Inaugural-Dissertation

zur

Erlangung der Doktorwürde

der

Naturwissenschaftlich-Mathematischen

Gesamtfakultät

der

Ruprecht-Karls-Universität

Heidelberg

vorgelegt von

Dipl-Ing. Mag.rer.soc.oec. Ing. Hubert Mara

geb. in Neunkirchen, Österreich

Tag der mündlichen Prüfung:

28. September 2012

Multi-Scale Integral Invariants for Robust Character Extraction from Irregular Polygon Mesh Data

Advisors:

Prof. Dr. Dr. h.c. mult. Willi Jäger

Prof. Dr. Dr. h.c. Hans-Georg Bock

Zusammenfassung

Hunderttausende von antiken Dokumenten in Keilschrift befinden sich in Museen, und täglich werden weitere bei archäologischen Grabungen gefunden. Die Auswertung dieser Dokumente ist wesentlich für das Verständnis der Herkunft von Kultur, Gesetzgebung und Religion. Die Keilschrift ist eine Handschrift und wurde in den Jahrtausenden vor Christi Geburt im gesamten alten Orient benutzt. Der Name leitet sich von den keilförmigen Eindrücken eines Schreibgriffels in den weichen Beschreibstoff Ton ab. Das Anfertigen von Handzeichnungen und Transkriptionen dieser Tontafeln ist eine langwierige Aufgabe und verlangt nach Unterstützung mittels automatisierter rechnergestützter Verfahren.

Das Ziel dieser Arbeit ist die präzise Extraktion von Schriftzeichen mit variablen Formen in 3D. Die für die Merkmalsextraktion aus 2D-Mannigfaltigkeiten in 3D entscheidenden Schritte sind Kantenerkennung und Segmentierung. Robuste Techniken in der Signalverarbeitung und dem *Shape Matching* benutzen hierfür Integralinvarianten in 2D. In aktuellen Arbeiten werden die Integralinvarianten grob geschätzt, um wenige prägnante Merkmale zu finden, mit denen sich zerbrochene 3D-Objekte zusammensetzen lassen.

Mit dem Ziel der exakten Bestimmung der 3D-Formen von Zeichen, wurde die aus der Bildverarbeitung und Mustererkennung bekannte Verarbeitungskette an 3D-Modelle angepasst. Diese Modelle bestehen aus Millionen von Messpunkten, die mit optischen 3D-Scannern aufgenommen werden. Die Punkte approximieren Mannigfaltigkeiten durch ein irreguläres Dreiecksnetz. Verschiedene Typen von integralinvarianten Filtern in mehreren Skalen führen zu verschiedenen hochdimensionalen Merkmalsräumen. Faltungen und kombinierte Metriken werden auf die Merkmalsräume angewandt, um Zusammenhangskomponenten zu bestimmen. Diese Komponenten stellen die Zeichen genauer als die Messauflösung dar. Parallel zum Design der Algorithmen werden die Eigenschaften der verschiedenen Integralinvarianten analysiert. Die Interpretation der Filterergebnisse sind von großem Nutzen zur Bestimmung von robusten Krümmungsmaßen und zur Segmentierung. Die Extraktion von Keilschriftzeichen wird mit einer *Voronoi* basierten Berechnung von minimalen normalisierbaren Vektordarstellungen vervollständigt. Diese Darstellung ist eine wichtige Grundlage für die Paläographie. Weitere Abstraktion und Normalisierung der Darstellung führt zur Zeichenerkennung. Die Einbettung der Algorithmen in das neu entworfene mehrschichtige *GigaMesh Software Framework* erlaubt eine Vielzahl von Anwendungen. Die Algorithmen nutzen den Speicher effektiv und die Verarbeitungskette ist parallelisiert. Die konfigurierbare Verarbeitungskette hat nur einen relevanten Parameter, nämlich die maximale Größe der zu erwartenden Merkmale.

Die vorgestellten Verfahren wurden an Hunderten von Keilschrifttafeln, so wie weiteren realen und synthetischen Objekten getestet. Repräsentative Ergebnisse sowie Aufwands- und Genauigkeitsabschätzung der Algorithmen werden gezeigt. Ein Ausblick auf künftige Erweiterungen und Integralinvarianten in höheren Dimensionen gegeben.

Abstract

Hundreds of thousands of ancient documents with cuneiform script are known to be in museum collections and are found on a daily basis at archaeological excavations. Analyzing these documents is essential to understand the origins of civilization, legislation and religion. This script is a handwriting and was used for several millennia in the ancient Middle East. Its name is derived from the Latin word for wedge, which is the 3D-shape left by an ancient scribe's stylus, when it was pressed into the soft surface of a clay tablet. Manually drawing and transcribing these tablets is a laborious and tedious task and assistance by an automated and computerized system is highly demanded.

The aim of this thesis is extracting these handwritten characters, i.e. 3D-shapes with high variability. The crucial steps for feature extraction from 2D-manifolds in 3D-space are reliable edge detection and segmentation. This can be achieved using integral invariant filtering, a robust technique known from signal processing and shape matching in 2D-space. In 3D-space the state-of-the-art systems roughly estimate integral invariants for determining small numbers of highly distinctive features to solve puzzles of fractured objects.

In order to precisely determine 3D-shapes of characters the pipeline known from image processing and pattern recognition is adapted for 3D-models. These models have millions of vertices, which are acquired by optical 3D-scanners. The vertices approximate manifolds with an irregular triangular mesh. Different types of integral invariant filtering in multiple scales lead to different high-dimensional feature spaces. Convolutions and combined metrics are applied to the feature spaces to determine connected components i.e. characters with sub-triangle accuracy within a manifold. Concurrently with the design of novel algorithms, the properties of the integral invariants are investigated. Understanding these properties is highly relevant for robust curvature measures and segmentation. The extraction of characters is completed with a *Voronoi* inspired method resulting in a minimal meaningful vector representation. This representation is an important basis for paleography. Further abstraction and normalization lead to character recognition. The embedment of the proposed methods in the novel and layered *GigaMesh* software framework enables a wide variety of applications. Memory efficiency and parallel processing were taken into account in the design of the configurable mesh processing pipeline. The pipeline has only one relevant parameter, which is the maximum size of the expected features.

The proposed methods were tested on hundreds of cuneiform tablets as well as on other objects including synthetic datasets. Representative results are shown and an evaluation regarding accuracy and performance of the algorithms are given. Finally observations about integral invariants in higher dimensions are shown and an outlook is given.

Danksagung

An erster Stelle möchte ich all jenen meinen Dank aussprechen, die mich in den letzten Jahren fernab der alten Heimat unterstützt haben. Allen voran danke ich meinem Doktorvater Professor Willi Jäger für die Freiheit, die er mir gewährt hat, für seinen fachlichen Rat und vor allem für das Vertrauen, das er in mich gesetzt hat. Mein besonderer Dank geht an meine Mentorin Susanne Krömker, die mit viel Geduld und konstruktiven Nachfragen die Fertigstellung meiner Arbeit unterstützte. Weiters danke ich Professor Hans Georg Bock, Michael Winckler und Johannes Schlöder für das Wohlwollen gegenüber meiner Promotion und der gemeinsamen Planung von zugehörigen laufenden und künftigen Projekten.

Mein besonderer Dank geht an die vielen KollegInnen in den Arbeitsgruppen im fünften Stockwerk unseres Zentrums. Im Rahmen gegenseitiger fächerübergreifender Unterstützung entstanden mehrere erfolgreiche Kooperationen und Publikationen. Auch viele Studenten aus den Geisteswissenschaften und der Informatik haben sich bei der Datenakquise und der Weiterentwicklung des *GigaMesh* Framework engagiert und verdient gemacht. Meinem Büromitbewohner Thomas Metz bin ich sehr dankbar für seine ruhige Art und seine politologische Sicht auf technische und soziale Themen. Den Sekretariaten des IWR und der HGS MathComp bin ich zu Dank verbunden, da ich ohne deren Hilfe an den vielen Stolpersteinen der traditionsreichen Bürokratie sicherlich gescheitert wäre. Durch den kompetenten Einsatz der Pressesprecherin Sabine Kluge hat meine Arbeit mehrfach ein großes Presseecho im gesamten deutschsprachigen Raum erhalten.

Bereits in den ersten Tagen an der Volluniversität Heidelberg lernte ich die KollegInnen der Assyriologie kennen, deren skeptische und interessante Fragestellungen eine ständige Motivation dieser Arbeit waren. Die Assur-Forschungsstelle von Professor Stefan Maul war dabei von großer Bedeutung. Recht herzlich bedanke ich mich bei Stefan Jakob, Frauke Weiershäuser und Kamran Zand für die geduldige Beantwortung all meiner Fragen zum alten Orient. Trotz fehlender Kenntnisse der assyrischen Sprache wurde ich nach einiger Zeit als ein Mitglied der Forschungsstelle akzeptiert. Gebührender Dank geht an all die vielen anderen des Zentrums für Altertumswissenschaften, die ich im Laufe der Zeit kennen und schätzen gelernt habe.

Großen Dank möchte ich meinen Eltern aussprechen, die durch unsere ausgiebige Urlaubsreisen an Orte wie Ephesos, Hattuscha, Ani oder den Berg Nimrod in meiner Kindheit einen Grundstein für mein Interesse an den Altertumswissenschaften gelegt haben. Gleichzeitig haben sie mir mit einem *Commodore 64* den Einstieg in die Informatik geschaffen. Für die zusätzliche familiäre Unterstützung durch Edith Seckler und Werner Hirtl bedanke ich mich sehr. Die vorliegende Arbeit widme ich meinen Großvater Erwin Kitzberger, der kurz vor meinem Aufbruch zu neuen Ufern verstorben ist.

Contents

1 Introduction **1**
- 1.1 Reading and Transcribing 3D-Characters . . . 2
- 1.2 Ancient Handwritings using Cuneiform Characters . . . 7
- 1.3 Related Projects . . . 10
- 1.4 Thesis Contribution & Structure . . . 15

2 Textual Information Embedded in 3D **16**
- 2.1 Discrete Surfaces from Optical 3D-Scanners . . . 16
- 2.2 Irregular Non-Manifold Meshes – 2D-Manifolds in 3D-Space . . . 25
- 2.3 Displacement Maps and Curvature Estimation . . . 33
- 2.4 Summary . . . 40

3 MSII – Multi-Scale Integral Invariants **41**
- 3.1 Principles of Curvatures and Integral Invariants . . . 42
- 3.2 Invariant Integration of Volume using Regular Sampling . . . 45
- 3.3 Surface and Line based Integral Invariants . . . 60
  - 3.3.1 Fast Estimation for Semi-Regular Dense Data . . . 63
  - 3.3.2 Efficent and Precise Computation for Irregular Data . . . 64
- 3.4 Volume Integral Invariant Computation using the Divergence Theorem . . . 72
- 3.5 Multiple Scales and Feature Vectors . . . 78
  - 3.5.1 Feature Metrics . . . 81
  - 3.5.2 Convolution of Feature Vectors . . . 82
  - 3.5.3 Combining Metrics . . . 85
- 3.6 Summary . . . 87

4 Feature Extraction embedded in the GigaMesh Framework **89**
- 4.1 Determination of Local Neighborhoods and Feature Outlines . . . 90
  - 4.1.1 Connected Components Labeling . . . 94
  - 4.1.2 Areas of Interest – Feature Boundaries . . . 98
- 4.2 Detecting Characteristic Points in Polygonal Lines . . . 100
  - 4.2.1 Signed Curvature Estimation . . . 101
  - 4.2.2 Sharp Corner Detection using Integral Invariants . . . 104
- 4.3 Extraction of Skeletons – Decoding Character Elements . . . 109
- 4.4 Filter and Visualization Framework . . . 115
  - 4.4.1 Memory Management and Parallel Processing . . . 116
  - 4.4.2 Layered Concept for Data Handling, Filtering and User Interface . . . 120
- 4.5 Summary and the Framework's Overall Workflow . . . 125

5 Results **127**
5.1 MSII Feature Visualization – Raster Images . . . 128
5.2 Geometric Character Representation – Vector Drawings . . . 130
5.3 Normalized Minimal Vector Representation . . . 134
5.4 Evaluation of Performance and Accuracy . . . 137
5.4.1 Computing and Estimating the Surface based Integral Invariant . . 137
5.4.2 Determining the Volume based Integral Invariant Feature Space . . 141
5.4.3 Combined Calculation of the Multi-Scale Integral Invariants 144
5.5 Summary . . . 148

6 Conclusion and Outlook **149**

A Pseudocode and Datastructure Examples **155**
A.1 Synthetic Cunei . . . 155
A.2 Imaginary numbers, $\not\geq$ and < in *C++* . . . 159
A.3 Synthetic 3D-Model of the Cuneiform Sign *kaskal* . . . 160
A.4 Bézier Spline Representation of the Cuneiform Sign *kaskal* . . . 162

B 3D Models **164**
B.1 Sammlung der Assyriologie Heidelberg . . . 164
B.1.1 Curvature Estimation Methods — Inv. No. 88/677 . . . 165
B.1.2 Unwrapping a Cone — Tonnagel, Text Ur-Baba 7 . . . 178
B.1.3 Unwrapping a Spherical Clay Envelope — Tonhülle einer Tafel . . . 179
B.2 Uruk-Warka Sammlung, Heidelberg . . . 180
B.3 Vorderasiatisches Museum, Berlin . . . 186
B.4 Landesmuseum Joanneum, Graz . . . 192
B.5 Key Data of the 3D-Models . . . 194

List of Acronyms **195**

Glossary **197**

Bibliography **201**

Chapter 1

Introduction

Motivated by the requirements of modern epigraphy and paleography, this thesis provides new robust methods to improve the processing and analysis of handwritings. The presented methods allow the automatized extraction of engraved or impressed characters from 3D-models of "writings in 3D", which are as numerous and important as their two dimensional counterparts written with ink on paper. Reading "writings in 3D" requires a different approach than recognition of characters represented by color, because characters have to be visualized first to be recognized by the human eye. Since ancient time this is done using the sun as light source, producing patterns of light and shadow as function of the geometry. Projecting light from the sun – or any other real or virtual light source – is a global method in object space. It is strongly depending on the position of the light source(s) in relation to the overall shape of the surface. To read characters on an object with a curved surface requires the reader to steadily re-position the object to achieve a proper contrast of light and shadow. This means that only parts of the textual information are readable. To overcome this drawback, the surface is processed locally in order to extract all characters with the same filter. This process is independent of illumination and works on arbitrary surfaces. Additionally the presented methods compensate for errors in measurements as well as damages to the objects, which are often weathered over centuries and millennia.

The acquisition of 3D-data, the development of the methods for data processing and the final analysis have been conducted as part of the *Heidelberg University Excellence Initiative* at the *Interdisciplinary Center for Scientific Computing* (IWR). Additionally the presented work was done as part of the *IWR Pioneering Projects* (IPP) and partially supported by the Heidelberg Graduate School Mathematical and Computational Methods for the Sciences (HGS MathComp). The 3D-scanners used for this work were provided by the HGS MathComp and the Heidelberger Akademie der Wissenschaften (HAW).

The objects used in this thesis belong to indexed collections and their numbers are given. The most prominent index is the *Tafelsignatur der Vorderasiatischen Abteilung der Berliner Museen* (VAT), which is an unique index on international level. Other indices from the collections and research projects in Heidelberg have HOS, W and TCH as prefix. For an explanation of the abbreviations see the list of acronyms on page 195.

The following sections describe the motivation; traditional methods to read characters in 3D; related and previous projects; the structure and contribution of this thesis.

1.1 Reading and Transcribing 3D-Characters

Printing and writing on paper is today's common practice, but information was always chiseled, scratched or stamped into all kinds of materials. A common type used throughout history of representing textual information by geometry and not by color are inscriptions. Chiseling characters into stone ensures the durability of important texts. For larger quantities of documents, people achieved lower production costs by using materials like wax or clay. Characters, e.g. imprinted into clay, become visible, when a light source is placed on the top-left of a clay tablet.

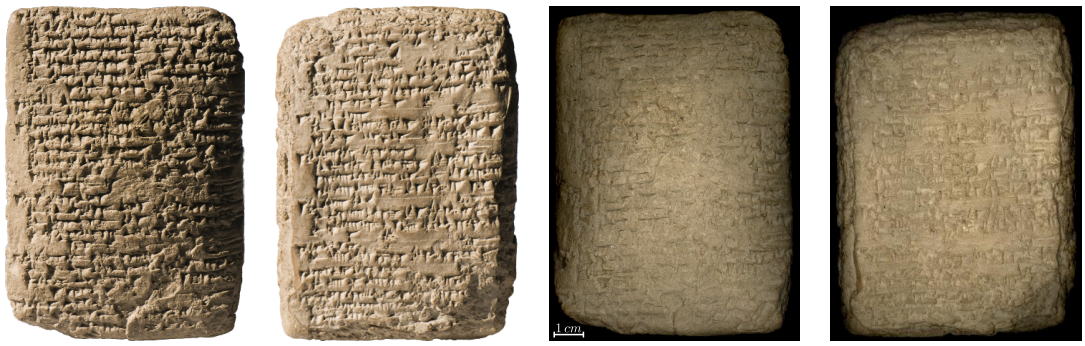
With a photograph using a proper setup of light and tablet we can store this traditional state of visualization used to decipher the contents. Figure 1.1a and 1.1b show typical photographs with one light source located at the top left. The most common "writings in 3D" are inscriptions in virtually any language and can be found on any kind of shape. Depending on the shape and state of preservation reading inscriptions using a lightsource can get time consuming and difficult [Bod01]. Furthermore characters are often camouflaged by discoloration and are impossible to recognize even for experts.

Throughout history inscriptions exist in relatively small numbers compared to handwritings. But cuneiform script [Sod94, DB96] is a major "handwriting in 3D" used for several millennia. There are hundreds of thousands of cuneiform tablets within excavation storages and museums all around the world. As it takes hours, even days to read each individual tablet, most of their contents is barely known. To properly understand the content of a tablet, its characters are typically manually drawn and transliterated.

Assyriologists steadily discover new and important information about ancient life or new text fragments of high political or cultural relevance, e.g. from the famous epic of *Gilgamesh* [Mau05] or the ancient financial crises [Neu07]. To assist in this task we decided to focus on cuneiform script and develop new technologies and new methodologies to automatically detect and extract characters. This shortens the time for acquisition of the tablets and gives an automated draft of the tablets' contents i.e. epigraphic evidence.

Besides using photographs of clay tablets, a cheap alternative solution is the use of flatbed scanners – both methods are common practice in Assyriology. Figure 1.1c shows the front of a cast copy of a clay tablet acquired by a flatbed scanner. Figure 1.1d shows the better preserved backside of the tablet. The 2D-scan takes only a few seconds, while the photograph and the proper position of the light source requires already a few minutes for acquisition.

As the Figures 1.1a and 1.1b indicate, the position and orientation of a directed light source makes the characters visible. These photographs were taken in bright sunlight and therefore the characters on the tablet are easier to distinct than in the images acquired using a flatbed scanner. However at the right-hand side of the photographs shadows discard characters due to the slightly curved surface. This can be avoided by either taking additional photographs or adding light sources – both are no practical solutions, because of two reasons: Multiple photographs will have a large overhead as only a fraction of the



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

Figure 1.1: Photograph of the (a) front and (b) back of a cast copy of the heavily eroded tablet TCH.92,G.127. (c) Front and (d) back acquired using a flatbed scanner. Scale is 1 : 2.5. For manual autography and transcription see [Jak09].

photograph is of practical use. Adding light sources to improve contrast on shadowed areas will also add light to properly illuminated parts, canceling the volitional shadows.

Figure 1.2 shows photographs and their brightness distributions (histograms) of a tablet with round shape. For the first image one light source was used leaving characters in the dark. The second photograph was taken with an additional light source showing some of the missing characters. While the second photograph is considered a good compromise, adding one or more light sources will have the same effect as using a flatbed scanner, where reading is difficult, because no shadows are visible.

The typical workflow for processing "writings in 3D" is shown in Figure 1.3. This Figure also shows that with each step information about the original is reduced to its contents, which then is transformed/translated into a modern language. Reducing and transforming information has to be done in all conscience, because further research heavily depends on the information of these secondary sources. Examples for further research are the analysis of trading relations, politcal events, etc. As a consequence the workflow from the original object to the translation into a modern language should be documented as precise as possible in case a review has to be done and the original object is not avialable – or accessible. The latter is a common case as it often involves travelling and special permissions to get access to original objects. Even by present day, these objects may not be available at all, due to bad storage conditions and looting during modern wars [Kap10].

