view saliency which is based on mesh saliency.

3.5.1 Viewpoint Entropy

Viewpoint entropy [Vázquez01] of a scene estimates the amount of information contained in the scene as the minimum number of bits required to represent it. It is computed as

$$-\sum_{i=0}^{N_f} \frac{A_i}{A_t} log \frac{A_i}{A_t}$$

where N_f is the number of faces in the scene, A_i is the projected area of face i over the view sphere, A_0 is the projected area of the background over the view sphere, and A_t is the total area of the sphere. This expression is maximized when all projected areas are equal.

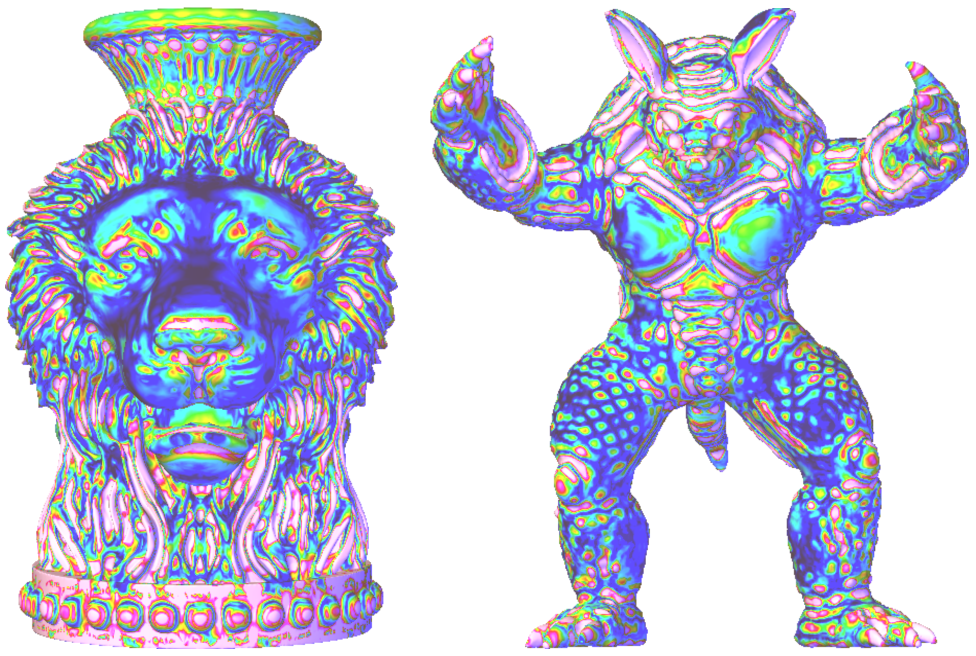


Figure 3.4: Saliency values for the Lion vase and Armadillo models, increasing from cold (blue) to warm (red) colurs.

3.5.2 Mesh Saliency

Mesh saliency [Lee05] predicts the amount of attention a user would pay to parts of a shape when viewing it. It is based on an older image saliency technique [Itti98] that computes visual importance of image regions to a human observer. Higher saliency values are assigned to more attention-grabbing parts. The motivation is that high curvature areas that “stand out” in their neighborhood are more salient than low curvature areas or periodically repeating patterns.

The saliency, $S(v)$, of each vertex, v, in a triangle mesh is computed as an aggregate of its saliency values, $S_i(v)$, at different scales, $i = 1 \dots n$. Saliency of v at each scale is defined in terms of its mean curvature, $C(v)$, as

$$S_i(v) = \|G(C(v), \sigma_i) - G(C(v), 2\sigma_i)\|,$$

where $G(C(v), \sigma)$ is the Gaussian weighted average of mean curvatures of vertices within a distance 2σ of v. Thus, $G(C(v), \sigma)$ represents a smoothing of mean curvatures around v and $S_i(v)$, computed at progressively larger scales, measures the change when smoothing over increasing radii around v. A high saliency value at a low scale indicates that the vertex belongs to a small shape feature, and a high value at a higher scale indicates a large feature.

$S(v)$ is computed as a weighted average of $S_i(v)$s where the weight for each level depends on the saliency maxima at that level and is meant to suppress spurious maxima at that level. Figure 3.4 shows our computed mesh saliencies of two shapes. For a given view, *view saliency* is simply the sum of saliencies of visible vertices.

3.6 Shape Descriptor: Shape Distributions

Just like view descriptors quantify a view of a shape, shape descriptors quantify a whole shape. In Chapter 8, we illustrate our efficient shape retrieval approach using the *D2 shape descriptor* as an example. The D2 shape descriptor is one of several *shape distributions* [Osada02]. Using the D2 distribution, the authors obtained more accurate results on a shape similarity benchmark than they did using other distributions.

The descriptor is computed by sampling the surface of the shape and repeatedly picking random pairs of samples and noting the distance between the points in the pair. The distribution of these distances forms the feature vector which, for convenience, is binned into a histogram.

Chapter 4

Learning the Shape of a Point Cloud

Stochastic methods and concepts are increasingly being found to model natural phenomena better than the hitherto used strictly logical methods, and a "sea change in our perspective" is envisioned when stochastic methods eventually overshadow traditional methods in use and application [Mumford99]. Parallels between statistical learning [Hastie01] and the workings of the human brain lead mathematicians to believe that such methods could one day help us understand the nature of intelligence itself [Poggio03].

The promise and efficacy of statistical learning methods has also been harnessed for Surface Reconstruction methods, and a review of some such methods was presented in Section 2.1. Although such methods are usually slower than their traditional astochastic counterparts, their superior handling of noisy, incomplete and uncertain data makes them especially attractive for Surface Reconstruction, where the input point cloud typically contains such artefacts.

In this chapter, we present *Neural Meshes* [Ivrissimtzis03, Jeong03], work on which we earlier published in [Isgro05, Saleem04, Saleem07a]. A neural network is initialized as a triangle mesh, $\mathcal{M}$, representing a tetrahedron. Vertices of $\mathcal{M}$ correspond to nodes and edges to connections of the neural network. For the rest of this chapter, no distinction is made between $\mathcal{M}$, which we call the neural mesh, and the neural network. $\mathcal{M}$ is trained to learn the shape of an input point cloud, $\mathcal{P} := \{\vec{p}_i(x_i, y_i, z_i) \; : \; i = 1, \ldots, N\}$, using an unsupervised learning scheme similar to Growing Cell Structures [Fritzke93]. Inspired by topology learning

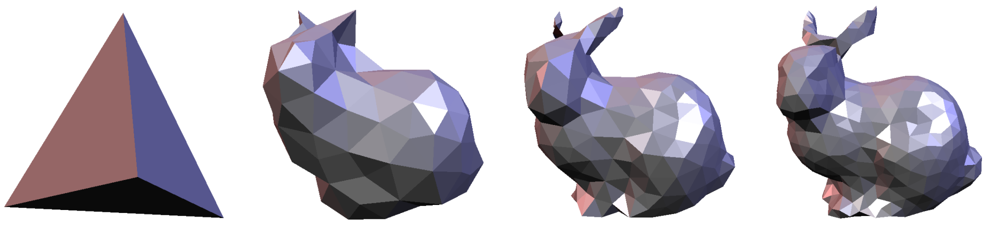


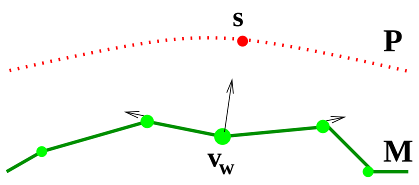
Figure 4.1: The neural mesh, $\mathcal{M}$, rapidly learns the general shape represented by $\mathcal{P}$. From left to right are the base mesh with 4 vertices followed by various learning stages at 100, 250 and 500 vertices.

methods [Fritzke94, Martinetz94] and in contrast to other learning based methods from Section 2.1, $\mathcal{M}$ can also learn topologically complex shapes from $\mathcal{P}$.

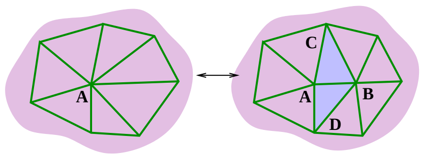
During training, some vertices learn more than others. Learning activity of all vertices is tracked and "active" vertices are rewarded in the form of addition of extra vertices in their neighborhood. This is different from [Barhak01b, Várady99] where entire rows/columns of new vertices are introduced. The method in [Yu99] also grows the mesh by adding new vertices, but it does so by globally subdividing the entire mesh, without regard to relative learning activities of vertices. By adding new vertices only in local neighborhoods of active vertices, we ensure added vertex population in only those regions of $\mathcal{M}$ that correspond to currently under-represented areas of the target shape. Thus, fine details of the target shape are represented more accurately by a larger number of vertices than coarser areas of the shape.

Once $\mathcal{M}$ has been initialized, learning proceeds by iterating over a few steps, namely the Geometry Learning, Node Addition, Node Removal and Topology Learning steps. The Geometry Learning step is the core step of the algorithm and is responsible for steering $\mathcal{M}$ to the shape represented by $\mathcal{P}$. However, as $\mathcal{M}$ is initialized as a tetrahedron, i.e. it initially has only four vertices, it cannot adequately learn any meaningfully complex shape. The Node Addition step is in charge of increasing vertex population adaptively. New vertices are added in those regions of $\mathcal{M}$ that under represent their corresponding areas of the target shape. Vertices in $\mathcal{M}$ that contribute to over representation of the target shape, either due to incorrect node addition, or those that are stuck in local minima (Figure 4.3) degrade mesh quality. These are removed from $\mathcal{M}$ in the Node Removal step. The Topology Learning step analyzes the current state of $\mathcal{M}$ to possibly make topological changes in it, namely removal of triangles to form boundaries, or merging of boundaries to form handles.

To perform these operations, activity information for each vertex, v, needs to be



(a) Geometry Learning



(b) Vertex addition/removal

Figure 4.2: (a) Positions of the winner and its neighbors are updated for each training sample. (b) Vertices are added/removed using complementary vertex split (left to right) and half-edge collapse (right to left) operations.

maintained. Each v is equipped with an *activity counter*, $\tau(v)$, that indicates the amount of learning v is performing, and a *winning sample number*, $S_w(v)$, which remembers the last time v learnt from $\mathcal{P}$. At initialization, both these quantities for all vertices are set to zero.

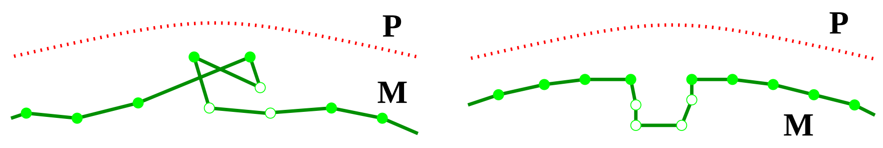
One of the most frequent operations in the above steps is to find the vertex in $\mathcal{M}$ that is closest to a given sample point. To optimize this computation, vertices of $\mathcal{M}$ are copied to an octree [Subramanian92]. This octree is updated each time vertices are repositioned, added or removed.

As Geometry Learning is the most important step, it is invoked in every iteration. Node Addition, Node Removal and Topology Learning are invoked with decreasing frequencies respectively.

The basic Neural Mesh algorithm yields triangle mesh reconstructions of the target shape whose vertex density mimics that of the input point cloud. With a small alteration, it is possible to obtain reconstructions where the vertex population follows curvature of the target shape instead. In [Saleem04], we also showed how the algorithm lends itself to a more efficient implementation that reduces its complexity from $O(N^2)$ to $O(N \log N)$ with little change in reconstruction quality, where N is the number of vertices in $\mathcal{M}$.

4.1 Geometry Learning

A sample, s, is obtained uniformly at random from $\mathcal{P}$, and the vertex in $\mathcal{M}$ that is closest to s is selected. The selected vertex is termed the *winner*, v_w. $\mathcal{M}$ learns from s by moving v_w towards s and applying smoothing to the 1-ring neighbors of v_w. Smoothing the neighborhood of v_w helps avoid foldovers and local minima in $\mathcal{M}$. Illustrations for the 1D case are given in Figures 4.2a and 4.3. The



(a) Foldover (b) Local minimum

Figure 4.3: Unwanted artifacts degrade mesh quality. Vertices represented by unfilled circles will never be selected as winners.

information contained in v_w is then updated.

4.1.1 Moving v_w

The new position of v_w is given as

$$\vec{v}_w \leftarrow \vec{v}_w + \alpha_w \cdot F(\vec{d}),$$

where $\vec{d} = \overrightarrow{v_w s}$ and α_w is a parameter between 0 and 1. $F(\vec{d})$ is a variant of Hubber's filter [Black98] that filters out the effects of outliers in $\mathcal{P}$. A moving average, μ_d, and standard deviation, σ_d, of $|\vec{d}|$ over the past 1000 training samples are maintained. An outlier threshold is then calculated as

$$\epsilon_d = \mu_d + \alpha_d \sigma_d,$$

using an input tolerance α_d, and $F(\vec{d})$ is then defined as

$$F(\vec{d}) = \begin{cases} 1 & \text{if } |\vec{d}| \leq \epsilon_d \\ \frac{\epsilon_d}{|\vec{d}|} & \text{if } |\vec{d}| > \epsilon_d \end{cases} .$$

4.1.2 Smoothing neighbors

For each vertex, v_i, in the 1-ring of v_w, its Laplacian [Taubin95] is calculated,

$$\vec{L}(v_i) = \frac{1}{n(v_i)} \sum_{v_k} (\vec{v}_k - \vec{v}_i),$$

followed by the displacement,

$$\vec{L}_s(v_i) = \vec{L}(v_i) - (\vec{L}(v_i) \cdot \vec{n}_i)\vec{n}_i,$$

where $n(v_i)$ is v_i's valence, v_ks its 1-ring neighbors and $\vec{n}_i$ its approximated normal. v_i's position is then updated as

$$\vec{v}_i \leftarrow \vec{v}_i + \alpha_s \vec{L}_s(v_i),$$

where α_s is a smoothing parameter between 0 and 1.

4.1.3 Updating Activity Information

Each time a new s is chosen from $\mathcal{P}$, a *current sample number*, S_c, is incremented. So, when $\mathcal{P}$ is sampled for the first time, S_c is set to one, at the second sample $S_c = 2$, and so on. S_c is used to update the stored activity information of the current v_w as follows.

First, the number of samples since the v_w's last "win" are computed,

$$x = S_c - S_w(v_w) - 1.$$

Recall, that when $\mathcal{M}$ is initialized, $S_w(v) = 0$ for all vertices v in $\mathcal{M}$. The activity counter is then updated as

$$\tau(v_w) \leftarrow \alpha_\tau(\alpha_\tau^x \tau(v_w) + 1),$$

where α_τ is calculated as $\alpha_\tau = (\frac{1}{2})^{\frac{1}{\lambda N}}$, N is the current number of vertices in $\mathcal{M}$ and λ is an input parameter such that a vertex loses half its counter value if it is not the winner for λN samples. Finally, v_w's winning sample number is set to the current sample number,

$$S_w(v_w) \leftarrow S_c.$$

4.2 Node Addition

As intended, "active" vertices, i.e. vertices that participate actively in learning, are identified by their high counter values. The presence of active vertices in a region of $\mathcal{M}$ indicates under representation of the target shape in that region. To remedy this, new vertices are added in the region.

First, activity information for all vertices is updated. Then, the vertex with the highest activity counter is selected and a new vertex is added in its neighborhood.

4.2.1 Updating Activity Information

For each vertex, v_i, in $\mathcal{M}$, the number of previous non-winning samples is calculated,

$$x = S_c - S_w(v_i),$$

and its activity counter is updated,

$$\tau(v_i) \leftarrow \alpha_{ctr}^{x}\tau(v_i),$$

along with its winning sample number,

$$S_w(v_i) \leftarrow S_c.$$

4.2.2 Adding a Vertex

The vertex, v_s, with the highest value for the activity counter is selected for a vertex split operation. The longest edge, e_s, incident on it is chosen and the edge-star of v_s is traversed in both clockwise and anti-clockwise directions to find two edges, e_1 and e_2 that divide the edge star in half. The vertex split operation is then performed on v_s along e_1 and e_2 with the new vertex, v_n, added at the midpoint of e_s. This is illustrated in Figure 4.2b where A corresponds to v_s, AC and AD to e_1 and e_2 and B to v_n.

In order to decide activity counter values for v_s and v_n, their *restricted Voronoi cells* (RVCs) are considered. A vertex's RVC is the intersection of the vertex's Voronoi cell with the surface of the target shape. The activity counter of v_s is distributed among itself and v_n in the ratio of the areas of their RVCs. In keeping with [Fritzke93], the RVC area of a vertex, *v*, is approximated by the area, F_v, of a square as

$$F_v = (l_v)^2$$

where

$$l_v = \frac{1}{valence(v)} \sum_{v_i \in 1-ring(v)} \|v_i - v\|$$

4.3 Node Removal

Vertices that are not participating in learning are either lying in areas of $\mathcal{M}$ that over represent the target shape, or are stuck in local minima or foldovers (Figure 4.3). These are "lazy" vertices and need to be removed from $\mathcal{M}$.

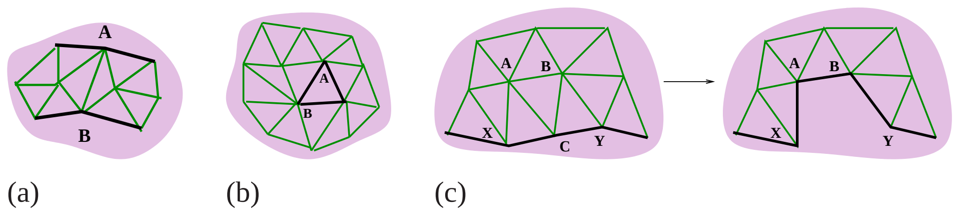


Figure 4.4: Preserving manifoldness (boundaries are shown in black). Collapsing edge AB (a,b) or removing triangle ABC (c) results in a non-manifold mesh. The half-edge collapses (a,b) are not performed, and the triangle removal is corrected by removing neighboring triangles.

Once again, the activity counter helps identify such vertices. After performing a counter update as in Section 4.2, the vertex with the lowest activity counter and other vertices whose counter values are below a threshold are chosen for removal through a half-edge collapse operation (Figure 4.2b).

The half-edge collapse operation, in addition to removing a vertex, changes the valences of three other vertices, as shown in Figure 4.2b, where B is the node to be removed, and A, C and D are the affected vertices. For that reason, when an edge around a vertex is to be collapsed, the algorithm selects from the vertex's edge star that edge whose collapse will cause the affected nodes to become as close to regular (valence 6) as possible. Such an edge has the least *regularity error*, E, of all the edges in the edge star, where E is given by

$$E = \frac{1}{3}\sqrt{(a+b-10)^2 + (c-7)^2 + (d-7)^2}$$

and a, b, c and d are the valences of A, B, C and D respectively in Figure 4.2b. For boundary edges, E is computed differently. If, in Figure 4.2b, we imagine a boundary running from left to right through the edge AB such that the part of $\mathcal{M}$ containing the vertex D does not exist, the regularity error for AB would be computed as

$$E = \frac{1}{2}\sqrt{(a+b-7)^2 + (c-7)^2}.$$

Apart from accommodating for boundary edges, the validity of a candidate half-edge collapse is also checked. As illustrated in Figure 4.4a,b, some collapses can lead to topological anomalies in $\mathcal{M}$. Such cases are easy to detect – endpoints of the edge to be collapsed lie on different boundaries, the edge to be collapsed is part of a boundary with only three edges. Invalid collapses like these are not carried out.

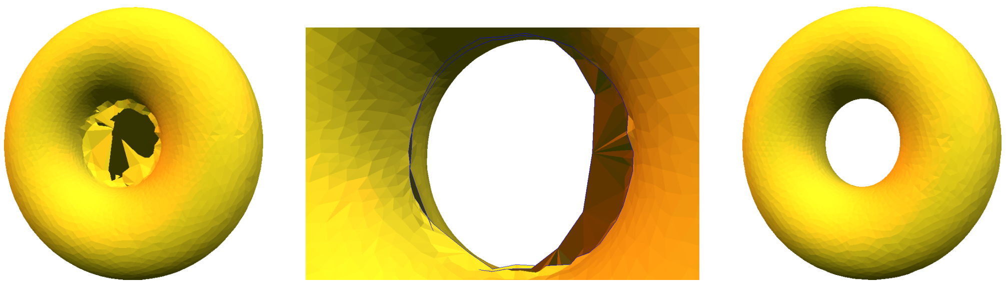


Figure 4.5: Learning topology. The hole is learnt (left) as large self-intersecting triangles, which are removed to form boundaries (center). With continued training, the boundaries grow close to each other and are merged to form the handle (right).

4.4 Topology Learning

Topology Learning consists of two sub-steps, removing large triangles from $\mathcal{M}$ to create boundaries, and merging boundaries close to each other to create handles. The motivation is that regions in $\mathcal{M}$ corresponding to holes in the target shape will contain very few vertices, as vertices in this region will typically be lazy and removed by the Node Removal step. This will lead to large triangles in such areas. Thus, large triangles in $\mathcal{M}$ are indicative of holes in the target shape. Following this strategy, a handle in the target shape will be learnt by $\mathcal{M}$ as two separate boundaries, that will grow close to each other during training. Therefore, when two boundaries in $\mathcal{M}$ get too close, they are merged to form a handle.

4.4.1 Triangle Removal

The average triangle area, $\overline{A}$, in $\mathcal{M}$ is computed and a triangle removal threshold, T_r, is computed as

$$T_r = \alpha_r \overline{A},$$

where α_r is an input parameter. All triangles in $\mathcal{M}$ that have an area greater than T_r are selected for removal. However, removing some triangles can cause $\mathcal{M}$ to become non-manifold. This is shown in Figure 4.4c. For such triangles, neighboring triangles are also removed to preserve manifoldness of $\mathcal{M}$.

The average triangle area, $\overline{A}$, in $\mathcal{M}$ is used to calculate a triangle removal and a boundary merging threshold. Triangles with area greater than the triangle removal threshold are removed, and boundaries whose Hausdorff distance to each other is less than the boundary merging threshold are merged. If the removal of a triangle causes $\mathcal{M}$ to no longer be manifold, neighboring triangles are removed to restore manifoldness (Figure 4.4c). Figure 4.5 shows the effect of these steps.

4.4.2 Boundary Merging

A boundary merging threshold, T_m, is computed in terms of $\overline{A}$ as

$$T_m = \alpha_m \sqrt{\overline{A}},$$

where α_m is an input parameter. Boundaries are merged when the Hausdorff distance between them falls below T_m. The Hausdorff distance between the boundaries is estimated as the Hausdorff distance between the sets of vertices representing them. The merging procedure is as follows. Starting from the two closest nodes, one from each boundary, both boundaries are traversed in the same orientation checking for possible triangles with the next vertex. The candidate which is closest to equilateral is selected and added to the mesh. Traversal followed by triangle addition continues until the two boundaries are completely connected with a set of triangles.

4.5 Feature Sensitive Reconstructions

Sharp features in a surface are characterized by high curvature values. Vertices in regions of $\mathcal{M}$ corresponding to such features in a target shape exhibit large changes in their normals during training. Thus, tracking changes in normals of vertices gives hints about curvature of the target shape. Therefore, rewarding, through Node Addition, those vertices that exhibit large variations in normals leads to feature sensitive reconstructions.

This is achieved by replacing the activity counter, τ, by a *normal counter*, η. In the Basic Step, a change in the winner's normal is measured as

$$\delta_w = 1 - \overrightarrow{\mathrm{n}_w}\overrightarrow{\mathrm{n}_w}',$$

where $\overrightarrow{\mathrm{n}_w}$ and $\overrightarrow{\mathrm{n}_w}'$ are the (normalized) normals estimated at v_w before and after its movement respectively. δ_w is then normalized as

$$n_\delta = \frac{\delta_w}{M_\delta},$$

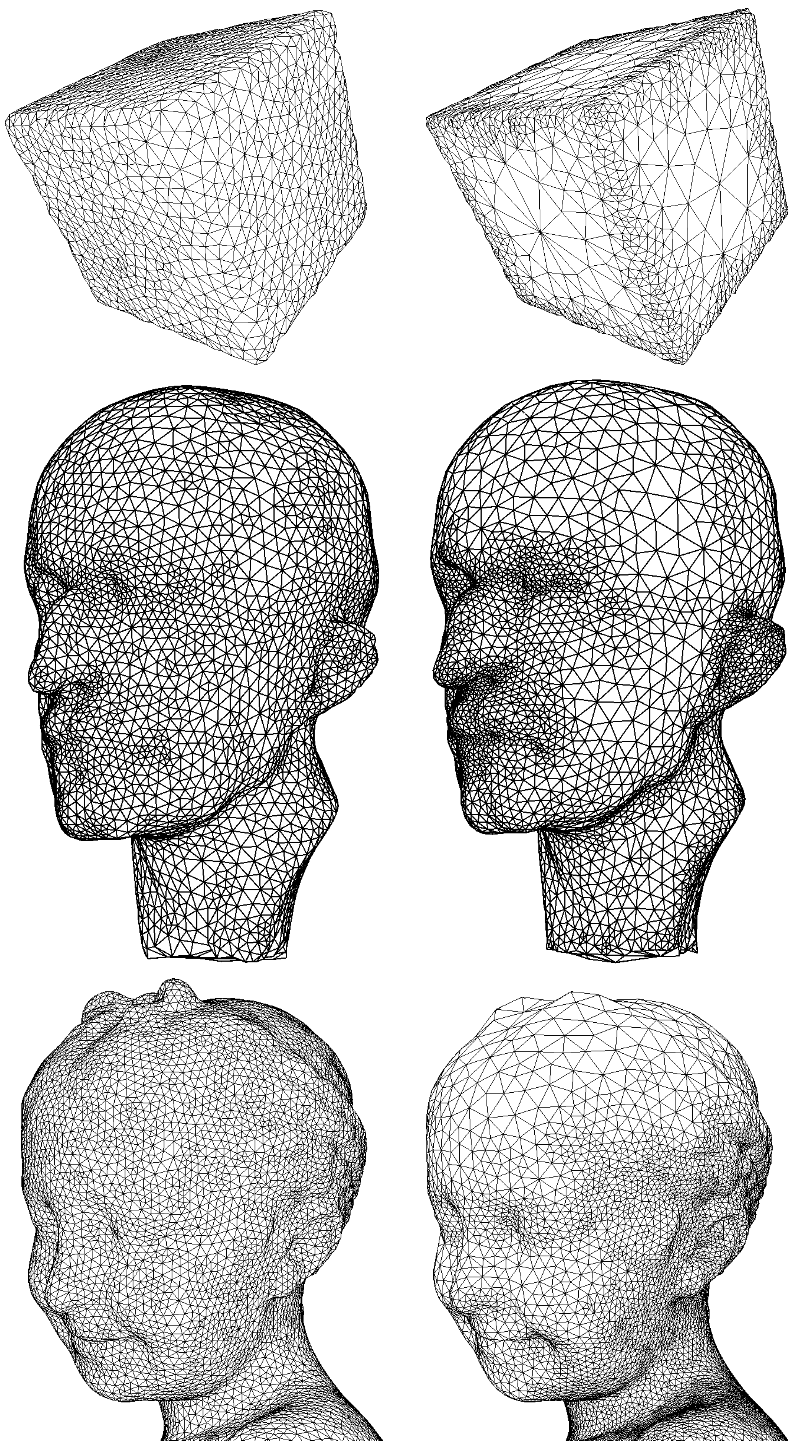


Figure 4.6: Neural Mesh reconstructions of the cube (top) and Bimba (bottom) models according to sampling density of $\mathcal{P}$ (left) and surface curvature (right). Reconstructions of the same size are compared.

where M_δ is the mean value of δ_w's over the last C_δ iterations and C_δ is a user defined constant, e.g. 1000. Again, the number of samples since the v_w's last "win" are computed,

$$x = S_c - S_w(v_w) - 1,$$

and finally $\eta(v_w)$ is updated as

$$\eta(v_w) \leftarrow \alpha_\eta(\alpha_\eta^x \eta(v_w) + n_\delta).$$

Note that when $\mathcal{M}$ is trained using the normal counter instead of the activity counter, it is not possible to apply the Topology Learning step, as large triangles in $\mathcal{M}$ no longer correspond to holes in the target shape. Instead, they now correspond to flat regions in the target shape. Some reconstructions of this sort are shown in Figure 4.6.

4.6 A Priority Queue Implementation

One core part of the Neural Mesh algorithm is the tracking of vertex activities using counters. These counters then need to be scanned during Node Addition and Node Removal steps to identify vertices with highest/lowest values of the counters. This is an $O(N)$ operation where N is the number of vertices in $\mathcal{M}$.

As the absolute values of the counters are not important for the algorithm, rather the relative values for different vertices, the counter can be done away with altogether by copying the vertices to a priority queue data structure, where the relative counter value of a vertex is modeled by the priority of the corresponding element in the queue. Changes in counter values can then be modeled by simply changing the priorities of the elements in the queue, or by "jumping" the corresponding elements ahead or behind in the queue by a certain amount. Addition of vertices to $\mathcal{M}$ with a particular counter value then corresponds to addition of a new element in the queue at a particular position. Vertex removal corresponds to element removal. The advantage offered by this approach is that vertices with high/low counter values can be spotted trivially at the head/tail of the queue.

Implementing the queue as a self-balancing AVL tree [Adel'son-Vel'skii62] allows changes in queue positions to run in $O(N \log N)$ time and selection of vertices to be split/removed in $O(1)$ time. A full treatment of this implementation is outside the scope of this thesis and we refer the reader to [Saleem04] for more details, where we also show that this implementation brings down the complexity of the algorithm from $O(N^2)$ to $O(N \log N)$.

4.7 Discussion

Neural Meshes effectively solve the problems unaddressed by previous surface learning methods. They can represent entire surfaces, not just patches. Learning is adaptive; it starts with a small, simple initial surface to which vertices are added only where needed, i.e. in the 1-ring neighborhood of active vertices. Vertices that become misplaced during training and over represent $\mathcal{P}$ are removed. Also, neural meshes possess the ability to learn topology.

Notice that the only step of the algorithm where $\mathcal{P}$ is required is in picking training samples, thus making the running time independent of the size of P. This is in direct contrast to methods that need to process all input points in order to output a surface. Independence from the input point cloud also allows out-of-core processing of large data sets.

Like most learning algorithms, Neural Meshes suffer from the need for user parameters. While a default set of values can be set, best results will be obtained by tuning the parameters in accordance to the input set. This could be seen as an advantage for the expert user. For an extensive treatment of this issue, we refer the reader to [Saleem04].

Despite the speedup offered by the priority queue implementation, the method is slow and non-competitive with contemporary geometry-based Surface Reconstruction methods [Schall05]. The majority of the running time is spent in Geometry Learning. We expect that a shrink-wrapping approach, similar to [Kobbelt99], with the Neural Mesh initialized as an inflated bounding sphere with number of vertices close to the final number could offer a solution to this problem.

Chapter 5

Static and Dynamic Shape Views

3D views are ultimately displayed on 2D media, invoking a projection from three dimensional shape space to two dimensional display space. As infinitely many projections are possible and display space is limited, it becomes important to choose a few good projections or *views*. Our interest in representative views is mainly in context of shape repositories, where a catalog of views of stored shapes is presented to the user for browsing. Possible applications of the methods presented in this chapter are to automatically compute such views for and to support view based similarity retrieval of shapes stored in the repository. However, the methods are general enough to be used in other contexts as well.

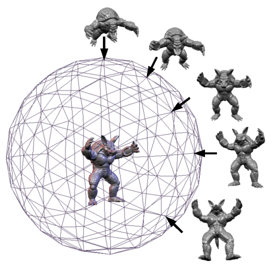
What distinguishes a good view from a bad one in the eyes of a human observer relies on the nature of 3D shape perception which is an active research topic in psychology, neuroscience, psychophysics, and computer science [Blanz99, Tarr01, Todd04] (see also references therein). As mentioned in Section 2.2.1, view selection, or the *best view problem*, receives attention from diverse areas. Broadly, we can distinguish between them as computer vision motivated techniques [Arbel99, Denton04, Hall05, Lee04, Mokhtarian00] that define representative 2D views for later recognition and representation, and computer graphics approaches [Bordoloi05, Gumhold02, Lee05, Podolak06, Shacked01, Takahashi05, Vázquez03a] that aim to effectively and economically present a shape to an observer. The former analyze similarity and stability relationships [Cutzu94, Cutzu96, Weinshall97] between different views while the latter maximize defined view descriptors. A comparison of various best view approaches in [Polonsky05] concludes that while each of the descriptors covered in the paper is reasonably good, it inevitably fails to produce satisfactory results for certain types of shapes. In Section 5.1, we suggest a method that combines similarity and goodness/saliency approaches to inherit the strengths of each approach while compensating for their individual disadvantages. This work was earlier published as [Yamauchi06].

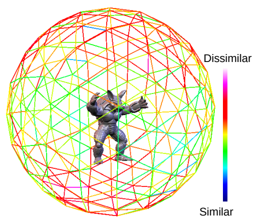
None of the above methods, including ours, consider shape orientation during view selection. In fact, as we mentioned in Section 2.4, in selecting views these methods employ rotation invariant techniques that cannot distinguish between views in which the shape is oriented differently. Therefore, the resulting views contain the shape in arbitrary orientation. This is a known problem and shape orientation in selected views is often corrected by hand [Polonsky05, Takahashi05]. An automatic heuristic we proposed and present in Section 5.1.1 turned out to work only in a few cases. We then tackled the problem of correcting the orientation of a shape in its view, in a systematic manner resulting in our example based method previously published as [Saleem07c] and presented here in Section 5.2. To the best of our knowledge, ours is the first attempt to automatically solve the problem of fixing shape orientation in views.

Static views limit exposition of the shape to just the shown parts. This problem is further exacerbated in methods that choose only a single view and thus convey no information on the rest of the shape. A dynamic view, on the other hand, presents a smooth animation of different parts of the shape. We surveyed the little existing work in this direction in Section 2.2.2 where we summarized the conditions that should be fulfilled as the camera recording the view travels (flies) along a computed path. In tradition of the best view problem, we term the problem of finding good dynamic views of a shape as the “best fly” problem. We present our work on extending best views to compute a best fly in Section 5.3, earlier published as [Saleem07b], whereby the speed and zoom of the camera along the computed path vary in accordance with viewed shape features. As far as we know, we are the first to alter the speed and zoom parameters of the camera to provide a more informative fly.

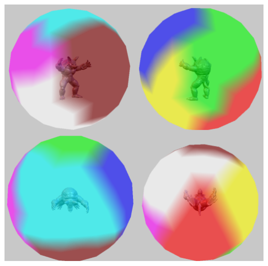
5.1 Stable and Salient Shape Views

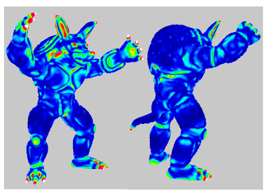
Like the computer vision methods mentioned above, our best view method is also based on similarities between shape views. We simplify similarity computation by considering only the binary silhouette of the shape in its views. The obtained similarity values are then used to guide a clustering process that partitions the view sphere into *stable view regions* where viewpoints in such a region share a similar view of the shape which is different from views from other regions. Using the

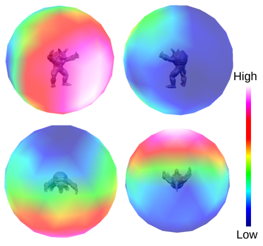

(a) a view sphere



(b) similarity weighted spherical graph


(c) view sphere partitioned into stable view regions


(d) mesh saliency



(e) view saliency

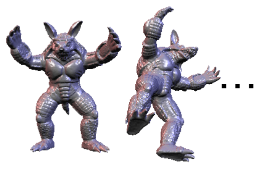

(f) computed views

Figure 5.1: Overview of our stable and salient view selection method. Top row: Partitioning the view sphere into stable view regions. (a) view sphere and observed object. (b) similarity weighted spherical graph. (c) colored stable view regions. Bottom row: using view saliency to select the final views. (d) mesh saliency. (e) visualization of view saliency. (f) selected representative views.

perceptually motivated view saliency measure [Lee05], we pick a representative from each of these regions and the most salient of these is our suggested best view of the shape. An overview of the method is presented in Figure 5.1.

We obtain 162 views of the shape as described in Section 3.1 corresponding to the vertices of our view sphere approximated by a doubly Loop-subdivided icosahedron. The view sphere itself forms a *spherical graph* in which an edge connects *neighboring views*. We discard all color information to consider only the silhouette of the shape in each view, as shown in Figure 5.2, and then perform pair wise image similarity comparison between all neighboring views using Zernike moments analysis, as described in Section 3.3.1. We use moments up to order 15. The similarity value between neighboring views is assigned as a weight to the corresponding edge in the spherical graph. Figures 5.1b and 5.3 show the spherical graph with edges colored by similarity weights. A blue edge represents high similarity between its incident views while red edges represent dissimilar views. Rotation invariance of our view similarity method is illustrated in Figure 5.3a, where edges between views that are rotated version of themselves have received

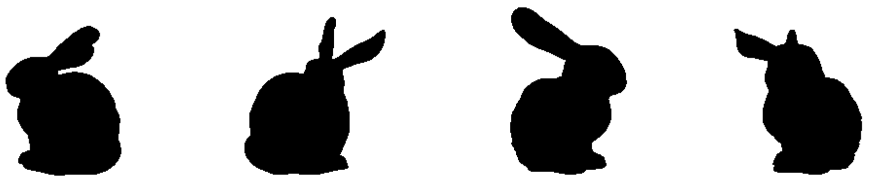
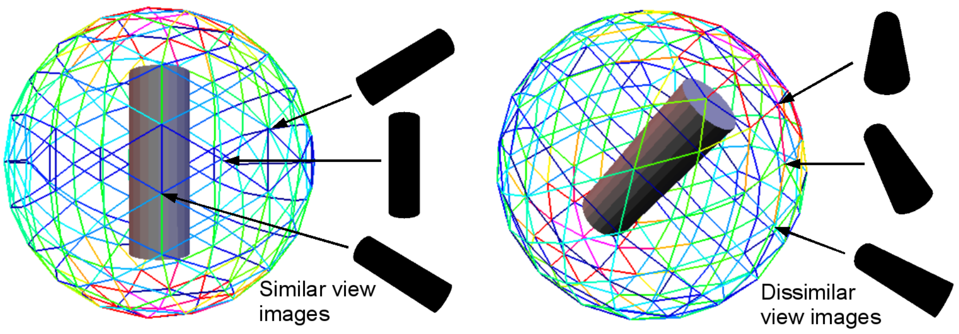


Figure 5.2: Sample of binary (silhouette) views.

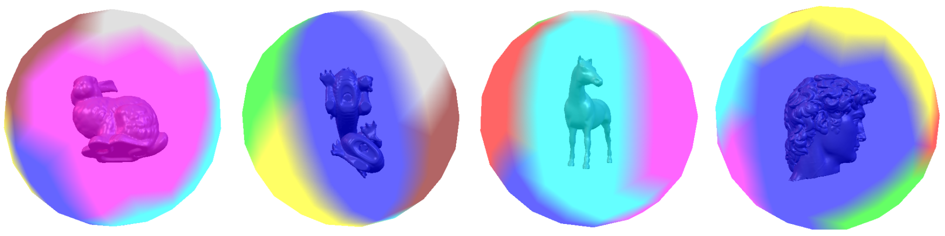


(a) similar views (b) dissimilar views

Figure 5.3: Similarity weighted spherical graph of a cylinder.

low weights.

In our similarity weighted view sphere, a stable view [Weinshall97] will be identifiable as a viewpoint with high similarity weights on all incident edges. An area of the view sphere that groups several such stable views together forms a *stable view region*. To find stable view regions, we partition the similarity weighted spherical graph based on its edge weights. One possible way to achieve this is to find an edge cut that segments the graph into the requested number of partitions while minimizing the total weights of edges in the cut. This way we prevent regions with high stability to be partitioned into two disjoint parts. MeTiS [Karypis98] is a graph partitioning application that partitions a graph into the requested number of sub-graphs. Given a graph with weighted edges, MeTiS finds an edge cut which minimizes total weights and generates sub-graphs with balanced number of vertices. Although our application does not require the balancing property, our experiments show that it does not create any bias. Figure 5.1c shows an example of this partition method (each partition has a different color). The partitioning quality is also influenced by the sampling density of the view sphere. Sparse sampling will result in a small graph which will be difficult to partition. In the field of object recognition, around 50 uniform distributed samples are


(a) each stable view region is represented by one of its views.

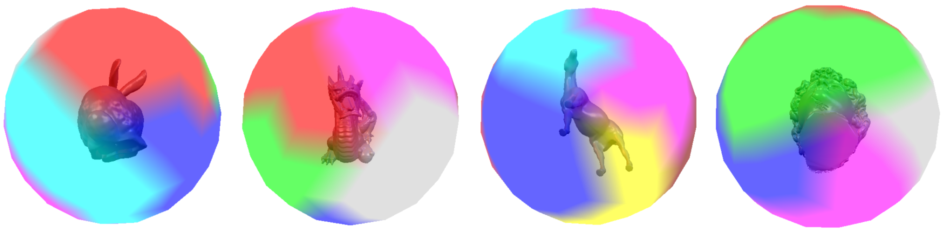

(b) unstable views lie at the intersection of stable view regions.

Figure 5.4: Stable view regions.

used [Seibert92, Mokhtarian00]. In our implementation, 162 samples are used, same as in [Takahashi05], and from our experience this number is sufficient.

We pick a single view from each region as its representative. This view is computed as the saliency weighted average of all viewpoints in the region. That is, the representative viewpoint, $\mathbf{R}_i$, of a partition, P_i, is given by

$$\mathbf{R}_i = \frac{\sum_{j \in P_i} s_j \cdot \mathbf{p}_j}{\sum_{j \in P_i} s_j},$$

where $\mathbf{p}_j$ is the position of a view sphere vertex and s_j is its corresponding view saliency. The representative views are then ranked according to their view saliencies.

5.1.1 A Suggestion for Model Orientation

One of the major challenges in view selection is to find the proper orientation of the 3D model. For example, when viewing a model of a four-legged animal, we expect the view-selection method to orient the view such that the animal will be on its feet and not on any other body-part. At the time of publication of this work, this problem was yet to be addressed successfully and as a full solution was outside the scope of the work, we suggested and tried the following heuristic.

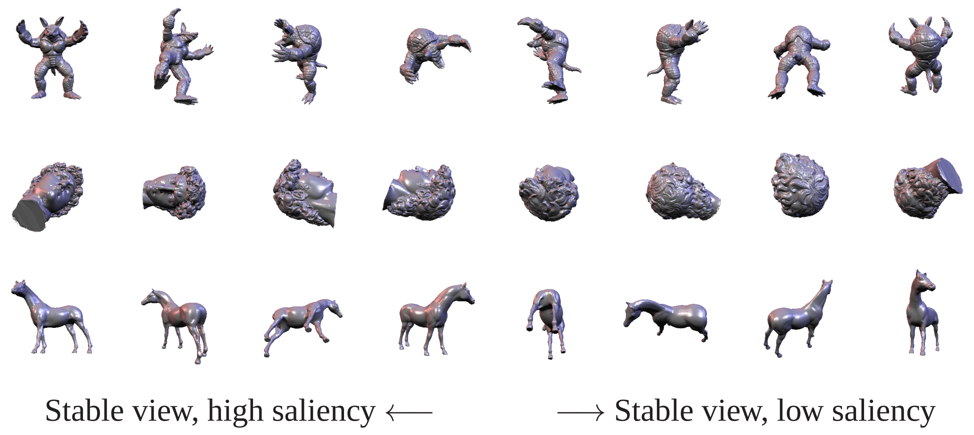


Figure 5.5: Selection of stable and salient views using binary (silhouette) views.

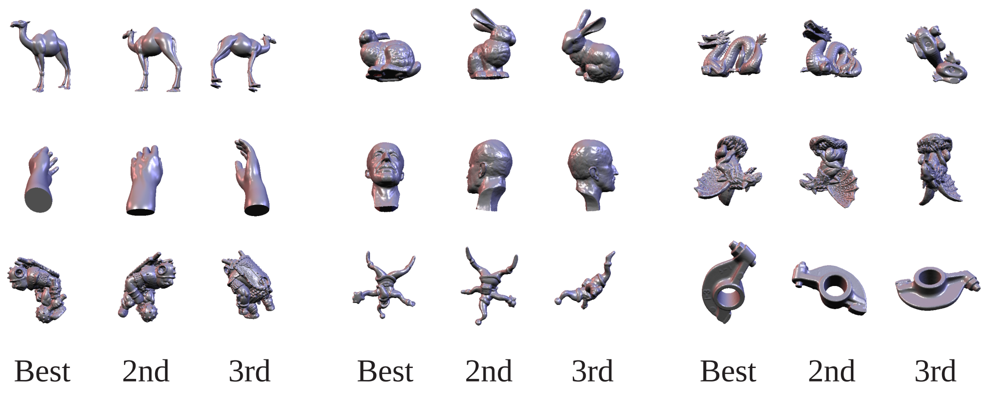


Figure 5.6: Best three views generated by our approach.

We claim that for an object to be properly oriented, its least important part should be facing down. The justification is that usually the lower part of an object is hidden from the viewer and the viewer would thus choose the least important part to be the lower one. Figure 5.7 shows several examples of model orientation selection based this our hypothesis. For all other view selection results in this chapter, the up vector was set to $(0, 1, 0)$. View orientation in the figures has not been adjusted in any other way.

As we see in Figure 5.7, our suggestion led to correct orientation for only 3 out of the 12 models that we tested. Our later work in this direction is presented in Section 5.2 where we obtain better results. After the publication of our two works, a classification based method [Fu08] was proposed to compute upright orientations of man made objects. We revisit the topic of shape orientation in 2D views in detail in Section 5.2.

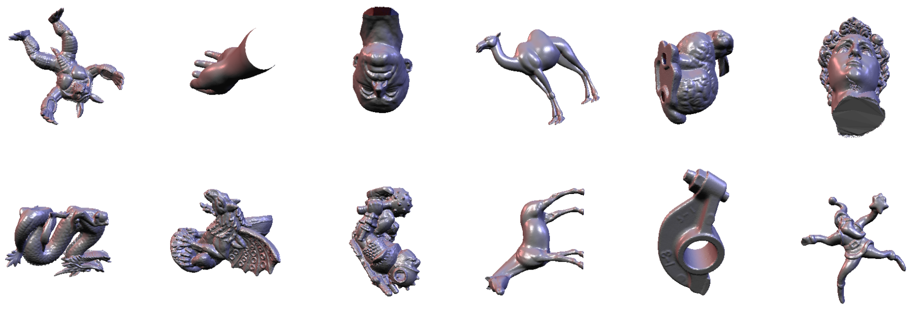


Figure 5.7: Best views generated by our approach together with our suggestion for model orientation.

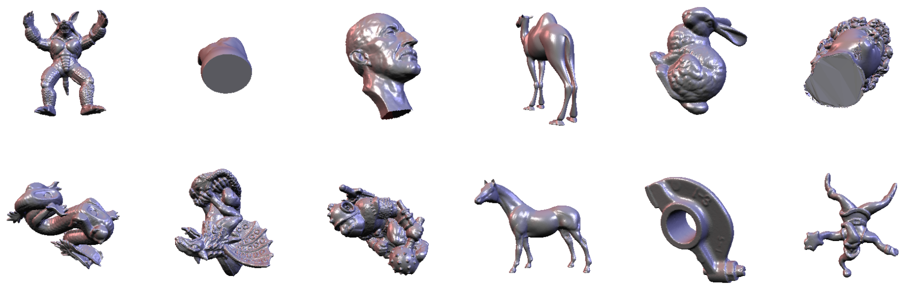


Figure 5.8: The best views selected by view saliency only.

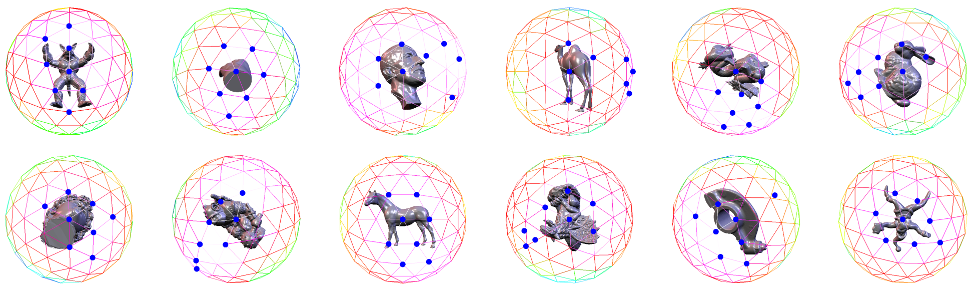


Figure 5.9: Top 8 most salient viewpoints. (Blue points) The view sphere's edge color reflects saliency value of the incident viewpoints. Notice the back of view sphere is culled for visualization.

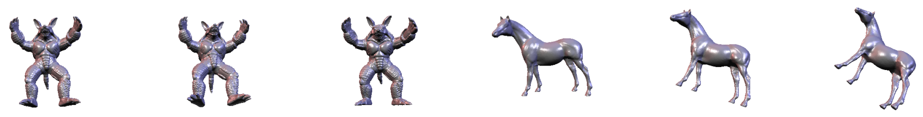


Figure 5.10: Top 3 salient view examples of Figure 5.9.

5.1.2 Results and Discussion

Figure 5.4 shows some results obtained by partitioning the similarity weighted spherical graph using MeTiS. Figure 5.4a shows a representative stable view for each model while Figure 5.4b shows an unstable view. Note, that the unstable view lies at the intersection of several stable view regions. We see that the stable regions for the dragon and horse models are stretched in the sphere's longitudinal direction. This implies that a movement of the viewpoint along a longitudinal line will result in minimal change in view compared to movement in a latitudinal direction.

We fixed the number of partitions (also the number of final views) to 8. From our experiments, this number usually proved sufficient to cover all interesting parts of the model. A larger number of partitions may not reveal any new information, while a smaller number might not suffice.

Figures5.5 and 5.6 show several results of our experiments using our automatic multi-view selection method. We also examined illuminance (grayscale) images using the Gouraud shading model besides silhouette (binary) images. However, the shaded images are sensitive to environmental conditions, such as lighting, and hence tend to bias the results. Due to this, we concluded that the use of binary images is more appropriate for our purposes.

Figure 5.9 shows a multi-view selection that is based on saliency alone. It is easy to see that all high saliency viewpoints are concentrated in a small region on the view sphere. This is because small deviations in viewpoint do not affect the saliency value much. By taking into account the stability of the view, we force the views to be spread all over the view sphere, resulting in a better distribution.

Figure 5.8 shows the best views selected by the mesh saliency method [Lee05]. Comparing Figure 5.8 with Figures 5.5 and 5.6, we can say that our results are comparable or, in some cases, better. For example, the bottom of the dragon model is the most salient (Figure 5.8). However, when stability is taken into account, the best view changes to the side of the dragon, which is a much better suggestion (Figure 5.6), although this is not the most salient view.

Similarly, the neck of David's head is also salient due to its high and consistent curvature. However, this view is unstable (Figure 5.4). Therefore, our method avoids the uninteresting view of the neck and recommends better views that cover the front and top of the head (Figure 5.5). Note that our method does not ignore the saliency recommendation in the neck region, but combines it with a stable view. These examples show the benefits of our method achieved by combining two human perception elements, stability and saliency.

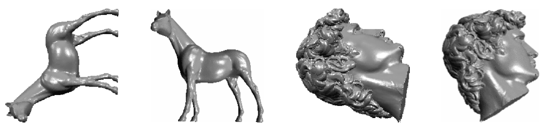


Figure 5.11: Two views each with different orientations of two shapes. Given the left image of each shape, our method automatically computes the right image, which has a more natural orientation for the shape.

No optimization was done throughout the implementation. Computation time of constructing the similarity weighted spherical graph is about 40 minutes. This does not depend on the shape since view similarity is computed on the rendered views whose number is fixed (number of vertices of the view sphere). View resolution is set to 256×256. The bottleneck here is the Zernike moment computation which is known to be slow. Adopting one of several optimizations suggested in [Chong03] will dramatically reduce the time needed for this step. Graph partitioning by MeTiS takes less than 10ms. Mesh saliency computation times depend on the number of vertices in the shapes and agree with those in the original paper [Lee05]. The view saliency computation takes less than a minute, depending on the rendering time of the model. All timings were measured on a 2.8GHz Pentium 4 PC with an ATI Radeon R300.

5.2 Example Based Shape Orientation

Humans know the correct orientation(s) of a shape through experience with objects of that shape. It is through previous interaction that we know, for example, that the correct orientation for a car is one in which the wheels are down, instead of up. Such information is difficult, if not impossible, to compute from the shape alone [Blanz99]. As view-finding techniques are insensitive to rotation, they cannot distinguish between the views in each row of Figure 5.11. However, a human observer would clearly prefer the views in the right column over those in the left column. In Section 5.1, we proposed a heuristic to automatically correct shape orientation in chosen views but the results, shown in Figure 5.7, turned out to be unsatisfactory.

In this section, we suggest a two step example based approach whereby a query shape is matched with exemplars in a database of classified, correctly oriented shapes. Correct orientation of database shapes is determined manually beforehand. In the first step, classification, the query shape is matched up with a candidate class from the database and a target shape from the candidate class is chosen.

In the subsequent alignment step, the query shape is re-oriented according to the target shape's orientation.

5.2.1 Setting up the Database

To set up our database, we choose the MPEG-7 dataset [Latecki00, Veltkamp06], which contains 1400 binary images organised into 70 classes with 20 members per class, where each image contains a single shape. As the dataset was compiled to test shape similarity techniques, members of the same class differ from each other in shape features. With regards to orientation, a lot of redundancy and false information had to be filtered out.

We found several classes of *non-orientable* shapes. These are artificial shapes for which no notion of correct orientation exists. These classes were removed. In some of the remaining classes, we found images with incorrect orientations. These images mostly contain natural objects in orientations which a human observer can easily specify to be incorrect. Such images were also removed, and later used as queries for our method.

Next, we filter out *redundant images* from the dataset. These are class members that differ from each other only in fine shape features and not necessarily in orientation. We removed redundancies automatically by identifying groups of very similar images within a class, and retaining one image from each group. This is done by choosing a class member initially at random as a query image and comparing it with all other class members. Member images that are too similar to the query, i.e. their similarity distance is below a threshold, are removed from the dataset. Note that the similarity measure used is rotation sensitive and is described in Section 3.3.2. The query image is flagged so that it can no longer be removed from the dataset. Next, the image which is the most dissimilar to the query image, i.e. has the maximum similarity distance, is taken as the query image and the process is repeated. Repetition stops when the class comprises only of flagged images. Figure 5.12 shows examples of the kinds of images that are removed from the database.

After the above filtering steps, the original dataset is reduced to 237 images in 56 classes. The smallest and largest class memberships are 1 and 14 respectively with a median of 4. The choice of 'similar' images is very much dependent on the shape similarity method used. We talk more about this in Section 5.2.4. Within the database, we represent each shape by its boundary contour, as described in Section 3.2. From the 237 images in our database, 20 images required blurring iterations, with only 4 images requiring 3 or more iterations.

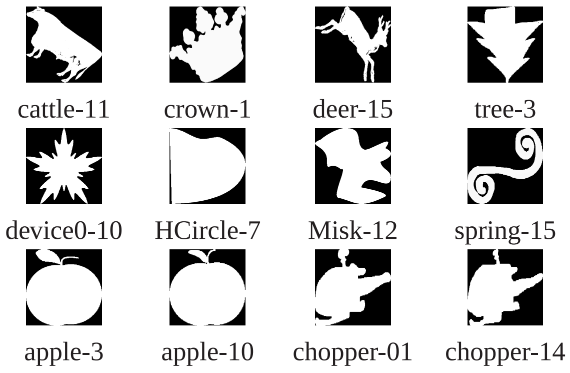


Figure 5.12: From the original dataset, we manually remove incorrectly oriented images (top row) and images containing non-orientable shapes (middle row). Redundantly oriented images (bottom row) within a class can be removed automatically.

5.2.2 Example based Shape Orientation

As mentioned earlier, our method proceeds in two steps – a classification followed by an alignment step. We use nearest neighbour classification. Using the CSS shape similarity method [Mokhtarian03], explained in Section 3.3.2, we retrieve a list of database images sorted according to their similarity distance from the query image. The candidate class is then chosen as the class containing the best match, i.e. the image least distant from the query.

From the candidate class, we choose the shape most similar to the query shape. This is the *target shape*. Recall that database shapes are correctly oriented, and that all shapes are represented by their boundary contours. To align the query shape to the target shape, we rotate the query shape such that the directions of its Principal Components (PCs) match those of the target shape.

To remove bias because of point density, shape contours are uniformly sampled before PCA calculation. An artefact of PCA alignment is that using PCs, a shape cannot be distinguished from its $180°$ rotation. Thus, during alignment, the query image could end up getting mis-aligned by $180°$. To avoid this, once the query shape has been rotated as described above, we match it with both the target shape and a $180°$ rotation of the target shape. If the $180°$ rotated target shape gives a better match, we rotate our final image by $180°$.

5.2.3 Results

We query our method with the incorrectly oriented shapes removed from the original MPEG-7 dataset, with user input images, and with commonly used shapes from Computer Graphics literature. We used images of commonly used models that were presented as best-view results in Section 5.1. Results are shown in Figures 5.13 to 5.15 (some artefacts may be visible because of image resizing). The columns show query images, corresponding target shapes chosen from the database and our final results. Recall that database shapes are assumed to be correctly oriented. The correctness of a result is evaluated visually, and is taken to be correct if the resulting image is similarly oriented as a database image of the same class, or if it agrees with the human notion of correct orientation for the contained shape.

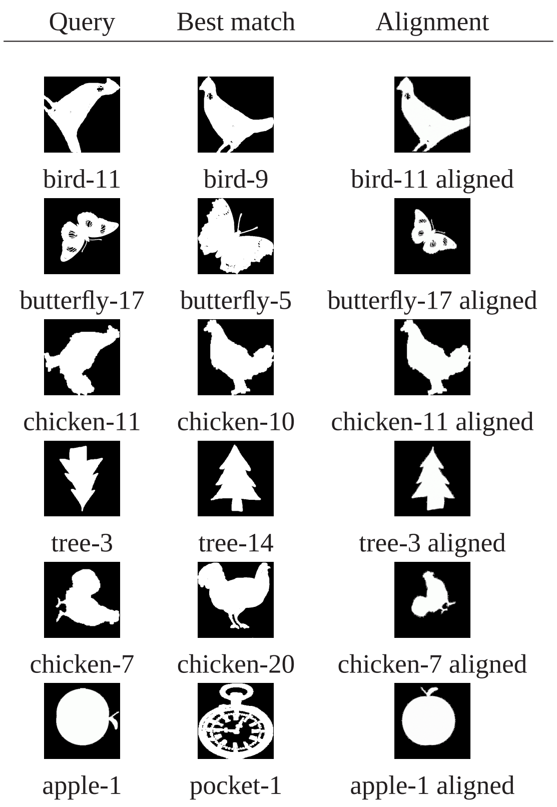


Figure 5.13: Results for incorrectly oriented shapes.

We have the most success when using incorrectly oriented shapes previously removed from the dataset, Figure 5.13, as they have similar class members still in the database. The tree-3 case shows the efficacy of the PCA rotation correction mentioned at the end of the last section. Even though the query and target images

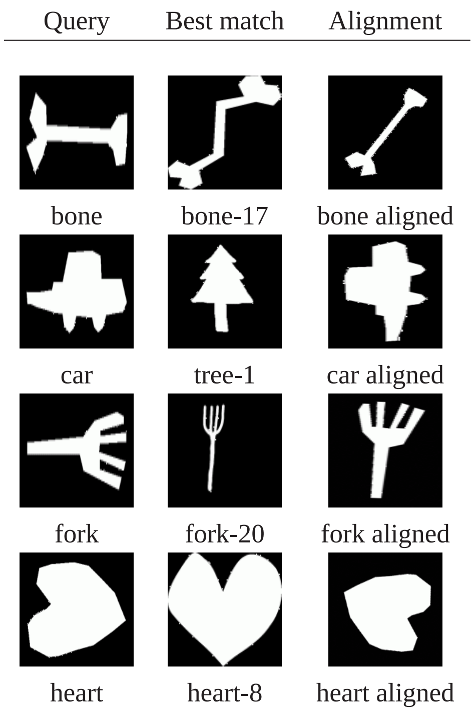


Figure 5.14: Results for user drawn queries.

have the same PCs, our method is able to detect and misalignment and rotates the query by $180°$ for better alignment. The chicken-7 case illustrates the limitations of a shape's PCs in estimating its orientation. While the target shape is very similar to the query shape, its PCs and those of the query do not match, thus ending up in an incorrectly oriented result. The apple-1 case illustrates our use of object boundaries for similarity computation. In terms of shape boundaries, the apple and pocket watch are hardly distinguishable.