While access to original objects is and also will be limited to a rather small number of researchers – the opposite is true for the objects' representations. Therefore the schematic workflow from Figure 1.3 indicates that photographs are the most prominent method used so far. The first two steps of the workflow are traditionally called autopsy, because the objects are visually inspected by the human experts. Cast copies have an advantage, when it comes to complex shapes as an autopsy can be performed similar to the original object, but they are not as easy to distribute as photographs. However there are other means of documentation for inscriptions. An example is the making of chalkings,

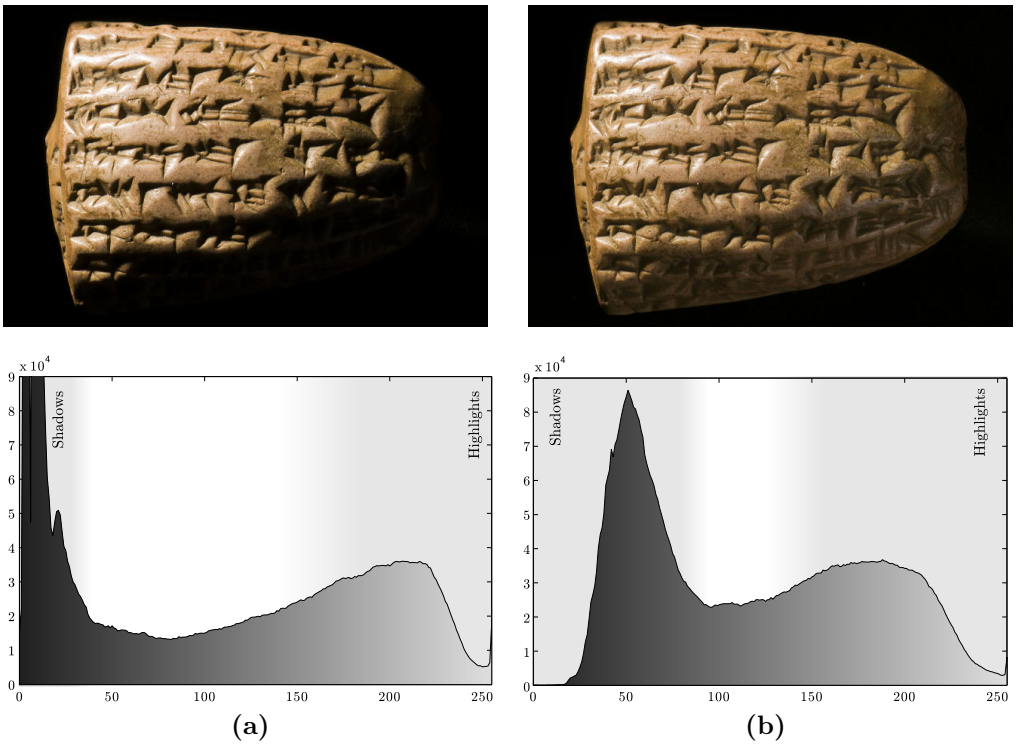


Figure 1.2: Photographs and their histograms of cuneiform tablet HOS G8 illuminated (a) with one light source (b) with a second light source highlightning the lower right corner. The x-axis of the histograms show the brightness of the photographs' pixels $[0 \ldots 255]$. The y-axis show the cardinality of pixels with a specific brightness. The background pixels in black (0) are not taken into account. The canceling of shadows from (a) to (b) can be seen in the histograms: The brightness between pixles considered in the shadowed and in the highlighted parts are narrowed towards 128 (medium brightness).

which are a rapid mean of documentation with a minimum of costs. This technique was used together with methods of two-dimensional image processing in [MHK09] for ancient Chinese sutras carved into rock walls [Led04]. However, the making of chalkings is not possible for the brittle cuneiform tablets, while making of large numbers of cast copies is a virtually impossible task.

Figure 1.4 shows the drawing, the transcription and the transliteration of the cuneiform tablet TCH.92,G.127 as well as a summary of the tablet's contents in German language. This summary already acts as a secondary source and tells us that this tablet is an administrative document assigning rations of barley and seed to families of archers. Additionally it is noted that the amount of barley appears to be too large compared to administrative decisions known from other tablets. The latter demonstrates the cross-linking of information and stresses the importance of correctly interpreting characters. The autopsy in Figure 1.4a also shows areas of uncertainties as dotted areas. Characters written around the corner are shown folded in the autopsy. In [Jak09] it can be seen that even the sides of the tablets contain textual information.

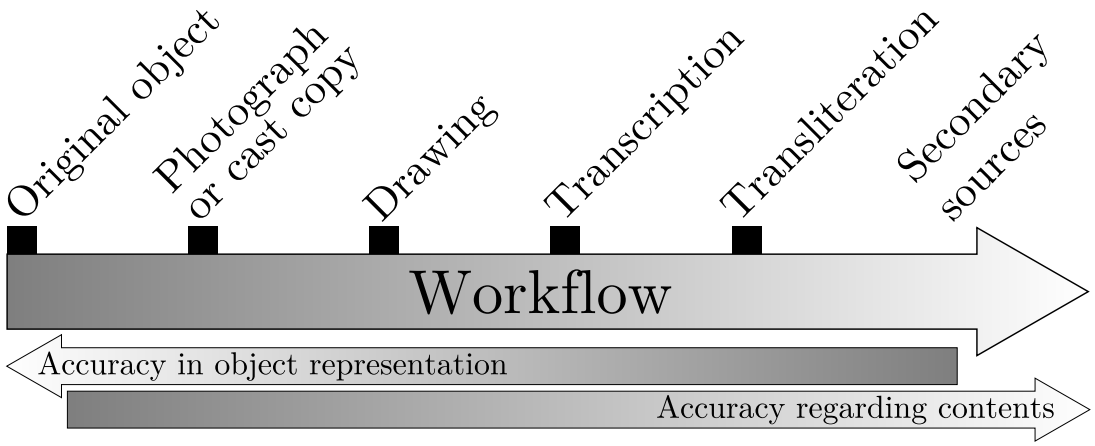


Figure 1.3: Schematic workflow for processing objects containing textual information like cuneiform tablets. Depending on factors like the complexity of an object or its contents, some of the steps may be skipped. The minimum workflow contains only the original object and its transliteration. Secondary sources typically provide analysis of trading relations, political situations, etc.

Summarizing the traditional methods to read "writings in 3D" and disseminate their contents we can conclude:

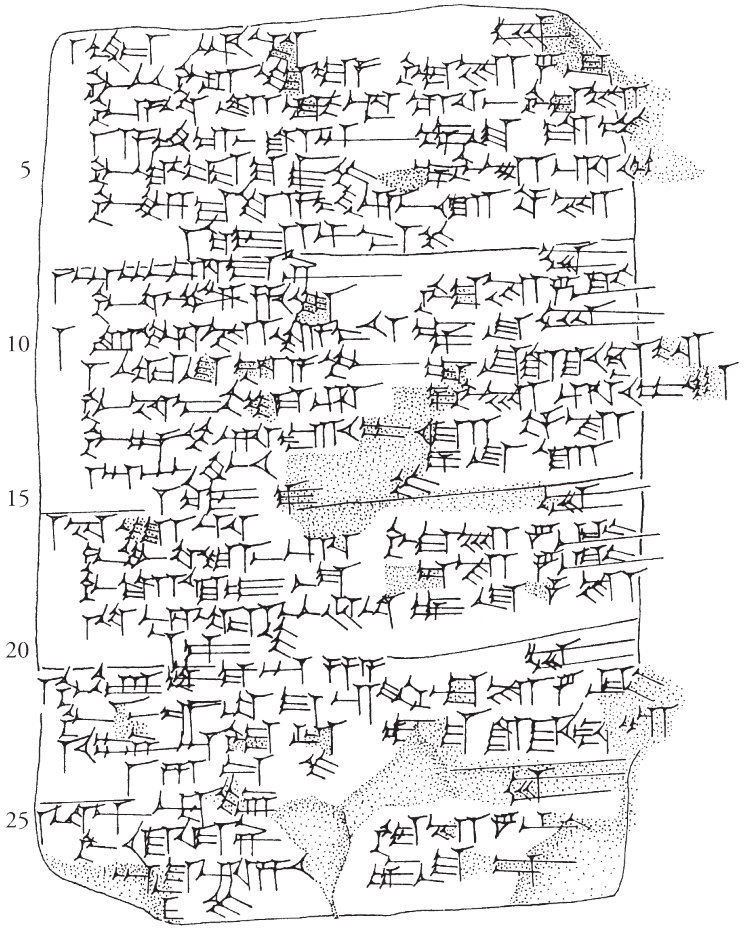
- Cast copies are very useful to an author of a secondary source, but they are nearly as difficult to handle and access as original objects. By the nature of cast copying, the cast copies' surfaces and the characters embedded are less distinct than on the original due to a smoother surface. Instead of distributing cast copies, drawings are published, which already contain an interpretation by the craftsperson. Additionally these drawings are nonreversible projections similar to those used for the following methods.

- Chalkings can be applied to objects with complex shapes, but due to their binary nature (black & white) they are not useful for weathered inscriptions [KMT08].

- 2D-scans are fast, cheap and true to scale, but the light source is fixed and the depth of field is too shallow for objects in general. Non-flat, heavy objects and objects larger than e.g. *DIN A3* ($297 \times 420\,mm$) are very difficult to acquire.

- Photographs are perspective projections onto a plane leaving no possibility to change important settings like the position and direction of light source or the position of the object. They also lack a proper scale and are error prone to lens distortion, depth of field, overexposure and underexposure.

The next section provides information about cuneiform characters followed by a section describing research of related projects including methods using holography, 3D-acquisition and Computer Graphics.



(a)

Ihre Zusammengehörigkeit drückt sich darin aus, dass für sie eine gemeinsame Ration an Gerste und Saatgut berechnet wird. Die Höhe der Gerstezuteilung scheint allerdings problematisch. Selbst drei erwachsene Männer erhalten normalerweise nicht mehr als 3 *sūtu* pro Person und Monat. Hier sind aber wenigstens [1] *emāru* (= 10 *sūtu*) verzeichnet. Dabei besteht noch die Unsicherheit hinsichtlich Altersstufe bzw. Profession bei dem an dritter Stelle Genannten. Eine Ergänzung zu *kukullu* als Variante zu *ša kukulle* (cf. z. B. 70:11.23; 71:11) scheint sich angesichts der vorhandenen Spuren zu verbieten. Im zutreffenden Fall würde die Spanne zwischen der eigentlich zu erwartenden und der tatsächlich berechneten Ration im Übrigen noch größer.

Vs.	[1]NE-[d]*Iš$_8$-tár* BAN	*Nūr-Ištar* Bogenschütze
	[f]*Be-el-tu-ia* DAM-*su* *šá* KI[N]	*Bēltuja*, seine Ehefrau, Arbeiterin
	[f]*A-na-ru-qi-al-si-ši* DAM-*su* [*šá* KIN]	*Ana-rūqi-alsiši*, seine Ehefrau, [Arbeiterin]
	[1]*A-am-mar-ša*-DINGIR DUMU-*šu šá* GABA	*Ammar-ša-ilī*, sein Sohn, Säugling
5	[f]*La-ka-ma-áš-še* [r]DUMU[1].MUNUS-*su tal-mi-*[*tu*]	*Lakamašše*, seine Tochter, »Aufhebende«
	[f]*Im-ma-te-a-am-mar* KI.MIN *pír-su*	*Immāte-ammar*, dto., Kleinkind
	1 ANŠE 2 1/2 SÌLA ŠE	1 *emāru* 2 ½ *qa'u* (ist) seine Gerste(ration)
	[1.d]XXX-*re-ma-ni* BAN	*Sîn-rēmanni* Bogenschütze
	[f]*Pi-il-ḫa-tu* DAM[1]-*su šá* KIN	*Pilḫatu*, seine Ehefrau, Arbeiterin
10	[1]*Ru-us-su-kar-du* DUMU-*šu* BAN	*Russukardu*, sein Sohn, Bogenschütze
	[1]KA-*šu-ṭa*[?]*-a-bu* [r]DUMU[1]-*šu ša ku-kúl-*[r]*lí*[1]	*Pīšu-ṭābu*, sein Sohn, *ša kukulle*
	[f]*Ba-bi-*[r]*šar*[1]*-ri* [DUM]U.MUNUS-*su tal-mi-*[*tu*]	*Bābi-šarri*, seine Tochter, Jugendliche
	[f]*Nap-še-ri-um-mi* [K]I.MIN *pír-su*	*Napšēri-ummī*, dto., Kleinkind
	[1.d]XXX-KAM [D]UMU-*šu* KI.MIN	*Sîn-ēriš*, sein Sohn, dto.
15	1 ANŠE 3 BÁN ŠE	1 *emāru* 3 *sūtu* (ist) seine Gerste(ration)
	[1]*Ba-li-ḫu* BAN	*Balīḫu* Bogenschütze
	[f]*Pu-ut-tu-me-ri* DAM[1]-*su šá* KIN	*Puttumeri*, seine Ehefrau, Arbeiterin
	[f]*Ta-*[r]*ki*[1]*-šu* [D]AM[1]-*su šá* KIN	*Takīšu*, seine Ehefrau, Arbeiterin
	[1]*Na-al-ka-me-en-na* DUMU-*šu* [*p*]*ír-su*	*Nalkamenna*, sein Sohn, Kleinkind
20	NIGIDA 2 BÁN ŠE	8 *sūtu* (ist) seine Gerste(ration)
	[1]*Gu-ú-ug-ma-an-za-a* BAN	*Gugmanzâ* Bogenschütze
	[f]*Me-e-na-ma-gi* DAM[1]-*su šá* KIN	*Mēnamagi*, seine Ehefrau, Arbeiterin
	[1]*Ṣíl-lí-*[d]*Nin-urta* DUMU-*šu ša ku-*[r]*kúl-lí*[1]	*Ṣillī-Ninurta*, sein Sohn, *»Korbträger«*
	NIGIDA 5 SÌLA ŠE	6 *sūtu* 5 *qa'u* (ist) seine Gerste(ration)
	[[1]]*Na-pár-ak-še-er* [r]BAN[1]	*Naparakšer* Bogenschütze
	[f]*Ki-di-ni-*[*t*]*u* DAM[1]-*su šá* K[IN]	*Kidinītu*, seine Ehefrau, Arbeiterin
	[[1]SU]-[d]*Šu-ri-ḫa* DUMU-*šu* [r]BAN[1]	[*Erīb*]-*Šuriḫa*, sein Sohn, Bogenschütze
	[NIGIDA] [r]2 BÁN[1] ŠE	8 *sūtu* (ist) seine Gerste(ration)

(b)

Figure 1.4: (a) Manual drawing from the autopsy, (b) transcription, transliteration of the front side of tablet TCH.92,G.127 by Stefan Jakob according to the workflow. A summary of the whole tablet is included on top of (b) acting as secondary source.

1.2 Ancient Handwritings using Cuneiform Characters

Tablets containing cuneiform script as shown in previous Figures are the most common finds of ancient documents in the Middle East, used for approximatley four millenia before the time of Christ's birth. The vast numbers of tablets and other objects containing this important type of "handwriting in 3D" divide into different writing systems [DB96], which evolved over time. *Akkadian* is often referenced as the Lingua Franca of this time and area. It was strongly influenced by the *Sumerian* and it had *Babylonian* and *Assyrian* as major dialects. The earliest cuneiform script had pictograms (icons), which became abstract over time. Similar to the Japanese language, most of the languages of the ancient Middle East use cuneiform characters as syllabary having hundreds of characters and variants. The archaeological, historical, and linguistic study of the cultures using cuneiform writing is called *Assyriology*. For more information see [Sod94].

The basic element of this handwriting is the wedge – called *cuneus* in Latin – embossed into the wet clay tablet with a stylus. The stylus has a rectangular profile and is imprinted with one corner orientated in forward direction. This leads to tetrahedral impressions, which can also be described as a negative of a triangular pyramid. Assyriologists distinct between two types of wedges:

- Horizontal or vertical wedges are an imprint, where one side is elongated by pulling the stylus in one direction after the imprint. These wedges appear as Y-shaped marks (e.g. 𒁹 or 𒀸) and can also be diagonal at arbitrary angles.
- The *Winkelhaken* is a simple imprint of the stylus. It appears typically larger than the horizontal and vertical wedges and has the shape of a "<" (e.g. 𒌋 or 𒌋). Similar to the wedges the *Winkelhaken* also appears rotated by an arbitrary angle.

Figure 1.5 shows the cuneiform character *Ṣu* (syllabary) consisting of four horizontal wedges, two vertical wedges and two *Winkelhaken*. Three of the vertical wedges have an elongated tail. In cases of characters at the end of a line these wedges can be even longer, because of the *Horror vacui*. From a geometric point of view these wedges and *Winkelhaken* share the same topology, but have a different scaling.

Figure 1.5a is not a manual drawing with pen on paper like Figure 1.4a – it is a digital photograph with a vector drawing as an overlay. First attempts to use such a digital variant of a drawing were already suggested at the beginning of the nineties [NDE91] with the proliferation of computers having *Graphical User Interfaces* (GUIs). The most obvious benefit of such vector graphics is its unlimited scalability for any print size and resolution, when the subjacent photograph is discarded.

Having the wedges and *Winkelhaken* represented as vectors will be an important benefit for future archival indexing and processing of cuneiform tablets, because these graphics contain a large amount of information typically required for *Optical Character Recognition* (OCR) [CKLS07]. Even the non-character parts of the vector drawings contain relevant information for OCR like the areas of uncertainty. These areas are usually represented by arbitrary shapes as polylines filled with a specific pattern (e.g. dotted).

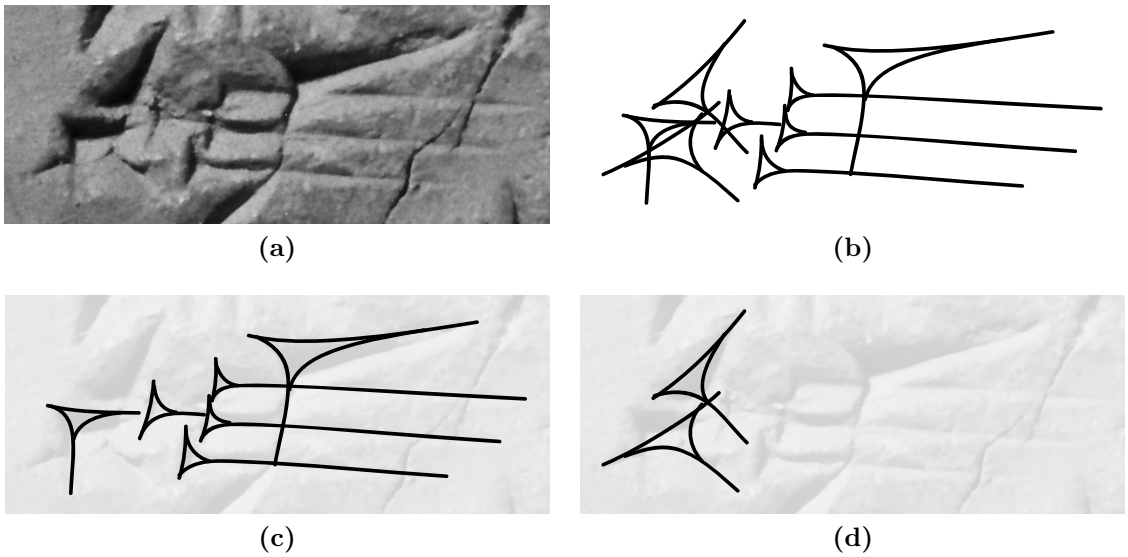


Figure 1.5: Cuneiform character *Ṣu*: (a) Photograph and (b) manual drawing of the character by Frauke Weiershäuser. (c) Vertical wedges, horizontal wedges and (d) *Winkelhaken* adding up the character.

The remaining parts of the vector description describe either cracks of the surface or rulings, which helped the ancient scribes to format their words. Figure 1.6 shows an example for a manual vectorized drawing from present day, which will be published in the series *Keilschrifttexte aus Assur literarischen Inhalts* (KAL) including a transliteration, translation and comments.

The benefit of using vectors to annotate the elements of a character and make it suitable for OCR is shown in Figure 1.7. In this figure the representation is reduced even more towards accuracy of content as each wedge is described by its corner points (arrowheads), the central point (dot) and their connections (line). For cuneiform characters, the central points are the points of the surface with the deepest impression. Additionally the textbook character is shown in contrast to its handwritten counterpart and each wedge is marked with a different color. Such a representation is suitable to use with e.g. *Support Vector Machines* (SVMs) [CV95, SV99] or *Random Decision Forests* [Ho95, Bre01, DBS+10]. To apply such methods we need tremendously large amounts of reliable vector data for training, which have to be computed first.

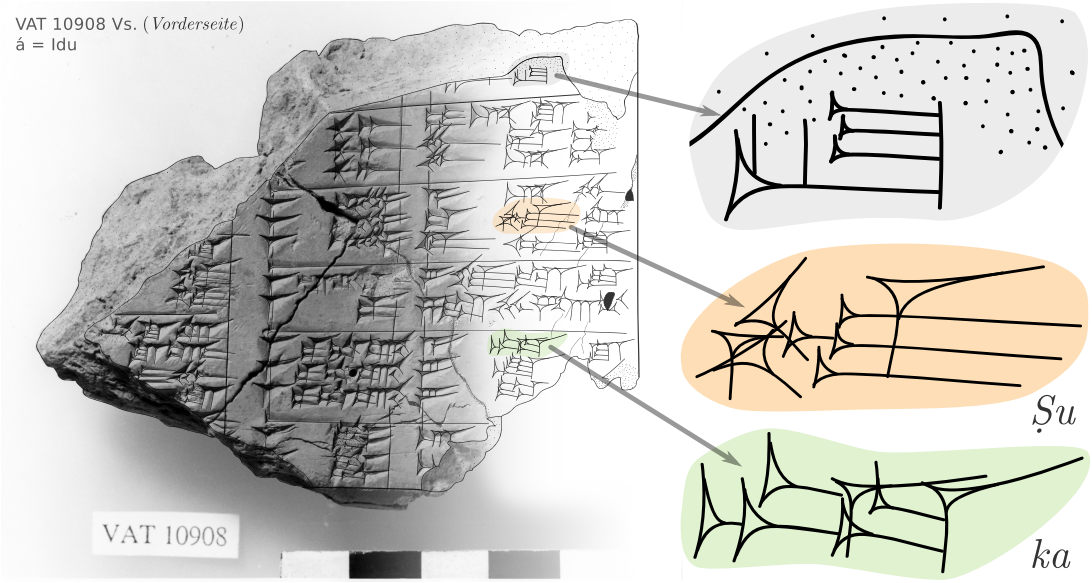


Figure 1.6: Frontside of cuneiform tablet VAT 10908 to appear in KAL. Composite of a digital photograph and manually vectorized cuneiform characters, areas of uncertainties (dense dots) and fracture surfaces (sparse dots). The right hand side shows from top to bottom: uncertain character at a fracture, *ṣu* and *ka*. Drawing by Frauke Weiershäuser, photographed by Marianne Kosanke.