Results for user sketches, Figure 5.14, are fairly good when the sketch matches an existing database shape closely. Limitation of orientation estimation by PCs is again illustrated in the heart case, and the car case shows the deficiency of our method when queried with a shape different from database exemplars (our database contains a personal_car class whose members are differently shaped than this query).

Major deficiencies of the method are fully exposed in Figure 5.15. Our database already contained a horse exemplar very similar to the query horse, thus yielding a good result. We expected the camel to match the horse as well, but its curvature properties caused our similarity method to deem it more similar to the frog, resulting in an incorrect output orientation. (Notice that the camel's front and hind

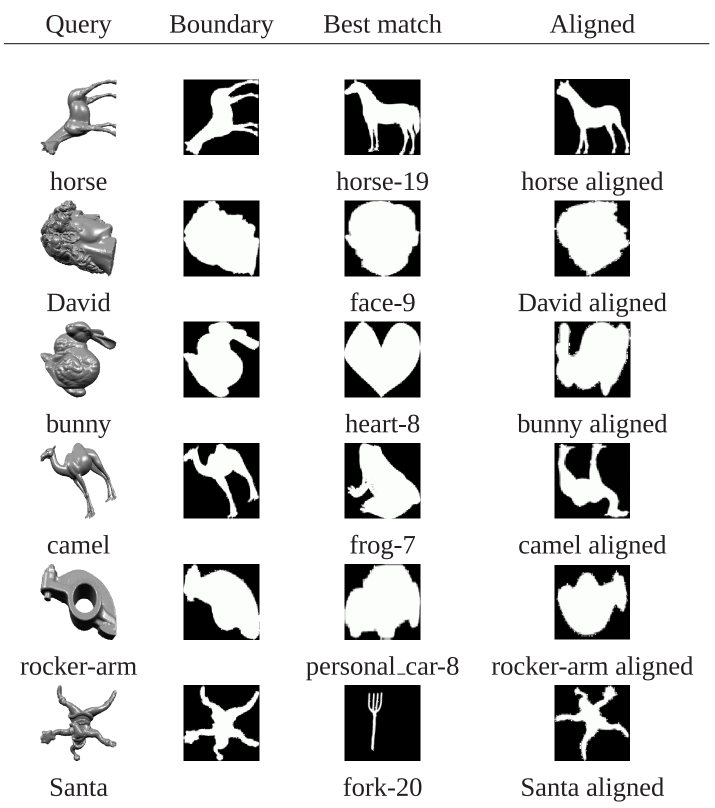


Figure 5.15: Results for common Computer Graphics models.

legs are not apart in the boundary image). Finally, our database simply does not contain suitable exemplars for many of the other models, so the results are more or less arbitrary. Santa and rocker-arm are examples of non-orientable shapes. No orientation can be regarded as correct for these shapes. The target shapes chosen are irrelevant, as any orientation imposed on these shapes will do.

5.2.4 Discussion

Typical running times to orient a query shape are about a minute. If the query shape is complex (the boundary has many concave/convex pairs), our method can take up to two minutes. The bulk of the running time is taken up by the similarity retrieval in the classification step. This can be optimised by performing an initial one-time probabilistic analysis of the database [Super04] and/or approximate nearest neighbor searches [Sebastian02]. Also, more efficient search strategies [Keogh06] can be employed. The search can further be optimised through indexing. Instead of matching a query image with the entire database, we could extract a 'prototype' for each class and match the query with the prototypes to find the candidate class. The target shape could then be found by matching the query with all members of the candidate class.

Many aspects of our method are dependent on the shape similarity method used. The most important of these is the retrieval of the similarity sorted list mentioned above. As seen in Section 5.2.3, accuracy of this list is crucial to our method. Artefacts in the similarity method lead to questionable sortings causing our method to yield implausible results. Throughout this chapter, we have used the CSS method, which had been reported earlier [Latecki00] to perform well and has in fact been included in the MPEG-7 standard. However, recent results [Veltkamp06] and our experience with the method indicate that the method can still be substantially improved.

Perhaps the most important ingredient of our method is the database used. It currently contains only 56 classes, extracted from an already compiled dataset. The limitations arising from this are visible in Figure 5.15. While the need for extending the dataset is obvious, it is not entirely clear how to do so without manual interaction. One solution could be to crawl the Internet for images. As these images are (mostly) posted by human users, we can assume them to be correctly oriented. However, automatically sorting through these images to extract the ones containing single objects against a plain background is non-trivial. Furthermore, each extracted image will need to be either classified into one of the existing classes in the database, or added as an example of a new class. This could be done using a similarity threshold.

Also, as highlighted in Section 5.2.3 and in [Podolak06], Principal Components are not robust estimators of shape orientation. The alignment step could instead make use of other methods like Iterative Closest Point (ICP) or symmetry axes [Podolak06].

One drawback our method suffers from is the absence of a systematic means of evaluation. The correctness of our result can be judged only by visual inspection by a human user, or by similarity to stored instances of the same class. But we believe that this flaw is intrinsic in our problem description, and will be resolved once we have a representative enough reference dataset.

A natural extension of our method is to work directly with 3D models. On closer inspection, this problem amounts to choosing a correct up-vector in 3D for the given shape. An example based approach as the one we presented has recently been proposed in [Fu08], where the authors form a feature vector composed of a few basic geometric properties of the 3D shape. They then train a classifier using 345 test models. The trained classifier was able to correctly re-orient 819 models with a success rate of about 90%.

We believe that the problem we tackle is a hard one, as we try, in essence, to mimic the human notion of correct orientation, which is a complex mix resulting mostly from user experience and partly from the object's shape. Such a notion is not entirely computationally replicable from the shape's geometry alone [Blanz99]. However, as has been shown in [Fu08], statistical methods that can to some extent learn this notion go a long way towards solving this problem.

5.3 Dynamic View Representation of a Shape

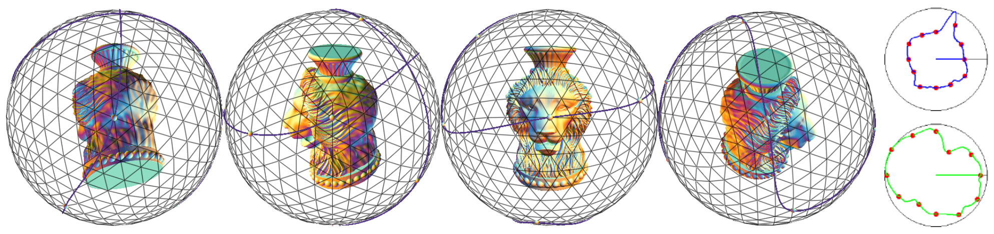


Figure 5.16: Left to right are scenes extracted in sequence from our animation of the Lion vase model. Speed (top right) and zoom (bottom right) of the camera vary along its path.

While static views limit shape exposition, interactive shape viewers, as offered by some repositories, require the entire shape to be streamed to the user's machine, which is inefficient for large, detailed shapes. We present an efficient, hybrid solution based on our best view method form Section 5.1. We compute a camera path, $\mathcal{P}$, on the view sphere as an interpolation between the shape's best views, $\mathcal{V}$, e.g. Figure 5.17. Our premise is that the shape is sufficiently described by $\mathcal{V}$ and what remains is to inform the user on the relation (transition) between these views. At each point along the path, corresponding to the view of the shape visible to the camera, the speed, $\mathcal{S}$, and zoom, $\mathcal{Z}$, of the camera are modified. The path is computed such that it fulfils the conditions listed earlier in Section 2.2.2 and reproduced below

1. *Brevity* – the animation should not be long,
2. *Information* – the animation must be maximally informative,
3. *Exploration* – the camera path should avoid fast returns to already visited viewpoints
4. *Smoothness* – the path should be smooth.

An animation recorded by a camera traveling on $\mathcal{P}$ observing $\mathcal{S}$ and $\mathcal{Z}$ can be made available on a shape repository's web page in a popular web format, e.g. animated GIF or Flash. A tunable parameter can allow a user to control the length of the animation. Most of the images in this section are stills from our results video which better illstrates our computed paths and is available at http://www.mpii.de/ wsaleem/SCCG07.

5.3.1 Computing the path

Given the set, $\mathcal{V}$, of representative viewpoints, we want the computed path, $\mathcal{P}$, to interpolate its members on the view sphere. Also, we require our animation to run continuously on a shape repository web page without any visible breaks. We formulate these requirements on $\mathcal{P}$ as follows:

- *Interpolation* – the camera path should interpolate a given set of viewpoints.
- *Looping* – the camera path should be a cycle.

In [Sokolov06b], the interpolation order is determined by first defining a distance function that favors viewpoints with higher view goodness values, computing all pairwise shortest paths between the viewpoints and then applying a TSP solver. This restricts the computed path to edges of the tessellated view sphere, and fails the Smoothness condition from Section 2.2.2.

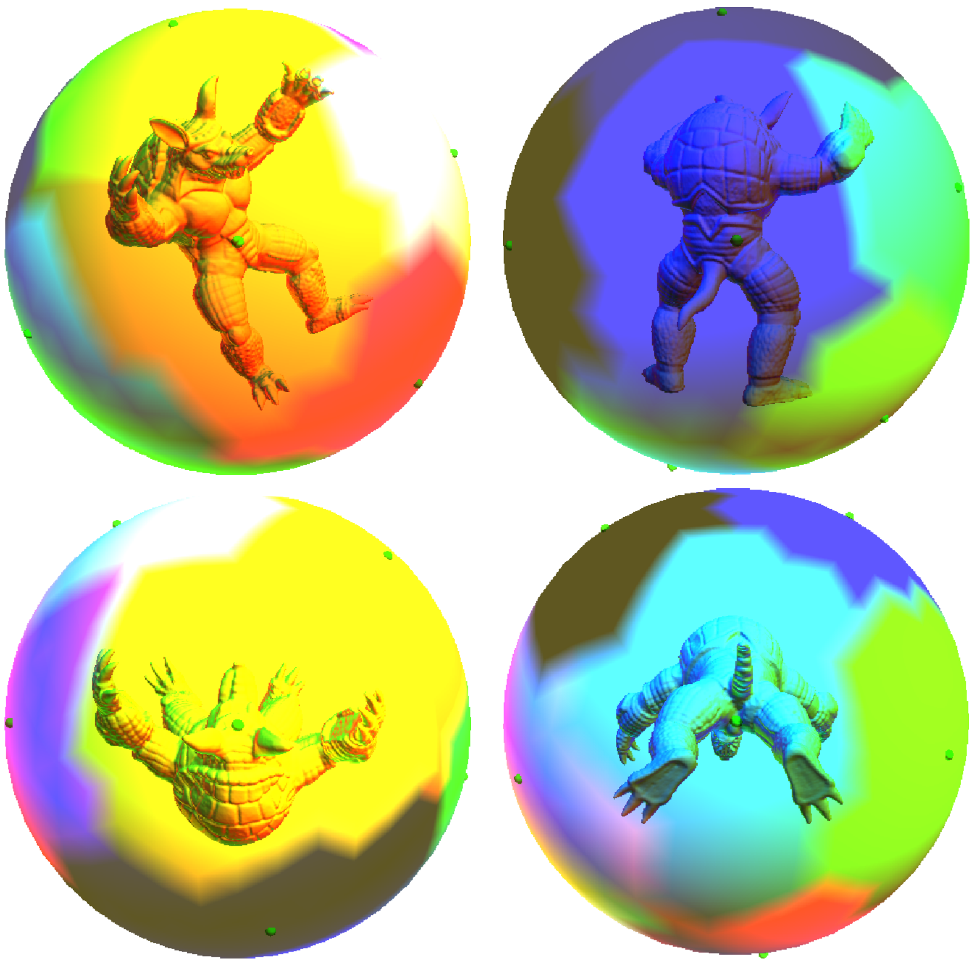


Figure 5.17: Stable view regions (colored) for the Armadillo model and their representative viewpoints (green dots). We compute 12 regions, and each of the four views above is taken from the representative viewpoint of one of the regions in the center of the image.

Looping is satisfied trivially by repeating the first point at the end when computing the ordering. As the path is a cycle, the choice of first point is arbitrary. Once the ordering is determined, we interpolate the points on the sphere using cubic spherical splines (Chapter 15 in [Watt91]). A few computed paths are shown in Figure 5.20. Our computation of viewpoint ordering is described below. A discussion on how well the resulting path satisfies the conditions listed in Section 2.2.2 is given in Section 5.3.6.

For every 3 consecutive points in a potential ordering, V_i, V_j, $V_k \in \mathcal{V}$, consider the quantity

$$\Theta_{ijk} = |d(V_i, V_j) + d(V_j, V_k) - d(V_i, V_k)|,$$

where $d(A, B)$ is the spherical distance between points A and B. Θ_{ijk} gives a measure of the turn at V_j. For V_i, V_j, V_k lying on the same great circle, $\Theta_{ijk} = 0$. The computed ordering is the one which minimizes $\sum_{V_j} \Theta_{ijk}$.

5.3.2 Up-vector consistency

Special consideration is given to the up-vectors of both the virtual camera and the models used. We use a default value for models' up-vector, $\vec{U}_m = (0, 1, 0)$, which is consistent with most scanning systems. However, this is not robust and we manually fixed $\vec{U}_m$ for one of the five models used in this chapter.

For proper orientation of the model in the animation, we keep $\vec{U}_c$, the up-vector of the camera, consistent with $\vec{U}_m$. At all times, $\vec{U}_c$ is chosen as the vector perpendicular to the viewing direction that is coplanar with $\vec{U}_m$. This gives two possible orientations for the up-vector. Indeed, there is a 'flip' in orientation at singular points, i.e. viewpoints with view direction parallel to $\vec{U}_m$. The flip in $\vec{U}_c$ is necessary to maintain correct orientation of the model, otherwise, once the camera passes through the singular point, the model appears to be upside down.

5.3.3 Computing camera speed

Camera speed, $\mathcal{S}$, determines the distance along $\mathcal{P}$ from the current viewpoint to the next one. The motivation is that the camera should quickly fly by uninteresting views. Formally, we pose the following condition on the speed.

- *Saliency respecting* – the camera should slow down when passing over visually important regions of the shape, and speed up for uninteresting views.

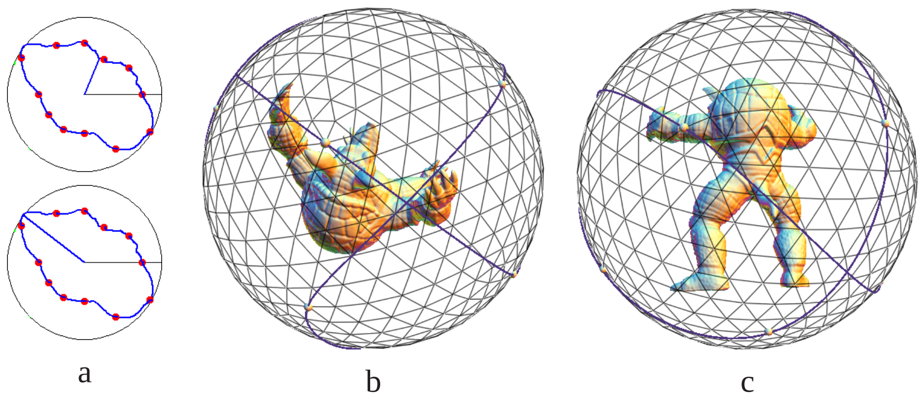


Figure 5.18: a) Minimum (top) and maximum (bottom) positions in the *speed clock* of the Armadillo animation. The 3 o' clock position represents the starting and ending point of the animation, during which the pointer moves clockwise. The length of the pointer represents the magnitude of the speed, and the dots represent the interpolated viewpoints. b) and c) show the views corresponding to the min and max positions respectively.

We use the perception based measure of view saliency, $\mathcal{VS}$, to compute visual importance. The above formulation suggests an inverse relationship, $\mathcal{S} \propto \frac{1}{\mathcal{VS}}$. Taking the view that the purpose of our animation is to aid human understanding of the shape, we use the Two-Thirds Power Law (cf [de'Sperati97] and references therein) from locomotion which relates tangential velocity, V, of free-hand movements to the radius of curvature, R, of the trajectory as follows:

$$V(t) = K \cdot \left(\frac{R(t)}{1 + \alpha \cdot R(t)} \right)^{1-\beta} \quad \alpha \geq 0,\ K \geq 0, \tag{5.1}$$

where K is a velocity gain constant, α is negligible if the trajectory does not have inflection points, and β is close to $\frac{2}{3}$ for adults. Putting in these values, we get

$$\begin{aligned} V(t) &= K \cdot (R(t))^{\frac{1}{3}} \\ &= K \cdot \left(\frac{1}{\kappa(t)} \right)^{\frac{1}{3}}, \end{aligned}$$

where $\kappa(t)$ is the curvature of the path. In our case, we want the speed to depend not on the curvature, but on $\mathcal{VS}$. Therefore we set

$$\mathcal{S}(t) = K_s \cdot \left(\frac{1}{\mathcal{VS}(t) + \gamma} \right)^{\frac{1}{3}}, \tag{5.2}$$

where γ is a constant offset to compensate for the 0 to 1 normalization of $\mathcal{VS}$. Putting $\gamma = 1$ makes $\mathcal{S}$ vary between K_s and $\frac{K_s}{\sqrt[3]{2}} \approx 0.79K_s$. A high value of K_s is thus needed for changes in speed to be discernible. Note that our use of cubic splines for interpolation technically invalidates the choice $\alpha = 0$ in Equation 5.1. However, we find that as a first approximation, the obtained results are quite satisfactory.

Equation 5.2 is in agreement with the inverse relation suggested earlier. The exponent dampens the effect of any irregularities in $\mathcal{VS}$. In Figure 5.18, we show an example of the computed speed function. As the static images only poorly convey the dynamic nature of the result, we urge the reader to view the accompanying video for a better understanding.

5.3.4 Computing camera zoom

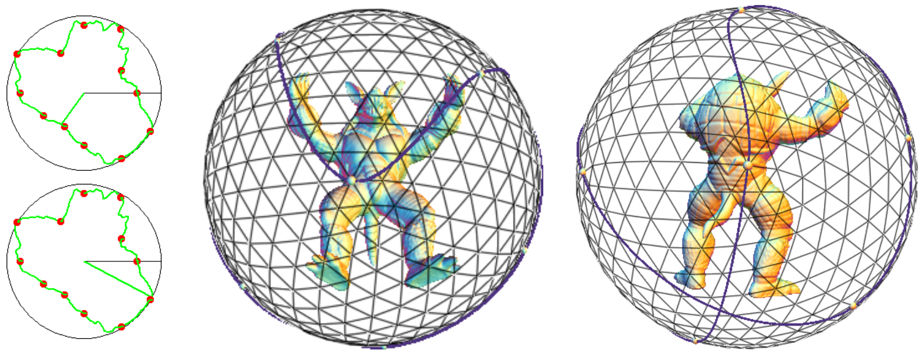


Figure 5.19: a) Minimum (top) and maximum (bottom) positions in the *zoom clock* of the Armadillo animation, where the zoom clock represents zoom in the same way as the speed clock in Figure 5.18 represents speed. b) and c) show the views corresponding to the min and max positions respectively.

The motivation behind a variable camera zoom, $\mathcal{Z}$, is the following.

- *Appropriate viewing scale* – the shape should be viewed at a scale that is in accordance with the size of the features being viewed.

In photography, zooming is achieved by changing the focal length of the camera lens. With the perspective projection of OpenGL that we use throughout this chapter, this is equivalent to varying the distance between the camera and the object, i.e. placing the camera in the corresponding position on a view sphere with a different radius; smaller radius for zooming in and larger for zoom out. Therefore, we compute $\mathcal{Z}$ by computing the corresponding radius, $\mathcal{R}$.

Recall that saliency is computed in a multi-scale way (Section 3.5.2), where a higher value at a small scale implies a small scale feature, for which the camera should zoom in (small viewing radius). Correspondingly, a high saliency value at a high scale implies a large scale feature, which requires a large viewing radius for proper inspection. We thus define, for each scale i, a corresponding viewing radius $r_i \propto k\sigma_i$, where k is a constant and σ_i is the size of the vertex neighborhood considered for saliency computation at scale i. The appropriate viewing radius, R, for a vertex, v, can then be computed as:

$$R(v) = \frac{\sum_i \mathrm{Sal}_i(v) r_i}{\sum_i \mathrm{Sal}_i(v)},$$

where $\mathrm{Sal}_i(v)$ is the saliency of v at scale i. For a view, V, an average viewing radius is computed as

$$\overline{R}(t) = \frac{\sum_{v \in V} R(v)}{n(v \in V)}. \tag{5.3}$$

After normalizing $\overline{R}$ to $[0, 1]$, we use Equation 5.1 to compute $\mathcal{R}$ as

$$\mathcal{R}(t) = K_z(\overline{R}(t)^{\frac{1}{3}} + 1),$$

where K_z corresponds to the minimum value of $\overline{R}$.

Results for the Armadillo model are shown in Figure 5.19.

5.3.5 Results

We tested our method on several models and the results are shown in Figures 5.16 and 5.20, and more comprehensibly, in the video mentioned earlier. A summary of computation times is given in Table 5.1.

In Section 5.1, the computation time for obtaining the similarity weighted view sphere was around 40 minutes, with the bottleneck being the unoptimized similarity computations between views of resolution 256×256. We reduce the similarity computation time to a couple of seconds by comparing not the binary views but their extracted boundaries at a resolution of 400×400. We also replace the area normalization step of the original method with the simpler one proposed in [Kamila05]. Note that as the images are compared using Zernike moments [Khotanzad90] which operate on the pixels of the image, we do not have to employ the full contour extraction method from Section 3.2. Just identifying pixels lying at shape boundaries and toggling the remaining shape pixels to background pixels is sufficient.

Most of the time in our current computation is spent on the mesh saliency calculation, which depends on the size of the mesh. For large models, e.g. the Buddha and Lion vase, this can be quite large. However, this is a one-time preprocessing step whose results are saved. This time also includes the computation of $R(v)$ from Section 5.3.4. The other preprocessing step – extracting $\mathcal{V}$ – uses image similarity and depends on the resolution of the views being compared. As we use the same resolution for all models, all of them take the same amount of time.

Time taken for viewpoint ordering depends on the number of viewpoints being considered, and for constant number of viewpoints (12 in our case) is independent of model size. View saliency computation requires identifying visible mesh vertices from each viewpoint. We interpolate 12 viewpoints using 12 spherical cubic splines, and sample 50 points on each spline. We thus have to compute visible vertices for 600 viewpoints. The times for view saliency computation also include computation time for $\overline{R}$ from Section 5.3.4.

	Vertices	Mesh Saliency	Extracting $\mathcal{V}$	Viewpoint ordering	View Saliency
Armadillo	172,974	761.42s	2s	15s	116.54s
Buddha	543,652	$<$1h	2s	15s	256.3s
Bunny	34,834	20.23s	2s	15s	41.0s
Elephant	20,007	12.73s	2s	15s	28.3s
Lion vase	800,002	$<$1.5h	2s	15s	576.5s

Table 5.1: Summary of computation times for a few models.

When the desired length of the animation is varied through the tunable parameter mentioned earlier, the pre-computed saliency and similarity values should be used. Extraction of stable view regions from the weighted view sphere [Karypis98] and computation of $\mathcal{V}$ then takes milliseconds. Viewpoint ordering and view saliency would have to be totally recomputed. The time for the former depends only on the chosen size of $\mathcal{V}$. In our experience, 12 viewpoints provide sufficient coverage of the object. View saliency computation, which also depends on model size, would also change markedly as varying the size of $\mathcal{V}$ varies the number of splines and hence the number of viewpoints in $\mathcal{P}$. Though this can be time consuming, we suffice with this solution as we do not aim to provide a real-time solution.

5.3.6 Discussion

We discuss how our method fares with respect to the four conditions mentioned earlier in Section 2.2.2 and reproduced below for convenience.

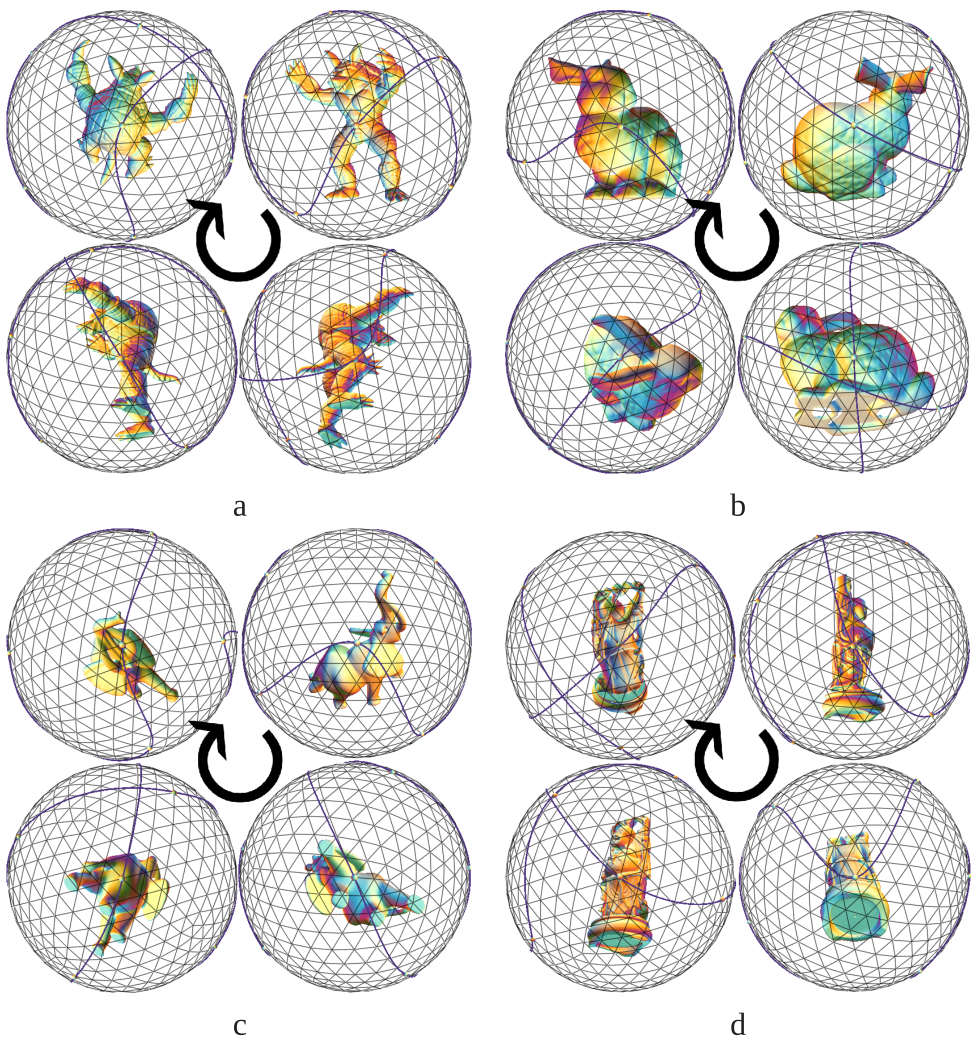


Figure 5.20: Scenes extracted in clockwise order from the computed animations of the (a) Armadillo, (b) bunny, (c) Livingstone elephant and (d) Happy Buddha models. The blue line on the view sphere denotes the computed path, $\mathcal{P}$, and the dots correspond to the interpolated viewpoints. In each case, the four images are extracted from the animation in clockwise order.

1. *Brevity* – the animation should not be long,
2. *Information* – the animation must be maximally informative,
3. *Exploration* – the camera path should avoid fast returns to already visited viewpoints
4. *Smoothness* – the path should be smooth.

Regarding Brevity, the length of the animation is tunable, as discussed in earlier sections. Once the points to be interpolated have been ordered, the shortest path consists of straight lines or geodesics. To ensure overall Smoothness, we interpolate using cubic splines. The extra length added by the smooth splines is compensated by speeding up the camera over low saliency views. Fulfilling the Smoothness condition and covering the view sphere in a cyclic path invariably lead to a self-intersecting $\mathcal{P}$. This seemingly violates the Exploration condition, but we claim that by interpolating points representative of different, non-overlapping regions of the view sphere, we have already fulfilled the condition. Lastly, we believe that by including the representative views of the shape, the path already conveys sufficient Information on the shape. Guiding it through intermediate 'good' views, as is traditionally done, will serve only to violate one of the other conditions. In addition, by allotting inspection times and viewing scales according to the visual importance of the shape features, we believe we are able to convey a lot more information about covered parts than previous methods that fly by the shape at fixed speeds and zooms.

Our objective, namely to generate a short but informative fly around a given shape model, has a long history in the movie industry. It seems promising to study the techniques used in that area. However, caution will have to be exercised as movies often have a story which serves as the context and aids the determination of camera parameters. In a shape repository framework, there are no such helping factors.

One of the biggest problems we faced while developing the method was lack of feedback. In the absence of any formal measures to judge the quality of our output, it was difficult for us to ascertain whether we were on the right track. While, in principle, we can always ask a human observer to compare two different flies of a shape, the human visual system is quite lenient and seems to automatically compensate for any missing information. Indeed, the few people in our lab whom we did ask to compare such flies were unable to give a confident answer, and showing more flies served only to confuse and disorient. In fact, recent research [Henderson07] even puts into doubt the role of computational models of visual saliency in determining a human observer's attention. Significant advances in the fields of human cognition and psychology are required (which may just go on to verify the heuristics proposed by the graphics community) before a ‘provably correct’ best fly of a given shape model can be computed. Until then, the only barometer computer graphics practitioners have towards that goal is how well they fulfil their heuristics derived through observation, common sense and application requirements.

Chapter 6

View Sphere Model

View based algorithms are quickly emerging as key tools for shape understanding and manipulation. While the most prominent application is focusing users' attentions on important shape parts [Bordoloi05, Lee05, Podolak06, Polonsky05, Takahashi05, Vázquez01], such algorithms have also been used for shape matching [Abbasi00], bas relief generation [Song07] and optimal camera path computation [Arbel99, Barral00]. The motivation behind these methods are longstanding results from human psychology [Bülthoff95, Koenderink79] which claim that humans perceive shapes as a set of 2D images from different viewpoints.