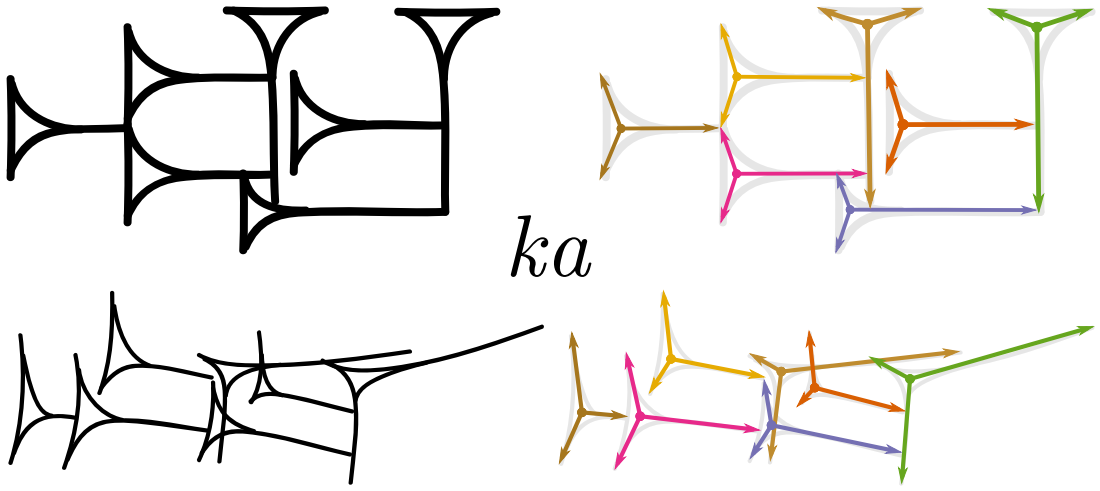


Figure 1.7: Top: Cuneiform character *ka* and its vector representation in textbook shape compared to Bottom: the same character as found on tablet VAT 10908.

1.3 Related Projects

Assyriology began in the middle of the 18th century, when the German mathematician and cartographer Karsten Niebuhr in the service of Denmark sparked an academic interest with the publication of his copies of the Persepolis cuneiform inscriptions[1]. Decipherment began at the end of the 18th century followed by important insights in the next century. By this tradition, Assyriology always involved mathematics and other sciences like geography, astronomy, linguistics, ancient history, etc. In consequence computer science projects using word processing, databases, pattern recognition and image processing follow this tradition. Already in the early days of computer science, processing of cuneiform characters was proposed for using a slide ruler called *Keilschriftmerkschieber* [Spe81], which is a mechanical analog computer. Even the first computers built by Konrad Zuse [Zus93] were used at this time.

This section shows projects using technologies from chemistry to computer science to answer research questions from assyriology.

Chemical coating is one of the oldest technique to improve visibility of cuneiform characters introduced by David I. Owen in the early 1970's [Owe75]. To enhance the contrast for photography the tablets are coated with a thin white layer of vaporized Ammonium Chloride (NH_4Cl), Afterwards the tablet has to be photographed immediately as the Ammonium Chloride dissipates quickly. In these images the characters appear in dark color on a white background. This procedure was improved for acquisition of large groups of tablets as shown in [VGL+05]. As the mist of Ammonium Chloride is toxic, it requires trained personnel and a proper working environment. Therefore chemical coating is rarely used. One of the few examples of its application can be found in [CD02].

Holography is an advanced photographic technique using wave optics (physical optics) to provides three dimensional impressions of objects utilizing the humans binocular vision [SG66]. This technology was used in [BVD+94, WGD+98, DKK+02] to acquire cuneiform tablets including pattern recognition and image processing methods to enhance the visualization of cuneiform characters [Ros97] with the use of Neural Networks [Bis05]. Holography works in principle like any other stereoscopic imaging using two photographs [Wel78]. In consequence we face the same limits as regular photography. Even today, as holography is part of many objects of daily life like banknotes, the process of taking holographic images requires large and power consuming equipment. This renders holography impracticable for the daily work of assyriologists.

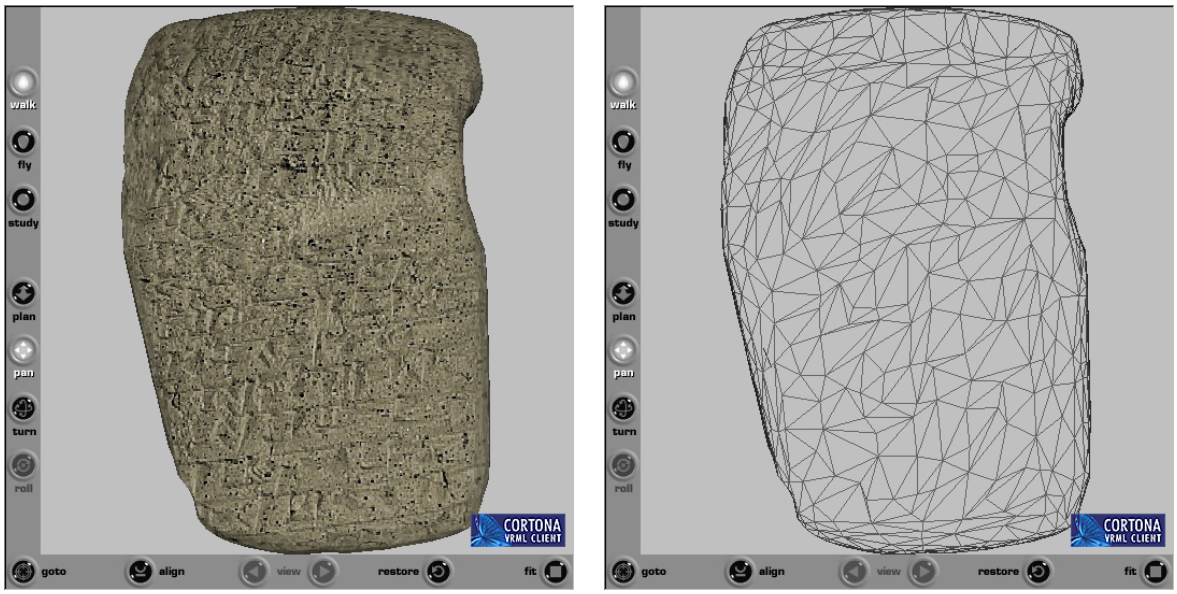
[1]The term inscription for cuneiform script is an exception, because in this case the characters are chiseled into stone mimiking the imprints of the handwritten clay tablets.

Computed Tomography also known as *Computer Tomography* (CT) is a well-known method used for medical checkup of the human body. A prominent example is the examination of ancient Egyptian mummies, which lead to new insights about the ancient life. As a consequence the use of *CT* became more and more popular in other archaeological research areas. [AA05] shows its application for unopened clay envelops holding letters in form of clay tablets with cuneiform script. The image stacks provided by a *Computer Tomograph* can be used to extract the surface of the letter e.g. by segmentation [Jun10] without opening the envelop. As *CT* uses radiation requiring a special building it is typically used for a small number of unique objects of high interest. [PCMA08] shows such a survey for an important cylinder in combination with digital photography, because *CT* images do not contain color information about an object's surface.

Close-Range 3D-Scanners became available with the new millennium, when the *Cuneiform Digital Forensic Project* (CDFP) [WDF+02] started. This project was continued and renamed to *Cuneiform Digital Palaeography Project* (CDP). The CDFP was experimenting with optical 3D-scanners based on the principle of structured light [SM92] consisting of a camera and a light source. As these early generations of 3D-scanners had to use camera sensors of their time, the resolution was not sufficient to capture the geometry of the wedges of the characters. In general the number of wedges outnumbered the measuring points describing the surface. Basically only the overall shape of the tablet was acquired, while the wedges themselves were represented as photographs mapped to the surface [PCK04] – the so-called texture map. Also the main memory and processing speed of the computers available at this time, would not have been sufficient to process 3D-models with higher resolution. The same applied to the bandwidth for data transfer via the Internet. Therefore the first 3D-models appear at websites in *Virtual Reality Modeling Language* (VRML), which was designed for representation of artificial 3D-models consisting of small numbers of primitives compared to the high-resolution of today's 3D-scanners.

Even with all these limitations, the idea to use 3D-scanners to acquire tablets and transfer them via the Internet to scholars and the public lead to the *Digital Hammurabi* [CDS+04, WS03] project. Summarizing the related publications it was an important step to demonstrate a modern approach for acquisition and new visualization methods [AL02, CDS+04]. The project was so successful that the *Johns Hopkins University Applied Physics Laboratory* (JHUAPL) continued to develop and improve a 3D-scanner for acquisition of cuneiform tablets [HBD07]. Despite of the pattern recognition and image processing methods for Holographic images proposed in [Ros97], no further work was conducted regarding character detection.

Figure 1.8a shows the 3D-model with texture map using VRML using the free, but proprietary *Cortona3D* viewer. Figure 1.8b shows the same 3D-model without texture map, revealing the low number of measuring points connected by triangles. These 3D-models were published on the Internet as part the *Digitalisierung der Ostrakasammlungen* project of the Universitätsbibliothek Gießen [Hec66, Lan03] funded by the Deutsche Forschungsgemeinschaft (DFG).



(a) **(b)**

Figure 1.8: VRML of cuneiform inscription *KUG 38* of the (a) with and (b) without texture map [Lan03].

A more recent, low-cost approach is the *Leuven Camera Dome System* (LCDS) as shown in [WVM+05]. This system is based on the principles of *Reflection Transformation Imaging* (RTI) [DCCS06] and *Polynomial Texture Mapping* (PTM) [MGW01]. The data acquired with LCDS provides virtual models with similar limitations as Holographic images, which can represent only one view of object consisting of flat surfaces. Additionally the size of objects acquired with the LCDS is limited by the Dome's radius. As RTIs use stacks of digital images it is possible to extract raster images mimiking line drawings [HW11]. However the use of vector drawings as shown in Figure 1.6 provides better means for automated character recognition. Furthermore vector drawings are already preferred over raster graphics by assyriologists as they can be used in any size for printed publications. It is also difficult to use the data acquired with the LCDS as the files provided by the LCDS in `.CUN` fileformat are binary and proprietary. A closed source viewer for *Microsoft Windows* and examples of files in `.CUN` format are provided at the website[2] of the *Katholieke Universiteit Leuven*.

[2]Last checked: 20th October 2011

Computational Linguistics is an interdisciplinary area of artificial intelligence and linguistics using statistics and/or rule-based modeling of natural languages. This area is known for e.g. automated spell-checking, search engines, automated translations, etc. – typically for modern languages. For cuneiform texts there is one recent statistical approach to decipher ancient languages [SBK10]. It was tested on the rare language of the city of *Ugarit* used only for two centuries until the city was destroyed. *Ugaritic* was chosen, because it uses only 30 cuneiform characters; it is related to modern languages by its structure and the characters are used as letters and not as syllabaries. The input data were already transliterated texts to learn alphabetic mappings, translate cognates, identifying cognates and morphological segmentation. Like in the other projects, the difficult task to identify and extract characters from tablets had to be done manually in painstaking detail by assyriologists.

Database Projects, Cuneiform Fonts and Unicode Thanks to the *Initiative for Cuneiform Encoding* (ICE) [Feu00] at the same time as the CDFP, cuneiform script became part of the *Unicode standard* [Uni06]. The *Unicode standard* assigns a hexadecimal number within the range `0x12000` to `0x123FF` for 824 out of 954 known cuneiform characters [Bor10]. This enables the integration of cuneiform fonts in *Operating Systems* (OSs) and ensures that the user can easily switch between different representations of the same character e.g. in word processing programs.

Figure 1.9 shows the *Unicode character* `0x12157` as manual drawing as font character as it was used in *Assur* and centuries earlier in *Babylonia*. Some *TrueType* fonts became available for LaTeX [Píš99] and commonly used word processing programs. These fonts do not cover all variants of characters developed over several millennia nor can they be used to (en)code characters.

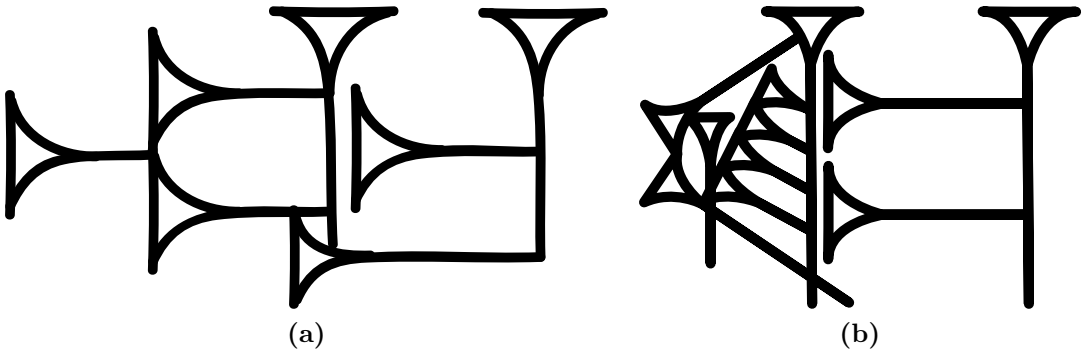
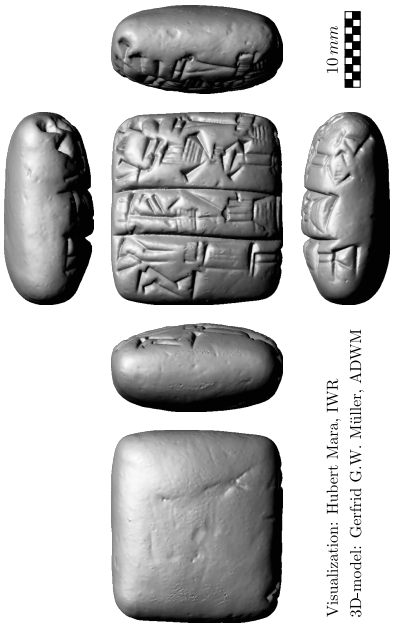


Figure 1.9: Textbook cuneiform character *ka*: (a) as used in *Assur* and (b) the older Babylonian notation as it is used within the *Unicode Standard* [Uni06]. The *Unicode* number is `0x12157`. This character is also referred to as No. *15* regarding the code of [Lab88] and *15-NAa15* regarding [Píš99].

The longest active project providing tablets and related research articles using the Internet is the *Hethitologie Portal Mainz* (HPM) of the Akademie der Wissenschaften und Literatur, Mainz (ADWM) and the *Julius-Maximilians-Universität Würzburg*. The project is coordinated by Gerfrid G.W. Müller, who conducted first experiments about 3D-acquisition of cuneiform tablets. The results – including 3D-models – were published in 2000, 2002 and 2005 in the web portal's section about 3D-technology[3]. By present day the most active project using modern technologies in a *Web 2.0* spirit is the *Cuneiform Digital Library Initiative* (CDLI) of the Max Planck Institut für Wissenschaftsgeschichte (MPIWG) and the *University of California at Los Angeles* (UCLA). The CDLI follows the idea of focusing more on working with the contents of the tablets like the database project from *Göttingen* [GWLU05, Wei09]. This led to a digital workflow for presenting and publishing contents like [Hil08] of the *CDLJ*ournal, still using cast copies, 2D-scans and photographs. Contributing to the CDLI the research groups of the ADWM and the MPIWG bought a modern 3D-scanner and began to compute side-, top- and bottom-views – the so-called *fatcross* – from 3D-models using standard *Computer Graphics* (CG) methods using virtual light. The tablet on the right hand side shows a recent *fatcross* visualization of one of the oldest 3D-models of the HPM. Further examples using the *fatcross* are shown in Appendix B.



[3]http://hethiter.net/3d – last checked: 20th October 2011

1.4 Thesis Contribution & Structure

As the previous Section shows, there was work done regarding acquisition of cuneiform tablets as well as storing, handling, processing and analyzing transcribed texts. During our collaboration with the *Forschungsstelle Assur* in Heidelberg we noticed that the previously shown workflow (Figure 1.3) has overlaps between its stages and can be rearranged. First of all, original objects and cast copies are processed in similar manner and can be combined as – general – objects. Handling and drawing an object already includes preparatory work for transcription.

This means that the acquisition of the object is directly connected to identifying and extracting characters. As the previous section shows, there is virtually no work done to automatically find and extract characters or components (wedges). Figure 1.10 shows the rearranged schematic workflow, indicating the bias of the workload blocking value able working hours of experts for a monotonous task.

Therefore this thesis focuses on character identification and extraction using a novel filtering approach applied to high resolution, non-regular data structures acquired using close-range 3D-scanners. The latter provide a maximum of accuracy in object representation.

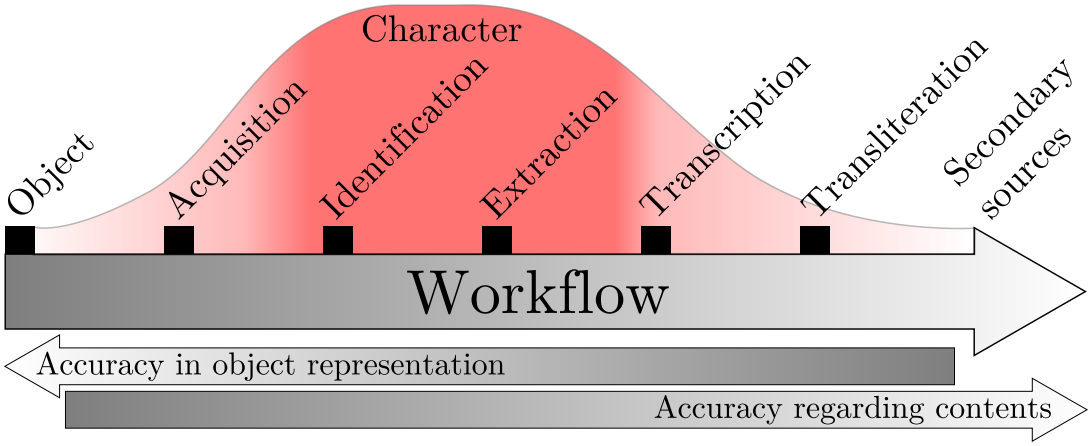


Figure 1.10: Rearranged schematic workflow indicating the essential steps of character identification and extraction leading to transciption.

This document is structured into the following chapters: Chapter 2 describes the necessary structures and geometric methods to handle and acquire textual information embedded in 3D. Processing of the data using local filters leads to Chapter 3, describing the invariant filtering in multiple scales to identify character components (wedges). The extraction of characters and their components are shown in Chapter 4 using a layered software framework, which combines all the algorithms developed within this interdisciplinary work. Results are shown in Chapter 5 for synthetic and real world data, like cuneiform tablets and inscriptions. Finally a conclusion for this thesis and an outlook for future enhancements is given in Chapter 6.

Chapter 2

Textual Information Embedded in 3D

As shown in the introduction the choice of using a close-range 3D-scanner creates more prospects to automatically process cuneiform tablets. By present day industry can supply us with off-the-shelf products having an accuracy as demanded by Assyriology. These 3D-scanners create a distinct data structure to provide as much geometric detail as possible with a minimum use of storage space. To achieve this, the data is irregular and discrete and by its definition it allows various types of non-manifoldness in describing a surface, as shown within this Chapter. This data structure strongly influences the theoretical foundation as well as the methods to be chosen to extract the textual information out of large meshes. The Chapter introduces the mathematical notation and common methods required to process and analyze meshes. Within this Chapter the 3D-scanner used for our practical experiments and to acquire real-world data is shown. As terms used in mathematics and computer science can have the a different meaning, this Chapter indicates their meaning in addition to the Glossary on Page 200. At the end of the Chapter a summary is given.

2.1 Discrete Surfaces from Optical 3D-Scanners

Like [WDF+02] we choose an optical 3D-scanner for data acquisition. As it can be seen as an advanced digital camera, most of the rules from photography apply like shutter time and depth of field. Images from 3D-scanner are called *range images* as the gray value of a pixel represents the distance to the objects surface. The gray values – typically in the range of $[0 \dots 255]$ – are mapped to an absolute distance, which is given in e.g. millimeter (mm). This means that the 3D-scanners can only acquire surfaces in its *Field of View* (FOV) and *range images* have to be stitched together to get a 3D-model. The process of stitching is called registration [BM92] and the most prominent methods are derivatives of the *Iterative Closest Point* (ICP) method, which minimizes the least square error between *range images* [AHB87].

The optics of the scanner also have a depth of field and a focal plane, which defines the so-called working distance. This distance is typically between $50\,cm$ and $2\,m$. Typically it is a fixed distance as optics with a fixed focal length are used, because these optics are

more accurate than a zoom optics, having moving parts and additional lenses. 3D-scanners having a variable focal length on the other hand are more flexible, when the working environment is very cramped. As surfaces of objects can have a wide range of bright and dark colors along the surface, the intensity of structured light and/or exposure time for the *range images* is altered similar to *High Dynamic Range Imaging* (HDRI)[RWPD05].