So far, the focus of such methods has been on defining good view descriptors (Section 2.2.1) with little attention paid to the construction and sampling of the view sphere. As mentioned in Section 3.1, a platonic solid, usually an icosahedron, favored for its triangle mesh structure, is taken as a base approximation of the view sphere, and is successively subdivided to obtain finer, fairly uniform samplings. The hope is that with a sufficiently large number of samples, the obtained views capture all necessary features of the shape.

In this chapter, we first present a derivation of the "optimal" view sphere of a shape, and then present a scheme to construct a smooth approximation of descriptor values over the view sphere using only a handful of samples. The samples chosen for the approximation depend on both the viewed shape and the chosen descriptor. In addition, we define a new shape based operation, *view transfer*, and demonstrate how our framework can be used to perform view transfer and other common view based shape operations.

For descriptors whose computation is known to be expensive, e.g. Zernike mo-

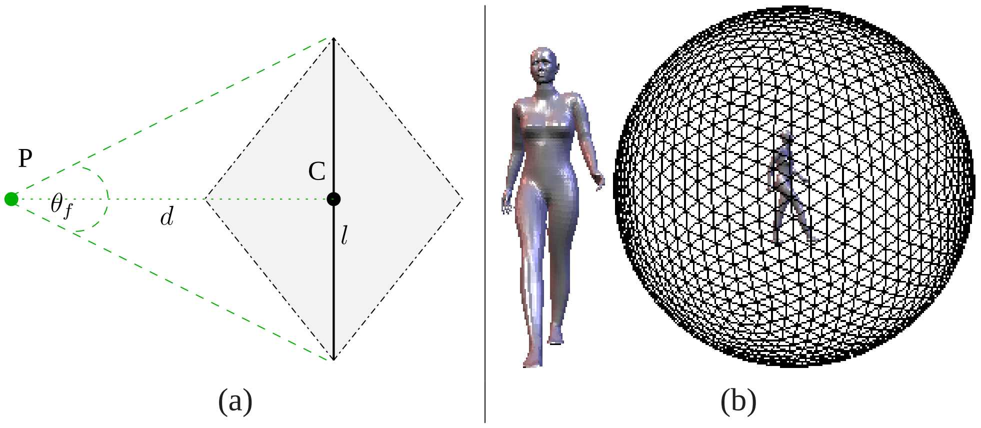


Figure 6.1: (a) A virtual camera at ***P*** with frustum θ_f viewing the center, ***C***, of a bounding box diagonal of length l from a distance d. (b) A model in its view sphere.

ments [Khotanzad90], computation for a few adaptively sampled viewpoints is far more efficient than for a dense sampling of the view sphere. In contrast to the traditional, discrete sampling, the continuous approximation we build from the adaptively sampled values can be trivially queried for descriptor values at any given point on the view sphere. The approximation can also be used for typical (representative views) and novel (view transfer) view based operations.

Our scheme is general and applies to any view descriptor. For demonstration purposes, we test our scheme on three descriptors, namely Zernike moments [Khotanzad90], viewpoint entropy [Vázquez01] and view saliency [Lee05]. All three have been presented earlier in Chapter 3.

6.1 Notation

A view descriptor, $\mathcal{D}$, for a shape model, $\mathcal{M}$, describes a function, $f_{\mathcal{D}} : \mathbb{S}^2 \to \mathbb{R}$, from the view sphere, $\mathbb{S}^2$, to $\mathbb{R}$. $\mathbb{S}^2$ is represented by surface samples, $S = \{\mathbf{s_i}\}$, and the representation of $\mathcal{M}$ w.r.t. $\mathcal{D}$ is the set of values $\{f_{\mathcal{D}}(\mathbf{s_i})\}$. S typically represents a dense sampling of $\mathbb{S}^2$, e.g. four subdivisions of an icosahedron (2562 view points) followed by re-projection to $\mathbb{S}^2$.

We iteratively approximate $f_{\mathcal{D}}$ by building up a set of *interpolation centers*, $\mathcal{C}$, and a corresponding interpolation function, f. In each iteration $k, k \geq 1$, we compute $\mathcal{C}^k$ and f^k which are used to update $\mathcal{C}$ and f at the end of the iteration. Note that the superscript indicates iteration number and not exponentiation.

For a given vector, $\mathbf{p}$, we indicate by $\|\mathbf{p}\|_n$ the n-th norm of $\mathbf{p}$, i.e. $\|\mathbf{p}\|_n = (\sum_i \mathbf{p_i}^n)^{\frac{1}{n}}$ where $\mathbf{p_i}$ is the i-th component of $\mathbf{p}$. For a set, S, $\|S\|$ denotes its size and for a complex number, z, $\|z\|$ or $|z|$ denotes the magnitude.

6.2 Optimal View Sphere

The size of $\mathbb{S}^2$ relative to $\mathcal{M}$ determines the shape to background ratio in obtained views. Clearly, we want to maximize this ratio while still viewing the entire shape. Typically, the radius, r, of $\mathbb{S}^2$ is taken to be some factor, F, of the length, l, of the bounding box diagonal of $\mathcal{M}$, $r = Fl$. In the best view literature we reviewed, no mention is made of the choice of F leading us to believe that it is chosen heuristically. We present here a formal derivation for an optimal value of F that meets the above constraints.

Shape views are captured by cameras placed on the surface of $\mathbb{S}^2$ and pointing at the center, **C**, of the bounding box of $\mathcal{M}$. (**C** is also the center of $\mathbb{S}^2$.) Thus, in order to fully view $\mathcal{M}$, the distance of the camera from **C** should be no smaller than d such that at distance d and with a viewing frustum of θ_f, the camera just fully views the bounding box diagonal. This is illustrated in Figure 6.1. We thus obtain $d = \frac{l}{2}\cot\frac{\theta_f}{2}$. Most mesh viewing softwares, including the one used throughout this thesis, model cameras with a default value of $\theta_f = \pi/4$. Thus,

$$d = \frac{l}{2\left(\sqrt{2}-1\right)} \approx 1.21l.$$

Setting $r = d$, i.e. $F = 1.21$ causes extremities of $\mathcal{M}$ to lie at view image boundaries. Therefore, for our experiments, we set $F = 1.3$ for all shapes.

6.3 Approximating $f_\mathcal{D}$ on $\mathbb{S}^2$

As mentioned in Section 6.1, one of the key components of our approximation scheme is a set, $\mathcal{C} = \{\mathbf{c_i}\} \subset \mathcal{S}$, of *interpolation centers*. These have the property that our final approximation, f, interpolates $f_\mathcal{D}$ at them, i.e. at all times

$$f(\mathbf{c_i}) = f_\mathcal{D}(\mathbf{c_i}), \qquad \mathbf{c_i} \in \mathcal{C}. \tag{6.1}$$

Furthermore, at all other $\mathbf{s_i} \in \mathcal{S}$, f approximates $f_\mathcal{D}$ up to an error η, i.e.

$$\mathbf{E}(\mathbf{s_i}) < \eta, \qquad \mathbf{s_i} \in \mathcal{S}, \mathbf{s_i} \notin \mathcal{C}$$

```
    $k \leftarrow 0; \mathcal{C} \leftarrow \emptyset; \mathcal{D} \leftarrow \emptyset$
    $f^k \leftarrow 1$
    $\mathcal{C}^k \leftarrow$ vertices of icosahedron on $\mathbb{S}^2$
    $\mathcal{D}^k \leftarrow \{f_{\mathcal{D}}(\mathbf{c_i}) \mid \mathbf{c_i} \in \mathcal{C}^k\}$   // actual descriptor computation
5:  $\mathcal{C} \leftarrow \mathcal{C} \bigcup \mathcal{C}^k; \mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}^k$
    repeat
      $k \leftarrow k+1; \mathcal{C}^k \leftarrow \emptyset; \mathcal{D}^k \leftarrow \emptyset$
      $f^k \leftarrow$ build_function $(\mathcal{C}, \mathcal{D})$  // interpolate at current centers
      $S' \leftarrow \mathcal{S} - \mathcal{C} -$ nghbr $(\mathcal{C}^{k-1})$  // $\mathbb{S}^2$ samples minus current centers, old neighbors
10:   while $((\|\mathcal{C}^k\| < N_v) \wedge (\|S'\| > 0))$ do
        $\mathbf{s} \leftarrow$ sample in $S'$ with highest error
        $\mathcal{C}^k \leftarrow \mathcal{C}^k \bigcup \{\mathbf{s}\}$  // new center
        $S' \leftarrow S' -$ nghbrs $(\mathbf{s}) - \{\mathbf{s}\}$  // discard center and neighbors
      end while
15:   $\mathbf{E}^k \leftarrow$ errors $(\mathcal{C}^k, f^k, f^{k-1})$  // errors for candidate centers
      $\mathcal{D}^k \leftarrow \{f_{\mathcal{D}}(\mathbf{c_i}) \mid \mathbf{c_i} \in \mathcal{C}^k\}$   // actual descriptor computation
      $\mathcal{C} \leftarrow \mathcal{C} \bigcup \mathcal{C}^k; \mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}^k$
    until $(\neg$ stop_condition $(\mathcal{C}, \mathcal{S}, \mathbf{E}^k))$
    $\mathcal{C} \leftarrow \mathcal{C} - \mathcal{C}^k$  // remove last, unused batch of new centers
```

Figure 6.2: Compute $\mathcal{C}$, set of interpolation centers.

where $\mathbf{E}(\mathbf{s_i})$ is a measure of error at $\mathbf{s_i}$.

Both $\mathcal{C}$ and f are updated iteratively. In each iteration $k, k \geq 1$, the pairs $(\mathbf{c_i}, f_{\mathcal{D}}(\mathbf{c_i}))$ for the current $\mathcal{C}$ are used to construct an approximation, f^k. The error, $\mathbf{E}^k(\mathbf{s_i})$, at each $\mathbf{s_i}$ is measured in terms of the variation between $f^k(\mathbf{s_i})$ and $f^{k-1}(\mathbf{s_i})$. The $\mathbf{s_i}$ with the highest error are added to the set, $\mathcal{C}^k$, of new centers. At the end of the iteration, $\mathcal{C}$ and f are updated as $\mathcal{C} \leftarrow \mathcal{C} \bigcup \mathcal{C}^k, f \leftarrow f^k$. Iteration stops when a stopping condition is met. Centers added in the last iteration have not yet been used for interpolation, and are therefore removed from $\mathcal{C}$. For initialization, $\mathcal{C}$ is set to the vertices of an icosahedron (12 centers) centered around the shape and with radius as discussed in Section 6.2, and $f \leftarrow f^0 \leftarrow 1$. Figure 6.2 presents the pseudo-code for the entire procedure.

Note that in our framework, the views corresponding to $\mathbf{c_i} \in \mathcal{C}$ form the final representation of $\mathcal{M}$ with respect to $\mathcal{D}$. This is different from other approaches that also represent $\mathcal{M}$ with a few views. These methods select their views based on visibility of faces [Jaubert06, Roberts98] or similarity among views [Abbasi00, Yamauchi06] of $\mathcal{M}$, whereas the views chosen in our approach are determined by the approximation of $f_\mathcal{D}$ by f.

Constructing f^k Given pairs, $(\mathbf{c_i}, f_\mathcal{D}(\mathbf{c_i}))$, f^k is calculated using implicit interpolation as follows. Using the Gaussian kernel $\varphi(t) := \exp(-t)$, $t \in \mathbb{R}^+$, f^k is defined as [Poggio90]

$$f^k(\mathbf{p}) := \sum_{\mathbf{c_i} \in \mathcal{C}} \alpha_i^k \varphi_i(\mathbf{p}), \quad \mathbf{p} := (x, y, z), \tag{6.2}$$

that is, a linear combination of the radial basis functions $\varphi_i(\mathbf{p}) := \varphi(\|\mathbf{p} - \mathbf{c_i}\|_2)$, centered at $\mathbf{c_i} \in \mathcal{C}$. Then, the coefficients in Equation 6.2 that uniquely satisfy Equation 6.1 are the solutions of the $r \times r$ square linear system $A^k\alpha = b$, where the entries of the matrix A^k are $a_{ij}^k := \varphi(\|\mathbf{c_i} - \mathbf{c_j}\|_2)$, $\alpha := (\alpha_i^k)_{i=1}^r$, and the constant term is $\mathbf{b} := [f_\mathcal{D}(\mathbf{c}_1), \ldots, f_\mathcal{D}(\mathbf{c}_r)]^T$. Here, r indicates the size of $\mathcal{C}^k$.

The function used in (6.1) involves the Euclidean distance between points of $\mathbb{S}^2$; an alternative is to replace it with the geodesic distance. In this case, it is enough to substitute $\|\mathbf{n} - \mathbf{m}\|_2$ with $\langle \mathbf{n}, \mathbf{m} \rangle$, $\mathbf{n}, \mathbf{m} \in \mathbb{S}^2$.

Measuring Error The approximation error at a sample, $\mathbf{s_i} \in \mathcal{S}$, in iteration k is measured as

$$\mathbf{E}^k(\mathbf{s_i}) = \frac{\|f^k(\mathbf{s_i}) - f^{k-1}(\mathbf{s_i})\|}{\|f^k\|_\infty}.$$

This leads to two artefacts. Firstly, samples chosen as centers in the previous iteration and their neighbors exhibit large errors in the current iteration. These errors do not signify approximation inaccuracies, rather the correction of f at these samples. The second artefact arises from the fact that approximation errors are usually clustered, i.e. samples with high error values often lie close to each other. Choosing any one of these samples as an interpolation center in this iteration will bring down the error in this neighborhood in the next iteration. To deal with these artefacts, two sets of vertices are barred from selection as new centers. The first set, corresponding to the first artefact, consists of interpolation centers from the previous iteration and their neighbors, and the second contains neighbors of current centers. In each iteration, we choose at most N_v new centers, where N_v is previously specified, i.e. $\|\mathcal{C}^k\| \leq N_v$ for all k. This is also indicated in Figure 6.2. The *iteration error* in iteration k is then measured as

$$\mathbf{E}_k = \|\mathbf{E}^k(\mathbf{c_i})\|_\infty, \qquad \mathbf{c_i} \in \mathcal{C}^k.$$

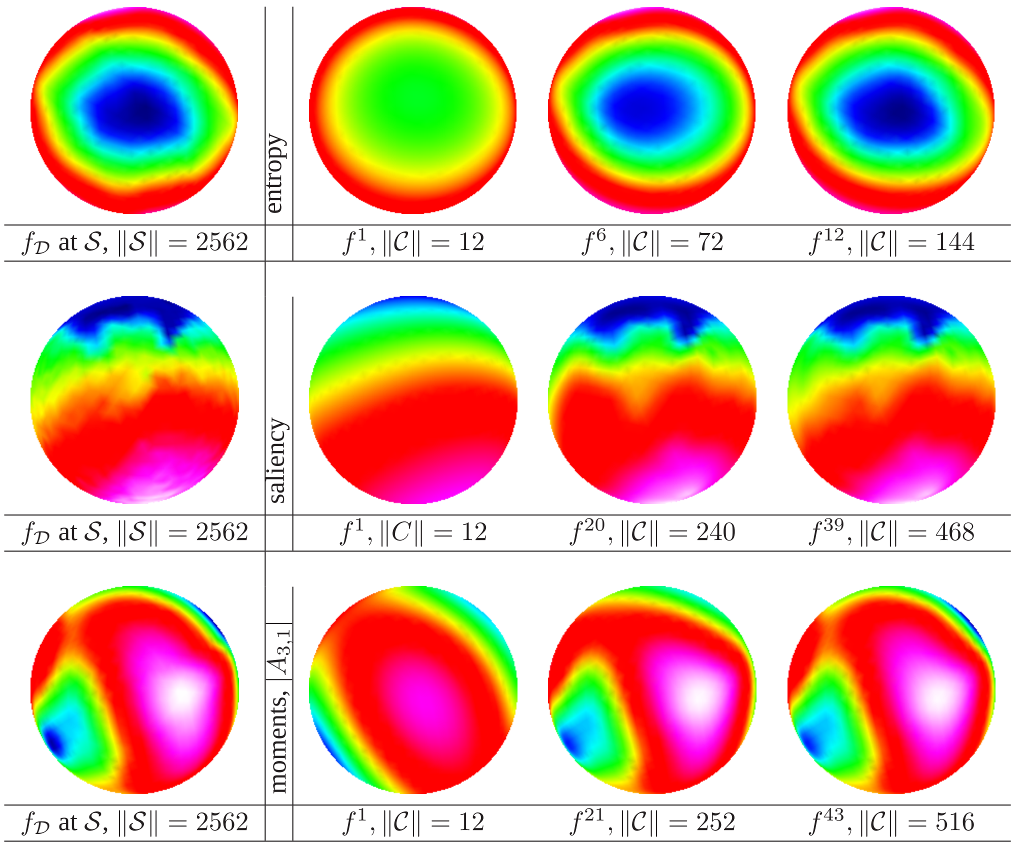


Figure 6.3: Descriptor approximation for the model from Figure 6.1b) for increasing k (left to right) and different descriptors.

Stopping Condition The algorithm terminates at the iteration $k = k'$ when, for the first time, the iteration error for two consecutive iterations falls below a threshold, η, i.e. $\mathbf{E}_{k'} \leq \eta$ and $\mathbf{E}_{k'-1} \leq \eta$. The final error for each sample is then set to its error in the last iteration, $\mathbf{E}(\mathbf{s_i}) = \mathbf{E}^{k'}(\mathbf{s_i})$ for all $\mathbf{s_i}$. As our approximation is smooth, it cannot approximate well descriptors whose corresponding $f_{\mathcal{D}}$ are non-smooth over $\mathbb{S}^2$. In such cases, if the error threshold is too low, iteration does not terminate quickly. Therefore, we add a second condition, namely that we terminate iteration also if $\|\mathcal{C}\| \geq \frac{\|\mathcal{S}\|}{3} + N_v$. The additional N_v term is to account for $\mathcal{C}^{k'}$, the last batch of candidate centers that will not be used for interpolation and are therefore discarded.

Results for an experiment are shown in Figure 6.3. We set $\eta = 1\%$ and $N_v = 12$. The leftmost view spheres represent the *ground truth* values at a quadruply subdivided icosahedron. The remaining view spheres in each row are shaded using

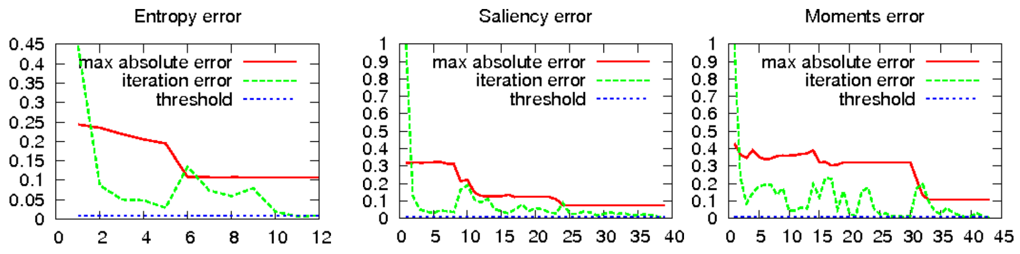


Figure 6.4: Approximation errors at sample points.

values approximated from $\mathcal{C}$ up to that iteration. Figure 6.4 shows approximation errors against iterations for the experiment. Absolute error is measured as the difference between ground truth and approximated values. Note that the ground truth is shown here only for purposes of exposition. Our algorithm does not rely on its computation.

6.4 Equivalent Views and View Likelihood

Views of $\mathcal{M}$ from viewpoints $\{\mathbf{p_1}, \ldots, \mathbf{p_n}\} \in \mathbb{S}^2$ are said to be *equivalent* [Bordoloi05, Weinshall97] w.r.t. a given $\mathcal{D}$ if their values $f_{\mathcal{D}}$differ by less than some threshold, ϵ, i.e.

$$|f_{\mathcal{D}}(\mathbf{p_i}) - f_{\mathcal{D}}(\mathbf{p_j})| < \epsilon \quad i, j \in 1, \ldots, n, i \neq j.$$

The *likelihood* of a given view is then measured in terms of the probability of existence of other views that are equivalent to it [Weinshall97].

Using our approximation, f, we are able to compute equivalent views and view likelihood by constructing iso-lines of f. For the given viewpoint, $\mathbf{p}$, all viewpoints lying on the iso-line $f = f(\mathbf{p})$ yield views equivalent to the one from $\mathbf{p}$. The likelihood of the view is given by the length of the iso-line.

To construct the iso-line, we collect all edges $\mathbf{e_i} = (\mathbf{s_j}, \mathbf{s_k})$ in $\mathcal{S}$, such that either

$$\begin{cases} f(\mathbf{s_j}) = f(\mathbf{p}), & \text{or} \\ f(\mathbf{s_k}) = f(\mathbf{p}), & \text{or} \\ (f(\mathbf{s_j}) - f(\mathbf{p})) * (f(\mathbf{s_k}) - f(\mathbf{p})) < 0 \end{cases}$$

, and add necessary points to a set, $\mathcal{I}$, of viewpoints. In the first (resp. second) case, $\mathbf{s_j}$ (resp. $\mathbf{s_k}$) is added to $\mathcal{I}$. In the third case, a linearly interpolated point, $\mathbf{q}$, on $\mathbf{e_i}$ is added to $\mathcal{I}$, i.e.

$$\mathbf{q} = \mathbf{s_j} + \frac{f(\mathbf{p}) - f(\mathbf{s_j})}{f(\mathbf{s_k}) - f(\mathbf{s_j})}(\mathbf{s_k} - \mathbf{s_j}).$$

Once $\mathcal{I}$ is populated, points that lie on the same triangle in $\mathcal{S}$ are connected by an edge. The set of all such edges forms the iso-line. The length of the iso-line can be taken as the sum of the great circle or Euclidean distances between points connected by edges. For convenience, the number of edges in the set may also be used. The cost of this operation is linear in the number of edges in $\mathcal{S}$.

6.5 Representative views

The three dimensional shape of an object is conveyed through its two dimensional views, whether on paper, screen or the retina. To this effect, some views are better than others, leading to the concepts of *best* and *worst* views. Views of a shape from neighboring viewpoints are typically similar. However, most nontrivial shapes have overall distinctly different views. *Unstable* views represent the transition from one view of the shape to another. Together, the best, worst and unstable views of the shape constitute its representative views.

Our framework can be used to compute representative views of $\mathcal{M}$ with respect to a given $\mathcal{D}$ by exploiting the critical points of f. Similar approaches are typically employed by best view techniques [Lee05, Vázquez01]. The critical points of $f$are first approximated according to the values of $f$on the neighborhoods of sphere samples, $\mathrm{s_i} \in \mathcal{S}$. More precisely, let $N(\mathrm{s_i}) := \{\mathrm{s_j} : (\mathrm{s_i}, \mathrm{s_j}) \text{ edge}\}$ be the 1-star of the sample $\mathrm{s_i}$, i.e. the set of vertices incident to $\mathrm{s_i}$ in the meshing of $\mathbb{S}^2$. Then, the view from $\mathrm{s_i}$ is the *local best* (resp. *worst*) *view* if $f(\mathrm{s_i}) \geq f(\mathrm{s_j})$ (resp. $f(\mathrm{s_i}) \leq f(\mathrm{s_j})$) , $\mathrm{s_j} \in N(\mathrm{s_i})$. Indicating with $N^\star(\mathrm{s_i})$ the anticlockwise (or clockwise) reordering of $N(\mathrm{s_i})$, the view from $\mathrm{s_i}$ is *unstable* if the number of sign changes, $(f(\mathrm{s_j}) - f(\mathrm{s_i}))$, $\mathrm{s_j} \in N^\star(\mathrm{s_i})$, is $2 + 2m$, $m \geq 1$. In this case, m closed iso-curves of f intersect at $\mathrm{s_i}$.

Note that computing these views takes linear time in the size of $\mathcal{S}$and does not require setting thresholds or other parameters. The global best (resp. worst) view is given as the maximum (resp. minimum) of the local best (resp. worst) views. Once the representative views have been approximated using the values of $f$on $\mathcal{S}$, we refine them by computing the critical points of f, thus making the computation of representative views independent of the sampling of $\mathbb{S}^2$.

If we are interested solely in the best (resp. worst) view, we need only find the maxima (resp. minima) of f. Choosing an approximated extremum, $\mathrm{s_0}$, we use the Nelder-Mead simplex search [Jr.87, Nelder90] with starting point $\mathrm{s_0}$. The algorithm performs a direct search method and does not require gradients or other derivative information. At each step of the search, a new point in or near the

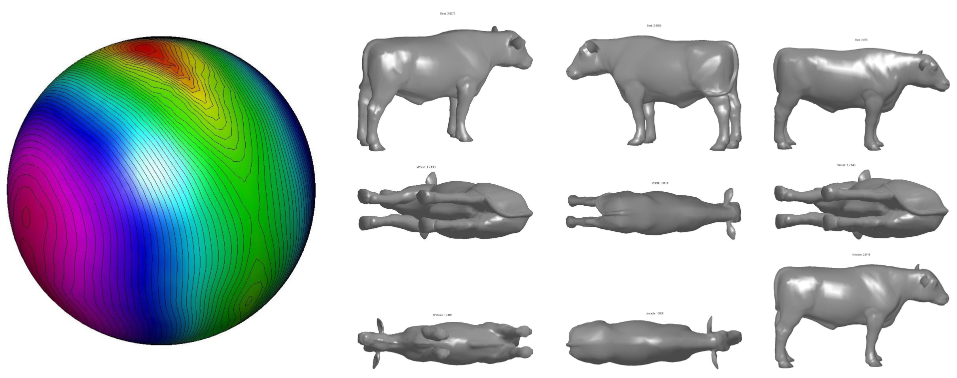


Figure 6.5: (**left**) View sphere with viewpoint entropy values and iso-lines. (**right**) From top to bottom, some of the best, worst and unstable views.

current simplex is generated. The function value at the new point is compared with the function values at the vertices of the simplex and one of the vertices is replaced by the new point, giving a new simplex. This step is repeated until the diameter of the simplex is less than a specified tolerance.

For the general case, we note that the gradient $\nabla f := (\partial_x f, \partial_y f, \partial_z f)$ of f is given by

$$\begin{cases} \partial_x f(\mathbf{n}) = -\sum_{i\in\mathcal{V}} \alpha_i \frac{x-n_i^x}{\|\mathbf{n}-\mathbf{n}_i\|_2} \exp(\|\mathbf{n}-\mathbf{n}_i\|_2), \\ \mathbf{n} := (x,y,z),\ \mathbf{n}_i := (n_i^x, n_i^y, n_i^z); \end{cases}$$

where $\partial_y f$ and $\partial_z f$ are achieved from the previous expression by replacing x with y and z. Then, we compute the critical points of f by imposing that $\nabla f = 0$. Choosing a representative view $\mathbf{n}_0$, which is an approximation of a minium, maximum, or saddle of f, we solve the implicit equation $\nabla f = 0$ in a neighborhood of $\mathbf{n}_0$ by using an iterative method with starting point $\mathbf{n}_0$.

Our tests show that if $\mathcal{S}$ is sufficiently dense, the locations of approximated and actual viewpoints on $\mathbb{S}^2$ corresponding to the representative views are almost coincident. This is due to the fact that the approximation f is continuous and its discrete critical points converge to the continuous ones by increasing the number of samples on $\mathbb{S}^2$.

6.6 View Transfer

Given two similar shape models, say a bicycle and a motorbike, and a particular view of one of them, e.g. front view of the bicycle, we would like to impose the same view on the motorbike, i.e. we would like to automatically compute a front

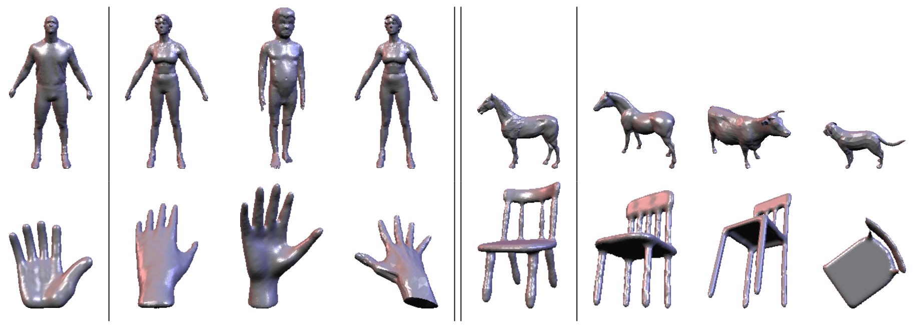


Figure 6.6: View transfer from reference (**left**) to three target (**right**) models using viewpoint entropy, view saliency and Zernike moments up to order 10 as descriptors.

view of the motorbike. In doing so, we say that the view of the bicycle model has been *transferred* to the motorbike model. Moreover, in this example, we refer to the front view of the bicycle as the *reference view* and to the bicycle and motorbike models as resp. the *reference* and *target* models.

A single reference view, $\mathbf{v}_{\text{ref}}$, may correspond to more than one transferred views. We notice however that, as the reference, $\mathcal{M}_{\text{ref}}$, and target, $\mathcal{M}_{\text{tar}}$, models are similar, descriptor values for $\mathbf{v}_{\text{ref}}$ and the transferred views must be alike. With this observation, the problem of transferring views becomes similar to the one of finding equivalent views in Section 6.4, i.e. we want to find those views of $\mathcal{M}_{\text{tar}}$ for which $f_{\mathcal{D}}$ is equal to some given value. In this case, the value that $f_{\mathcal{D}}$ should equal is $f_{\mathcal{D}}(\mathbf{v}_{\text{ref}})$. We therefore go about this problem in the same manner.

As a single descriptor captures only some aspects of the shape, a more accurate view transfer is achieved when several view descriptors are combined. Such a transfer ensures that a greater number of shape features of $\mathcal{M}_{\text{tar}}$ in the transferred view match those of $\mathcal{M}_{\text{ref}}$ in $\mathbf{v}_{\text{ref}}$.

Single Descriptor For a given descriptor, $\mathcal{D}$, and reference view, $\mathbf{v}_{\text{ref}}$, we compute the descriptor value, $f_{\mathcal{D}}(\mathbf{v}_{\text{ref}})$, and construct an approximation, f, w.r.t. $\mathcal{D}$ for the target model, $\mathcal{M}_{\text{tar}}$. We then construct iso-lines on the view sphere of $\mathcal{M}_{\text{tar}}$ corresponding to $f = f_{\mathcal{D}}(\mathbf{v}_{\text{ref}})$ as described in Section 6.4. Views corresponding to the points on the iso-line are then the transferred views.

Multiple Descriptors Using several descriptors, $\mathcal{D}_i, i = 1 \dots n$, for performing the transfer, for each $\mathcal{D}_i$, we build the corresponding approximation, f^i, for $\mathcal{M}_{\text{tar}}$ and then construct the iso-lines $f^i = f_{\mathcal{D}}(\mathbf{v}_{\text{ref}})$ on the view sphere of $\mathcal{M}_{\text{tar}}$.

Each iso-line represents transferred views w.r.t. to a single descriptor. The transferred view w.r.t. all descriptors thus lies at the point where all iso-lines intersect. In practice though, there is rarely a single point where all iso-lines intersect. Therefore, we assign to each point, $\mathbf{p}$, on the view sphere of $\mathcal{M}_{\text{tar}}$ a quantity computed as

$$g(\mathbf{p}) = \sum_{i=1}^{n} \left| f^i(\mathbf{P}\mathbf{p}) - \mathbf{f}_{\mathcal{D}_\mathrm{i}}(\mathbf{v}_{\text{ref}}) \right| .$$

The transferred view then corresponds to the point on the view sphere that minimizes g. Finding the minimizer is similar to the problem of finding the worst view in Section 6.5, so we adopt the same approach as in that section.

A few examples of transferred views are given in Figure 6.6 where we use three descriptors for the transfer. We notice that with the descriptors used, the method chooses the correct view of a left hand even when the reference model is a right hand. However, it can not distinguish between different orientations – in its transferred view, the cow is facing the other way as the horse in the reference view. This can be attributed to the descriptors used. The animal models used are roughly symmetric on either side, and two of the descriptors used, viewpoint entropy and view saliency, are insensitive to reflection. Thus they cannot distinguish between reflected views. Results for the chair models are progressively worse, especially the last chair. This is because this chair model is quite different from the reference model. It has long thin and curvy legs with only 2 beams in its back support. Less discriminative descriptors would have given more acceptable results for the model.

6.7 Discussion and future work

In this chapter, we have presented a technique to build a continuous approximation of a given view descriptor, and have shown how this approximation can be exploited to answer typical view based queries, in particular, to transfer views between similar shapes. Unlike traditional view finding schemes, we sample the view sphere adaptively based on both the model and the descriptor. This allows us to avoid descriptor computation in “uninteresting” parts of the view sphere, and leads to an economical view-based representation of the shape, namely our final set of interpolation centers.

6.7.1 Shape Comparison

One important view based shape operation that we have yet to address is shape comparison. Using our model, we envision the following approach to the problem of finding a matching cost between two shape models.

After rescaling the models, $\mathcal{M}_1$ and $\mathcal{M}_2$, to be compared, their view spheres, $\mathbb{S}^2(\mathcal{M}_1)$ and $\mathbb{S}^2(\mathcal{M}_2)$, are aligned and a correspondence, c, is established between a set, $\mathcal{S}_1$, of points on $\mathbb{S}^2(\mathcal{M}_1)$ and another set, $\mathcal{S}_2$, of points on $\mathbb{S}^2(\mathcal{M}_2)$. Two points pairs are sufficient to describe a unique alignment. Then, for a given descriptor, $\mathcal{D}$, we construct the approximations, f_1 and f_2, for both shapes. The cost of the alignment can then be described as

$$\sum_{\mathbf{p}\in\mathcal{S}_1} \|f_1(\mathbf{p}) - f_2(c(\mathbf{p}))\|.$$

A naive approach would be to try many alignments and then choose as the matching cost of the two models the smallest cost of the two alignments, i.e.

$$\text{cost}(\mathcal{M}_1, \mathcal{M}_2) = \min_i \text{cost}(\mathcal{A}_i),$$

where each $\mathcal{A}_i$ is an alignment. This is obviously quite expensive. A solution is to initially find a suitable alignment of the view spheres. This could be done by the aligning the PCA or symmetry axes [Podolak06] of the two shapes, or by pre-computation of a canonical orientation of the viewed shapes [Fu08].

Chapter 7

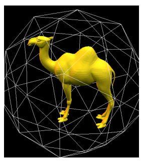
Shape Complexity

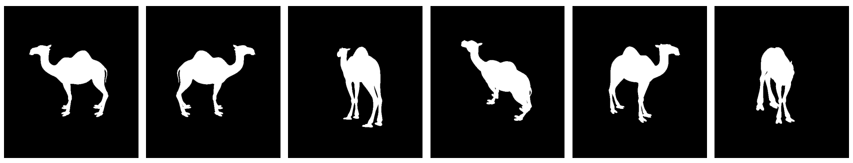
As even small amounts of noise can significantly perturb the curvature distribution of a shape, previous curvature based approaches to shape complexity [Page03, Sukumar06] turn out to be sensitive to noise–small amounts of noise on a shape's surface will increase computed complexity even though the shape itself will not have changed much.

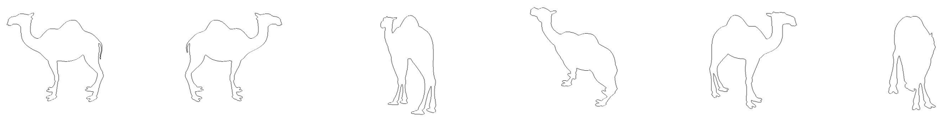
Our approach to computing shape complexity is based on evidence from human perception [Koenderink79] that humans construct internal representations of 3D shapes as 2D images in certain relation to each other. We thus claim that a complex shape is one with distinct, dissimilar views, while a simple shape has smaller variation in its views. In fact, the canonical simplest shape, the sphere, has exactly the same view, up to scale, from all viewpoints [Cutzu97].

We compute the complexity of a shape as follows. We obtain N views of it (Section 3.1), extract their boundary contours (Section 3.2) and obtain all pairwise similarity distances using the contour-to-centroid method (Section 3.3.3). This gives a $N \times N$ similarity matrix, $\mathcal{S}$, to which we apply SSA (Section 3.4) to obtain a 2D plot with N points. Each of the points represents a view and pairwise distances between points in the plot correspond to the similarity distance between the views. Thus, a simple shape will result in a tightly clustered SSA plot whereas the plot for a complex shape will contain highly dispersed points. We measure the complexity of the shape as the amount of dispersion of points in its SSA plot obtained in the above manner. Figure 7.1 gives a pictorial overview of the approach. As our methods considers shape views, it is insensitive to small variations in shape.

(a) shape to be analyzed

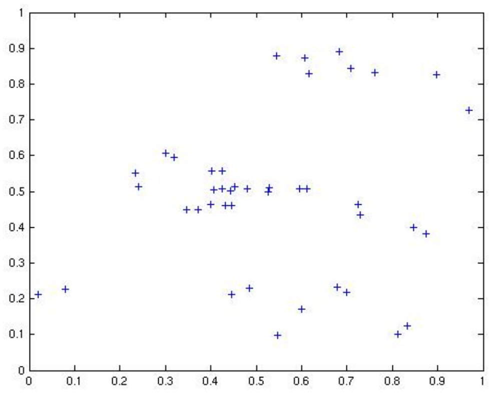

(b) shape with view sphere


(c) shape views obtained from its view sphere


(d) extracted boundary silhouettes from the views

$$\mathbf{S} = \begin{pmatrix} 0 & 0.130 & 0.246 & 0.236 & \dots \\ 0.130 & 0 & 0.222 & 0.220 & \dots \\ 0.246 & 0.222 & 0 & 0.123 & \dots \\ 0.234 & 0.220 & 0.123 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

(e) similarity distance matrix of boundary silhouettes



(f) 2D SSA plot corresponding to $\mathbf{S}$

Figure 7.1: A pictorial overview of our shape complexity method. a) shows the original 3D shape. b) shows the shape with its view sphere around it. c) shows several sampled views taken from the view sphere. d) shows the extracted silhouettes from above views. e) shows the symmetric matrix of similarity distances between the silhouettes. f) is the SSA plot calculated from the above similarity distance matrix.

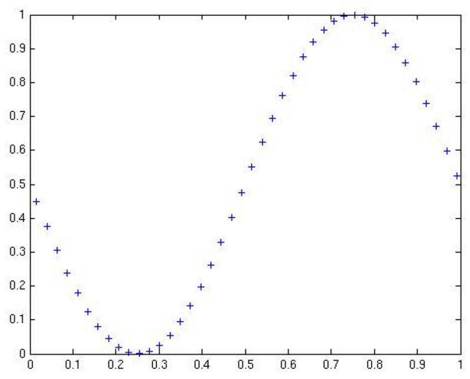


Figure 7.2: Starting configuration of points in the SSA plot.

7.1 Applying SSA to the Similarity Matrix

The original SSA method places no requirements on the initial point configuration in the SSA plot. However, the choice of starting points affects our final complexity result. A random selection of points may lead to slightly varying complexity values for the same shape each time it is computed. Therefore, we fix the initial point configuration as follows. We consider a sinusoidal function with x and y rescaled to the interval $[0, 1]$ and sample the N initial points, $\{\mathbf{p}_{i,0}), i \in \{1, \ldots, N\}\}$, uniformly on it along the x axis, i.e. the coordinates of $\mathbf{p}_{i,0}$, are given by

$$x_i = \frac{i-1}{N-1}, \quad y_i = \frac{1}{2}(1 - \sin 2\pi x_i)$$

As we take the same number of views for each shape, the initial point configuration in the SSA plots for all shapes is the same, shown in Figure 7.2. As per the SSA method, movement of points in subsequent iterations is guided by the relative magnitudes of entries in the shape's similarity matrix. Thus, it is not possible to distinguish between the plots obtained for two shapes whose similarity matrices differ only in scale. Therefore, when iteration stops, we rescale each plot according to its similarity matrix, $\mathcal{S}$, to obtain $\mathbf{Q} = \{\mathbf{q}_i\}$, the final set of points. Assuming the algorithm stopped after M iterations, we consider the last configuration matrix, $\mathcal{C}_M = D(\mathbf{P}_M)$, and obtain a rescaling factor

$$F = \frac{\text{largest entry in } \mathcal{S}}{\text{largest entry in } \mathcal{C}_M}.$$

The centroid of the points in $\mathbf{P}_M$ is computed, $\mathbf{c}_M = \frac{1}{N}\sum_i \mathbf{p}_{i,M}$, and the positions of the rescaled points are computed,

$$\mathbf{q}_i = \mathbf{c}_M + F \cdot (\mathbf{p}_{i,M} - \mathbf{c}_M).$$

7.2 Computing Shape Complexity

Once the points, $\mathbf{Q}$, in the SSA plot are obtained, we aim to measure complexity of the analyzed shape in terms of their dispersion. The motivation is that a simple shape will yield only a few distinct views, leading to a handful of tight, distinct clusters in the SSA plot, whereas a complex shape will have largely varying views which will lead to loose and overlapping clusters.

We use two measures to obtain a complexity value from the points, $\mathbf{Q}$, obtained in the previous section. The first method measures complexity as the dispersion of the points in the x and y directions,

$$C_\sigma = \sqrt{\sigma_x^2 + \sigma_y^2},$$

where σ_x and σ_y are standard deviations of the x and y coordinates resp. of the $\mathbf{q}_i$. The second measure relies on the convex hull of the points in $\mathbf{Q}$ which is a subset, $\mathbf{H} = \{\mathbf{h}_j | \, j \in \{1, \ldots, h\}\}$, of $\mathbf{Q}$. Shape complexity is then measured as

$$\begin{aligned} C_H &= \frac{1}{2} \begin{vmatrix} \mathbf{h}_{1x} & \mathbf{h}_{1y} \\ \mathbf{h}_{2x} & \mathbf{h}_{2y} \\ \vdots & \vdots \\ \mathbf{h}_{hx} & \mathbf{h}_{hy} \\ \mathbf{h}_{1x} & \mathbf{h}_{1y} \end{vmatrix} \\ &= \frac{1}{2} \left[(\mathbf{h}_{1x}\mathbf{h}_{2y} + \mathbf{h}_{2x}\mathbf{h}_{3y} + \ldots + \mathbf{h}_{hx}\mathbf{h}_{1y}) - \right. \\ &\qquad \left. (\mathbf{h}_{1y}\mathbf{h}_{2x} + \mathbf{h}_{2y}\mathbf{h}_{3x} + \ldots + \mathbf{h}_{hy}\mathbf{h}_{1x}) \right], \end{aligned}$$

where $\mathbf{h}_{jx}$ and $\mathbf{h}_{jy}$ are the x and y coordinates resp. of $\mathbf{h}_j$.

7.3 Results

We tested our approach on a set of shapes we obtained from the Internet. In Figure 7.3, we show each of these shapes alongside their corresponding SSA plots, and the obtained values for our two complexity measures, C_H and C_σ. As the shown

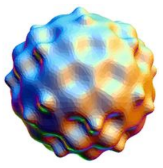
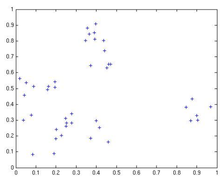
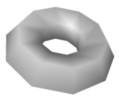
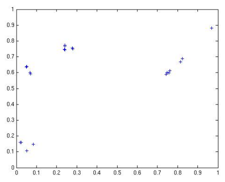
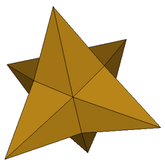
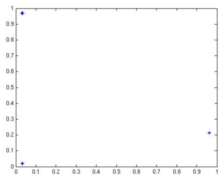
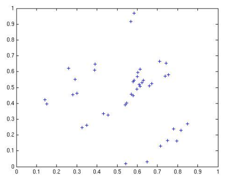
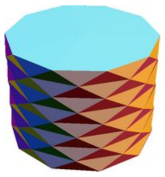
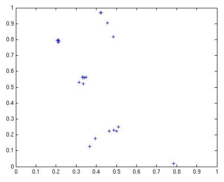
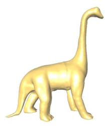
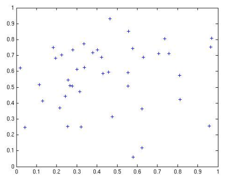
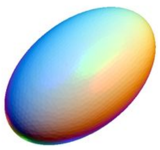
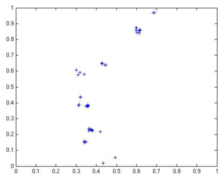
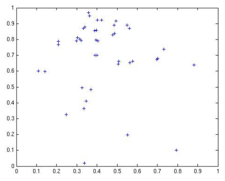
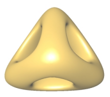
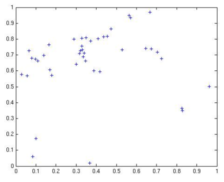
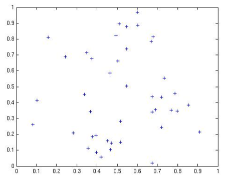
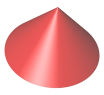
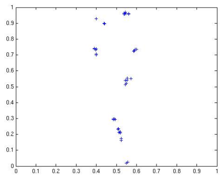
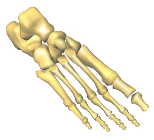
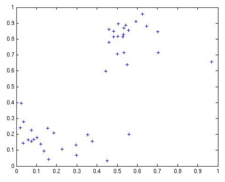
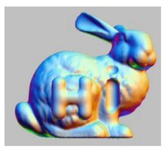
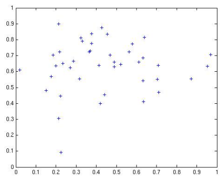
Shape	SSA plot	Shape	SSA plot
			
Bumpy sphere, $C_H = 1.40$, $C_\sigma = 6.04$		Torus, $C_H = 8.20$, $C_\sigma = 19.72$	
			
Star, $C_H = 1.41$, $C_\sigma = 9.88$		Camel, $C_H = 13.20$, $C_\sigma = 15.91$	
			
Schwarz's cyl., $C_H = 3.16$, $C_\sigma = 11.08$		Dinosaur, $C_H = 13.37$, $C_\sigma = 14.94$	
			
Ellipsoid, $C_H = 3.24$, $C_\sigma = 12.72$		Homer, $C_H = 15.86$, $C_\sigma = 15.85$	
			
Genus, $C_H = 7.34$, $C_\sigma = 11.09$		Armadillo, $C_H = 15.93$, $C_\sigma = 18.45$	
			
Cone, $C_H = 7.59$, $C_\sigma = 25.07$		Bones, $C_H = 19.54$, $C_\sigma = 25.51$	
			
Bunny iH, $C_H = 8.06$, $C_\sigma = 11.91$			

Figure 7.3: Shapes sorted top to bottom, left to right according to C_H.

Shape	C_H	Relative	C_σ	Relative
Bumpy sphere	1.4028	1	6.0435	1
Star	1.4091	1.0	9.8827	1.6
Schwarz's Cylinder	3.1565	2.3	11.0770	1.8
Ellipsoid	3.2412	2.3	12.7204	2.1
Genus	7.3383	5.2	11.0868	1.8
Cone	7.5897	5.4	25.0723	4.1
Bunny iH	8.0565	5.7	11.9106	2.0
Torus	8.2006	5.8	19.7177	3.3
Camel	13.2013	9.4	15.9134	2.6
Dinosaur	13.3709	9.5	14.9445	2.5
Homer	15.8560	11.3	15.8468	2.6
Armadillo	15.9370	11.4	18.4462	3.1
Bones	19.5370	13.9	25.5131	4.2

Table 7.1: Result sorted by C_H

values indicate, the shapes are sorted according to values of C_H from top to bottom and left to right, so the Bumpy Sphere is the simplest shape according to this measure, the Star is more complex and so on till the Bunny iH model. The next more complex model with respect to C_H is the Torus and then the Camel up to the Bones model. In the SSA plots shown, for better visualization, the points have been rescaled to fit inside the interval $x, y \in (0, 1)$. We use $N = 42$, i.e. we take 42 views of each shape. These results are summarized in Table 7.1, where the shapes are again sorted by C_H and we also show the relative complexities of the shapes, e.g. according to C_σ, the Camel is 2.6 times as complex as the Bumpy sphere.

C_H and C_σ do not give mutually consistent results. This can also be seen in Figure 7.4 where we compare our results with those obtained using our implementations of previous curvature based methods [Page03, Sukumar06]. We see especially that relatively simple shapes like the Cone and Torus are ranked quite high with C_σ. The reason for this is that points in the SSA plots for these shapes (Figure 7.3) lie in tight, distinct clusters. As C_σ relies on deviation in one dimension only (along the x and y axes separately), the final value comes out to be large. This is corrected when we consider two dimensional information by computing the area of the convex hull to calculate C_H. In Figure 7.4, the Cone and Torus models obtain much lower ranks according to C_H. As expected, the curvature based methods of [Page03, Sukumar06] are unable to deal with noise, the most prominent example of which is that they rank the Bumpy sphere as one of the most complex shapes, whereas our view based method ignores the noise and ranks the Bumpy sphere as the simplest.

7.4 Discussion

Our literature review on automatic computation of complexity of 3D shapes, presented in Section 2.7, yielded few other works. The ones among these that we tested are vulnerable to noise and slight irregularities in the shape. In contrast, our method which is motivated by results from human vision research [Cutzu97] is able to ignore these artefacts and produce a ranking of shapes that is more in agreement with human notions of shape complexity.

However, our method still has deficiencies, e.g. in the first two columns in Figure 7.4, the Bunny is ranked quite low compared to other, simpler shapes like Torus. We believe this is because of inadequate representation of the information contained in our SSA plots. A deeper understanding of the SSA plot reflected in a sophisticated measures to compute complexity from the plots will, in our opinion, relieve our method of the above problems.

The key to our complexity results is the SSA plot we obtain for each shape, which in turn depends on the shape similarity method used. A good shape similarity method, i.e. one that can compute similarities between shapes as humans perceive them, is thus crucial for the success of our approach.

As large numbers of 3D shape content become common, organizing them in a meaningful manner becomes important. Our approach can be used for this purpose to sort shapes in a 3D shape repository according to their complexities. Given a query shape, the repository can also be searched for stored shapes that are more, less or similarly complex.

One straightforward application of our SSA plots can be to compute shape symmetries [Podolak06, Mitra06]. Symmetries in a shape are a measure of the shape's self-similarities. A shape that has many symmetries will yield tight clusters of points in the SSA plot, e.g. the Star in Figure 7.3. This is because clusters correspond to views that are similar to each other. If views from different parts of the shape end up in the same cluster, that is indicative of a self-similarity within the shape between those parts. We could see each point in a cluster as a "vote" for a view. Different parts of a shape voting for the same view will be significant indicators of symmetry. Similar voting schemes have also been employed in previous works on symmetry [Podolak06, Mitra06]. A significantly large number of votes for a view could also be used as a cue for the best view of the object.

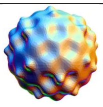
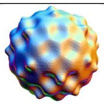
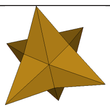
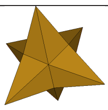
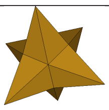
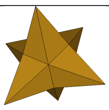
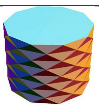
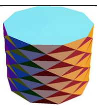
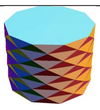
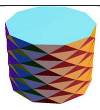
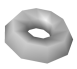
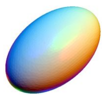
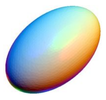
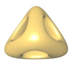
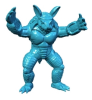
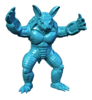
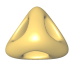
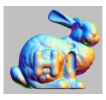
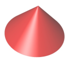
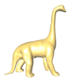
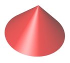
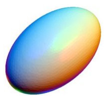
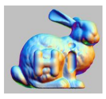
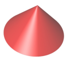
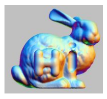
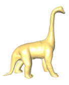
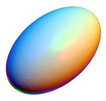
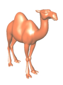
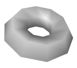
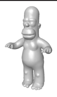
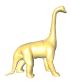
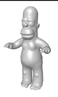
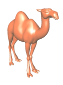
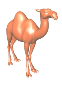
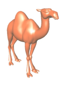
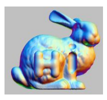
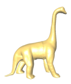
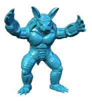
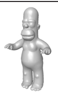
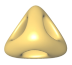
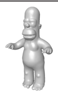
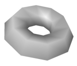
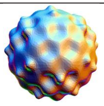
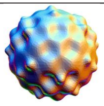
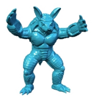
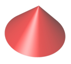
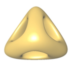
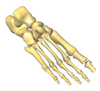
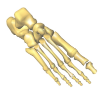
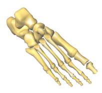
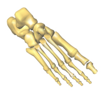
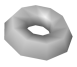
Rank	C_H	C_σ	[Page03]	[Sukumar06]
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				

Figure 7.4: Shapes sorted according to four complexity measures – our complexity measures, C_H and C_σ, and the methods from [Page03] and [Sukumar06].

Chapter 8

Shapes as Bags of Words

The aim of shape retrieval is to retrieve, given a collection of shapes and a query shape, a list of shapes from the collection ranked by similarity to the query shape. In general, a search query to a shape collection can comprise many things including key words, images and sketches. Each of these queries makes sense in different applications. In this chapter, we consider only queries that are shapes themselves.

When talking of searching and retrieval, we have to specify what we consider a successful search. In other words, we need to adopt notions of *precision* and *recall* for shape retrieval. These notions depend on measures of shape similarity, which themselves heavily depend on the application at hand. Very often shape (dis)similarity measures are categorized as either sensitive or oblivious to articulated motions, see Figure 8.1 for an example of an articulated motion. This means either the similarity measure depends on how the shape (and thus its surface) is embedded into Euclidean space, or it depends only on intrinsic properties of the surface that do not (or only slightly) depend on the embedding. In the first case we have a measure that is sensitive to articulated motions, and in the second case an oblivious measure. Note that it depends on the particular application if articulation invariant (dis)similarity makes sense or not.

As mentioned in Section 2.6, many communities work on 3D shape retrieval. There even exists an annual shape retrieval contest [SHREC]. The focus in these works and the contests has been mostly on search quality, but not on its efficiency. Our aim and purpose in this chapter is to scale 3D shape retrieval to large collections of shapes. Though publicly available shape collections at the moment contain only a few hundred shapes we are convinced that with some delay we will

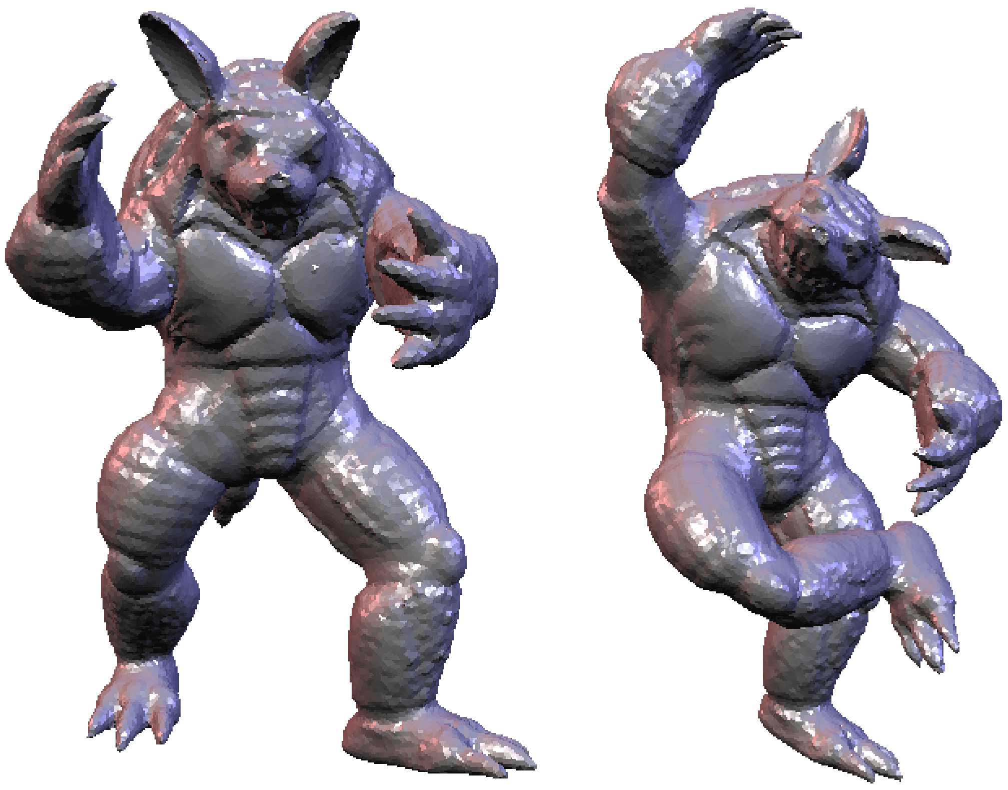


Figure 8.1: Articulated motion of the Armadillo model.

see them grow/explode like other multi-media collections as text, audio or images. Here we make the following contributions to scale shape retrieval to large shape collections:

1. We show how to derive, from each shape, a multi(bag) of artificial (feature) words with the two key properties that (1) there are only relatively few distinct words per shape, and (2) similar shapes have a relatively large number of words in common, whereas dissimilar shapes do not. These properties allow us to use standard top-k indexing and retrieval techniques, which scale well to a large number of objects. Many different geometric features that have been discussed in the shape retrieval literature can be used to derive the (feature) words. In that sense the derivation of (feature) words is generic. We discuss two examples: shape distributions [Osada02] for articulated motion sensitive shape retrieval, and spin images [Johnson97] for articulated motion invariant retrieval.

2. We show how to construct an arbitrarily large database of realistic shapes by variation of a given small number of base shapes with respect to the fol-

lowing: thickness, scale, stretch, and local deformation. This, we construct 1,000,000 shapes from the small SHREC collection [SHREC] that contains only 400 shapes.

3. We show how indexing and retrieval based on the artificial words easily scales to our artificially created 1,000,000 shape database, where we achieve an average query time of 27 milliseconds.
4. We show how text retrieval based on our artificial words gives results of the same quality as those obtained using the most successful of the original shape-distribution descriptors [Osada02].
5. We show how to use ideas from the spin-images descriptor [Johnson97] to derive an alternative bag of words for each document that permit articulated motion invariant shape retrieval, as called for by certain applications

Our approach can be characterized as *localize* and *quantize*. The same philosophy is being increasingly employed in the image retrieval domain (Section 2.6), and has been employed once for shapes [Biswas07], though in 2D. To test our approach, we generated a large collection of shapes (1,000,000) from a base set of only 400 shapes.

8.1 Scaling Shape Retrieval

In a nutshell our approach can be characterized as *localize* (geometric features) and *quantize* (the localized features). We observe that many geometric feature based shape descriptors—also global ones—can be localized in the following sense: first the shape is sampled, i.e. the boundary that represents it, and then the features are computed at each sample point. The localized features can be treated like words in a document—turning shape retrieval into a text retrieval problem. But localization alone is not enough to provide an efficient, robust and relevant 3D shape search engine. While localization essentially allows for efficient top-k search by enabling efficient indexing of the local features their quantization is needed for robustness (recall), accuracy (precision) and also efficiency. With almost no quantization a shape query might not provide any hit due to small shape variations, noise or even just numerical inaccuracies, whereas quantizing too aggressively can render many hits irrelevant and makes it harder to rank them properly. The two extremes—almost no hits when not quantizing and almost every shape in the collection being a hit—illustrate the virtue of quantization, namely trading-off accuracy and robustness. Note that sampling already introduces some sort of quantization, but this is not enough to ensure robustness (a high recall value). To achieve the latter, the computed words also need to be quantized.

Our *generic* approach follows the following pipeline to build an index:

1. Sample the shape which is given by a boundary description.
2. Compute features localized at the sample points.
3. Quantize the localized features, providing a (feature) word frequency vector representation.
4. Build a weighted inverted index on the (feature) words.