The differences to photography are: per definition we have two FOVs, where one is from a camera and the second is either from a camera (stereo) or a light source (structured light). Both FOVs have to overlap as shown in Figure 2.1 defining the 3D-scanners FOV. The pyramid frustum shown in green color is the space defined by:

$$x_{min},\, y_{min},\, z_{min},\, x_{max},\, y_{max} \,\text{and}\, z_{max}$$

in which an object can be acquired. For practical reasons the *range image* diagonal $sqrt(x_{min}^2 + y_{min}^2)$ equaling the diagonal of the pyramid's top surface is given by 3D-scanners' manufacturers as FOV in $[mm]$ instead of the focal length of the optics. Most of the 3D-scanners use the camera to acquire color or gray-scale information of the objects surface, which is used as texture-map for the 3D-models. This influences the setup of the illumination as shadows are to be avoided, because otherwise they will be permanently present in the texture-map. Shadows can be computed later on, in case they are required to visualize the 3D model of an object.

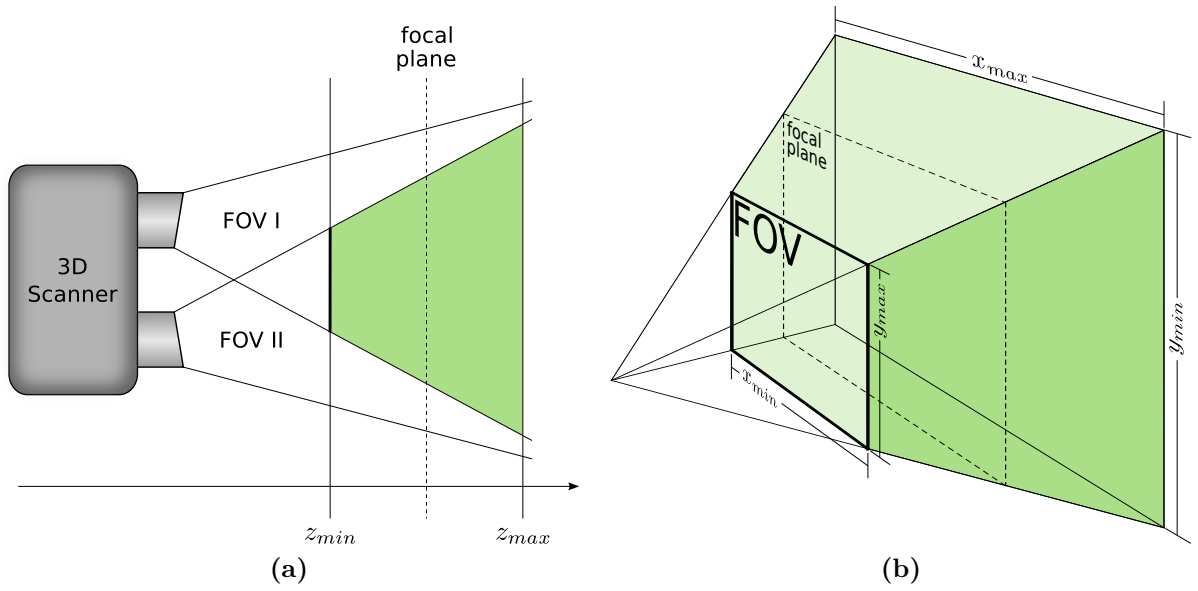


Figure 2.1: (a) Side view of the scheme of an optical 3D-scanner and its Field of View. The scheme is valid for stereo and for structured light. $FOV.I$ is the field of view of camera 1. $FOV.II$ is either the field of view of camera 2 (stereo) or the projection of structured light. The area in green shows the frustum of the overlapping areas defining the volume in which an object can be acquired. (b) Perspective drawing of the FOV and its dimensions [MP13].

Shadowing Effects are a limiting factor for optical 3D-scanners besides the limits of the pyramid frustum of the fields of views. If a surface can be placed within the boundaries of this frustum it may not be acquired completely in the presence of concavities. Enclosed concavities like the interior part of a bottle or undercuttings cannot be acquired as there exists no line of sight between device and surface. However, when working with 3D-scanners it occurs very frequently, that even open concave surfaces are not properly acquired. Having such a surface as shown in Figure 2.2 requires typically three positions for 3D-acquisition, one for the left slope, another for the right slope and a third for the bottom. These positions are sufficient in theory, but the placement of the 3D-scanner has to be done very carefully. While Figure 2.2a and 2.2b are correct, the position in Figure 2.2c is shifted slightly to the right leading to a gap within the acquired surface as shown in Figure 2.2d. These discontinuities of the surfaces called holes, introduce borders to the surface. Practically the hole on the right slope in Figure 2.2d will be even bigger than shown and there would also be a hole on the left slope. This is due to the fact that the lines of sight are almost parallel to the surface causing numerical problems within a 3D-scanner's internal acquisition algorithm.

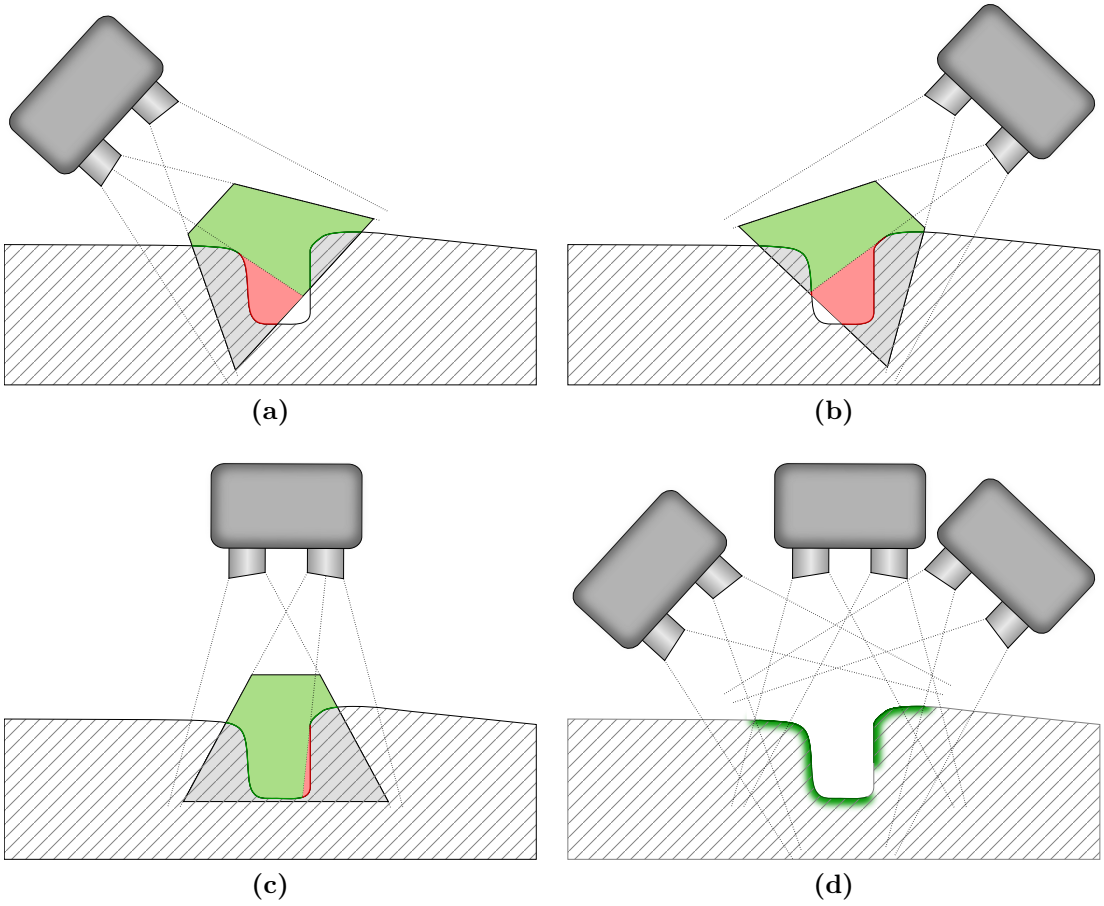


Figure 2.2: Positions of a 3D-scanner to acquire (a) the right flank, (b) the left flank and (c) the bottom of a concavity. The valid acquisition areas are shown in green, the invalid areas in gray and areas with shadowing effects are shown in red. Dotted lines show the limits of the fields of views containing all lines of sight.

Choosing a 3D-Scanner To acquire small human-made objects like cuneiform tablets having wedges with details sized smaller than $1\,mm$, a spatial resolution of $0.5\,mm$ or higher is required to represent these features. This can be proven using a version of the *Nyquist-Shannon sampling theorem* [Sha48, Sha84] for signals with frequencies given in *Hertz*, which is abbreviated as Hz or sec^{-1}:

> If a function $x(t)$ contains no frequencies higher than B *Hertz*, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart.

When we exchange the time t as parameter of signal to x, y as parameters of a surface f and the frequency B to spatial distances dx and dy in mm, we get:

> If a surface $f(x, y)$ contains no features smaller than $min(dx, dy)$, it is completely determined by giving its coordinates at a series of points spaced $1/(2 \cdot min(dx, dy))$ millimeter apart.

which gives us the theoretical minimal spatial resolution required, having a noise-free process of 3D-acquisition. The extension of the *Shannon theorem* from signals to grids leads to the Geometric sampling theorems [Köt08].

Another parameter to be considered for real-world application are the true costs for object acquisition consisting of working time in man-hours per object and initial cost of purchasing a 3D-scanner. These costs have to be minimized, but minimizing the costs for the one-time expense of purchasing a 3D-scanner can result in dramatically increased ongoing expenses for working time. The spatial resolution as well as the expected noise of 3D-scanner corresponds loosely to its price tag. Working time consisting of acquisition time as well as inspecting and post-processing (e.g. *range image* registration) the acquired data, can not be expressed by a formula as simple as the sampling theorem. This is similar to photography too as a better image quality will reduce cost and time for post-processing. Additionally images of high quality often have to be achieved in minimum of time as access to ancient objects in museums is limited – having access to an object is sometimes literally the only chance in a lifetime for acquisition. This fact can not even be expressed in terms of money.

With the *DAVID-Laserscanner* [WMW06] the *Braunschweig University of Technology* demonstrated how a 3D-scanner can be built at minimal cost. Virtually everyone can afford it as it requires a webcam, a line laser and the *DAVID* software package having a total cost of €400[1]. Even though this 3D-scanner uses a laser as a light source, its name is misleading as *Laserscanner* usually refers to devices using the *Time-Of-Flight* (TOF) principle – also known as *Light Detection And Ranging* (LIDAR) [DA09]. Typically *Terrestrial Laser Scanners* (TLSs) [SGFT12] and *Airborne Laser Scanners* (ALSs) [RHP08] are using TOF. The *DAVID-Laserscanner* package provides many possibilities for teaching at schools using 3D-models e.g. in VRML for display in the Internet. Having experimented with this low-cost 3D-scanner we came to the conclusion, that it is not practical for larger

[1] Website of the *DAVID Vision Systems GmbH*, 2011

numbers of highly detailed objects in a timely manner. The consideration of improving the *DAVID-Laserscanner* using e.g. larger optics quickly leads to solutions ten-folding or centupling the costs. To avoid reinventing the wheel, there are off-the-shelf low-cost 3D-scanners available, which can be used to rapidly acquire an object. As these 3D-scanners are designed for rapid acquisition, they are no instrument of measure [BBS10]. Finally we decided to investigate for 3D-scanners used in industry for measuring.

Measuring instruments in industry often have other requirements than for the use in the field of *Cultural Heritage* (CH), where the objects to be measured differ in size and detail; temperature and humidity cannot be controlled on excavation sites; weight and volume for transporting a device by air-travel are important. In our previous projects we already have used industrial 3D-scanners in-situ at excavation sites and museums in Asia Minor [MKS02], the Middle East [Mar06], Europe [MTKZ07, MMS08, BMV09] and Peru [Mar09]. Within these projects 3D-scanners from *Breuckmann*, *Minolta* and *Perceptron* were used – as well as within other projects from cuneiform tablets [WDF+02] to ancient temples [QTK+09].

Deciding on which 3D-scanner is best choice for cuneiform tablets having the most versatile use for future projects cannot be done based on the manufacturers data sheets, as there are no official engineer standards to measure accuracy. Typically manufacturers define their own accuracy specification, like relative and absolute distances in respect on various synthetic objects are given. Sometimes information about accuracy for repeated measurements is added. Alternatively specifications like scanlines per seconds; angular resolution; or sensor resolution in megapixel might be provided. Determining the usability for specific applications by a data sheet is de-facto not possible.

Therefore we invited the previously mentioned manufacturers to test their 3D-scanners on one specific cuneiform tablet with the inventory number 88/677. This tablet was chosen as it reassembles the worst case scenario for readability, because the characters are very small, partially broken-off or filled with dirt. The surface has a very dark color, which absorbs most of the light from the 3D-scanner. An additional challenge was to acquire a cylinder seal impression in the surface, which contains details smaller than the characters. Such seal impressions are important as they give information about the authenticity and the origin of ancient documents. The particular tablet for the test has a size of $44 \times 49 \times 18\,mm$ – a complete visualization is shown in Appendix B. For usability on larger objects with complex shape, the 3D-scanners were tested on a ceramic cast copy of the *Charioteer of Delphi* [Ham41] of the *Museum of Antiquity and Cast Court* of the Institute for Greek and Roman archaeology in Heidelberg.[2]

As photographs of cuneiform tablets are taken with scales and images from flatbed scanners provide a scale, it is evident to draft an accuracy measure for 3D-models related to the scale of 2D-images. Cuneiform tablets are sometimes flat and the accuracy of their 2D-images is typically given as pixels per (scan)line divided by its actual length, known as *Dots Per Inch* (DPI). Having a 3D-model we can estimate a similar measure, by

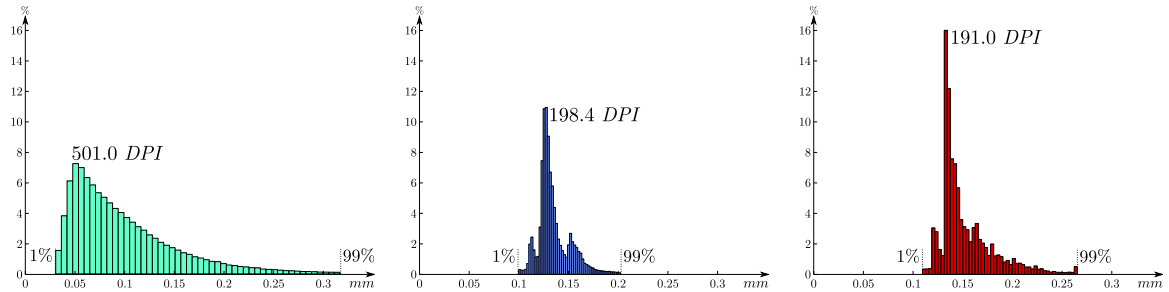
[2]Thanks to Prof. Diamantis Panagiotopoulos and Dr. Hermann Pflug

computing the area of a 3D-model's surface divided by the cardinality of the measuring points ($points/mm^2$ or mm^{-2}). This measure is an average as the measuring points of a 3D-model are not necessarily uniformly distributed like the pixels of an image. As 3D-scanners use a cameras, we can assume that the measuring points are approximately uniform distributed over the surface and therefore we can convert mm^{-2} to DPI (or in^{-1}) by extracting the root and scaling from mm to $inch$. Table 2.1 shows the average resolution as well as the maximum, 1% quantile and 99% quantile of the locally computed accuracy per measuring point of the tested 3D-scanners.

		Resolution $[DPI]$			
3D-scanner		Average	Maximum	Quantile	
Manufacturer	Model			1%	99%
Breuckmann	smartSCAN	305,8	501,0	80,2	829,1
Minolta	RANGE5	261,9	198,4	125,6	254,3
Perceptron	Infinite SC	229,0	191,0	95,7	232,0

Table 2.1: Estimated resolutions of the tested 3D-scanners in DPI assuming a flat surface with equidistant measuring points. The *RANGE5*'s data has such extreme outliers, that the globally computed average is shifted beyond the 99% quantile.

An alternative representation for resolution is the distance between adjacent points of the surface. Therefore it is computed in the same way as the values given in Table 2.1, showing the reciprocal value using mm as unit. Figure 2.3 shows the distribution of these distances for the $[1\%, 99\%]$ quantile range with the maximum in DPI for the tested 3D-scanners.



(a) **(b)** **(c)**

Figure 2.3: Distributions of distances of adjacent measuring points in mm for the same object acquired with (a) *Breuckmann smartSCAN*, (b) *Minolta RANGE5* and the (c) *Perceptron Infinite SC* 3D-scanner. Smaller distance means higher resolution.

The values for average resolution in Table 2.1 already indicate that the *Breuckmann* 3D-scanner appears to be more accurate than the others. As it is the most expensivest, it leaves the question if one of the other 3D-scanners is sufficient for our work. This can be examined by an autopsy of the surface as shown in Figure 2.4. This Figure shows the same part of the tablet rendered with the same, solid surface color (gray-level 73.3%) and the same illumination. The lower images show the enlarged top-left section of the tablet, with the edges of all the triangles.



(a) (b) (c)

Figure 2.4: Details of the cuneiform tablet Inv. No. 88/677 acquired using (a) a *Breuckmann smartSCAN*, (b) a *Minolta Range5* and (c) a *Perceptron Infinite SC* 3D-scanner. The upper images show an area of $26 \times 20\,mm$ (scale $2:1$). The lower images show the faces of the mesh of the top-left corner covering an area of $11.1 \times 5\,mm$ (scale $4.5:1$).

In comparison with the other 3D-models, the 3D-model of the *smartSCAN* shows the details of the characters much more distinct. It is also the only image, where the cylinder seal impression on the top-left corner can be seen. This impression features smaller characters appearing as grid-like concave structure. Additionally the mesh appears to be reduced for parts with low curvature preserving higher resolutions for surface details. This corresponds with the distribution shown in Figure 2.3a. The surfaces from the *RANGE5* and the *Infinite SC* show the same level of detail, which corresponds to the similar quantile range and average of their resolution. It appears that the data from *Minolta* had smoothing and sub-sampling [PR08] applied leading to least distinct representation of surface details. Therefore the effective sensor resolution is either lower or the post-processing was done improperly. The surface from *Infinite SC* shows minor resolution changes depending on curvature partially providing equidistant points. Smoothing to remove noise appears not to be applied.

Summarizing Figure 2.3, 2.4 and Table 2.1, we can conclude that values for average resolution may only give a hint about the smallest details a 3D-scanner can acquire. We choose to use *Breuckmann smartSCAN* as it provided sufficient resolution and colored texture-maps, which is important for other CH objects like polychrome ceramics [MKT08]. The *Minolta RANGE5* can capture only gray-scale values, while the *Perceptron Infinite SC* does not acquire any color value at all. For complex objects with slightly larger details than cuneiform, the *Infinite SC* has a clear superiority, because of its mechanical arm. This arm has seven degrees of freedom and captures the position of the camera using it for *range image* registration. The result is a very fast and easy workflow to acquire objects like statues.

Optics and Spatial Resolution 3D-scanners can be configured with different optics (wide, middle, tele) having either a larger FOV and lower resolution or vice-versa. For many archaeological finds the FOV of 150 mm of the *Breuckmann smartSCAN* has proven to be the best choice to acquire small details like fingerprints. When objects contain smaller details, like seal imprints and toolmarks, the FOV with 60 mm is recommended. The FOVs of 470, 600 and 1.000 mm are suitable to acquire larger objects like reliefs and statues – even whole temples [QTK+09]. Having a 3D-scanner based on $5,0$ megapixels cameras we can achieve $\approx 1,000\ DPI$ for the FOV of 60 mm, $\approx 300\ DPI$ for the FOV of 150 mm down to $\approx 100\ DPI$ for the larger FOVs. Recently a FOV of 35 mm has been added for forensic sciences, which increases the resolution up to $\approx 2,500\ DPI$. This is equal to a spatial resolution below the 10 μm-range and gives about four times smaller details than in our initial tests.

Figure 2.5 shows the *smartSCAN-3D-HE* during acquisition using the 60 mm FOV projecting the structured light. Such a single 3D-scan requires a few seconds independent of the chosen optics. For first experiments with cuneiform tablets, the whole acquisition workflow took between 5 and 45 *minutes* depending on the size of the tablet as well as the quality, condition and utilization/density of the cuneiform characters.

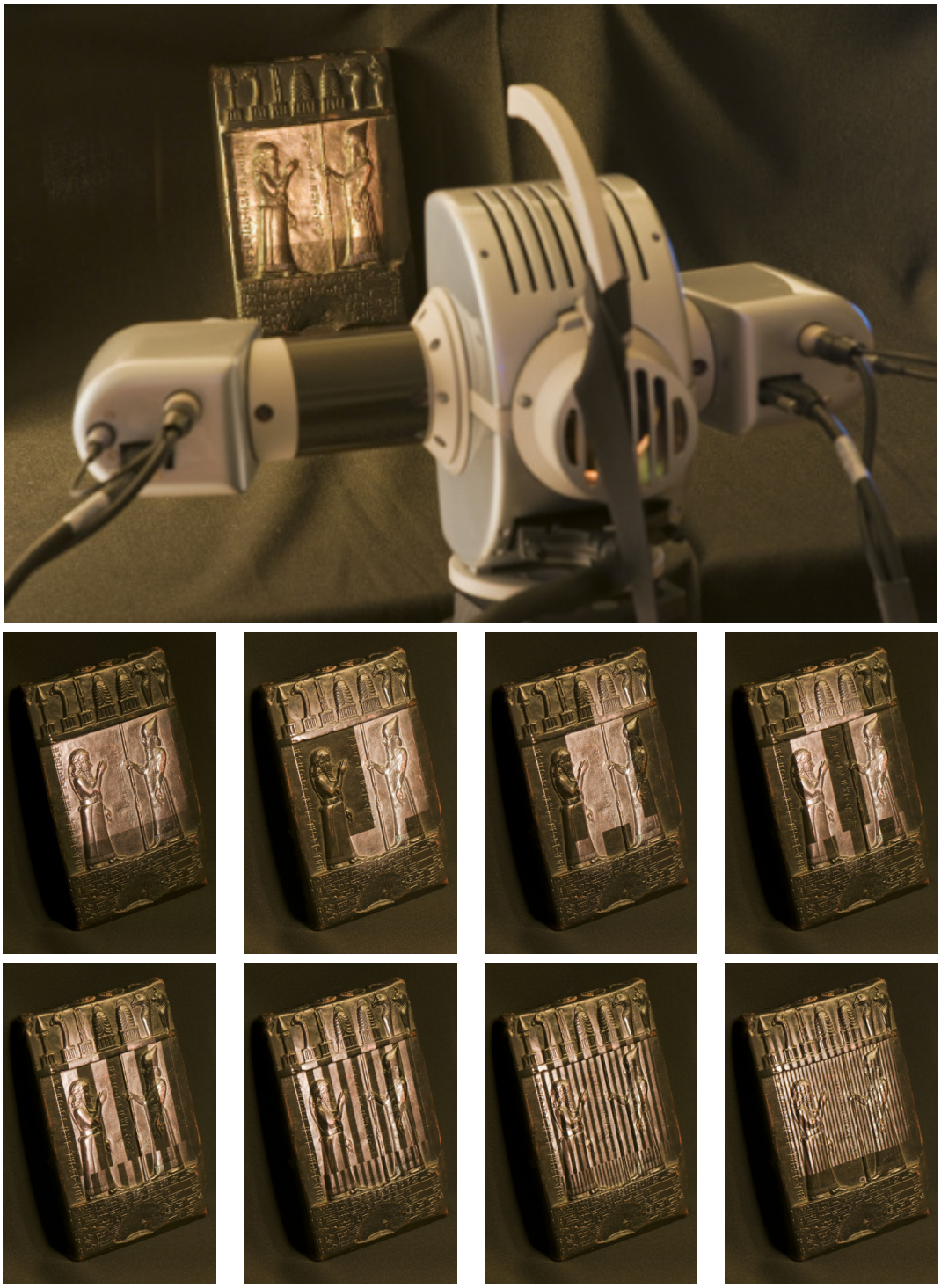


Figure 2.5: 3D-acquisition of *Inv. No. BM 90922, King, Boundary Stones Nr. XXVIII.* The image on the top shows a *Breuckmann smartSCAN-3D-HE* using the optics with a field of view (FOV) of 60 mm. The lower images show the structured light patterns projected on the object during the acquisition.

2.2 Irregular Non-Manifold Meshes and 2D-Manifolds in 3D-Space

Choosing an optical 3D-scanner for data acquisition is also a choice for a surface-based data representation. Virtually all the software packages accompanying 3D-scanners export a 2D-non-planar, manifold and non-regular surface mesh consisting of vertices connected as triangles, and having an optional texture map. This kind of a surface mesh is often and shortly called 3D-data by *Computer Vision* (CV) [BB82] groups. In the field of geography a similar data structure is called *Triangulated Irregular Networks* (TINs), which also can be acquired with non-optical methods (e.g. LIDAR). As for our test-case the cuneiform script is generally defined by pure geometry and the consideration of a texture-map is per-se not necessary.

We have to mention that a common misunderstanding is that acquired 3D-objects are similar or even identical to modeled 3D-objects – this is not true! Both utilize the same methods for organizing the data, but per-se the concepts of primitives and other simplifications do not exist for acquired 3D-objects. Additionally acquired 3D-objects contain noise, which partially prevents simplification. Furthermore this adds to the complexity as simple objects, which can be modeled with a few primitives, are acquired as point-clouds, which can – and do – have thousands to millions of points. For worst cases even the assumption of a non-manifold mesh[3] may not apply due to incorrect registration of multiple 3D-scans [BR07]. As we know from our previous work, there is always at least a minor number of non-manifold points within our data, which has to be taken care of to achieve a robust system [Mar06]. Reducing the resolution for acquisition contradicts with the *Shannon theorem*, while the rule of thumb from our previous projects is: the resolution for acquisition has to be 5 times the size of the smallest detail to be acquired.

Data Representation

The most primitive element of our data is a measuring point $\mathbf{p}$, having three Cartesian coordinates x, y and z. In Computer Graphics and Computer Vision these points are called vertices, while they are called position vectors in $\mathbb{R}^3$ in *Linear Algebra* (LA). As generally all vertices are unique and in no particular order, we address them by the index i and store them as a list $\mathcal{L}_v$:

$$\mathcal{L}_v = \{\ \mathbf{p}_1,\ \ldots,\ \mathbf{p}_i,\ \ldots,\ \mathbf{p}_{i_{max}}\ \} \text{ with } i_{max} = |\mathcal{L}_v| \text{ and } i = \mathbb{Z}^+ \wedge i \leq i_{max} \tag{2.1}$$

The indexing in Equation 2.1 corresponds to the usage of 1 to address the first element of a list. Alternatively the first element can also be addressed using 0 to address the first element of a data structure in programming languages like C/C++: $i = \mathbb{Z}_0^+ \wedge i < i_{max}$. For compliance with textbook mathematics we use 1 to address the first element of a data structure, vector, etc. for all other equations.

[3] A mesh having more than two triangles connected by one edge.

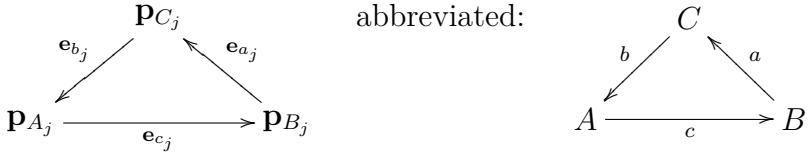
$|\mathcal{L}_v|$ is the cardinality of the list of vertices, while $|\mathbf{p}_i|$ is the length of the position vector. In case of position vector, its length is equal to the distance of a vertex from the origin $\mathbf{o} = (0,0,0)^T$ of the coordinate system: $\sqrt{x_i^2 + y_i^2 + z_i^2}$. These position vectors sampling the surface $\mathcal{M}_2$ in $\mathbb{R}^3$ are identified within homogeneous coordinate system [Gra30] having $w_i = 1$ as fourth coordinate:

$$\mathbf{p}_i = (x_i, y_i, z_i)^T \qquad = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \qquad \text{in } \mathbb{R}^3 \tag{2.2}$$

$$\mathbf{p}_i = (x_i, y_i, z_i, w_i)^T \quad = \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \wedge w_i = 1 \quad \text{in } \mathbb{R}^4 \tag{2.3}$$

The second important class of primitive elements are the so-called faces, which are triangles $\mathbf{t}$ having an orientation provided by the data-structure of the 3D-models, e.g. for visualization using virtual illumination. The triangles are addressed using the index j, while each triangle has three implicit edges: $\mathbf{e}_{a_j}$, $\mathbf{e}_{b_j}$ and $\mathbf{e}_{c_j}$.

$$\mathbf{t}_j \coloneqq \left\{\mathbf{p}_{A_j}, \mathbf{p}_{B_j}, \mathbf{p}_{C_j}\right\} \equiv \{A_j, B_j, C_j\} \qquad \text{abbreviated:} \qquad \mathbf{t} = \{A, B, C\} \tag{2.4}$$



The faces are stored in the list $\mathcal{L}_f$:

$$\mathcal{L}_f = \{\ \mathbf{t}_1,\ \ldots,\ \mathbf{t}_j,\ \ldots,\ \mathbf{t}_{j_{max}}\ \} \text{ with } j_{max} = |\mathcal{L}_f| \text{ and } j = \mathbb{Z}^+ \wedge j \leq j_{max} \tag{2.5}$$

So the list of vertices $\mathcal{L}_v$ and the list of faces $\mathcal{L}_f$ describe the discrete, meshed geometry of the surface $\mathcal{M}_2$ as provided by our 3D-models (triangulation networks):

$$\mathcal{M}_2 \coloneqq \mathcal{L}_\mathcal{M} = \{\mathcal{L}_v, \mathcal{L}_f\} \tag{2.6}$$

This basic list $\mathcal{L}_\mathcal{M}$ will be extended by other surface properties (e.g. color) in the following sections and chapters. In case no $\mathcal{L}_f$ is provided, it is a necessity to perform a point set triangulation [BE92, HK09] to compute $\mathcal{L}_f$ from $\mathcal{L}_v$.

Volume and Surface of triangular 2D-Manifolds in 3D-Space

Computing the area $A_{\mathcal{M}}$ of the surface $\mathcal{M}_2$ of a 3D-model and its enclosed volume $V_{\mathcal{M}}$ is an elementary step to analyze and classify objects. Having $\mathcal{M}_2$ represented only by triangles, the surface can be computed by:

$$A_{\mathcal{M}} = \sum_j A(\mathbf{t}_j) = \sum_j \frac{|\mathbf{n}_j|}{2} \text{ with the normal } \mathbf{n}_j = (\mathbf{p}_{B_j} - \mathbf{p}_{A_j}) \times (\mathbf{p}_{C_j} - \mathbf{p}_{A_j}) \tag{2.7}$$

An alternate notation for computing an area is the use of $|\cdot|$. This means $A_{\mathcal{M}} \equiv |\mathcal{M}_2|$.

Regarding the divergence theorem (*Gaußscher Integralsatz*) the volume $V_\square$ for a unit cube enclosing a region Ω can be computed by:

$$V_\square = \int_\Omega 1 dV = \int_\Omega \nabla \cdot \begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix} dV = \int_\Omega \nabla \cdot \begin{pmatrix} 0 \\ y \\ 0 \end{pmatrix} dV = \int_\Omega \nabla \cdot \begin{pmatrix} 0 \\ 0 \\ z \end{pmatrix} dV \tag{2.8}$$

Having $\partial\Omega$ as the border of the volume, which equals the surface of our 3D models, we can compute its volume $V_{\mathcal{M}}$ by:

$$V_{\mathcal{M}} = \int_\Omega \nabla \cdot \begin{pmatrix} x_j \\ 0 \\ 0 \end{pmatrix} dV = \int_\Omega \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} \cdot \begin{pmatrix} x_j \\ 0 \\ 0 \end{pmatrix} dV = \int_{\partial\Omega} \begin{pmatrix} x_j \\ 0 \\ 0 \end{pmatrix} \mathbf{n}_j d\mathbf{s}_j \tag{2.9}$$

For any manifold and closed $\mathcal{M}_2$ with a balance point (also known as center of gravity) $\mathbf{s}_j$ for each face $\mathbf{t}_j$, the volume can be computed using:

$$V_{\mathcal{M}} = \sum_j \begin{pmatrix} s_{x_j} \\ 0 \\ 0 \end{pmatrix} \frac{\mathbf{n}_j}{2} \text{ with } \mathbf{s}_j = \begin{pmatrix} s_{x_j} \\ s_{y_j} \\ s_{z_j} \end{pmatrix} = \frac{\mathbf{p}_{A_j} + \mathbf{p}_{B_j} + \mathbf{p}_{C_j}}{3} \tag{2.10}$$

As normals of primitives, like triangles, are typically given with a length of 1 in Computer Graphics, the normalized normal of a triangle $\mathbf{t}_j$ is defined as:

$$\hat{\mathbf{n}}_j = \frac{\mathbf{n}_j}{|\mathbf{n}_j|} \quad \Rightarrow \quad |\hat{\mathbf{n}}_j| = \sqrt{\hat{n}_{j_x}^2 + \hat{n}_{j_y}^2 + \hat{n}_{j_z}^2} = 1 \tag{2.11}$$

and therefore Equation 2.10 becomes:

$$V_{\mathcal{M}} = \sum_j A(\mathbf{t}_j) \begin{pmatrix} s_{x_j} \\ 0 \\ 0 \end{pmatrix} \hat{\mathbf{n}}_j \tag{2.12}$$

The alternate notation for computing such a volume is $|\mathbb{V}_{\mathcal{M}}|$ allowing a notation in accordance to areas, which is used later within this thesis e.g. $|B|$ is the volume of a ball B.

As $\mathcal{M}_2$ measured using real world 3D-scanners, we cannot assume manifoldness nor a closed surface. The next subsections show how holes and related irregluarities of meshes can be determined.

Borders and Edges

Proper detection and treatment of borders of $\mathcal{M}_2$ is important even for watertight 3D-models as we will encounter borders for subsets $\mathcal{M}_2$ in Chapter 4. These subsets are typically connected components also called labeled regions – see Chapter 2 in [BB82]. In general we have to expect holes within the surfaces of our 3D-models, because the data structure is designed to handle real world data, where measurements of the surface may be missing.

Holes within the surfaces of a 3D-model mean mathematically, that $\mathcal{M}_2$ can have one or more borders. Consequently there is an area along the borders, which have to be determined. Borders have to be treated differently than the rest of the surface. The border area contains faces with vertices, which can be computed by examination of the edges $\mathbf{e}$ of $\mathbf{t}$. A list of orientated edges $\mathcal{L}_e$ can be derived from $\mathcal{L}_f$ by splitting each triple $\{j_A, j_B, j_C\}$ into three tuples:

$$\begin{aligned}
\mathbf{t_j} &\mapsto \mathcal{L}_{e_j}(\mathbf{t_j}) = \{\{A_j, B_j\}, \{B_j, C_j\}, \{C_j, A_j\}\} \equiv \{\mathbf{e}_{c_j}, \mathbf{e}_{b_j}, \mathbf{e}_{a_j}\} && (2.13)\\
\mathcal{L}_e &= \{\, \mathbf{e}_{c_1}, \mathbf{e}_{b_1}, \mathbf{e}_{a_1}, \ \ldots, \ \mathbf{e}_{c_{j_{max}}}, \mathbf{e}_{b_{j_{max}}}, \mathbf{e}_{a_{j_{max}}} \,\} && (2.14)\\
&\Rightarrow |\mathcal{L}_e| = 3|\mathcal{L}_f| \text{ , with } k = \mathbb{Z}^+ \wedge k \leq k_{max} \wedge k_{max} = 3j_{max} \text{ we get:}\\
\mathcal{L}_e &= \{\, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \ \ldots, \ \mathbf{e}_{k_{max}-2}, \mathbf{e}_{k_{max}-1}, \mathbf{e}_{k_{max}} \,\} && (2.15)
\end{aligned}$$

Edges not on the border appear twice within a mesh with inverted orientation. An edge $\mathbf{e}_m = \{A_m, B_m\}$ is on the border $\partial\mathcal{M}_2$ of $\mathcal{M}_2$, when:

$$\nexists\, k \quad \text{with} \quad \mathbf{e}_k = \{B_m, A_m\} \in \mathcal{L}_e \tag{2.16}$$

These edges are stored in the list $\mathcal{L}_{\partial e}$, which holds the indices of edges along the boundary of the mesh. A mesh with an empty list of edges $\mathcal{L}_{\partial e} = \emptyset$ is called a closed mesh.

Consequently the face $\mathbf{t}_{\frac{k-(k \mod 3)}{3}}$ and the vertices $\mathbf{p}_{A_m}$ and $\mathbf{p}_{B_m}$ are on the border, when $\mathcal{M}_2$ is a manifold. The only exceptions are the so-called solo vertices, which are not connected to any face. These vertices are extreme cases of holes and typically occur as outliers for surfaces with properties like translucency, high reflectance and/or dark color. Let I be an index set for solo vertices, the list of solo vertices $\mathcal{L}_{v_s}$ is be determined by:

$$\begin{aligned}
\mathcal{L}_{v_s} := \{\mathbf{p}_s \mid \nexists\, \mathbf{t}_j \in \mathcal{L}_f \text{ with} && (2.17)\\
\mathbf{t}_j = \{A_j = s, B_j, C_j\} \vee && (2.18)\\
\mathbf{t}_j = \{A_j, B_j = s, C_j\} \vee && (2.19)\\
\mathbf{t}_j = \{A_j, B_j, C_j = s\}\}_{s \in I} \subseteq \mathcal{L}_v && (2.20)
\end{aligned}$$

When $\mathcal{L}_f = \emptyset$ than $\mathcal{L}_{v_s} = \mathcal{L}_v$, which is the definition of a point cloud. This is the contrary of an ideal $\mathcal{M}_2$ having $\mathcal{L}_{v_s} = \emptyset$ as a necessary criteria.

Adjacencies, Singularities and k-ring Neighborhoods

Knowing the neighbors of a measuring point is the key to analyze our data. For regular data structures, e.g. 2D-images a pixel has four neighbors in orthogonal direction with a (relative) geometric distance of 1 and four further neighbors with a geometric distance of $\sqrt{2}$. Neglecting the geometric distance we can apply the same for vertices and faces and determine their neighbors using the orientation of the triangles:

- A face $\mathbf{t}_j$ is adjacent to a face $\mathbf{t}_i$, when $\mathbf{t}_j$ has an edge $\mathbf{e}_m = \{A_m, B_m\}$ and $\mathbf{t}_i$ has an edge $\mathbf{e}_k = \{B_m, A_m\}$.
- The vertices adjacent to a vertex $\mathbf{p}_i$ are those $\mathring{\mathbf{p}}_i$, belonging to the faces $\mathring{\mathbf{t}}_i := \{A_i, B_{\neq i}, C_{\neq i}\} \vee \{A_{\neq i}, B_i, C_{\neq i}\} \vee \{A_{\neq i}, B_{\neq i}, C_i\}$.

Faces $\mathring{\mathbf{t}}_i$ and vertices $\mathring{\mathbf{p}}_i$ belong to the so-called 1-ring as they have distance of 1 within the graph of the mesh. The 1-ring can be extended to a 2-ring by using the vertices $\mathring{\mathbf{p}}_i$ as $\mathbf{p}_i$, which adds vertices and faces having a distance of 2 within the graph. This iterative concept can be repeated k times and the neighborhood is than refered to as k-ring. The distance within the graph can not be assumed equal to the geometric distance nor geodesic distance. However it is used throughout literature as experiments are often done on synthetic data – especially in the field of Computer Graphics. Figure 2.6 shows the scheme of the 1-ring and an example for a 5-ring on a synthetic sphere in contrast to 5-rings on an acquired mesh. Even the triangles of the sphere are almost equal, we see that the outlines of the k-rings have a hexagonal shape, while geodesic ring would have the shape of a circle. Already the 1-rings of the acquired mesh have completely arbitrary shapes and sizes.

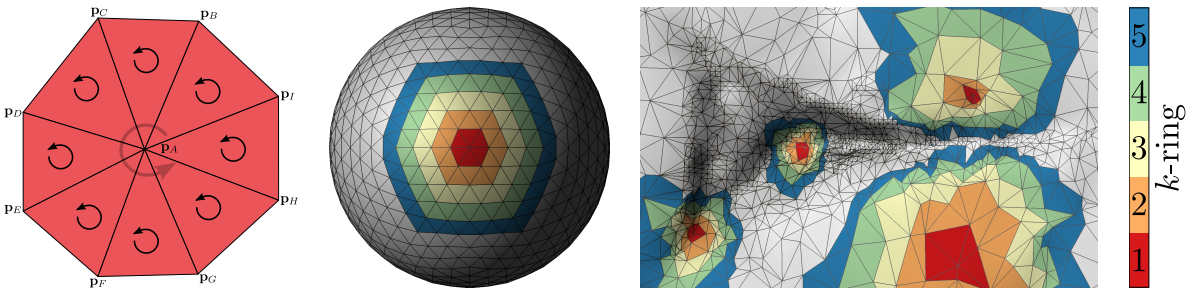


Figure 2.6: k-ring neighborhood: (a) Scheme of a 1-ring neighborhood and 1 to 5-ring neighborhood for (b) a synthetic sphere and (c) for a wedge acquired using a 3D-scanner.

The 1-ring shown in Figure 2.6a is closely related to the so-called triangle-fan used by *Open Graphics Language* (OpenGL). While the Figure represents the most common case of an 1-ring equal to a triangle-fan as most parts of $\mathcal{M}_2$ are organized like this. The second common case looks the same, with one or more **adjacent** faces missing – this is the case, when $\mathring{\mathbf{p}}_i$ is on the border of the mesh. However the data structure allows that **non-adjacent** faces are missing leading to singular vertices – not to be confused with solo vertices.

We can identify singular vertices connecting different parts of $\mathcal{M}_2$ by determining one triangle-fan per vertex and compare them to the 1-ring neighborhood. The faces of the 1-ring are determined just by the presence of a shared central vertex [MB05] as shown before. Having the 1-ring we need to test if we can walk around the shared vertex using the faces and their neighbors. This is shown in Figure 2.7a, where we can start at any of the vertices $\{B, \ldots, G\}$ following the directional edges between them. We than will arrive at vertex $\mathbf{p}_G$. If we have not started at $\mathbf{p}_B$, we will have to backtrack the edges in opposite direction as $\mathbf{p}_A$ is a vertex on the border $\partial\mathcal{M}_2$. In the end we will have visited all vertices of the triangle fan $\{B, \ldots, G\}$.