This pipeline is generic as it allows many different geometric features to be used in its second step. For our experiments we used a localized version of the D2 shape distribution [Osada02] which we described in Section 3.6. The D2 shape distribution is a global feature in the sense that it takes into account the relation of the sample point at which the feature is localized to all other sample points spread out over the entire surface of the shape. In that sense we are dealing in our experiments with a localized global feature that is not invariant under articulated motions of the shape. We are also discussing an example of a localized local feature, namely spin-images [Johnson97]. So far we have not conducted experiments using our pipeline and spin-images, but we expect them to perform much better for articulated motion invariant shape search.

The rest of the pipeline is standard. We used a standard weighting scheme in our index, namely "term-frequency-inverse document frequency" (tf-idf) with is defined as follows: let n_i^j be the frequency of word j in shape i, let $n_i = \sum_j n_i^j$ be the total number of words for shape i (in our implementation this is a constant), let m^j be the number of shapes in the collection having word j, and let m be the number of shapes in the collection. The weighted frequency of word j in document i is given as

$$\bar{n}_i^j = \frac{n_i^j}{n_i} \log \frac{m}{m_j},$$

i.e., as the product of the word frequency within the document $\frac{n_i^j}{n_i}$ and the inverse document frequency $\frac{m}{m_j}$ (scaled logarithmically).

A query shape is processed exactly the same way as the shapes in the collection, i.e. a weighted (feature) word frequency vector is computed for the query shape. Shapes s in the collection are ranked with respect to the query shape q by the normalized inner product of the weighted word frequency vectors for s and q, i.e. by $\sum_j \bar{n}_s^j \cdot \bar{n}_q^j$.

8.1.1 Example: a localized global descriptor

Localization We assume that we are given a boundary representation of a 3D shape that is normalized to fit into the unit cube. Our localized shape descriptor for each point on the boundary is the distribution of distances to other points on the boundary. Globally the shape is described by the function that maps each point on the boundary to its associated distribution of distances. Note that this function is very similar to the D2 shape function studied by Osada et al. [Osada02].

Quantization Obviously our localized descriptor cannot be used or even computed in practice. To discretize it, we sample 5000 points from the shape uniformly (with respect to surface area) at random. For each sample point we compute the distance to another 100 sample points (sampled uniformly at random from the 4999 remaining sample points). The distribution of these distances is our discretized version of the localized shape descriptor. Globally the shape is now represented by the distributions at all the sample points.

Sampling the shape and the distance computations is already a quite aggressive quantization. Quantization is not only necessary to discretize our shape descriptor—as we argued before quantization is also needed to trade-off robustness (recall) and accuracy (precision). We quantize further (sacrificing accuracy for robustness) as follows: the distances at each sample point are binned into five bins each of width $1/20$ of the length of the diagonal of the unit cube, i.e. $\sqrt{3}$. Then the number of elements in each bin is multiplied by $1/4$ giving number in the interval $[0, 25]$. This number is rounded to the next integer providing a number in $\{0, \dots, 25\}$ that can be represented by a letter in $\{A, \dots, Z\}$. Hence the final quantized, localized descriptor is a five letter word in $\{A, \dots, Z\}^5$.

8.1.2 Example: a localized local descriptor

Localization Again we assume that we are given boundary representation of a 3D shape that is normalized to fit into the unit cube. Furthermore we assume that we have access to the surface normal at each point. Our localized local shape descriptor is for each point on the boundary a distribution of (α, β)-pairs that are defined as follows: fix a radius $\varepsilon > 0$ for any point p of the boundary and let $\vec{n}_p$ be the unit surface normal at p. For any point q on the boundary with distance less than ε to p, i.e., $\|p - q\| < \varepsilon$, let $\vec{q} := q - p$ and define

$$(\alpha_p^q, \beta_p^q) = \left(\sqrt{\|\vec{q}\|^2 - \langle n_p, \vec{q} \rangle^2}, \langle \vec{n}_p, \vec{q}/\|\vec{q}\| \rangle \right) .$$

Our localized shape descriptor (also known as spin-image [Johnson97]) is the distribution of the (α_p^q, β_p^q) for the points q in the ε-neighborhood of p on the surface.

Quantization Spin-images cannot be computed in practice, but they can be discretized easily by sampling the shape: take the k nearest sample points of the sample point at which the spin-image needs to be discretized and just compute the histogram of spin-image values for these neighbors.

8.1.3 Difference to feature hashing

At first glance our general approach looks very similar to the feature hashing approach from [Biswas07]—note that at this level we do not care about the particularities of the chosen features, but there are some differences that affect the efficiency as well as the search quality:

1. Both approaches sample the shapes. If n is the number of sample points per shape (called landmarks in [Biswas07]), then with feature hashing one generates n^2 words per shape (the hash table bins exactly correspond to our words), whereas we generate only n words per shape. Note that the sampling requirements for feature hashing and our word generating mechanism are roughly the same since they are basically determined by the local geometries of the shapes. Hence, there is a factor of n difference in the total number of words to be indexed for the two approaches.

2. Feature hashing uses a very different distance metric. For comparing shapes s_1 and s_2 with (normalized) frequencies n_1^j and n_2^j of the j'th word, feature hashing uses $\sum_j (n_1^j - n_2^j)^2/(n_1^j + n_2^j)$, whereas our approach uses (with a distinctly different normalization $\bar{n}_{\cdot}^{j}$) $\sum_j \bar{n}_1^j \cdot \bar{n}_2^j$. The first measure is a dissimilarity measure for shapes, whereas the second measures the similarity between shapes, i.e., a large value means very similar shapes. The advantage of our measure is that it is monotone in the word frequencies, which is an essential prerequisite for doing top-k retrieval, which, in turn, is the key to scale search to very large data sets.

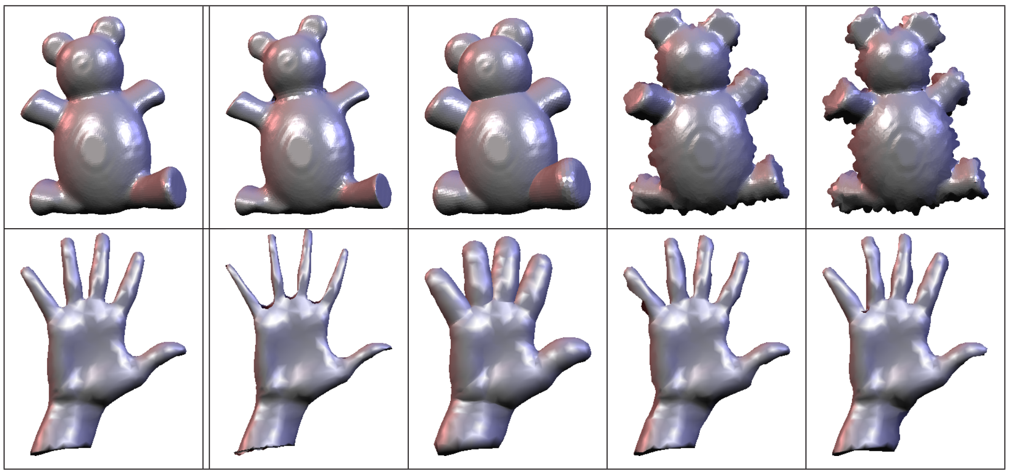


Figure 8.2: An example of our shape deformation pipeline. Form left to right: original shape from the SHREC collection, thinned, thickened, and locally deformed for two different parameter settings.

8.2 Experiments

8.2.1 Datasets

We used the SHREC [SHREC] collection of the AIM@SHAPE consortium as our main source of 3D shapes, see Figures 8.1, 8.2, 8.3 and 8.4 for examples of shapes in this collection. Currently this collection contains about 400 shapes represented as polygonal surface meshes. A collection of only 400 shapes is not enough to test the scalability of our (or any) shape retrieval scheme to the size of collections that we expect in the not too distant future. Hence we created our own large scale shape collection from the shapes in the SHREC collection. The goal when creating the large collection was to model some real world variety. We considered global variations like thickening and thinning, as well as local, small scale variations (which can also be seen as noise).

Creating a large shape collection

We took the shapes from the SHREC collection as base shapes and applied the following transformations (we arrived at our choice of the parameter values after some experimentation, e.g., we wanted to make sure that the deformed shapes are not self intersecting):

1. Rotation about the x,y and z-axes with three angles chosen uniformly at random from $[0, 2\pi)$ for each axis, then
2. scaling along the x,y and z-axes with three scaling values each chosen uniformly at random from the interval $[0.5, 1.50]$, and finally
3. one of
    (a) thickening or thinning by moving the vertices of the mesh along the surface normals at these vertices. The offset value is chosen from an exponential distribution with mean value 0.02. We cut off the values at 0.01 and 0.03, thus taking only values from the interval $[0.01, 0.03]$.

    (b) local deformations at seed vertices from the polygonal mesh. We pick $1/k$ seed vertices uniformly at random from the set of all vertices of the mesh. The value k itself is chosen from an exponential distribution with mean value 20. The offset values at the seed vertices is then chosen uniformly at random from an interval $[o_{\text{min}}, o_{\text{max}}]$, where $-o_{\text{min}}$ and o_{max} are chosen from exponential distributions with with mean value 0.02. Again these values are cut off at 0.01 and 0.04. The off-set values are propagated with decreasing weight to the neighbors of the seed vertices. The neighborhood is taken as the direct neighbors and their neighbors, i.e., the one- and two ring in the mesh. If neighborhoods of seed vertices overlap, then maximum of the magnitudes of all off-sets is chosen. Finally, vertices are moved along their unit normals by the offset amount. A negative off-set means moving the vertices to the interior of the shapes.

8.2.2 Quality

We compared the quality of the localized D2 shape distribution descriptor and of our implementation of the original D2 descriptor as described in Section 3.6). For the comparison we used the labelled SHREC shape collection with 400 shapes, and computed standard relevance measures (mean average precision (map) and precision at k (pr. at k)). Our results show that relevance does not degrade when localizing the D2 shape descriptor, see also Figure 8.4 for visual results. At this point we should remark that neither the D2 descriptor nor its localization are particularly well suited for the SHREC collection. We expect much better results for articulated motion invariant descriptors. We summarize our findings in the following table.

	map	pr. at 5	pr. at 10	pr. at 20
Orig. D2	0.40	0.66	0.49	0.36
Our method	0.41	0.66	0.52	0.38

8.2.3 Symmetries

A benefit of our localization scheme is that it allows for symmetry detection in shapes. As expected, we observe that the same word occurs at sample points that reflect some shape symmetry, see Figure 8.3. Note that this is a feature of the localization, since one cannot hope to detect symmetries with the corresponding global descriptor—in our case the D2 shape distribution.

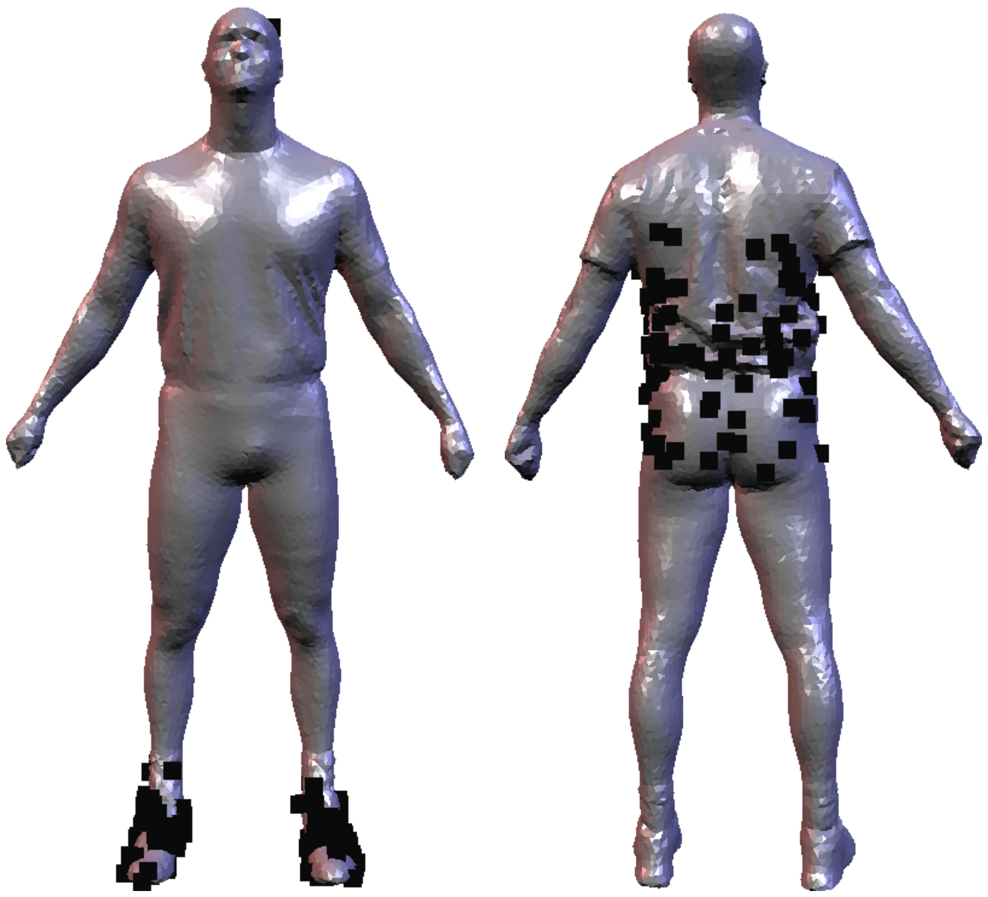


Figure 8.3: The localization scheme respects symmetries in the shape. Shown are the sample points at which a single, frequent word is located—the word is not the same for the two figures.

8.3 Conclusions and Future Work

We have presented a generic scheme to scale feature based shape retrieval to large collections of 3D shapes. To demonstrate our generic approach we have implemented a localized version of the so called D2 shape distribution descriptor and tested it on two collections. The first one is a standard collection that has 400 shapes, and the second contains 1,000,000 shapes that we have generated for the purpose of testing our method.

To exploit the genericity of our approach more shape features should be localized and integrated into our framework. Especially, it will be interesting to study the different behavior of local and global features—one can expect a significant difference with respect to invariance under articulated motions. We also want to gain a better understanding of how well we can detect shape symmetries with our approach and if this can be used for partial shape matching.

Also the generic set-up can be extended. So far we do not take any spatial coherence into account when ranking the retrieved shapes. It is not obvious that taking spatial coherence into account will improve the search quality much though since by the nature of our approach similar words should be located at neighboring sample points on a shape. Nevertheless, it is an interesting question if the ranking can be improved by considering particularities of the sampling approach.

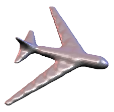
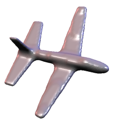
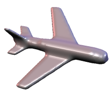
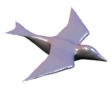
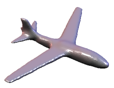
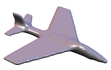
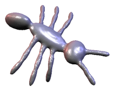
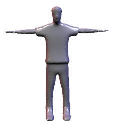
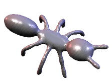
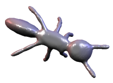
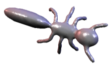
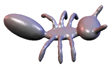
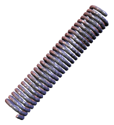
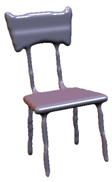
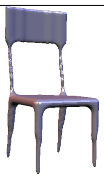
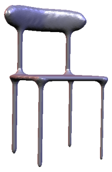
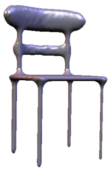
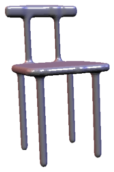
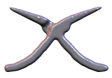
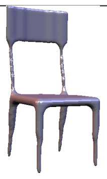
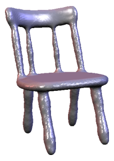
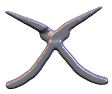
Query		Results 2 to 6 (first result is always the query itself)				
	D2 adapted					
		plane-26	plane-273	bird-28	bird-133	plane-376
	D2 original					
plane-169		plane-145	plane-48	plane-82	hand-278	ant-339
	D2 adapted					
		human-389*	ant-325	mech_part-348	hand-35	ant-15
	D2 original					
human-37		human-389*	ant-275	ant-202	spring-102	mech_part-141
	D2 adapted					
		chair-96	chair-6	chair-125	chair-232	chair-87
	D2 original					
chair-38		pliers-320	chair-96	human-43	chair-22	pliers-392

Figure 8.4: Query results on the original SHREC collection. Shown are three queries (the query shape is always shown as the left most shape) and the top six results (the top result always was the query shape itself so we do not show it here). For each query the first row shows the results retrieved our localized version of the D2 shape distribution descriptor and the second row shows the results for the D2 shape distribution descriptor itself. Note also that the search quality does not degrade by localization (also note that a articulated motion sensitive descriptor as D2—localized or not— is not well suited for some of the queries).
* – human-389 in the dataset is a scaled version of the human-37 model.

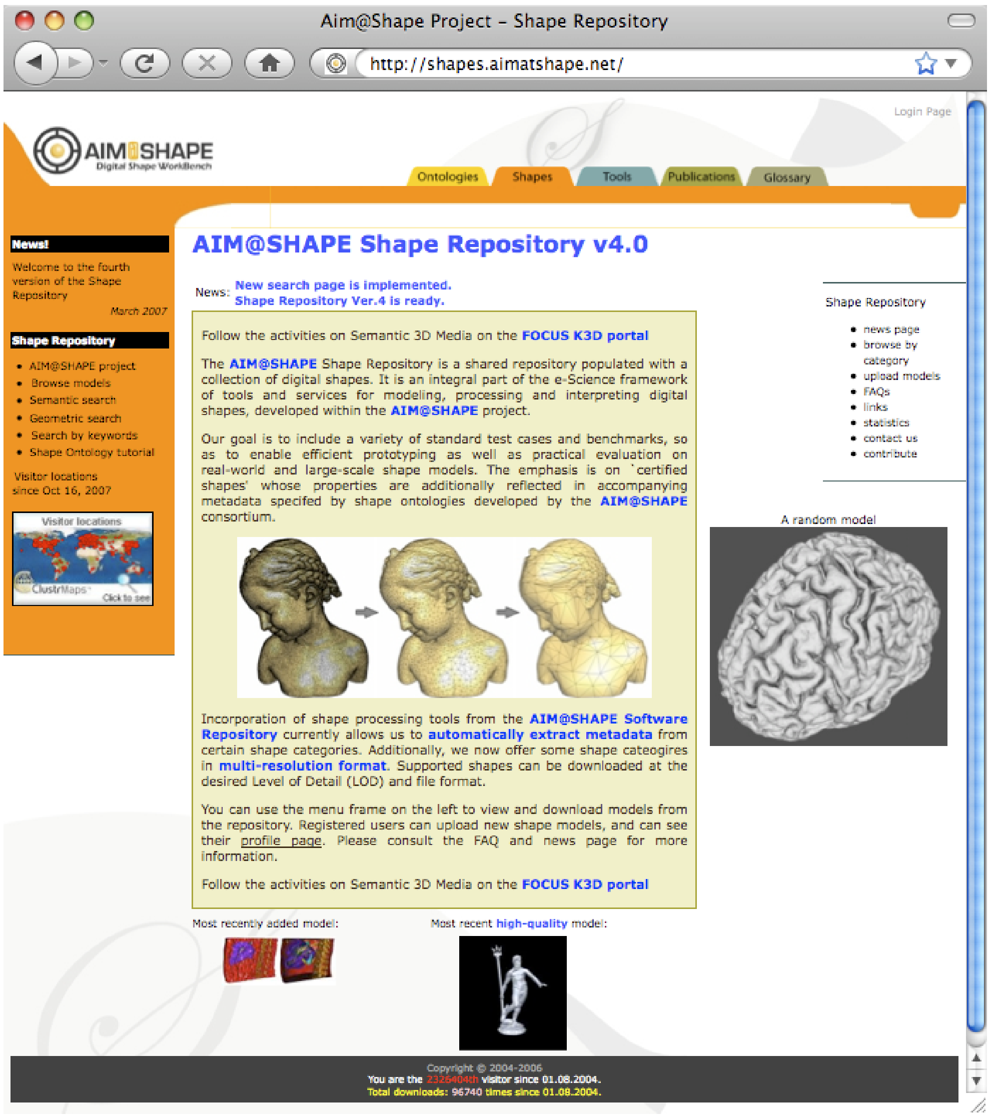
Chapter 9

Managing a Shape Repository

A significant portion of the research effort for this thesis was spent in contributing to the development, maintenance and upkeep of the Shape Repository [Repositorya] of the EU IST NoE project, AIM@SHAPE [AIM@SHAPE]. The repository is a result of efforts of the entire project, not ours alone. However, we feel that the underlying principles can be applied to any shape repository and thus, an overall exposition of the ideas behind the repository will be beneficial to the digital shape research community. The following ideas have previously been outlined in project internal reports available through the AIM@SHAPE web portal [AIM@SHAPE]. All of the figures in this chapter are screen captures from the repository website [Repositorya]. Readers are encouraged to visit the website themselves to further explore the repository.

9.1 Knowledge Management of Shapes

The first and foremost concern in maintaining a sizable collection of digital shapes is their organization. Although, the most common form of shape representation is the triangle mesh, other representations like parametric surfaces, volume representations, point set surfaces and implicit representations also abound. Furthermore, if we slightly relax our definition of *shape* beyond surface models, we have to additionally deal with representations in the form of raster data, animations, contours and structural descriptors. Moreover, shapes vary among each other in terms of their size, complexity, topology, origin, means of creation and a multitude of other characteristics. In this context, to meaningfully present stored shapes and



News!

Welcome to the fourth version of the Shape Repository

March 2007

Shape Repository

- AIM@SHAPE project
- Browse models
- Semantic search
- Geometric search
- Search by keywords
- Shape Ontology tutorial

Visitor locations since Oct 16, 2007

AIM@SHAPE Shape Repository v4.0

News: **New search page is implemented.**
Shape Repository Ver.4 is ready.

Follow the activities on Semantic 3D Media on the **FOCUS K3D portal**

The **AIM@SHAPE** Shape Repository is a shared repository populated with a collection of digital shapes. It is an integral part of the e-Science framework of tools and services for modeling, processing and interpreting digital shapes, developed within the **AIM@SHAPE** project.

Our goal is to include a variety of standard test cases and benchmarks, so as to enable efficient prototyping as well as practical evaluation on real-world and large-scale shape models. The emphasis is on `certified shapes' whose properties are additionally reflected in accompanying metadata specifed by shape ontologies developed by the **AIM@SHAPE** consortium.

Incorporation of shape processing tools from the **AIM@SHAPE Software Repository** currently allows us to **automatically extract metadata** from certain shape categories. Additionally, we now offer some shape cateogires in **multi-resolution format**. Supported shapes can be downloaded at the desired Level of Detail (LOD) and file format.

You can use the menu frame on the left to view and download models from the repository. Registered users can upload new shape models, and can see their profile page. Please consult the FAQ and news page for more information.

Follow the activities on Semantic 3D Media on the **FOCUS K3D portal**

Shape Repository

- news page
- browse by category
- upload models
- FAQs
- links
- statistics
- contact us
- contribute

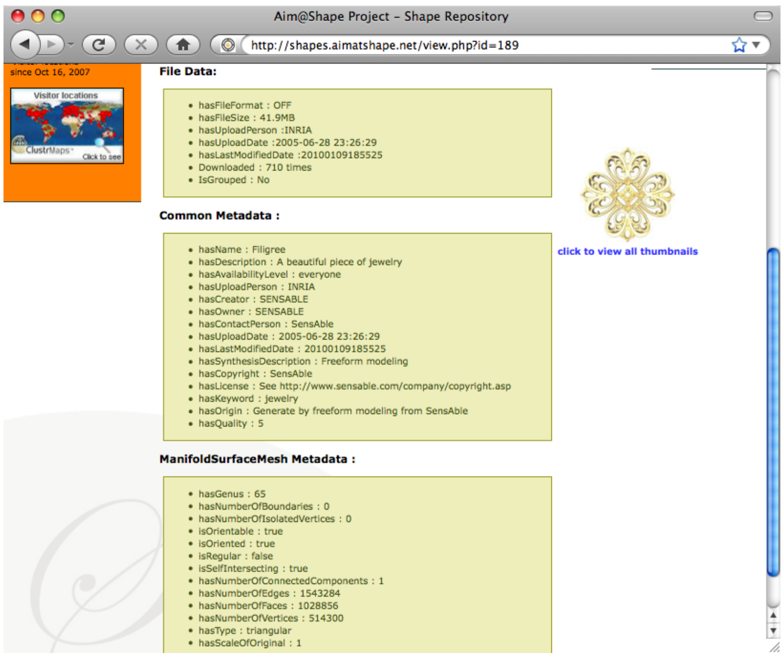
A random model

Most recently added model:

Most recent **high-quality** model:

Copyright © 2004-2006
You are the 2326404th visitor since 01.08.2004.
Total downloads: 96740 times since 01.08.2004.

Figure 9.1: The AIM@SHAPE Shape Repository is accessible online at http://shapes.aimatshape.net.


Aim@Shape Project – Shape Repository

http://shapes.aimatshape.net/view.php?id=189

since Oct 16, 2007

File Data:

- hasFileFormat : OFF
- hasFileSize : 41.9MB
- hasUploadPerson :INRIA
- hasUploadDate :2005-06-28 23:26:29
- hasLastModifiedDate :20100109185525
- Downloaded : 710 times
- IsGrouped : No

Common Metadata :

- hasName : Filigree
- hasDescription : A beautiful piece of jewelry
- hasAvailabilityLevel : everyone
- hasUploadPerson : INRIA
- hasCreator : SENSABLE
- hasOwner : SENSABLE
- hasContactPerson : SensAble
- hasUploadDate : 2005-06-28 23:26:29
- hasLastModifiedDate : 20100109185525
- hasSynthesisDescription : Freeform modeling
- hasCopyright : SensAble
- hasLicense : See http://www.sensable.com/company/copyright.asp
- hasKeyword : jewelry
- hasOrigin : Generate by freeform modeling from SensAble
- hasQuality : 5

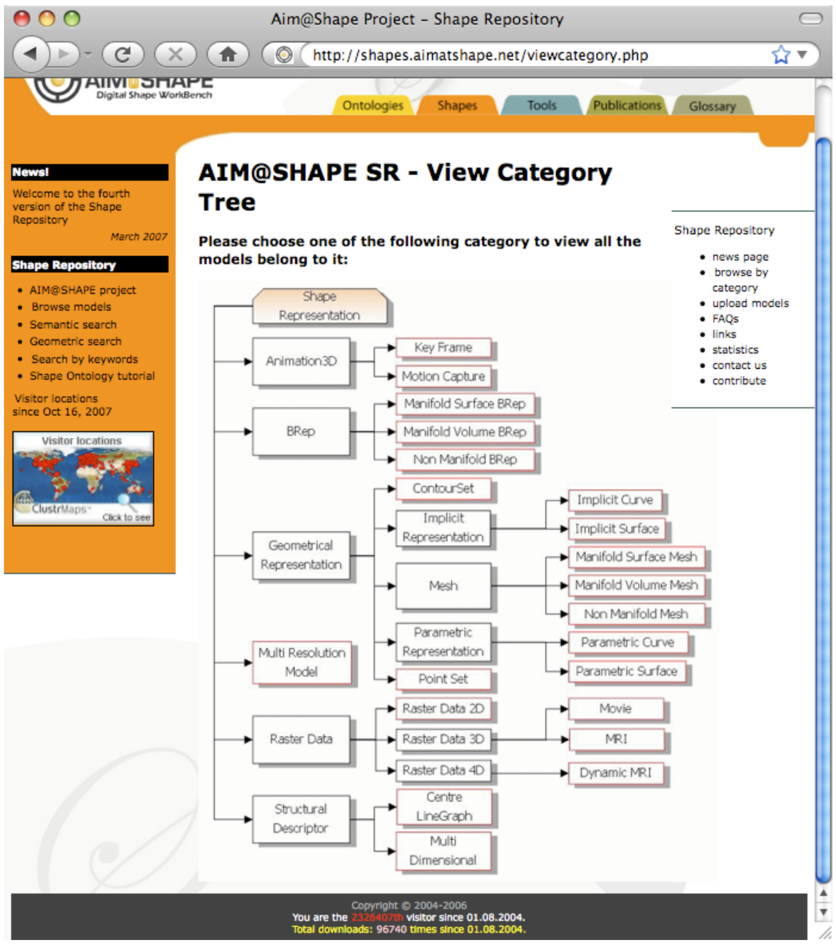
ManifoldSurfaceMesh Metadata :

- hasGenus : 65
- hasNumberOfBoundaries : 0
- hasNumberOfIsolatedVertices : 0
- isOrientable : true
- isOriented : true
- isRegular : false
- isSelfIntersecting : true
- hasNumberOfConnectedComponents : 1
- hasNumberOfEdges : 1543284
- hasNumberOfFaces : 1028856
- hasNumberOfVertices : 514300
- hasType : triangular
- hasScaleOfOriginal : 1

Figure 9.2: Shapes in the repository are considered instances of ontology concepts and have associated metadata fields which have to be entered when adding the shape to the repository and are subsequently displayed with the shape.

to allow a user to browse through them in some organized fashion is a non-trivial task.

In the AIM@SHAPE Shape Repository, this problem is solved using a knowledge management approach. A Common Shape Ontology has been developed that defines a hierarchical classification of different shape representations. Each node in the hierarchy represents a *concept* in the ontology and has a list of associated attributes, or *metadata fields*, in addition to the metadata it inherits from higher level concepts. Specific shape types are concepts residing at the lowest level of the hierarchy, and only these concepts are allowed to have *instances*. For a shape type concept, an instance is a shape of that type. Thus, all shapes are added to the repository as instances of some concept. As each concept has some related metadata, an instance has to have values for each of the metadata fields. Therefore, when instances are added to the repository, the associated metadata has also to be entered. This metadata is stored in the repository along with the shapes. Different instances of the same concept are distinguished based on their individual metadata values. Figure 9.2 shows an example "view page" of a shape which shows its thumbnail(s) along with stored metadata.