When $\mathbf{p}_A$ is not on the border, because a triangle $\{A, G, B\}$ exists, we can visit all vertices without backtracking. Figure 2.7b shows the scheme for a border vertex, which is singular, because we can visit either the $\{B, C, D, E\}$ or $\{F, G, H\}$ of these two triangle-fans. Figure 2.7c shows an example from a 3D-model, where a non-border, singular vertex $\mathbf{p}_A$ connects different parts of the mesh.

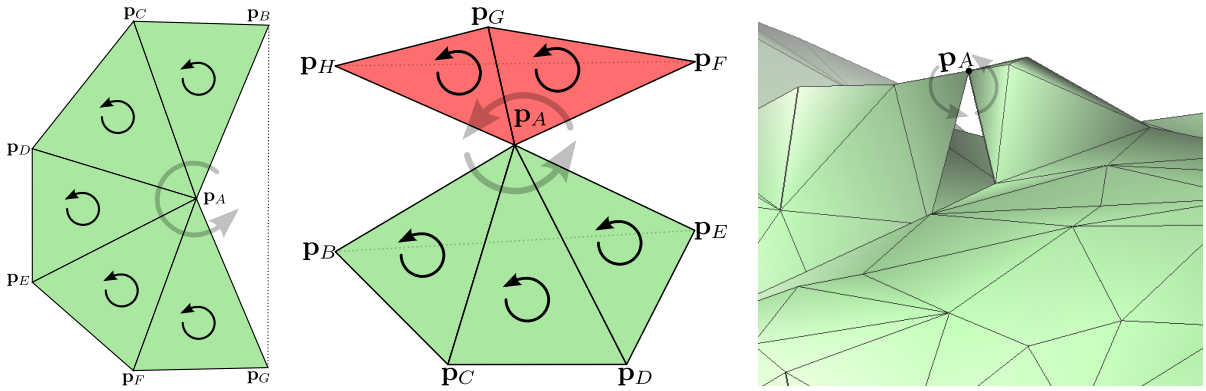


Figure 2.7: Schematic visualization of (a) a non-singular vertex $\mathbf{p}_A$ with its triangle fan equaling its 1-ring neighborhood. When there is a triangle $\{A, G, B\}$ the fan is closed – if not $\mathbf{p}_A$, $\mathbf{p}_B$ and $\mathbf{p}_G$ are vertices on the border of $\mathcal{M}_2$. (b) Singular vertex $\mathbf{p}_A$ connecting two triangle fans. (c) Real world example for a singular vertex $\mathbf{p}_A$.

Non-Manifold Edges

Previous Sections have shown that 3D-models from optical 3D-scanners consist of a discrete surface representation and are generally not regular having singular vertices due to the missing constraints from file formats organized by $\mathcal{L}_v$ and $\mathcal{L}_f$. The lack of these constraints also allows for non-manifold structures as faces can have more than one neighbor per edge. This is useful for synthetic 3D-models from *Computer Aided Design* (CAD) systems, e.g to connect primitives like walls of buildings. However these non-manifold structures have to be detected and treated. Methods to analyze 3D-data have to search along the graph of the mesh. Applying e.g. a search algorithm designed for manifold

surfaces may not reach all parts of the mesh as it will ignore junctions caused by non-manifoldness. In worst-case scenarios an algorithm may be trapped in an infinite loop. By definition of our data structure, non-manifold edges, singular vertices and agglutinated faces can occur.

Manifold edges are edges shared by exactly two faces. Exceptions are edges at the border of the surface, having no adjacent face. As each edge derived from $\mathcal{L}_f$ has an orientation, a non-manifold edge $\mathbf{e}_m^{2+}$ is found, when there are two edges from different triangles defined by the same vertices, having the same direction:

$$\mathbf{e}_m^{2+} \;:=\; \mathbf{e}_k \in \mathbf{t}_u \;\vee\; \mathbf{e}_m \in \mathbf{t}_v \;\vee\; \mathbf{e}_k = \mathbf{e}_m = \{A_m, B_m\} \tag{2.21}$$

Such a non-manifold edge can connect an arbitrary number of faces $\mathbf{t}_{\neq v}$, which leads to:

$$\mathbf{e}_m^{+} \;:=\; \mathbf{e}_{\neq m} \in \mathbf{t}_{\neq v} \;\vee\; \mathbf{e}_m \in \mathbf{t}_v \;\vee\; \forall \mathbf{e}_{\neq m} : \mathbf{e}_{\neq m} = \mathbf{e}_m = \{A_m, B_m\} \tag{2.22}$$

As the faces $\mathbf{t}$ have an orientation, a special case of wrong connections can be found using the above definition: pairs of faces sharing one edge having the edge orientated in the same direction. This means that one of the faces has a wrong orientation resulting in a normal vector pointing in the negative direction. Such cases become immediately visible, when rendered using a virtual illumination and have to be removed or corrected. Figure 2.8a shows the scheme for one edge $\mathbf{e}_{k\neq m}$ connecting two manifold faces ($\mathbf{t}_u$, $\mathbf{t}_v$). Figure 2.8b shows the same faces having a face $\mathbf{t}_w$ added sharing a non-manifold edge $\mathbf{e}_{k=m}$ and Figure 2.8c shows the special case for a manifold edge connecting the face $\mathbf{t}_w$ having a wrong orientation.

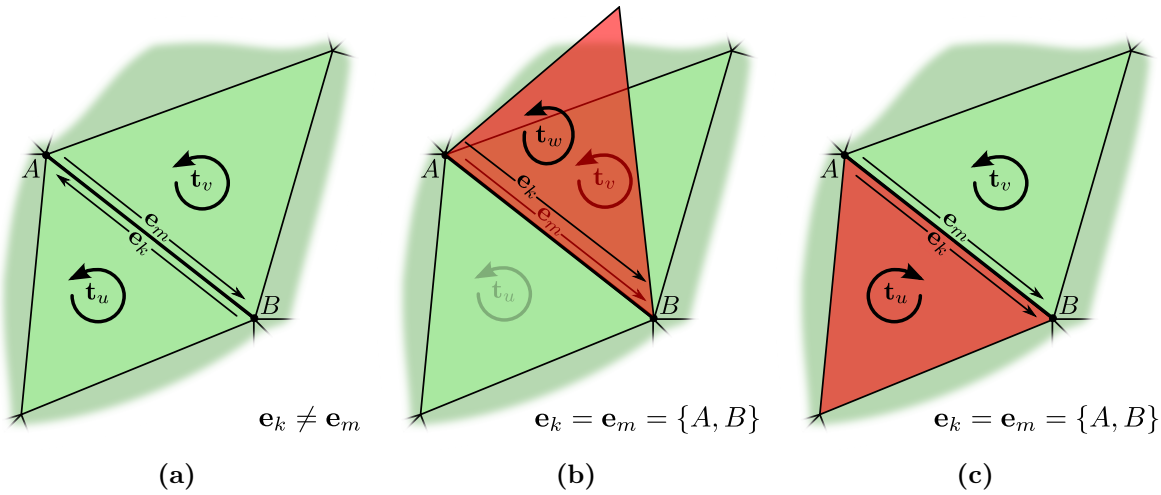


Figure 2.8: Schematic visualization of $\mathcal{M}_2$ in green with an edge connecting (a) two manifold faces ($\mathbf{t}_u$, $\mathbf{t}_v$) and (b) the face $\mathbf{t}_w$ added along the non-manifold edge $\mathbf{e}_k = \mathbf{e}_m$ and (c) having a face $\mathbf{t}_w$ embedded with a wrong orientation.

Agglutinated Faces and Faces with Zero Area

Faces literally sticking together having a proper orientation (no $\mathbf{e}^+$) are typical remains of outliers from acquisition. They can be removed as they describe parts of objects without any volume. This can be shown using Equation 2.10 with two faces sharing the same vertices in opposite orientation. Let the first face be $\mathbf{t}_1 := \{\mathbf{p}_a, \mathbf{p}_b, \mathbf{p}_c\}$ and the agglutination face be $\mathbf{t}_2 := \{\mathbf{p}_c, \mathbf{p}_b, \mathbf{p}_a\}$, we can simplify: $\mathbf{s} = \mathbf{s}_1 = \mathbf{s}_2$, $A(\mathbf{t}) = A(\mathbf{t}_1) = A(\mathbf{t}_2)$ and $\hat{\mathbf{n}}_2 = -\hat{\mathbf{n}}_1$ and substitute:

$$V_{\mathcal{M}} = A(\mathbf{t}_1)\begin{pmatrix} s_{x_1} \\ 0 \\ 0 \end{pmatrix}\hat{\mathbf{n}}_1 + A(\mathbf{t}_2)\begin{pmatrix} s_{x_2} \\ 0 \\ 0 \end{pmatrix}\hat{\mathbf{n}}_2 = A(\mathbf{t})\begin{pmatrix} s_x \\ 0 \\ 0 \end{pmatrix}(\hat{\mathbf{n}}_1 - \hat{\mathbf{n}}_1) = 0 \tag{2.23}$$

An indicator for an agglutinated face are edges with faces having a wrong orientation as shown in Figure 2.8c. In practical examples agglunating faces often appear along non-manifold edges. However agglunating faces can be found and removed using the following definition, which should be done before removing non-manifold edges and removing faces with wrong orientation to minimize the changes to the mesh.

$$\exists\ \mathbf{t}_i \vee \mathbf{t}_j \in \mathcal{L}_f \text{ with } \mathbf{t}_i := \{A_i, B_i, C_i\} \text{ and } \mathbf{t}_j := \{C_i, B_i, A_i\} \wedge \{B_i, A_i, C_i\} \wedge \{A_i, C_i, B_i\} \tag{2.24}$$

Besides agglutinated faces having enclosing no volume, there also exist degenerated faces having no area. This happens, when two or all three vertices of a face have the same position vector. For example, when we substitue $\mathbf{p}_A = \mathbf{p}_B$ in Equation 2.7 we get:

$$\mathbf{n}_j = (\mathbf{p}_{B_j} - \mathbf{p}_{B_j}) \times (\mathbf{p}_{C_j} - \mathbf{p}_{B_j}) = (0,0,0)^T \text{ and therefore } A(\mathbf{t}_j) = \frac{|\mathbf{n}_j|}{2} = 0 \tag{2.25}$$

Summarizing this Section we can detect the worst case scenarios within the data structure of files provided by 3D-scanners, which can render an algorithm for local surface filtering instable, because of e.g. an area of zero leading to a division by zero. The case of self-intersecting surfaces as known from synthetic data and their treatment [JSC04] is not necessary, as it does not change or influence the critical surfaces properties in respect to the methods presented within this thesis.

2.3 Displacement Maps and Curvature Estimation

Having acquired a 3D-model, its visualization is the consequent next step for an autopsy. For the cuneiform tablets this can be done by placing a parallel light source above the top-left corner and an optional light source in the opposite direction – like the setup using a real camera. As for these objects the color information as texture-map is rather distracting, we can use a solid gray color instead. To increase contrast on the surface we can virtually change the surface materials using OpenGL [MB05]. In heuristic experiments we determined that surface properties similar to metal surfaces with 30% specular reflectance gives the best contrast for cuneiform tablets.

The 3D-model of the object photographed in Figure 1.2 is shown in Figure 2.9. In this Figure the characters of the part of the tablet within the shadows are highlighted, because raytracing [Whi80] was disabled. This effect is generally avoided as it creates nonnatural looking scenes, but it is useful for cuneiform script. The histograms of these images show, that shadows of the flanks of the wedges are very distinct, while the rest of the surface is perfectly lit showing lots of details. Adding a second light source without raytracing dramatically cancels shadows as shown for the image and its histogram on the right hand side. The intensity of the virtual light was trimmed to use the maximum range of gray values resulting in a maximum of contrast.

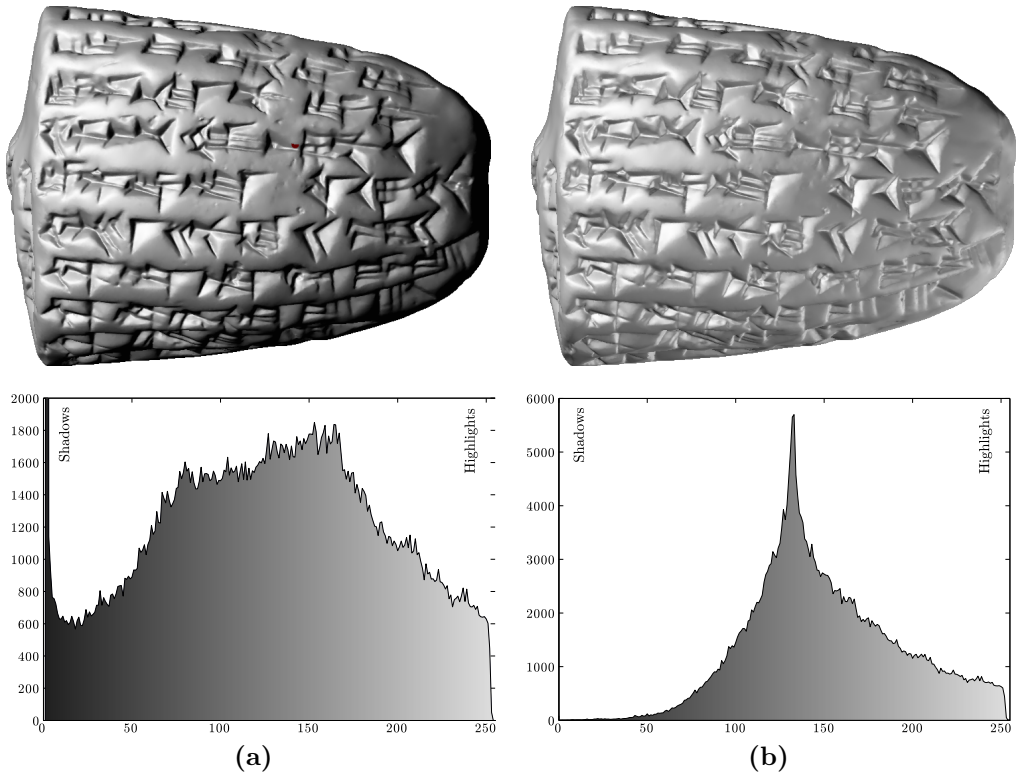
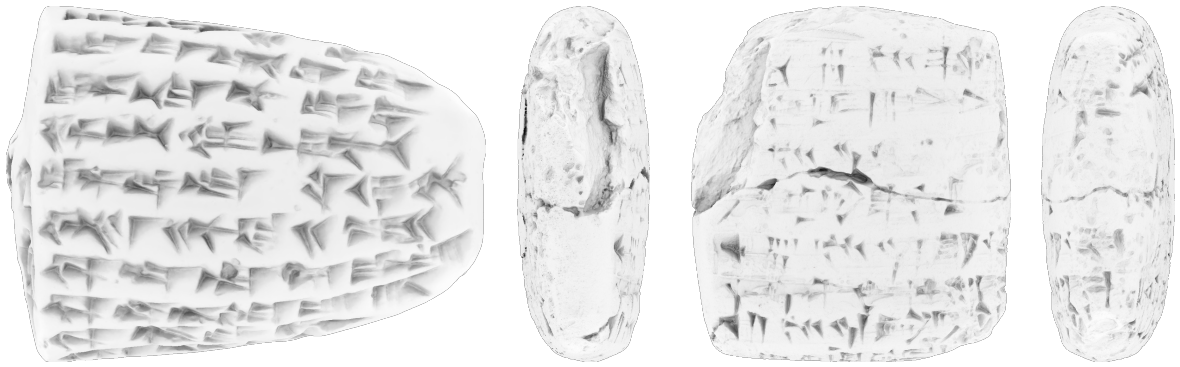


Figure 2.9: Visualizations and their gray-level histograms of the 3D-model of HOS G8 using (a) one virtual light source and (b) two opposing virtual light sources.

Ambient occlusion [Bun05] is a Computer Graphics' rendering technique promising

to shade corners darker, but the differences in contrast between wedges and the overall surface is to low for any further analysis. Figure 2.10 shows two tablets texture mapped using ambient occlusion and no virtual light source. For tablet HOS G8 all the wedges become visible regardless of their position on the surface, but the contrast is low and the tablet can only be read because the characters are very well written and preserved. For tablets like 88/677 we can only visualize some of the larger wedges.



(a) **(b)**

Figure 2.10: Visualizations using ambient occlusion instead of virtual light sources for tablet (a) HOS G8 and (b) Inv. No. 88/677 with side views. For the latter only larger wedges become visible, while details cannot be seen.

A better approach for cuneiform tablets is shown in [AL02], where spheres are fit onto the surface, which trace the wedges like a virtual stylus. [AL02] uses B-spline surfaces [Boo72, Por10] to provide a canvas for the virtual stylus, which show good results for very well preserved tablets. In case of damaged tablets this method is not well-suited as the next section will show for even more generic surfaces. Generally the use of rendering methods always leads to projections of our 3D-models to 2D raster images, which limits further processing as spatial information is lost. Therefore analyzing the surface $\mathcal{M}_2$ itself is required.

Displacement Map Filtering the surface using a smoothing method can be applied to 3D-models to determine its representation with lower details. Lowering the *Level of Detail* (LOD) will remove the wedges resulting in an empty tablet, like nobody has ever written anything on it. This can be achieved using the Poisson surface reconstruction [KBH06] with a low LOD to estimate the empty tablet $\delta\mathcal{M}_2$. Computing the distance between the surface with text $\mathcal{M}_2$ and $\delta\mathcal{M}_2$ using the Hausdorff distance [Ruc96, CRS01] $d_H(\mathcal{M}_2, \delta\mathcal{M}_2)$ for each vertex $\mathbf{p}_i$ will be a function $f_{d_H}(\mathbf{p}_i)$ with values $\ll 0$ for wedges. Mapping the function values $f_{d_H}(\mathbf{p}_i)$ to a color ramp e.g. from black to white to red, enables a visualization of the differences between the surfaces $\mathcal{M}_2$ and $\delta\mathcal{M}_2$ using a false-color texture map. Figure 2.11 shows this approach applied to tablet Inv. No. 88/677.

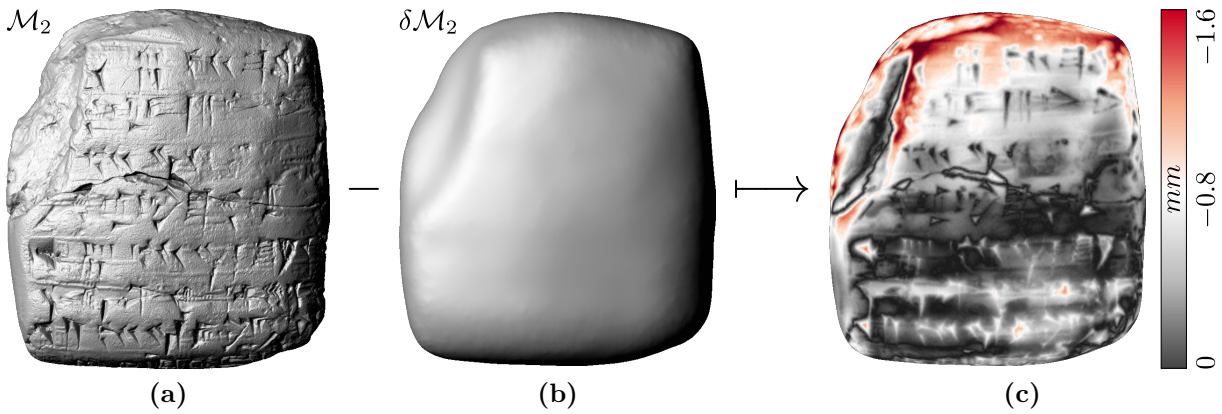


Figure 2.11: Tablet Inv. No. 88/677: (a) Acquired surface $\mathcal{M}_2$, (b) smoothed surface $\delta\mathcal{M}_2$ and (c) the Hausdorff distance $d_H(\mathcal{M}_2, \delta\mathcal{M}_2)$ as false-color texture map using a black-white-red color ramp.

In Figure 2.11 we see that $\delta\mathcal{M}_2$ is slightly shrunken. Using Equation 2.10 we can compute that $V_{\delta\mathcal{M}_2} = 22.71\,cm^3$ and therefore 4% less than $V_{\mathcal{M}_2} = 23.54\,cm^3$. The chipped of edges on the top and the left side are influencing the smoothing leading to additional shrinkage near these areas as shown in in red color. Therefore the wedges can be distinguished by color variations within the false-color texture map, but they cannot be segmented from the rest of the surface using thresholding. As the tablets shapes and especially their fractures are arbitrary there is nothing like an underlying model. Experiments with alternative smoothing methods (e.g. [OBB00]) also led to the same result as well as the fact that smoothing does not provide an empty tablet – it mimics its look for unbroken tablets.

Curvature Estimation Computing the curvature κ for the vertices of $\mathcal{M}_2$ is an alternative approach to determine the edges of the tetrahedrons impressed into the surface. A common measure for the curvature is named after *Carl Friedrich Gauß*. The so-called Gaussian curvature κ_G can be determined for a vertex $\mathbf{p}_i$ using a geodesic disc D_r having an radius r, an area $A(r)$ and a length $L(r)$ of ∂D_r. It is computed using the *Bertrand-Diquet-Puiseux* theorem [BDP48, Spi79]:

$$\kappa_G(\mathbf{p}_i) = \lim_{r \to 0^+} 3\frac{2\pi r - L(r)}{\pi r^3} = \lim_{r \to 0^+} 12\frac{\pi r^2 - A(r)}{\pi r^4} \tag{2.26}$$

In case $\mathbf{p}_i$ is on a plane or any other developable surface, the geodesic disc is equal to a circle and we get: $L(r) = 2\pi r$ and $A(r) = \pi r^2$. Examples of primitive objects with a fixed sign of the curvature for $\mathcal{M}_2$ are: a sphere has positive curvature, a plane has curvature zero and a hyperboloid has negative curvature. As our wedges are piecewise flat, the curvature is expected to be zero except for the corner points. This is illustrated in Figure 2.12 for a synthetic wedge showing a front view, side view and the circular sectors of the geodesic discs. We see for arbitrary points on an edge, like $\mathbf{p}_{AB}$ and $\mathbf{p}_{BD}$, that the geodesic disc consists of two halves of a circle resulting $A(r) = \pi r^2$ and $\kappa_G = 0$. For $\mathbf{p}_A$ we can compute $A(r)$ as sum of three circle sectors, which is larger than πr^2 resulting in $\kappa_G < 0$. $\kappa_G(\mathbf{p}_B) = \kappa_G(\mathbf{p}_A)$, because the wedge is symmetric about the horizontal axis. The sum for the three circle sectors of $\mathbf{p}_D$ leads to $A(r) < \pi r^2$ and therefore a positive curvature. The computation of the angles of the circle sectors and κ_G is shown in Section A.1.

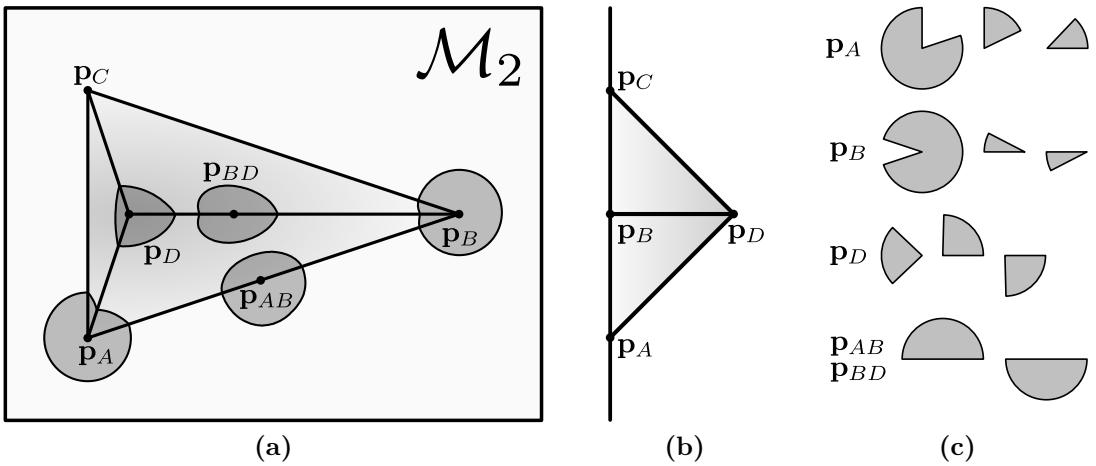


Figure 2.12: Ideal wedge embedded in a flat surface $\mathcal{M}_2$ as (a) front view, (b) side view and (c) the circle segments of the geodesic discs for points at the corners and edges.

While the Gaussian curvature can only be used to determine corner points of a wedge, differential geometry [Pre10] provides the so-called *mean curvature* κ_H by using the embedding of $\mathcal{M}_2$ in $\mathbb{R}^3$. For each $\mathbf{p}_i$ we can locally approximate a tangent plane in $\mathbb{R}^3$, orthogonal to the normal vector $\mathbf{n}_i$. As points – in contrast to faces – have no normals within triangulated meshes, such an approximation can be done by e.g. normal vector

voting [PSK+02]. The normal curvatures $\kappa_i^{\mathbf{n}}(\phi)$ describe the local bending for each unit direction $\hat{\mathbf{u}}_{i,\phi}$. The mean curvature κ_H is computed:

$$\kappa_H(\mathbf{p}_i) = \frac{1}{2\pi} \int_0^{2\pi} \kappa_i^{\mathbf{n}}(\phi) d\phi \tag{2.27}$$

The two principal curvatures $\kappa_{i,1}, \kappa_{i,2}$ with their associated principal directions $\hat{\mathbf{u}}_{i,1}$, $\hat{\mathbf{u}}_{i,2}$ are computed using the *Principal Component Analysis* (PCA) [Jol02]. As $\hat{\mathbf{u}}_{i,1}$ and $\hat{\mathbf{u}}_{i,2}$ are orthogonal we can express the normal curvature as $\kappa_H(\mathbf{p}_i) = \kappa_{i,1} cos^2(\phi) + \kappa_{i,2} sin^2(\phi)$ leading to the well-known definition:

$$\kappa_H(\mathbf{p}_i) := \frac{\kappa_{i,1} + \kappa_{i,2}}{2} \quad \text{and} \quad \kappa_G(\mathbf{p}_i) := \kappa_{i,1}\kappa_{i,2} \tag{2.28}$$

Figure 2.13 shows the principal directions $\hat{\mathbf{u}}_{i,1}$ and $\hat{\mathbf{u}}_{i,2}$ scaled using the principal curvatures for points at the corners and edges of the ideal wedge.

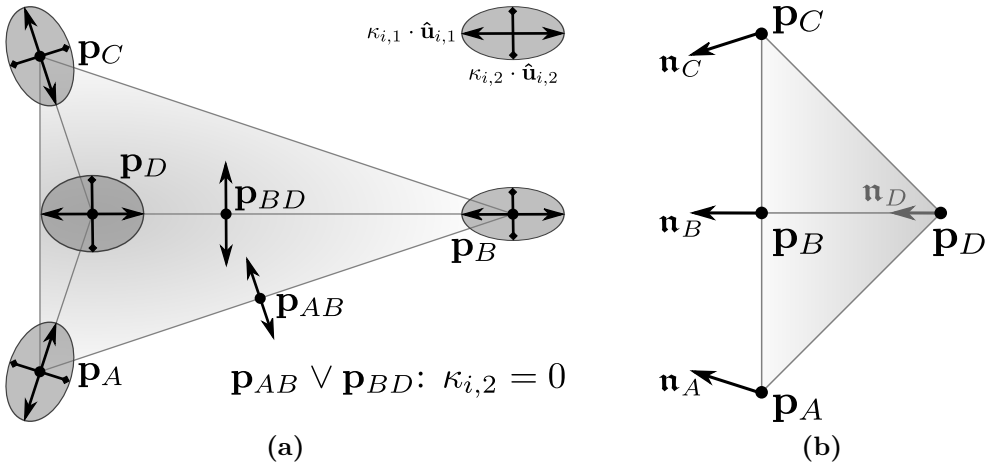


Figure 2.13: Ideal wedge with schematic drawing of (a) the principal curvatures $\kappa_{i,1}$ and $\kappa_{i,2}$ scaling the principal directions $\hat{\mathbf{u}}_{i,1}$ and $\hat{\mathbf{u}}_{i,2}$. (b) Side view with estimated normals $\mathbf{n}$ used to compute the mean curvature κ_H.

There exist comprehensive surveys of concepts for triangular meshes [Kil04] and implementations of discrete operators to estimate the mean curvature and the Gaussian curvature for such meshes [MDSB03]. The latter is often referenced within literature and implemented in *MeshLab* by the *Visual Computing Lab, Istituto di Scienza e Tecnologie dell'Informazione* in Pisa, Italy.

Representing our minimal and ideal wedge, with its surrounding flat surface as a synthetic mesh, requires 17 triangles and 15 vertices. The source of the mesh structure for the minimal wedge is shown in Section A.1. Regarding the Shannon theorem [Sha48] a measuring instrument has to sample at least twice the number of vertices in x and y direction resulting in ≈ 60 vertices. Depending on the *FOV* of the 3D-scanner we will get an even larger number of vertices.

To evaluate methods available for discrete surfaces, we can increase the number of vertices of our synthetic wedge using five iterations of the mid-edge subdivision scheme [PR97, HW99]. Therefore our synthetic wedge for testing consists of 8 881 vertices and 17 408 faces. Testing the estimation of mean curvature and Gaussian curvature were done using [MDSB03] in *MeshLab* Version 1.2.2. A random spatial displacement of 0.05% of the overall size of the wedge was added to the vertices positions to simulate a marginal amount of noise. Figure 2.14 shows the visualization of the results for curvature estimation of the synthetic wedge.

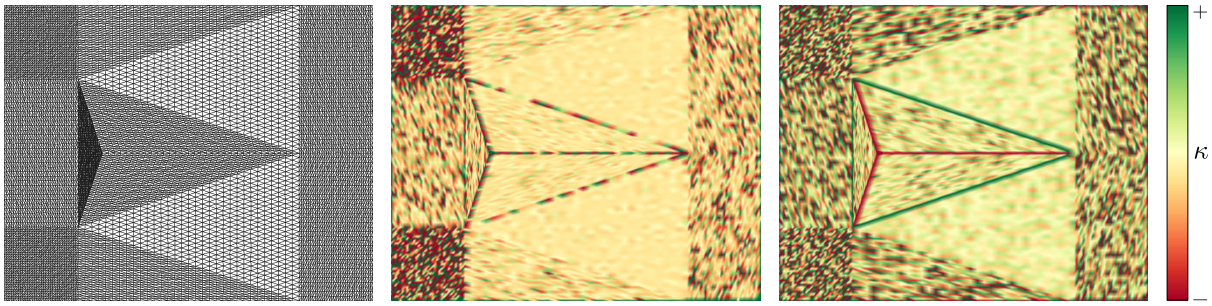
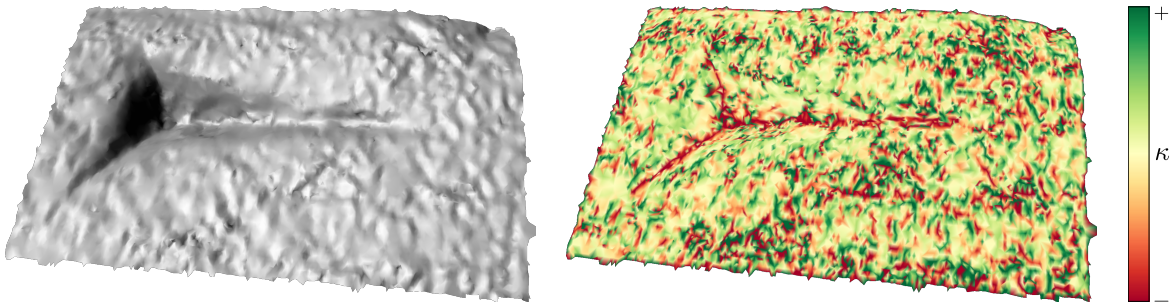


Figure 2.14: Synthetic wedge (a) subdivided with the mid-edge subdivision scheme and a marginal amount of noise. (b) The wedge with estimated Gaussian curvature and (c) estimated mean curvature. (d) Color ramp used to visualize the curvature. The subdivision and the curvature estimations were done using [MDSB03] in *MeshLab* Version 1.2.2.

In Figure 2.14b we see that the outline of the wedge is stippled, which means that this distinct edge is not properly estimated. The centerlines (also known as skeleton) are also error prone. Furthermore the estimation depends on the shape and size of the triangles. Therefore areas with smaller and/or non-equilateral triangle show the same curvature values than the edges of the wedge. This is also reflected in the curvature values, as they range from −3.2 up to 35.4, which are expected to be −1.3, −0.59 and +3.2 (as computed on page 157). The mean curvature in Figure 2.14c does not have the stippling, but its value ranges from $-12,7$ to $+14,8$ showing additional noise. For testing on real data we choose a horizontal wedge from tablet **Inv. No. 88/677**, which has 14 526 vertices and 28 516 faces. The result for a real rough surface and noise from the acquisition is shown in Figure 2.15.

For the real wedge the values scatter between −116.64 and +147.81 with the 90%-centile of −6.99 to +6.73. Smoothing the real wedge as suggested in [MDSB03] improves the result visually, but at the same time we loose small details like the seal impressions on the tablet. Additionally the amount of smoothing can not be determined as the noise present in a 3D-model depends strongly on the objects surface reflectance properties. This may vary along the surface and there is no information gathered by the 3D-scanner about the reflectance like shown in [GTHD03] for planar objects.



(a) **(b)** **(c)**

Figure 2.15: Horizontal wedge from tablet Inv. No. 88/677 (a) with virtual illumination and (b) estimated Gaussian curvature as surface color and no illumination. (d) Color ramp used to visualize the curvature.

MeshLab provides two additional curvature estimations:

- The root mean squared curvature $\kappa_{RMS} := \sqrt{\frac{\kappa_{i,1}+\kappa_{i,2}}{2}}$ and
- the absolute curvature $\kappa_{ABS} := |\kappa_{i,1}| + |\kappa_{i,2}|$,

which do not improve the results as they depend on the previously shown results for κ_H and κ_K. Page 165 in Section B.1.1 shows the results of the provided curvature estimations by *MeshLab* for tablet Inv. No. 88/677.

To achieve high performance and assuming that the surface was sampled regarding the *Shannon theorem*, [MDSB03] uses the 1-ring neighborhood of a vertex to estimate the curvature. As we have details to be sampled in different scales and a fixed 3D-scanner resolution per optics, the surface is generally oversampled like the synthetic wedge with a ratio up to $1:1\,000$. This means, that the 1-ring spans only 1/1000 of a corner or an edge and can explain, why the skeleton of the wedge in Figure 2.14b is visualized by a stippled line.

Using a k-ring would be an alternative solution, when the mesh would be regular. An applicable alternative is to compute a geodesic ring, fit a second degree polynomial surface and compute its curvature. We have shown this approach in [Mar06] using *Single Value Decomposition* (SVD) for orientation of the geodesic ring $U \subset \mathcal{M}_2$ to the xy-plane to fit the second degree polynom using [VH96] having $u = x$ and $v = y$ as parameters. Having a parametrized surface the Gaussian curvature can be computed as ratio of the determinants of the second fundamental form and the first fundamental form using *Brioschi's formula* – as proofen in Gauss' *Theorema Egregium* [GMH07]. The key to apply this method is a proper choice for the radius of the geodesic ring. In case of surface segmentation [TF95, Mar06] into relatively simple and large concave, convex and planar parts the choice of the radius can be done using a priori knowledge. The method should be applied on datasets reduced to $\approx 10\%$ of the amount of vertices describing the overall shape of an object. Otherwise the computational complexity becomes $O(n^2)$ and the computing time increases dramatically.

2.4 Summary

All the shown approaches are useful for application on industrial made real world objects or synthetic 3D-models, where certain a priori knowledge about the surface features are present. This knowledge can be used to set smoothing parameters, compute parametrized surface, reduce the size of the datasets, estimate the normal vectors or choose the radius of a geodesic ring to estimate the curvature. In other words: these methods are robust to process discrete irregular meshes in reasonable time, when certain criteria are fulfilled (e.g. noise removed). These criteria typically require manual parameterization (e.g. radius for a geodesic disc) or rely on complete (unbroken) objects. [MDSB03] address the intrinsic chicken or the egg causality dilemma common to feature recognition based on Gaussian or mean curvature, because the required preprocessing requires features to be recognized in advance. One can argue that watershed methods [BL79] can solve this dilemma, but watersheding relies on the topology of the surface [CB97, Ber05] – for example: if a method does not distinguish between a valley of a fracture and a valley of a character both will be treated equally. Having to process meshes, which are often incomplete due to measuring errors and/or parts of the object missing with wide range of character shapes and tremendous amounts of objects, we can not apply the presented methods as they require too much manual processing time to get parameters right. Therefore we require a method to extract characters by their features being robust against all adversities of 3D-data provided by 3D-scanners and the objects themselves with a minimum of parameters.

The next chapter shows that integral invariants are a robust alternative to curvature estimation required for visualization of characters and computing feature metrics. These metrics are used for feature extraction shown in Chapter 4.

Chapter 3

MSII – Multi-Scale Integral Invariants

Integral invariant operations are relatives of the previously shown differential operations on manifolds, sharing the same desirable properties like local computation and uniqueness of representation, but they are less sensitive to noise [MHYS04]. In theory and on relatively small, synthetic data sets the robustness for feature classification was shown for one scale [LZH+07]. The most prominent application to find small numbers of distinct features to reassemble broken objects was demonstrated in [HFG+06].

While all these articles demonstrate the superiority of integral invariants over differential operations, there is little information about fast and robust computation of the invariants on high-resolution meshes having large numbers of areas of interest with varying shapes, like it is the case for characters on cuneiform tablets. Computing integral invariants relies on determining the local neighborhoods along the manifold, which strongly influences its performance and numerical reliability. The demand for a robust algorithm with respect to performance is increased by the fact, that the underlying and elementary shapes e.g. the wedges of cuneiform characters exist in different sizes leading to a multi-scale approach. This means that a performance improvement by estimating the local neighborhood using Principal Component Analysis as shown in [HFG+06] can not be applied, because it leads to wrong results with increasing complexity of an object's shape, as it is pointed out by the authors. An alternate approach is to use reduced data-sets, which works well to extract only small numbers of points of interest, like corresponding points for solving a puzzle of broken objects [PWHY09]. To solve these puzzles no more than seven scales were used on 3D-models with less than 100 000 vertices.

As there exist different integral invariant operators for 2D-manifolds in 3D-space, this Chapter shows the robust computation using volume, surface and line based integrals for irregular, triangulated meshes in multiple scales. This includes local neighborhood determination and border treatment, as the meshes are not flawless as assumed in theoretical work. Therefore the presented algorithms are robust against the imperfections shown in the previous Chapter. Finally a new method to efficiently compute all these integral invariants at once is shown and their properties are discussed.

3.1 Principles of Curvatures and Integral Invariants

Computing integral invariants can be done for curves and surfaces in any dimension and we can integrate different representations of curves or surfaces. Let us begin with curves embedded in 2D as archaeologists and assyriologists use curves like cross-section or silhouettes of objects within their drawings. The silhouette simply represents the object's outline, while the cross-section – also known as profile-line – is used to determine specific features of an object. The relevance of profile-lines within archaeology is shown in [OTV93]. Cross-sections of cuneiform tablets provide insights in respect to paleographic questions as geometric features of the characters can correspond to the scribe. Such features can be the angles left by the stylus, when impressed into the surface.

Having our synthetic cuneus, we can compute a cross-section from its 3D-model as described in [MK03] using a plane dividing the wedge along the writing direction – from left to right. The result is a piecewise linear curve $\mathcal{M}_1$ embedded in 2D enclosing a domain $D_{\mathcal{M}_1}$, which is a slice of the 3D-model's volume. Such a curve can be numerically derivative along its run-length, gives a measure about the changes of the function. In theory these changes will be a function of impulses – sum of Dirac functions – indicating the edges of the wedges. Practically most of these impulses will be from quantization errors. Smoothing these impulses is not an option, because of the reasons shown in Section 2.3.

To avoid derivation and smoothing, the alternative is integration along the run-length of a curve or in general the parameters of a surface. Using the parameters means invariance against translation in Euclidean space. Rotation invariance is achieved by using the spherical space about a point of a curve or surface. The concept of balls and spheres exists for arbitrary dimensions and they can be defined for the natural numbered dimensions n. A n-ball is denoted as $B_r^n(\mathbf{p}_i)$, where r is its radius and $\mathbf{p}_i$ its center. The n-sphere is the boundary ∂B_r^n of a ball and is denoted $\mathbb{S}_r^{n-1}$. Furthermore a n-ball B^n has a volume of $|B^n|$ and a n-sphere an area of $|\mathbb{S}_r^{n-1}|$.

Because the data to be processed is the given measurement data $\mathcal{M}_2$, the dimension n will only be annotated, when a ball is not of dimension three. Therefore $B_r(\mathbf{p}_i)$ is a ball embedded in $\mathbb{R}^3$ having a two-dimensional surface $\mathbb{S}_r$ as shown in Equation 3.1c. The properties of n-spheres and their boundaries are:

$$\mathbb{R}^1: \quad |B_r^1| = 2r \qquad \partial B_r^1 = \mathbb{S}_r^0 \qquad |\mathbb{S}_r^0| = 0 \tag{3.1a}$$

$$\mathbb{R}^2: \quad |B_r^2| = \pi r^2 \qquad \partial B_r^2 = \mathbb{S}_r^1 \qquad |\mathbb{S}_r^1| = 2\pi r \tag{3.1b}$$

$$\mathbb{R}^3: \quad |B_r| = \frac{4}{3}\pi r^3 \qquad \partial B_r = \mathbb{S}_r \qquad |\mathbb{S}_r| = 4\pi r^2 \tag{3.1c}$$

$$\dots$$

$$\mathbb{R}^n: \quad |B_r^n| = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}+1\right)} r^n \qquad \partial B_r^n = \mathbb{S}_r^{n-1} \qquad |\mathbb{S}_r^{n-1}| = \frac{2\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}\right)} r^{n-1} \tag{3.1d}$$

with the gamma function $\Gamma(x) = \int_0^\infty t^{x-1} \mathrm{e}^{-t} dt$, which becomes $\Gamma(n) = (n-1)!$ for $n \in \mathbb{Z}^+$.