AIM@SHAPE SR - View Category Tree

Please choose one of the following category to view all the models belong to it:

Welcome to the fourth version of the Shape Repository

March 2007

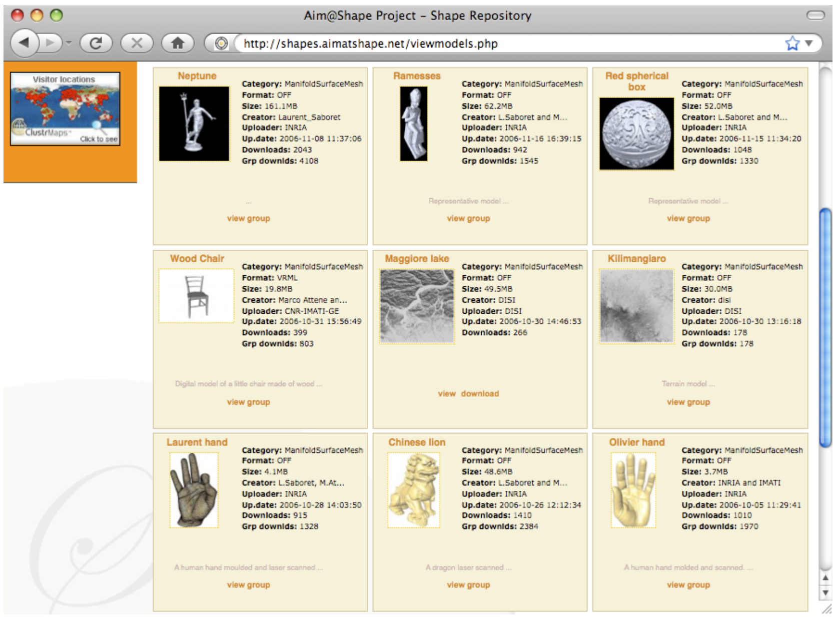
Visitor locations since Oct 16, 2007

Copyright © 2004-2006
You are the 2326407th visitor since 01.08.2004.
Total downloads: 96740 times since 01.08.2004.

Figure 9.3: Shapes in the repository are organized using a Common Shape Ontology developed within the project. Considering only the bottom level concepts of the ontology results in a Shape Category Tree.

A condensed version of the ontology showing only the lower level concepts that correspond to concrete shape types is presented on the repository website as a clickable “Shape Category Tree” shown in Figure 9.3. Clicking on a shape type in the tree allows users to “browse by category” by displaying all stored shapes on that type.

Storing metadata simply as database fields allows us to trivially sort shapes according to several criteria (shape quality, number of times downloaded, identity of uploader etc.) by issuing single SQL queries to the database. The quality of the repository depends on the presence and quality of the metadata provided for the instances. Two steps are taken to uphold these requirements. Firstly, addition


Aim@Shape Project – Shape Repository
http://shapes.aimatshape.net/viewmodels.php
Visitor locations
ClustrMaps
Click to see

Neptune
Category: ManifoldSurfaceMesh
Format: OFF
Size: 161.1MB
Creator: Laurent_Saboret
Uploader: INRIA
Up.date: 2006-11-08 11:37:06
Downloads: 2043
Grp downlds: 4108
...
view group

Ramesses
Category: ManifoldSurfaceMesh
Format: OFF
Size: 62.2MB
Creator: L.Saboret and M...
Uploader: INRIA
Up.date: 2006-11-16 16:39:15
Downloads: 942
Grp downlds: 1545
Representative model ...
view group

Red spherical box
Category: ManifoldSurfaceMesh
Format: OFF
Size: 52.0MB
Creator: L.Saboret and M...
Uploader: INRIA
Up.date: 2006-11-15 11:34:20
Downloads: 1048
Grp downlds: 1330
Representative model ...
view group

Wood Chair
Category: ManifoldSurfaceMesh
Format: VRML
Size: 19.8MB
Creator: Marco Attene an...
Uploader: CNR-IMATI-GE
Up.date: 2006-10-31 15:56:49
Downloads: 399
Grp downlds: 803
Digital model of a little chair made of wood ...
view group

Maggiore lake
Category: ManifoldSurfaceMesh
Format: OFF
Size: 49.5MB
Creator: DISI
Uploader: DISI
Up.date: 2006-10-30 14:46:53
Downloads: 266
view download

Kilimangiaro
Category: ManifoldSurfaceMesh
Format: OFF
Size: 30.0MB
Creator: disi
Uploader: DISI
Up.date: 2006-10-30 13:16:18
Downloads: 178
Grp downlds: 178
Terrain model ...
view group

Laurent hand
Category: ManifoldSurfaceMesh
Format: OFF
Size: 4.1MB
Creator: L.Saboret, M.At...
Uploader: INRIA
Up.date: 2006-10-28 14:03:50
Downloads: 915
Grp downlds: 1328
A human hand moulded and laser scanned ...
view group

Chinese lion
Category: ManifoldSurfaceMesh
Format: OFF
Size: 48.6MB
Creator: L.Saboret and M...
Uploader: INRIA
Up.date: 2006-10-26 12:12:34
Downloads: 1410
Grp downlds: 2384
A dragon laser scanned ...
view group

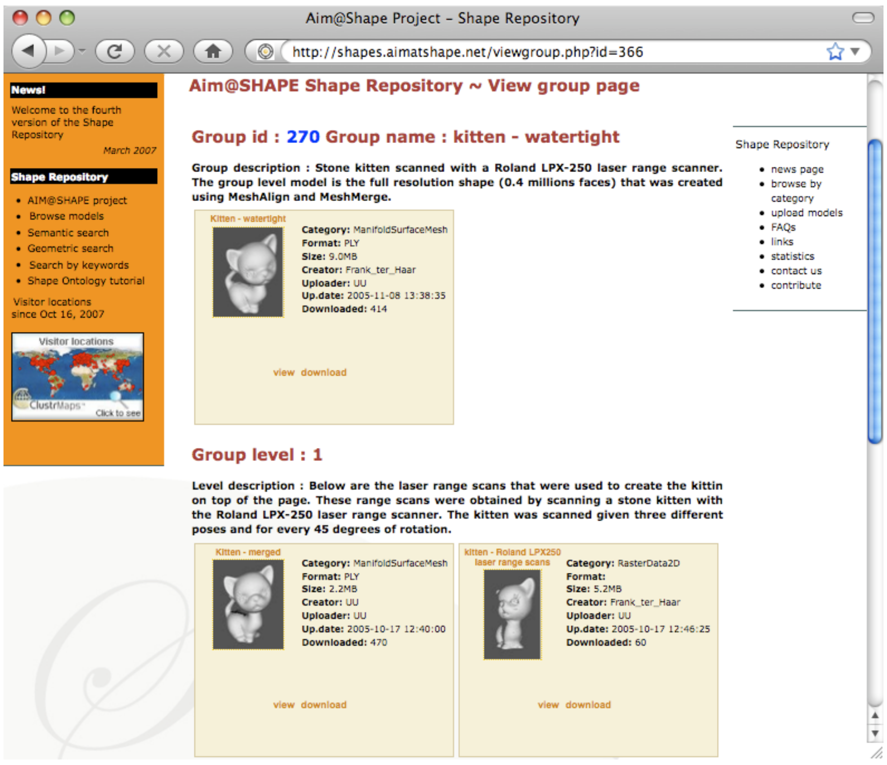
Olivier hand
Category: ManifoldSurfaceMesh
Format: OFF
Size: 3.7MB
Creator: INRIA and IMATI
Uploader: INRIA
Up.date: 2006-10-05 11:29:41
Downloads: 1010
Grp downlds: 1970
A human hand molded and scanned. ...
view group

Figure 9.4: The repository has been populated with shape representations acquired from a large variety of objects.

of instances to the repository is restricted to a handful of project internal and external partners who have agreed to uphold the metadata requirement when adding instances. Secondly, for common shape types, tools have been incorporated into the repository that automatically extract values for certain metadata fields from shapes that are being added. This eases the burden on the user to enter a long list of metadata values.

9.2 Populating the Repository

Of course, the primary requisite of a truly useful and general purpose Shape Repository is to provide a large number of shapes ranging in their attributes: size, complexity, topology, representation format, etc. The shapes must have known properties and should be aesthetically pleasing. Preferably, the shapes should be unique to the repository, at least at the time of addition. To remain current, the repository must be updated periodically with new shapes. To be of special use to the scientific community, current trends in digital shape research should be monitored, and an effort should be made to provide data suitable for experimentation and validation of algorithms in these fields.


Aim@Shape Project – Shape Repository

http://shapes.aimatshape.net/viewgroup.php?id=366

News!

Welcome to the fourth version of the Shape Repository

March 2007

Shape Repository

- AIM@SHAPE project
- Browse models
- Semantic search
- Geometric search
- Search by keywords
- Shape Ontology tutorial

Visitor locations since Oct 16, 2007

Aim@SHAPE Shape Repository ~ View group page

Group id : 270 Group name : kitten - watertight

Group description : Stone kitten scanned with a Roland LPX-250 laser range scanner. The group level model is the full resolution shape (0.4 millions faces) that was created using MeshAlign and MeshMerge.

Kitten - watertight

Category: ManifoldSurfaceMesh
Format: PLY
Size: 9.0MB
Creator: Frank_ter_Haar
Uploader: UU
Up.date: 2005-11-08 13:38:35
Downloaded: 414

view download

Group level : 1

Level description : Below are the laser range scans that were used to create the kittin on top of the page. These range scans were obtained by scanning a stone kitten with the Roland LPX-250 laser range scanner. The kitten was scanned given three different poses and for every 45 degrees of rotation.

Kitten - merged

Category: ManifoldSurfaceMesh
Format: PLY
Size: 2.2MB
Creator: UU
Uploader: UU
Up.date: 2005-10-17 12:40:00
Downloaded: 470

view download

kitten - Roland LPX250 laser range scans

Category: RasterData2D
Format:
Size: 5.2MB
Creator: Frank_ter_Haar
Uploader: UU
Up.date: 2005-10-17 12:46:25
Downloaded: 60

view download

Shape Repository

- news page
- browse by category
- upload models
- FAQs
- links
- statistics
- contact us
- contribute

Figure 9.5: "Groups" in the repository link related shapes, e.g. this group contains a reconstruction at the topmost level followed by range data at lower levels.

For the purposes of the AIM@SHAPE Shape Repository, the above issues have been dealt with in a variety of ways. As the repository is part of a larger project involving researchers working on different aspects of digital shape processing for diverse applications, collecting an initial set of shapes in various formats and sizes was straightforward. Participating researchers contributed shapes they typically use for experiments and benchmarking, and shapes they generated as results of their research. Properties of shapes are supposed to be documented through accompanying metadata. As it is unfeasible to manually moderate metadata for all shapes, the burden of entering correct metadata values for shapes has to be left on the users adding the shapes to the repository. This is somewhat alleviated for some fields of some shape types by the automatic metadata extraction tools mentioned in Section 9.1, but the problem of totally freeing the user from entering metadata is far from solved. It should however also be noted that some metadata fields, like origin of the digital shape, cannot be automatically computed and user interaction will always be needed for such values.

Great effort has been put into populating the repository with unique and interest-

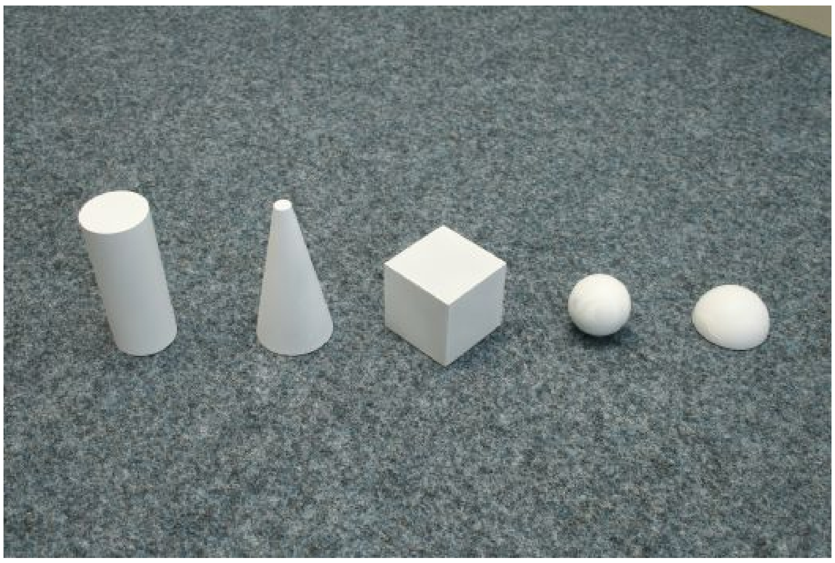


Figure 9.6: Accurate scans of a few canonical shapes have been added to allow benchmarking of algorithms.

ing shapes. To this effect, exclusive scanning sessions have been held regularly to acquire shapes ranging from toys to furniture items to cultural artifacts. Some of these shapes are shown in Figure 9.4. Both raw and processed data from these sessions is made available through the repository, with the accompanying meta-data providing details about the scanners used for acquisition and origin of the shapes. Figure 9.5 shows one such group. In an effort to provide shapes with *certified* properties, original CAD models of some machine parts, along with scans of the manufactured part are also provided. This is also done for a few simple, canonical shapes, shown in Figure 9.6, which were later custom-manufactured for the repository.

Scans from a new (at the time of writing) type of 3D scanner that uses a scanner mounted on an articulated arm and thus eliminates the need for subsequent scan alignment and registration, were acquired exclusively for and made available through the repository. To provide highly detailed shapes, alginate moulds of hands were prepared and subsequently scanned. Figure 9.7 shows a combined thumbnail of one such model. The resulting scans provide very fine details, in some cases up to the resolution of individual dermal ridges. *Super resolution reconstruction* is a recent (as of writing) line of research whereby an object is scanned many times from the same position. Noise inherent in the measuring device, namely the 3D scanner, causes all scans to vary slightly from each other. This is exploited to reconstruct the scanned object at a resolution much higher

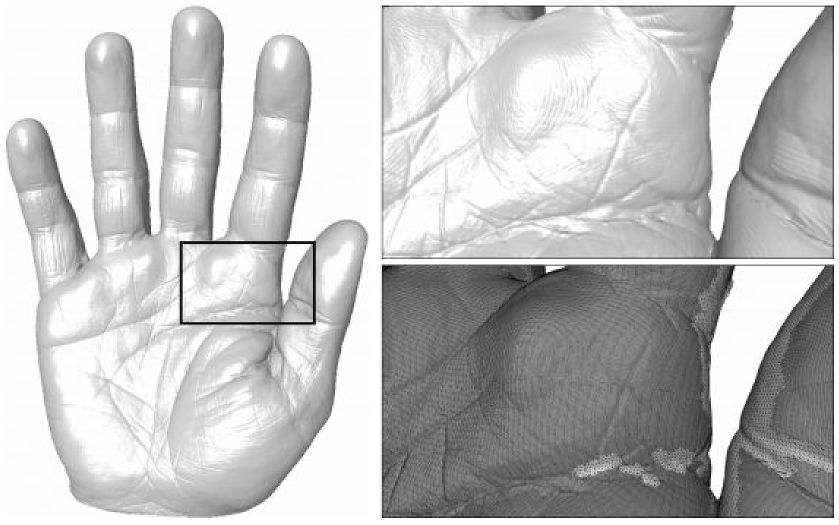


Figure 9.7: Scans of highly detailed alginate moulds of hands have been added to the repository. The figure shows the scanned model.

than that of any of the individual scans. Data for such algorithms was acquired for the repository in the form of 100 scans of a single coin from the same side.

9.3 Grouping Shapes

We saw in Section 9.2 some of the ways in which the Shape Repository may contain different representations of the same shape. In such cases, it is reasonable to group these different representations together into a single logical whole.

This idea is implemented in the AIM@SHAPE Shape Repository by introducing the concept of hierarchical *shape groups* illustrated earlier in Figure 9.5. The topmost level contains a single "representative" model for the group. Depending on how the user adding the shapes wants to organize the group, there might be one or more sub-levels in the group, each containing one or more shapes. Thus, the repository comprises of *individual models* and *group models*, and when adding to the repository, the user has to indicate whether an individual or group model is going to be added.

Group models essentially consist of individual models coupled with organizational information, i.e. number of levels in the group, and membership of each level. They are modeled as a separate concept in the Common Shape Ontology and each shape group in the repository is an instance of this concept, with references to shape instances.

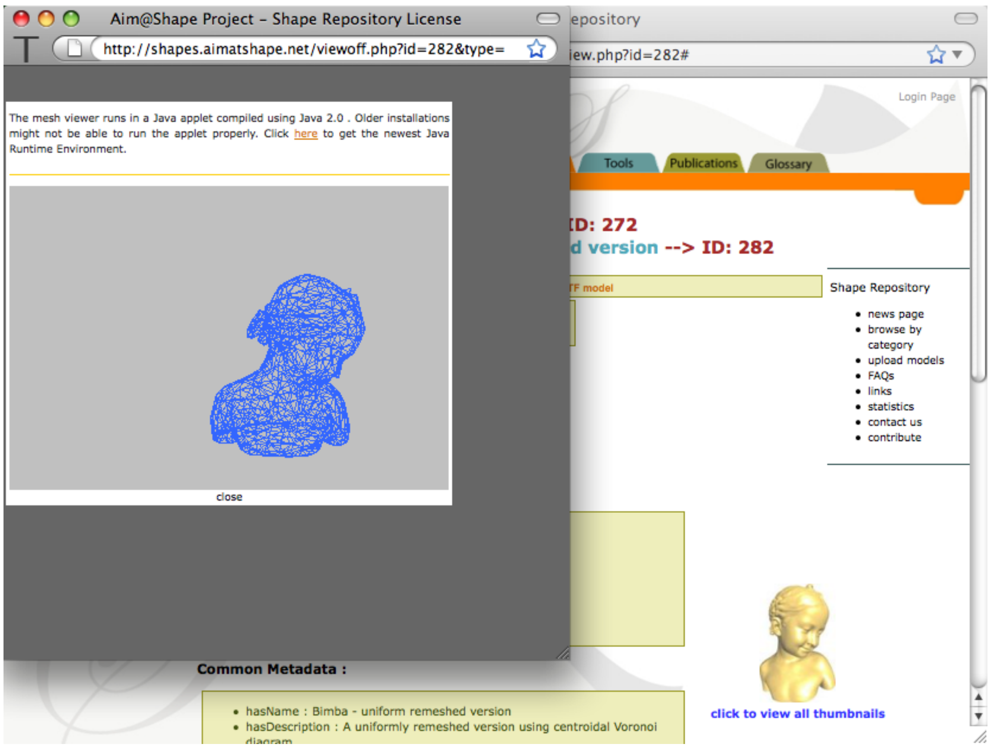
Other than providing a means for linking related shapes, shape groups have emerged to be starting points for collaboration. In our experience, models added to the repository by one user have later been used by other users to test their own algorithms. The results of these algorithms are then added to the group containing the original shape, possibly in a new sub-level. Another use has been to conveniently disseminate all models used and/or produced in a research project. The authors of the project can then simply direct inquirers to the repository to obtain desired models.

9.4 Legal Issues

Having a large number of shapes from diverse sources necessarily entails issues on ownership and dissemination rights. Within the AIM@SHAPE Shape Repository, the user who added a model becomes its *owner*. To deal with dissemination and reuse issues, an AIM@SHAPE General Licence has been put together which covers most of the general requirements of model owners. Any user wishing to download a model from the repository has to indicate agreement to the licence for the download to proceed successfully. The downloaded archive contains the chosen shape model, its thumbnail(s) and metadata. The terms of the licence are that

- the Shape Repository and owner of the model, whose information is present in the accompanying metadata, be acknowledged whenever the model or its derivatives are further disseminated,
- the accompanying metadata be stored with the shape model,
- shape models representing culturally or religiously sensitive artefacts, e.g. Chinese dragon, be used conscientiously, and
- the Shape Repository team and the model owner be contacted if the model is to be used for any non-research or non-commercial purpose.

All shapes added to the repository are subject to the above licence. An online service, Creative Commons (CC) [Commons], is used to help model owners enforce additional terms, namely No Derivative Works (ND) and Share Alike (SA). ND forbids a user to disseminate works derived from the downloaded model. SA, on the other hand, allows users such dissemination but only if the derived works are subject to the same conditions. For the chosen option, CC provides a legally valid licence, which, when specified by the model owner, is linked to in



The mesh viewer runs in a Java applet compiled using Java 2.0 . Older installations might not be able to run the applet properly. Click here to get the newest Java Runtime Environment.

ID: 272
d version --> ID: 282

TF model

Common Metadata :

- hasName : Bimba - uniform remeshed version
- hasDescription : A uniformly remeshed version using centroidal Voronoi diagram

Figure 9.8: Using an integrated mesh viewer, visitors to the repository can view simplified versions of shapes online before choosing to download them.

the AIM@SHAPE General Licence for the shape. A user wishing to download the model then has to indicate agreement to this licence as well.

9.5 Helper Tools

Detailed, high quality shapes typically correspond to large files. Prior to committing to a download, most users would like to have a better idea of the shape of the model. Additionally, a stored shape may be too large or too detailed for the user's purposes, in which case the user would prefer to download a simplified version of the shape.

While the metadata and thumbnails accompanying a model in the AIM@SHAPE Shape Repository give a rough idea of its shape and complexity, a true idea can be had only by directly viewing the 3D shape itself. Therefore, a lightweight, Java based viewer, shown in Figure 9.8, has been incorporated into the repository that allows users to view stored shapes that are represented as manifold triangle meshes. For the same shapes, a more advanced viewer [Danovaro07] based on Java 3D is also available which allows users to interactively and selectively simplify the viewed shape. When the user is satisfied with the simplification results, the resulting shape can be directly downloaded.

The repository also offers staple features like site statistics, FAQs, links to other online repositories and a keyword search that searches through some metadata fields. As it is developed within a larger project, links to other parts of the project are also provided. There is also a link to a short tutorial on the Common Shape Ontology, and a functionality to browse through stored shapes according to their shape types.

Shapes and metadata stored in the Shape Repository are replicated respectively in a dedicated geometric search engine server and a separate Ontology & Metadata Repository (OMR). The first offers functionality to search through stored shapes using other shapes as queries, where search results are stored shapes ranked with respect to similarity to the query shape. The OMR provides a *smart search* that can reason about the metadata and thus be able to handle advanced queries, e.g. to retrieve all triangle mesh models with more than 20,000 vertices. However, issuing queries for *semantic search*, i.e. search based on metadata, is tedious. To facilitate this, a natural language query interface has been incorporated that lets a user enter the query in natural language (English) which is then transparently broken down into a form suitable for the semantic search.

9.6 Discussion

We presented the general principles behind the AIM@SHAPE Shape Repository and believe that these can be helpful in the implementation of any large scale shape library. Maintaining a shape library spurs research into helper tools that automatically analyze contained shapes to extract interesting information for users. The tools presented in other chapters of this thesis are cases in point.

We verify these principles by measuring the popularity of the repository in terms of numbers of visitors and downloaded shapes. Since its launch in August 2004, the repository has had more than two million visitors who have downloaded shapes almost a hundred thousand times. Shapes from the repository have been featured in publications in all major digital geometry related conferences. This suggests to us that the repository is useful and the underlying principles sound.

Chapter 10

Conclusion

The number and popularity of large scale shape repositories is on the rise. With growing interest in 3D shapes from diverse application areas, we can safely assume this trend to hold in the foreseeable future. This necessitates new kinds of tools for the analysis and management of stored shapes. This thesis has presented a few such digital shape processing techniques. Looking at shape processing from this novel view point led us to tackle some problems that were as yet unaddressed, e.g. shape orientation and view transfer, or had received little attention, e.g. best fly and shape complexity. At the same time, it directed us towards classic (surface reconstruction, shape retrieval) as well as recent (best view) problems. We also studied in detail a common shape analysis construct used also in several of our above methods, namely the view sphere.

Our learning based surface reconstruction method from Chapter 4 can be applied to a point set contained in a repository and the obtained reconstruction then added back to the repository in the same group as the original raw data. As the method is statistical, it, on the one hand, offers high robustness to noise and other input artifacts and, on the other, suffers from low speed. However, this can be alleviated in light of current computational advances, namely parallel computing. Our method comprises chiefly of iteratively finding and updating the positions of a *winner* and its neighbors where each winner is independent of others. Such a setting is perfectly suited to the SIMD parallel model implemented by modern GPUs. A GPGPU version of the algorithm will therefore bring the algorithm's running time down to competitive levels while retaining its robustness.

Presenting a few, carefully chosen views of a shape is a very common operation in both the digital and real worlds. Chapter 5 described our proposed techniques to automatically select static and dynamic views of a given shape. We are the first to compile a formal list of requirements for the latter and to compute an animation that satisfies these requirements. In the process, we coined the term “best fly” for the problem. Our methods are motivated by results from psychology and shape perception, and thus provide intuitive shape views.

In the same chapter, we address the problem of 2D shape orientation and present our example based approach to it. Our method relies on a database of correctly oriented shapes and we demonstrate how an existing shape data set can be used for the purpose. Not surprisingly, we need human input to specify correct orientations for shapes. Our method can then clean up redundantly oriented shapes automatically. User input is also required to identify non-orientable shapes to be removed. For query shapes, we find the closest matching exemplar from the data set and transfer its orientation to the query. We note that we were the first to address this problem and that our work was later extended to 3D shapes in [Fu08] where shapes labeled as correctly oriented by a human observer are used to train a classifier. Then, instead of matching a query shape to an exemplar, its target orientation class is found and it is reoriented accordingly.

While maintaining the AIM@SHAPE Shape Repository we found it useful to highlight complex shapes to visitors. For that purpose, complexity or “quality” values were manually added to each shape. Visitors could then sort stored shapes by their quality and quickly access the more complex shapes. We then set out to automatically compute such a quality measure. More specifically, in Chapter 7 we aimed at automatic computation of shape complexity. It turned out that there had been little work on this problem and that too suffered from sensitivity to common shape defects like noise. Once again, we leveraged results from human shape perception to motivate our approach and comparing our results to those of previous methods, we found that our complexity ranking of an example set of shapes was more intuitive and robust.

In addition to the usual keyword search, the AIM@SHAPE Shape Repository also provides a geometric search option whereby shapes in the repository can be sorted according to similarity to a submitted query shape. Shape retrieval methods so far have focused on accuracy of the retrieved list. However, with sizes of repositories expected to grow, efficiency also becomes an important concern. Once again, we are among the first to consider fast end efficient similarity retrieval of 3D shapes. In Chapter 8, we introduced a framework that couples existing shape retrieval techniques with powerful and proven methods from text retrieval. The sub-second query time we achieve on a collection of one million shapes is far beyond anything reported in existing shape retrieval literature. Though our collection of shapes is admittedly a toy example, we believe it aptly simulates a database of similar shapes and that real collections of this size are just around the corner.

Shape repositories provide natural showcases for shape processing tools. This approach also fits with the emerging paradigm of cloud computing whereby users expect more and more services to be provided online. A shape repository should therefore incorporate tools for common shape operations like surface reconstruction, mesh viewing, smoothing and simplification. At the same time, the repository should provide time saving tools to the user that automate menial tasks like entering shape metadata and that enable efficient browsing of shapes through various means and criteria. In Chapter 9, we described how we were able to implement some of our ideas in the AIM@SHAPE Shape Repository. The apparent popularity and success of the repository serves us as a validation of our approach.

10.1 Future trends

We have paced this thesis such that techniques dealing with single shapes appear first followed by techniques that deal with large collections of shapes culminating in the management of a full fledged publicly accessible online shape library. This reflects our vision of the direction taken by research activity in digital geometry research. This trend is closely coupled with the digital shape pipeline, which we reproduce below from Chapter 1. When faster and cheaper computers first started

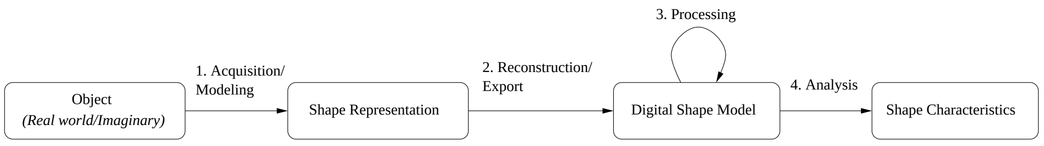


Figure 10.1: The digital shape pipeline, reproduced from Fig. 1.1

coming out, interest arose in the nascent computer graphics community towards the display and analysis of 3D geometry. At that time, this was still a new kind of content and research focus was appropriately on its creation, acquisition and reconstruction. The first general purpose surface reconstruction paper appeared in 1992 [Hoppe92]. As the community now matures, methods for these steps abound and attention has now moved on to novel ways of using and analyzing this content. At the same time, now that we can safely assume a proliferation of such content, the time is also right to think about how to reason about collections of shapes, instead of single shapes in isolation. In a way, this also represents our level of comfort with the content. Early, low level handling of the data of the sort, “How do we reconstruct a triangle mesh from the given point cloud?”, has been replaced by a semantically higher level of reasoning where we can completely abstract out geometric details to process queries like, “Given two shapes representing a dinosaur and a humanoid, which one is more complex?”

Of course, the shift in research activity is not entirely discrete. Just like in today’s age of video streaming and online photo editing, it is still worthwhile to make a better photo camera, there is still ongoing research on the earlier stages of the digital shape pipeline. If anything, their significance is all the more emphasized. However, those stages no longer generate as much interest as they once did, because they have already paved the way to the later stages which are currently newer and hence more exciting.

At the same time, as we venture into higher level processing of 3D content, we necessarily invoke more ambiguity and vagueness. When performing a ranking by shape similarity, is a cup more similar to a fish or an automobile? Is the view of a bottle from the top better than the one from the bottom? These and similar questions are ones that even a human cannot answer with confidence and probably, two different human subjects will end up giving two different sets of answers. This leads to a paradox whereby, when our algorithms set out, for example, to compute “best” views of a given shape, or to perform a “similarity” based ranking, it is not entirely clear what the best view is, and what is meant by similarity.

We realize that we want our algorithms to mimic human behavior; a shape similarity or shape complexity method should rank objects as a human would, a best view or shape orientation method should choose views that a human would. The solution therefore lies in understanding human behavior. That is why a growing amount of digital geometry research is looking into results from human psychology. Our own view based methods presented in Chapters 5, 6 and 7 rely on results from [Blanz99, Koenderink79, Tarr01, Todd04].

10.1.1 A Sociological Approach to Shapes

As large collections of shapes are on the rise, methods leveraging information from them have also begun to emerge. Instead of deeply analysing a single shape, it is less laborious to simply infer the required information from a large, easily accessible population of shapes. Sometimes, such information might not even be deducible from the single analyzed shape. For example, in Section 5.2, we infer the correct orientation of a 2D shape from a larger data set. In [Fu08], a collection of shapes is used to train a classifier which then indicates a correct orientation for a query shape. [Laga07] also trains a classifier with features extracted from a large collection of shapes to perform shape classification, retrieval and best view computation.

It is clear that such methods need to maintain some kind of structure that can both “absorb” and “apply” information. Information pertaining to individual members of the collection should be absorbed and formulated such that it can be readily applied to new, unseen shapes. This is very similar to the human process of deciding based on past experience. And as statistical learning methods have rich parallels with human intelligence [Poggio03], it is not surprising that learning methods are popular in this context.

Another class of methods we see gaining importance in context of shape repositories is those that impose some ordering on the contained shapes, e.g. by ranking them according to their complexities, or sorting them according to similarity to a query shape. Such methods simply provide another view of the contained shapes and quickly give a broad overview of the entire population with respect to different criteria.

10.1.2 Relevance to Shape Repositories

Increasing focus on analysis and processing of digital shapes is good news for shape repositories, which, as we argued earlier in the chapter, provide natural showcases for shape processing tools. In accordance with current computing trends, users would like, even expect, to modify shapes stored in a repository online and view the results to the their satisfaction before downloading the shape to their computer. To serve these needs, a shape repository has to store not just shapes but also a considerable cache of tools that can operate on these shapes quickly and efficiently to deliver results online to a user in real time. In addition, emerging methods that make use of large collections of shapes make us view shape repositories no longer as passive containers of shapes, but as entities in themselves that actively play a part in shape analysis.

While seamless synergy of such tools and repositories is still a distant vision, the need for this synthesis is already present and in order to keep pace with advances in other kinds of media, managers of 3D shape repositories would do well to take note of it. We believe that shape repositories are still in their infancy. With a little above a thousand shape models, the AIM@SHAPE Shape Repository is among one of the largest, general purpose, public shape repositories. This is nowhere close to public libraries of other content, e.g. videos on YouTube or images on Flickr. Thanks to advances made in digital shape research, creation of digital shapes no longer lies in the realm of specialists alone. We therefore foresee an explosion in the number and sizes of shape repositories. At such time, it will be handy to have existing tools like the ones developed in this thesis.

Bibliography

[Abbasi00] S. Abbasi and F. Mokhtarian. Automatic View Selection in Multi-view Object Recognition. In *International Conf. on Pattern Recognition (ICPR'00)*, pages 1013–1016, August 2000.

[Adel'son-Vel'skii62] G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1263, 1962.

[AIM@SHAPE] AIM@SHAPE. http://www.aimatshape.net/.

[Amenta98] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-Skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135, 1998.

[Arbel99] T. Arbel and F. P. Ferrie. Viewpoint Selection by Navigation through Entropy Maps. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 248–254, 1999.

[at Georgia Tech] Large Geometric Models Archive at Georgia Tech. http://www.cc.gatech.edu/projects/large_models/.

[Attalla05] Emad Attalla and Pepe Siy. Robust shape similarity retrieval based on contour segmentation polygonal multiresolution and elastic matching. *Pattern Recognition*, 38(12):2229–2241, 2005.

[Barhak01a] J. Barhak and A. Fischer. Parameterization and reconstruction from 3D scattered points based on neural network and PDE techniques. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):1–16, 2001.

[Barhak01b] JACOB BARHAK AND ANATH FISCHER. Adaptive Reconstruction of Freeform Objects with 3D SOM Neural Network Grids. In *PG '01: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, page 97, 2001.

[Barral00] PIERRE BARRAL, GUILLAUME DORME, AND DIMITRI PLEMENOS. Scene understanding techniques using a virtual camera. In A. de Sousa and J. C. Torres, editors, *Proceedings of Eurographics 2000*, volume 19, 2000.

[Benchmark] PRINCETON SHAPE BENCHMARK. http://shape.cs.princeton.edu/benchmark/.

[Bishop95] CHRISTOPHER M. BISHOP. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[Biswas07] SOMA BISWAS, GAURAV AGGARWAL, AND RAMA CHELLAPPA. Efficient Indexing For Articulation Invariant Shape Matching And Retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[Black98] MICHAEL J. BLACK, GUILLERMO SAPIRO, D. MARIMONT, AND DAVID HEEGER. Robust Anisotropic Diffusion. *IEEE Transactions on Image Processing*, 7(3):421–432, March 1998.

[Blanz99] VOLKER BLANZ, MICHAEL J. TARR, HEINRICH H. BÜLTHOFF, AND THOMAS VETTER. What Object Attributes Determine Canonical Views? *Perception*, 28(5):575–600, 1999.

[Bordoloi05] UDEEPTA BORDOLOI AND HAN-WEI SHEN. View Selection for Volume Rendering. In *16th IEEE Visualization Conference (VIS 2005)*, pages 487–494, 2005.

[Borg87] I. BORG AND J. LINGOES. *Multidimensional similarity structure analysis*. Springer, Berlin, 1987.

[Borg05] I. BORG AND P.J.F. GROENEN. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[Browser] 3D CAD BROWSER. http://www.3dcadbrowser.com/.

[Bülthoff95] H. H. Bülthoff, S. Y. Edelman, and M. J. Tarr. How are three-dimensional objects represented in the brain? *Cerebral Cortex*, 5(3):247–260, 1995.

[Cacheforce] Cacheforce. http://www.cacheforce.com/.

[Chong03] Chee-Way Chong, P. Raveendran, and R. Mukundan. Translation invariants of Zernike moments. *Pattern Recognition*, 36(8):1765–1773, 2003.

[Colin88] C. Colin. Towards a system for exploring the universe of polyhedral shapes. In *Proceedings of Eurographics*, pages 209–220, 1988.

[Commons] Creative Commons. http://creativecommons.org/.

[Cutzu94] F. Cutzu and S. Edelman. Canonical views in object representation and recognition. *Vision Research*, 34(22):3037–3056, 1994.

[Cutzu96] F. Cutzu and S. Edelman. Faithful representation of similarities among three-dimensional shapes in human vision. *Proc. Natl. Acad. Sci., Psychology*, 93(21):12046–12050, 1996.

[Cutzu97] F. Cutzu and M. J. Tarr. The representation of three-dimensional object similarity in human vision. In *In SPIE Proceedings from Electronic Imaging: Human Vision and Electronic Imaging II, 3016*, pages 460–471. SPIE, 1997.

[Danovaro07] Emanuele Danovaro, Laura Papaleo, Davide Sobrero, Marco Attene, and Waqar Saleem. Advanced remote inspection and download of 3D shapes. In *Web3D '07: Proceedings of the twelfth international conference on 3D web technology*, pages 57–60. ACM Press, 2007.

[Databasea] The GAMMA Database. http://wwwc.inria.fr/gamma/.

[Databaseb] The SAMPL Database. http://sampl.ece.ohiostate.edu/data/3DDB/Models/.

[Demand] 3D On Demand. http://www.3dod.org/.

[Denton04] Trip Denton, M. Fatih Demirci, Jeff Abrahamson, Ali Shokoufandeh, and Sven Dickinson. Selecting

Canonical Views for View-Based 3-D Object Recognition. In *ICPR '04: Proceedings of the 17th International Conference on Pattern Recognition*, volume 2, pages 273–276, Cabridge, UK, August 2004. IEEE Computer Society.

[de'Sperati97] CLAUDIO DE'SPERATI AND PAOLO VIVIANI. The relationship between curvature and velocity in two-dimensional smooth pursuit eye movements. *The Jounral of Neuroscience*, 17(10):3932–3945, 1997.

[do Rego07] RENATA L. M. DO REGO, ALUIZIO F. R. ARAÚJO, AND FERNANDO BUARQUE DE LIMA NETO. Growing Self-Organizing Maps for Surface Reconstruction from Unstructured Point Clouds. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN*, pages 1900–1905. IEEE, 2007.

[do Rego09] RENATA L. M. E. DO REGO, HANSENCLEVER DE F. BASSANI, DANIEL FILGUEIRAS, AND ALUIZIO F. R. ARAJO. Surface Reconstruction Method Based on a Growing Self-Organizing Map. In Cesare Alippi, Marios M. Polycarpou, Christos Panayiotou, and Georgios Ellinas, editors, *ICANN (1)*, volume 5768 of *Lecture Notes in Computer Science*, pages 515–524. Springer, 2009.

[Feixas09] MIQUEL FEIXAS, MATEU SBERT, AND FRANCISCO GONZÁLEZ. A unified information-theoretic framework for viewpoint selection and mesh saliency. *ACM Transactions on Applied Perception*, 6(1):1–23, 2009.

[Flash Fire Designs] FLASH FIRE DESIGNS. http://www.flashfire.com/.

[Fraundorfer07] FRIEDRICH FRAUNDORFER, HENRIK STEWNIUS, AND DAVID NISTR. A Binning Scheme for Fast Hard Drive Based Image Search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2007.

[Fritzke93] BERND FRITZKE. Supervised Learning with Growing Cell Structures. In *Advances in Neural Information Processing Systems (7th NIPS Conference 1993)*, pages 255–262. MIT Press, 1993.

[Fritzke94] BERND FRITZKE. A Growing Neural Gas Network Learns Topologies. In *Advances in Neural Information Processing Systems (8th NIPS Conference 1994)*, pages 625–632. MIT Press, 1994.

[Fu08] HONGBO FU, DANIEL COHEN-OR, GIDEON DROR, AND ALLA SHEFFER. Upright orientation of man-made objects. *ACM Transactions on Graphics (SIGGRAPH proceedings)*, 27(3), 2008.

[Gu95] P. GU AND X. YAN. Neural network approach to the reconstruction of freeform surfaces for reverse engineering. *Computer-Aided Design*, 27(1):59–64, 1995.

[Gumhold02] STEFAN GUMHOLD. Maximum entropy light source placement. In *VIS ’02: Proc. of the conference on Visualization ’02*, pages 275–282, Washington, DC, USA, 2002. IEEE Computer Society.

[Hall05] P. M. HALL AND M. J. OWEN. Simple Canonical Views. In *The British Machine Vision Conf. (BMVC’05)*, volume 1, pages 7–16, Oxford, UK, 2005.

[Hastie01] TREVOR HASTIE, ROBERT TIBSHIRANI, AND JEROME FRIEDMAN. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, August 2001.

[Henderson07] J. M. HENDERSON, J. R. BROCKMOLE, M. S. CASTELHANO, AND M. MACK. Visual saliency does not account for eye movements during search in real-world scenes. In R. van Gompel, M. Fischer, W. Murray, and R. Hill, editors, *Eye Movements: A Window on Mind and Brain*. Elsevier, 2007.

[Hoffmann98] M. HOFFMANN AND L. VÁRADY. Free-form Surfaces for Scattered Data by Neural Networks. *Journal for Geometry and Graphics*, 2:1–6, 1998.

[Hoppe92] HUGUES HOPPE, TONY DEROSE, TOM DUCHAMP, JOHN MCDONALD, AND WERNER STUETZLE. Surface reconstruction from unorganized points. In *SIGGRAPH ’92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 71–78, New York, NY, USA, 1992. ACM.

[Iglesias04] A. Iglesias, G. Echevarría, and A. Gálvez. Functional networks for B-spline surface reconstruction. *Future Gener. Comput. Syst.*, 20(8):1337–1353, 2004.

[Isgro05] Francesco Isgro, Francesca Odone, Waqar Saleem, and Oliver Schall. Clustering for Surface Reconstruction. In Bianca Falcidieno and Nadia Magnenat-Thalmann, editors, *1st International Workshop on Semantic Virtual Environments*, pages 156–162, Villars, Switzerland, 2005. MIRALab.

[Itti98] Laurent Itti, Christof Koch, and Ernst Niebur. A Model of Saliency-Based Visual Attention for Rapid Scene Analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, Nov 1998.

[Ivrissimtzis03] Ioannis P. Ivrissimtzis, Won-Ki Jeong, and Hans-Peter Seidel. Using Growing Cell Structures for Surface Reconstruction. In *Shape Modeling International*, pages 78–88, 288. IEEE Computer Society, 2003.

[Jaubert06] Benoit Jaubert, Karim Tamine, and Dimitri Plemenos. Techniques for off-line scene exploration using a virtual camera. In *International Conference 3IA'06*, May 2006.

[Jenke06] Philipp Jenke, Michael Wand, Martin Bokeloh, Andreas Schilling, and Wolfgang Strasser. Bayesian Point Cloud Reconstruction. *Computer Graphics Forum (Proceedings of Eurographics 2006)*, 25(3):379–388, 2006.

[Jeong03] Won-Ki Jeong, Ioannis P. Ivrissimtzis, and Hans-Peter Seidel. Neural Meshes: Statistical Learning Based on Normals. In *Pacific Conference on Computer Graphics and Applications*, pages 404–408. IEEE Computer Society, 2003.

[Johnson97] A. Johnson. *Spin-Images: A Representation for 3D Surface Matching*. PhD thesis, Carnegie Mellon University, 1997.

[Jr.87] J. E. Dennis Jr. and D. J. Woods. New Computing Environments: Microcomputers in Large-Scale Computing. *SIAM*, 7, 1987.

[Kamada88] TOMIHISA KAMADA AND SATORU KAWAI. A simple method for computing general position in displaying three-dimensional objects. *Computer Vision, Graphics and Image Processing*, 41(1):43–56, 1988.

[Kamila05] N. K. KAMILA, S. MAHAPATRA, AND S. NANDA. Invariance image analysis using modified Zernike moments. *Pattern Recogn. Lett.*, 26(6):747–753, 2005.

[Karypis98] G. KARYPIS AND V. KUMAR. MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Version 4.0. In *http://www-users.cs.umn.edu/~karypis/metis/*. Univ. of Minnesota, Dept. of Computer Science, 1998.

[Keogh06] EAMONN KEOGH, LI WEI, XIAOPENG XI, SANG-HEE LEE, AND MICHAIL VLACHOS. LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 882–893. VLDB Endowment, 2006.

[Khotanzad90] A. KHOTANZAD AND Y. H. HONG. Invariant Image Recognition by Zernike Moments. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(5):489–497, 1990.

[Kim08] YOUNGMIN KIM AND AMITABH VARSHNEY. Persuading Visual Attention through Geometry. *IEEE Transacations on Visualization and Computer Graphics*, 14(4):772–782, 2008.

[Knopf04] G. K. KNOPF AND A. SANGOLE. Interpolating scattered data using 2D self-organizing feature maps. *Graphical Models*, 66(1):50–69, 2004.

[Kobbelt99] LEIF KOBBELT, JENS VORSATZ, ULF LABSIK, AND HANS-PETER SEIDEL. A Shrink Wrapping Approach to Remeshing Polygonal Surfaces. *Computer Graphics Forum (Proceedings of Eurographics 99)*, 18(3):119–130, 1999.

[Koenderink79] J. J. KOENDERINK AND A. J. VAN DOORN. The Internal Representation of Solid Shape with Respect to Vision. *Biological Cybernetics*, 32(4):211–216, 1979.

[Laga07] HAMID LAGA AND MASAYUKI NAKAJIMA. A boosting approach to content-based 3D model retrieval. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 227–234. ACM, 2007.

[Latecki00] L. J. LATECKI, R. LAKÄMPER, AND U. ECKHARDT. Shape Descriptors for Non-rigid Shapes with a Single Closed Contour. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 424–429, 2000.

[Lee04] JINHO LEE, BABACK MOGHADDAM, HANSPETER PFISTER, AND RAGHU MACHIRAJU. Finding Optimal Views for 3D Face Shape Modeling. In *6th IEEE Int. Conf. on Automatic Face and Gesture Recognition*, pages 31–36, 2004.

[Lee05] CHANG HA LEE, AMITABH VARSHNEY, AND DAVID W. JACOBS. Mesh Saliency. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3):659–666, 2005.

[Liu07] YU-SHEN LIU, MIN LIU, DAISUKE KIHARA, AND KARTHIK RAMANI. Salient critical points for meshes. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 277–282. ACM, 2007.

[Lowe99] DAVID G. LOWE. Object Recognition from Local Scale-Invariant Features. In *ICCV*, pages 1150–1157, 1999.

[Luo05] J. LUO AND M. BOUTELL. Automatic Image Orientation Detection via Confidence-Based Integration of Low-Level and Semantic Cues. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(5):715–726, 2005.

[Martinetz94] THOMAS MARTINETZ AND KLAUS SCHULTEN. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[Mitra06] N. J. MITRA, L. GUIBAS, AND M. PAULY. Partial and Approximate Symmetry Detection for 3D Geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 06)*, 25(3):560–568, 2006.

[Modelsa] THE123D MODELS. `http://www.the123d.com/`.

[modelsb] VRML MODELS. `http://www.ocnus.com/models/models.html`.

[Mokhtarian96] F. MOKHTARIAN, S. ABBASI, AND J. KITTLER. Robust and efficient shape indexing through curvature scale space. In *British Machine Vision Conference*, pages 53–62, Edinburgh, UK, 1996.

[Mokhtarian00] FARZIN MOKHTARIAN AND SADEGH ABBASI. Automatic Selection of Optimal Views in Multi-view Object Recognition. In *Proceedings of the British Machine Vision Conf. (BMVC'00)*, pages 272–281, 2000.

[Mokhtarian03] F. MOKHTARIAN AND M. BOBER. *Curvature Scale Space Representation: Theory, Applications and MPEG-7 Standardization*. Kluwer Academic, 2003.

[Mortara09] MICHELA MORTARA AND MICHELA SPAGNUOLO. Technical Section: Semantics-driven best view of 3D shapes. *Computers and Graphics*, 33(3):280–290, 2009. Proceedings of IEEE SMI.

[Mostafa99] MOSTAFA G.-H. MOSTAFA, SAMEH M. YAMANY, AND ALY A. FARAG. Integrating Shape from Shading and Range Data Using Neural Networks. *cvpr*, 02:2015–2020, 1999.

[Mukundan98] R. MUKUNDAN AND K. R. RAMAKRISHNAN. *Moment Functions in Image Analysis: Theory and Applications*. World Scientific Publishing Co Pte Ltd., Singapore, September 1998.

[Mumford99] DAVID MUMFORD. The dawning of the age of stochasticity. In V. I. Arnold, M. Atiyah, P. Lax, and B. Mazur, editors, *Mathematics: Frontiers and Perspectives 2000*, pages 197–218. American Mathematical Society, 1999.

[Nelder90] J. A. NELDER AND R. MEAD. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–313, 1990.

[Nistér06] D. NISTÉR AND H. STEWÉNIUS. Scalable Recognition with a Vocabulary Tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2161–2168, June 2006.

[Osada02] ROBERT OSADA, THOMAS A. FUNKHOUSER, BERNARD CHAZELLE, AND DAVID P. DOBKIN. Shape distributions. *ACM Trans. Graph.*, 21(4):807–832, 2002.

[Page03]	DAVID L. PAGE, ANDREAS KOSCHAN, SREENIVAS R. SUKUMAR, B. ROUI-ABIDI, AND MONGI A. ABIDI. Shape analysis algorithm based on information theory. In *ICIP (1)*, pages 229–232, 2003.
[Patané06]	G. PATANÉ. SIMS: a Multi-Level Approach to Surface Reconstruction with Sparse Implicits. In *IEEE International Conference on Shape Modeling and Applications*, pages 222–233, 2006.
[Philbin07]	J. PHILBIN, O. CHUM, M. ISARD, J. SIVIC, AND A. ZISSERMAN. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.
[Philbin08]	J. PHILBIN, O. CHUM, M. ISARD, J. SIVIC, AND A. ZISSERMAN. Lost in Quantization: Improving Particular Object Retrieval in Large Scale Image Databases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
[Plemenos96]	DIMITRI PLEMENOS AND MADJID BENAYADA. Intelligent Display in Scene Modeling. New Techniques to Automatically Compute Good Views. In *Proceedings of GraphiCon*, 1996.
[Podolak06]	JOSHUA PODOLAK, PHILIP SHILANE, ALEKSEY GOLOVINSKIY, SZYMON RUSINKIEWICZ, AND THOMAS FUNKHOUSER. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, pages 549–559, 2006.
[Poggio90]	T. POGGIO AND F. GIROSI. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.
[Poggio03]	T. POGGIO AND S. SMALE. The mathematics of learning: Dealing with data. *Amer. Math. Soc. Notice*, 50(5):537–544, 2003.
[Polonsky05]	OLEG POLONSKY, GIUSEPPE PATANÉ, SILVIA BIASOTTI, CRAIG GOTSMAN, AND MICHELA SPAGNUOLO. What's in an Image? *The Visual Computer*, 21(8–10):840–847, 2005. Proc. of Pacific Graphics 2005.

[Repositorya] AIM@SHAPE SHAPE REPOSITORY. http://shapes.aimatshape.net/.

[Repositoryb] THE STANFORD 3D SCANNING REPOSITORY. http://graphics.stanford.edu/data/3Dscanrep/.

[Rigau05] JAUME RIGAU, MIQUEL FEIXAS, AND MATEU SBERT. Shape Complexity Based on Mutual Information. In *SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005*, pages 357–362, Washington, DC, USA, 2005. IEEE Computer Society.

[Roberts98] D. R. ROBERTS AND A. DAVID MARSHALL. Viewpoint Selection for Complete Surface Coverage of Three Dimensional Objects. In *Proceedings of the British Machine Vision Conference*, volume II, pages 740–750, 1998.

[Rossignac05] JAREK ROSSIGNAC. Shape complexity. *The Visual Computer*, 21(12):985–996, 2005.

[Saleem04] WAQAR SALEEM. A Flexible Framework for Learning-based Surface Reconstruction. Master's thesis, Computer Science Department, University of Saarland, 2004.

[Saleem07a] WAQAR SALEEM, OLIVER SCHALL, GIUSEPPE PATANÈ, ALEXANDER BELYAEV, AND HANS-PETER SEIDEL. On Stochastic Methods for Surface Reconstruction. *The Visual Computer*, 23(6):381–395, June 2007. AIM@SHAPE Best Paper Award, 2007.

[Saleem07b] WAQAR SALEEM, WENHAO SONG, ALEXANDER BELYAEV, AND HANS-PETER SEIDEL. On Computing Best Fly. In Mateu Sbert, editor, *Proceedings of the 23rd Spring Conference on Computer Graphics, 2007*, pages 143–149, 2007.

[Saleem07c] WAQAR SALEEM, DANYI WANG, ALEXANDER G. BELYAEV, AND HANS-PETER SEIDEL. Automatic 2D Shape Orientation by Example. In *2007 International Conference on Shape Modeling and Applications (SMI 2007)*, pages 221–225. IEEE, 2007.

[Saleem08] WAQAR SALEEM, GIUSEPPE PATANÉ, MICHELA SPAGNUOLO, BIANCA FALCIDIENO, AND HANS-PETER SEIDEL.

On continuously approximating discretely sampled view descriptors. Technical report, IMATI, 2008.

[Schall05] OLIVER SCHALL AND MARIE SAMOZINO. Surface from Scattered Points: A Brief Survey of Recent Developments. In Bianca Falcidieno and Nadia Magnenat-Thalmann, editors, *1st International Workshop on Semantic Virtual Environments*, pages 138–147, Villars, Switzerland, 2005. MIRALab.

[Schindler07] GRANT SCHINDLER, MATTHEW BROWN, AND RICHARD SZELISKI. City-Scale Location Recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–7, 2007.

[Sebastian02] THOMAS B. SEBASTIAN AND BENJAMIN B. KIMIA. Metric-Based Shape Retrieval in Large Databases. In *Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02)*, pages 291–293. IEEE Computer Society, 2002.

[Seibert92] MICHAEL SEIBERT AND ALLEN M. WAXMAN. Adaptive 3-D Object Recognition from Multiple Views. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(2):107–124, February 1992.

[Shacked01] RAM SHACKED AND DANI LISCHINSKI. Automatic Lighting Design using a Perceptual Quality Metric. *Comput. Graph. Forum*, 20(3), 2001.

[SHREC] SHREC. http://www.aimatshape.net/event/SHREC/.

[Sivic03] JOSEF SIVIC AND ANDREW ZISSERMAN. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *ICCV*, pages 1470–1477, 2003.

[Sivic06] JOSEF SIVIC AND ANDREW ZISSERMAN. Video Google: Efficient Visual Search of Videos. In *Toward Category-Level Object Recognition*, pages 127–144, 2006.

[Sokolov05] DMITRY SOKOLOV AND DIMITRI PLEMENOS. Viewpoint quality and scene understanding. In *VAST 2005: Eurographics Symposium Proceedings.*, pages 67–73, Pisa, Italy, November 2005.

[Sokolov06a] DMITRY SOKOLOV, DIMITRI PLEMENOS, AND KARIM TAMINE. Methods and data structures for virtual world exploration. *The Visual Computer*, 22(7):506–516, 2006.

[Sokolov06b] DMITRY SOKOLOV, DIMITRI PLEMENOS, AND KARIM TAMINE. Viewpoint quality and global scene exploration strategies. In *International Conference on Computer Graphics Theory and Applications (GRAPP 2006)*, Setúbal, Portugal, February 2006. INSTICC - Institute for Systems and Technologies of Information, Control and Communication.

[Song07] WENHAO SONG, ALEXANDER BELYAEV, AND HANS-PETER SEIDEL. Automatic Generation of Bas-reliefs from 3D Shapes. In *2007 International Conference on Shape Modeling and Applications (SMI 2007)*, pages 211–214, 2007.

[Subramanian92] K. SUBRAMANIAN AND D. FUSSEL. A search structure based on k-d trees for efficient ray tracing. Technical Report Technical Report Tx 78712-1188, The University of Texas at Austin, 1992.

[Sukumar06] SREENIVAS R. SUKUMAR, DAVID PAGE, ANDREI GRIBOK, ANDREAS KOSCHAN, AND MONGI A. ABIDI. Shape Measure for Identifying Perceptually Informative Parts of 3D Objects. In *3DPVT*, pages 679–686, 2006.

[Super04] BOAZ J. SUPER. Learning Chance Probability Functions for Shape Retrieval or Classification. In *CVPRW '04: Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop*, page 93. IEEE Computer Society, 2004.

[Takahashi05] SHIGEO TAKAHASHI, ISSEI FUJISHIRO, YURIKO TAKESHIMA, AND TOMOYUKI NISHITA. A Feature-Driven Approach to Locating Optimal Viewpoints for Volume Visualization. In *16th IEEE Visualization Conference (VIS 2005)*, pages 495–502, 2005.

[Tangelder04] JOHAN W. H. TANGELDER AND REMCO C. VELTKAMP. A Survey of Content Based 3D Shape Retrieval Methods. In *SMI*, pages 145–156, 2004.

[Tarr01] M. J. TARR AND D. J. KRIEGMAN. What Defines a View? *Vision Research*, 41:1981–2004, 2001.

[Taubin95] GABRIEL TAUBIN. A signal processing approach to fair surface design. In *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 351–358, 1995.

[Todd04] J. T. TODD. The visual perception of 3D shape. *TRENDS in Cognitive Science*, 8(3):115–121, 2004.

[Turbo Squid] TURBO SQUID. http://www.turbosquid.com/.

[Vailaya00] ADITYA VAILAYA AND ANIL JAIN. Reject Option for VQ-Based Bayesian Classification. *ICPR*, 02:2048, 2000.

[Várady99] L. VÁRADY, M. HOFFMANN, AND E. KOVÁCS. Improved free-form modeling of scattered data by dynamic neural networks. *Journal for Geometry and Graphics*, 3:177–181, 1999.

[Vázquez01] PERE-PAU VÁZQUEZ, MIQUEL FEIXAS, MATEU SBERT, AND WOLFGANG HEIDRICH. Viewpoint Selection using Viewpoint Entropy. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001*, pages 273–280, 2001.

[Vázquez03a] P.-P. VÁZQUEZ, M. FEIXAS, M. SBERT, AND W. HEIDRICH. Automatic View Selection Using Viewpoint Entropy and its Applications to Image-based Modelling. *Computer Graphics Forum*, 22(4):689–700, 2003.

[Vázquez03b] PERE-PAU VÁZQUEZ. *On the Selection of Good Views and its Application to Computer Graphics*. PhD thesis, Dept. LSI, Technical University of Catalonia, Barcelona, 2003.

[Vázquez03c] PERE-PAU VÁZQUEZ AND MATEU SBERT. Fast Adaptive Selection of Best Views. In *Lecture Notes in Computer Science (ICCSA 2003)*, volume 2669, pages 295–305. Springer Verlag, 2003.

[Veltkamp06] R. C. VELTKAMP AND L. J. LATECKI. Properties and Performance of Shape Similarity Measures. In *10th IFCS Conf. Data Science and Classification*, 2006.

[Wang04] Y. M. WANG AND H. ZHANG. Detecting image orientation based on low-level visual content. *Computer Vision and Image Understanding*, 93(3):328–346, 2004.

[Wang08a] DANYI WANG. 3D Shape Complexity using View Similarity. Master's thesis, Computer Science Department, University of Saarland, 2008.

[Wang08b] DANYI WANG, ALEXANDER BELYAEV, WAQAR SALEEM, AND HANS-PETER SEIDEL. Estimating complexity of 3D shapes using view similarity. Research Report MPI-I-2008-4-002, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, September 2008.

[Watt91] ALAN WATT AND MARK WATT. *Advanced Animation and Rendering Techniques*. ACM Press, 1991.

[Weinshall97] DAPHNA WEINSHALL AND MICHAEL WERMAN. On View Likelihood and Stability. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2):97–108, 1997.

[Welke07] KAI WELKE, ERHAN OZTOP, GORDON CHENG, AND RÜDIGER DILLMANN. Exploiting similarities for robot perception. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 3237–3242. IEEE, 2007.

[Welke08] KAI WELKE, TAMIM ASFOUR, AND RÜDIGER DILLMANN. Object separation using active methods and multiview representations. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 949–955. IEEE, 2008.

[Yamauchi06] HITOSHI YAMAUCHI, WAQAR SALEEM, SHIN YOSHIZAWA, ZACHI KARNI, ALEXANDER BELYAEV, AND HANS-PETER SEIDEL. Towards Stable and Salient Multi-View Representation of 3D Shapes. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI2006)*, pages 265–270, Matsushima, Japan, 2006.

[Yang00] MINYANG YANG AND EUNGKI LEE. Improved neural network model for reverse engineering. *International Journal of Production Research*, 38(9):2067–2078, 2000.

[Yu99] Y. YU. Surface reconstruction from unorganized points using self-organizing neural networks. In *IEEE Visualization 99, Conference Proceedings*, pages 61–64, 1999.