Let $\mathbf{1}_{D_{\mathcal{M}_1}}$ be the indicator function for $D_{\mathcal{M}_1}$, which is defined by the cross-section $\mathcal{M}_1$ of the synthetic cunei shown in Figure 3.1b. Then the area of the intersection between $D_{\mathcal{M}_1}$ and a 2-ball $B_r^2(\mathbf{p}_i)$ shortly called disk, becomes a curvature related measure:

$$A_r(\mathbf{p_i}) = \int_{B_r^2(\mathbf{p}_i)} \mathbf{1}_{D_{\mathcal{M}_1}} dxdy\,, \quad \text{with} \quad \mathbf{p}_i \in \mathcal{M}_1 \quad \text{and} \quad \mathbf{1}_{D_{\mathcal{M}_1}}(x,y) = \begin{cases} 1 & \text{if } (x,y) \in D_{\mathcal{M}_1} \\ 0 & \text{else} \end{cases} \tag{3.2}$$

This function is normalized to the interval $]-1,+1[$ using the area of the disk shown in Equation 3.1b:

$$\hat{A}(\mathbf{p}_i) = \frac{A_r(\mathbf{p}_i)}{\frac{1}{2}|B_r^2|} - 1 = 2\frac{A_r(\mathbf{p}_i)}{\pi r^2} - 1 \tag{3.3}$$

If $A_r(\mathbf{p_i})$ equals the half of disk than $\hat{A}(\mathbf{p_i})$ equals zero, which corresponds to a flat segment. Positive values mean concave segments and negative values convex segments. The 3D-model and its intersection as well as the integral invariant function along an intersection of the synthetic wedge is shown in Figure 3.1. At the points of interest i.e. sharp corners like $\mathbf{p}_D$ or $\mathbf{p}_C$ the integral invariants $\hat{A}$ were computed as the area of circular sectors. This can be done as long as the circular disks do not contain any other point with such corners. This also means, that the curvature maxima and minima are independent of the radius as long as the radius r is smaller than the distances between these extremal points.

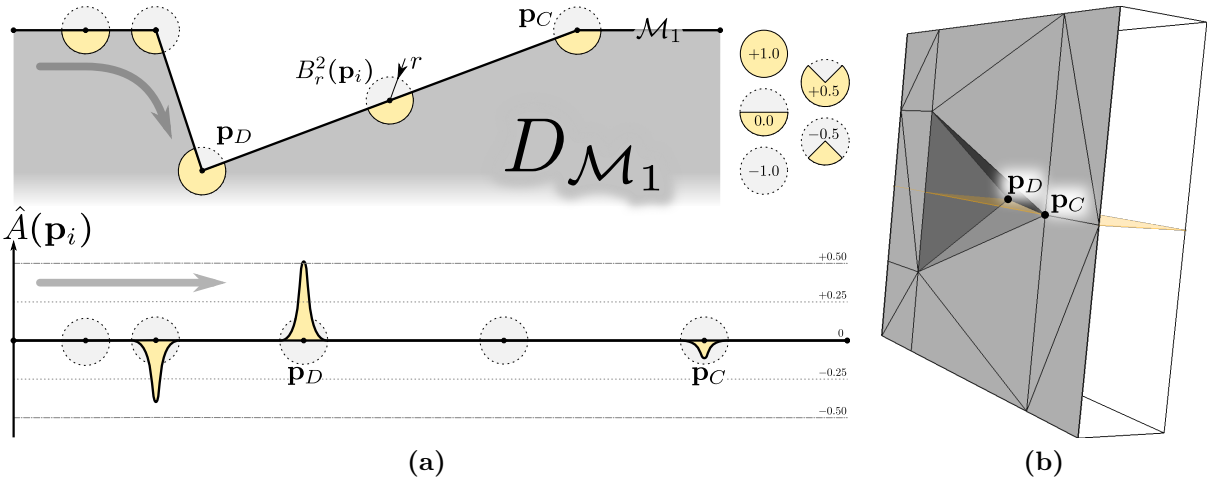
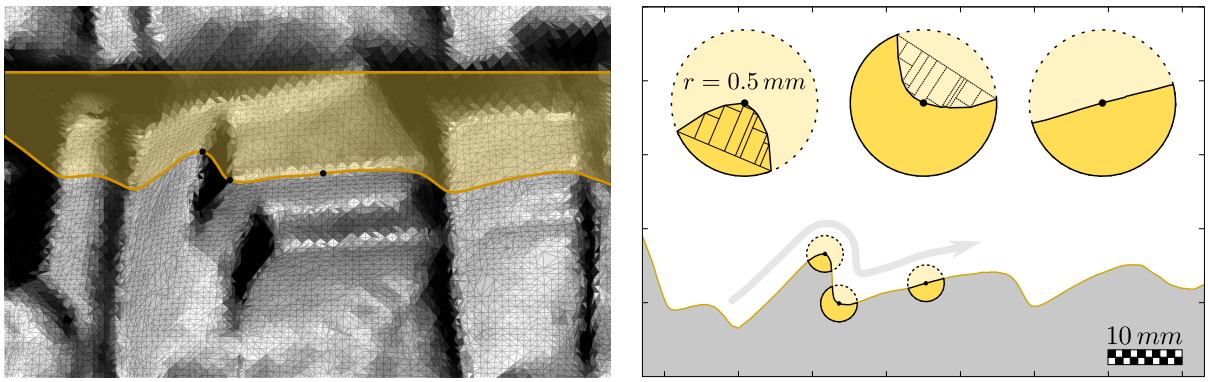


Figure 3.1: Intersection of the 3D-model of the ideal wedge and the integral invariant function being a curvature metric $]-1,+1[$ along the run-length of the intersection. Zero means flat, positive values concave and negative values convex segments.

For practical reasons the radius r is typically given with a unit like millimeter (mm) or meter (m). The unit and the size of r relates to the metric used for feature detection. For the application on cuneiform tablets r is given mm. Figure 3.2 shows an example for a cross-section of a tablet and some of the disks for curvature estimation with a radius r of $0.5\,mm$. As the faces of the mesh in this example have edges with a length of $0.01 \ldots 0.2\,mm$, the circular disk will contain approximately 5 to 10 triangle intersections. This number of intersections of a piecewise flat surface is also the minimum for robust feature detection.

Choosing smaller radii can be used to estimate errors from noise and quantization. The visualization in Figure 3.2a was rendered using flat shading to emphasize the triangular structure of the surface leading to the piecewise linear cross-section. Computing such a cross-section of a 3D-model is described in [MK03]. Figure 3.2b shows, that the intersection of the circular disks with the cuneiform tablet contain several piecewise linear segments. Therefore the area A_r as well as $\hat{A}$ can be computed using triangles, rectangles and arc segments.



(a) **(b)**

Figure 3.2: Cross-section of (a) a triangular mesh of a real wedge impression and a plane. Plot of (b) the two-dimensional cross-section and the three of the circular disks used to compute the surface integral along the run-length.

The orientation of the plane influences the result of the curvature estimation as it skews the curve describing the intersection. Therefore we extend this concept for curves embedded in 2D to surfaces embedded in 3D in the next subsection.

3.2 Invariant Integration of Volume using Regular Sampling

Having surfaces in 3D, spheres B_r can be used to compute the volume below the surface of a 3D-model in an analog manner as the disks B_r^2, which were used to compute the area below a curve embedded in 2D in the previous section. This means that the center $\mathbf{p}_i$ of the spheres $B_r(\mathbf{p_i})$ are always placed at the surface and $\mathbf{1}_{D_\mathcal{M}}$ is the indicator function for the domain $D_\mathcal{M}$ defined by a 3D-model's surface, which leads to the volume integral invariant:

$$V_r(\mathbf{p_i}) = \int\limits_{B_r(\mathbf{p}_i)} \mathbf{1}_{D_\mathcal{M}} dxdydz \quad \text{with} \quad \mathbf{p_i} \in \mathcal{L}_v \quad \text{and} \quad \mathbf{1}_{D_\mathcal{M}} = \begin{cases} 1 & \text{if } (x,y,z) \in D_\mathcal{M} \\ 0 & \text{else} \end{cases} \tag{3.4}$$

The integral is normalized using the volume of the entire sphere to the interval of $]-1,+1[$ having no unit using:

$$\hat{V}_r(\mathbf{p_i}) = \frac{V_r(\mathbf{p_i})}{\frac{1}{2}|B_r|} - 1 = \frac{3V_r(\mathbf{p_i})}{2\pi r^3} - 1 \tag{3.5}$$

Then $\hat{V}_r$ denotes the normalized volume integral invariant, while r indicates its scale respectively an index referring to its scale.

An empty intersection corresponds to -1, which cannot occur for surfaces with an area > 0. However the volume integral can become 0 in case of numeric errors. The same applies to $+1$ for a full intersection. The threshold between concave and convex is 0, which can be interpreted as a flat surface dividing the sphere in two halfs. It can also be interpreted as surface patch with different curvatures $\kappa_i^{\mathbf{n}}(\phi)$ having a mean curvature of 0 using Equation 2.27 (on page 37). This can happen at inflection points of surfaces.

To compute $\hat{V}_r$ it is sufficient to use only the vertex positions $\mathbf{p}_i \in \mathcal{L}_v$ as the number of vertices is smaller than the number of faces. Having a piecewise flat surface the integral can be linearly interpolated for any point on a face. Figure 3.3 shows an example for a parametrized surface using $f(x,y) = \frac{x^3+y^3}{2}$ having a saddle point at $f(0,0)$ resulting in a normalized volume integral $\hat{V}_r$ of 0. Additionally the surface's mean curvature is shown.

The visualization of curvature values using a color-ramp in Figure 3.3 shows the relation of the volume integral to the mean curvature. The most obvious difference is the inverted sign due to the previous definition, secondly the area between high curvatures and low curvatures ($\to 0$) appears to have a more distinct separation for the volume integral. This can be confirmed by computing the distribution of curvature values for this ideal surface for both methods. Figure 3.4 shows the area-percental distribution of curvature values of the surface similar to a gray-level histogram. Beside a more distinct separation of areas of high and low curvatures, we also see that the volume integral is less prone to quantization errors.

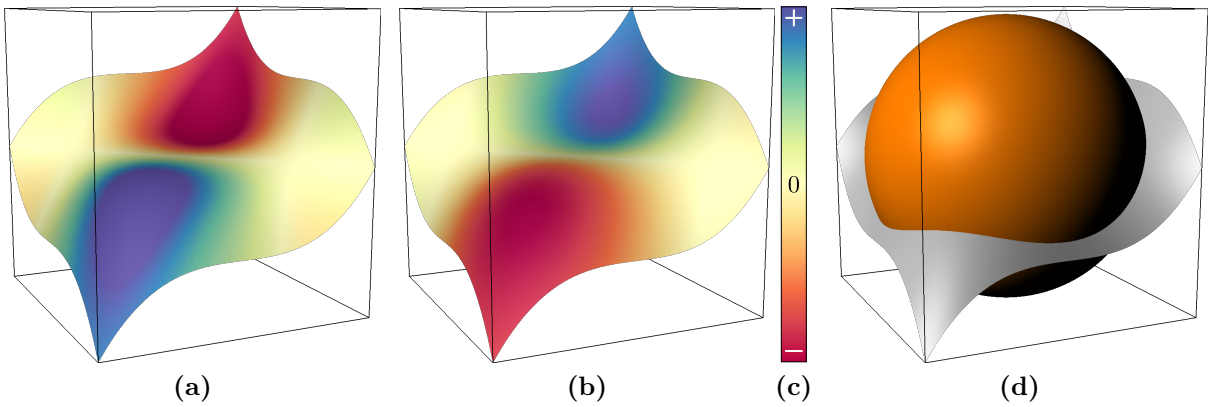


Figure 3.3: Surface $f(x,y) = \frac{x^3+y^3}{2}$ with $x,y = [-1,+1]$ having a saddle point at $f(0,0)$. The false-color texture-map visualizes (a) the mean curvature and (b) the volume integral using (d) spheres with a radius of 1. The color-scale (c) shows the range from negative to positive values.

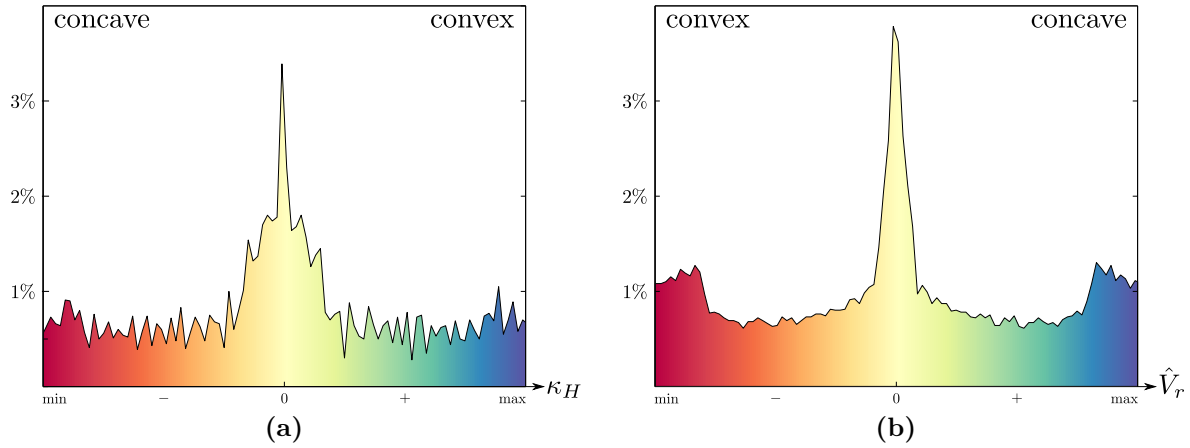


Figure 3.4: Distribution of curvature values (x-axis) as percentage of the surface (y-axis) for (a) mean curvature κ_H and (b) the volume integral invariant $\hat{V}_r$.

Considering that a vertex $\mathbf{p}_i$ is along a sharp edge, while the surface between the edges are flat triangles, we get κ_H^+ and κ_H^- for the mean curvatures to either side of the edge. This means that the edge connects two flat areas having a so-called opening angle α between these areas. Examples for such points along sharp edges are $\mathbf{p}_{AB}$ and $\mathbf{p}_{BD}$ in Figure 2.13 (on page 37). In this case, mean curvature can be estimated for $r \to 0$ as

$$\kappa_H(\mathbf{p}_i) \approx \frac{8}{3r} - \frac{4V_r(\mathbf{p}_i)}{\pi r^4} \tag{3.6}$$

because the higher order terms of

$$\kappa_H(\mathbf{p}_i) = \frac{4}{\pi r^4}\left(\frac{2\pi}{3}\cdot r^3 + O(r^5) - V_r(\mathbf{p}_i)\right) \tag{3.7}$$

become irrelevant.

Equation 3.7 was derived from

$$V_r = \frac{2\pi}{3} \cdot r^3 - \frac{\pi\kappa_H}{4} \cdot r^4 + O(r^5) \tag{3.8}$$

which becomes

$$V_r = \frac{2\alpha}{3} - \frac{\pi\left(\kappa_H^- + \kappa_H^+\right)}{8} \cdot r^4 + O(r^5) \tag{3.9}$$

using *Taylor expansion* for the case of a simple edge. The result contains terms $O(r^n)$ of higher orders of n. The relation to curvature of V_r in Equations 3.8 and 3.9 including the *Taylor expansion* are proven in [HT03, PWHY09].

While solving Equation 3.9 for sharp corners is more complicated, it becomes virtually impossible to provide a solution for all possibilities a triangular mesh can provide. Regarding [Dig10, DM11] the usage of integral invariants is considered to have additional serious numerical drawbacks, because of the influence of $O(r^n)$ having $r \gg 0$ for curvature computation on such meshes. Due to the fact that there is already a quantification error present within these meshs depending on the 3D-scanner's properties, the consequent decision was to design a filter, which makes the best use of the available data and has a negligible error. The filtering algorithm has to provide a robust numerical solution with reasonable performance for practical use with the 3D-models of cuneiform tablets.

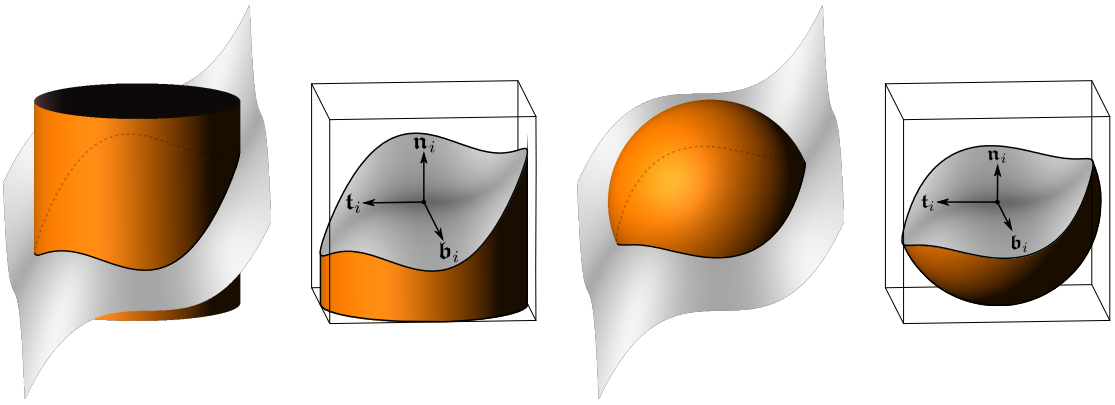
Let us assume we have already a localized surface patch $\mathcal{P}_i \subset \mathcal{L}_\mathcal{M}$ with all vertices and faces in a circular neighborhood of a vertex $\mathbf{p}_i$:

$$\mathcal{P}_i = \mathcal{P}_{i,r} := \{\, \mathbf{t}_j \in \mathcal{L}_f \,|\, \exists \mathbf{p}_k \in \mathbf{t}_j \vee \mathbf{p}_k \cap B_r(\mathbf{p}_i) \neq \emptyset \,\} \tag{3.10}$$

Then Figure 3.5 shows two variants to determining a $\mathcal{P}_i$ for computation of the volume integral as well as a bounding box of $\mathbf{1}_{D_\mathcal{M}}$. The first uses a cylinder, which results in a $\mathcal{P}_i$ consisting of a slightly larger area than required. However in the case of $\mathfrak{n}_i = (0,0,1)^T$, which means an orientated $\mathcal{P}_i$, an implementation can determined the patch faster than the second, more precise variant using a sphere.

As shown in Figure 3.5 the normal $\mathfrak{n}_i$ at a point $\mathbf{p}_i$ is part of the *Frenet-Serret frame* (TNB frame) together with the tangent $\mathfrak{t}_i$ and the bi-tangent $\mathfrak{b}_i$. The TNB frame defines a bounding box for the spherical neighborhood of $\mathbf{p}_i$. Determining $\mathfrak{t}_i$ and $\mathfrak{b}_i$ can be done e.g. by using PCA [YLHP06] or an octree [PWHY09]. Because of the drawbacks of using these methods, we choose to use relations between faces and vertices stored in $\mathcal{L}_f$ and $\mathcal{L}_v$ as it is shown in Section 4.1.

The naïve approach to compute the volume is to use an algorithm, which computes the volume V_r of the intersection using two regular voxel grids. These voxel grids are represented by three-dimensional arrays a (image stacks) of size $n_\boxplus^3$, where each cell $a[i,j,k]$ holds the density of each voxel.



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

Figure 3.5: Local surface patch $\mathcal{P}_i$ (a,b) approximated using a cylinder and (c,d) determined using the ball $B_r(\mathbf{p}_i)$. $\mathbf{n}_i$ is the surface normal and $\mathbf{t}_i$, $\mathbf{b}_i$ are the two tangent vectors at $\mathbf{p}_i$.

The first cell $a[i,j,k]$ represents the sphere, where cells inside the sphere have a density of 1 (full/solid) and the voxels outside the sphere are 0 (empty). As the voxel space has to contain the bounding box defined by the TNB frame, the center of the sphere is located in the center of the array and the radius $r_\boxplus = n_\boxplus/2$. Therefore the sphere's volume having voxel as unit is approximated:

$$|\widetilde{B}_\boxplus| = \sum_{i=1}^{n_\boxplus}\sum_{j=1}^{n_\boxplus}\sum_{k=1}^{n_\boxplus} a_\circ[i,j,k] \quad \approx \quad |B_{r_\boxplus}| = \frac{4\pi r_\boxplus^3}{3} \tag{3.11}$$

The cell values $a_\circ$ are numerically estimated based on $x^2+y^2+z^2=r^2$ describing a sphere

$$a_\circ[i,j,k] = \begin{cases} 1 & (i-1-r_\boxplus)^2+(j-1-r_\boxplus)^2+(k-1-r_\boxplus)^2 \le r_\boxplus^2 \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

where -1 compensates for the arrays indices starting with 1.

The second array $a_{\mathcal{P}_i}$ represents the volume below the surface patch – in other terms it represents a subset of $\mathbf{1}_{D_\mathcal{M}}$ about $\mathbf{p}_i$. $a_{\mathcal{P}_i}$ has the same size as $a_\circ$ and it represents the same subset of $\mathbb{R}^3$ as $a_\circ$. Therefore the cells below $\mathcal{P}_i$ have a density of 1 (full/solid) and the voxels outside the sphere would be 0 (empty). This can be achieved by tracing the arrays column bottom down (along the negative $\mathbf{n}_i$) setting all cells to 0 unless the surface is intersected and the following cells are set to 1. Note that for complexer shapes the tracer has to determine on what side the cells are in relation to the surface and set them accordingly.

Finally these two arrays are multiplied and normalized providing a numerical solution $\hat{V}_\boxplus$ for the volume integral invariant $\hat{V}_r$ in accordance to Equation 3.5:

$$\hat{V}_\boxplus = -1 + \frac{2}{|\widetilde{B}_\boxplus|}\left(\sum_{i=1}^{n_\boxplus}\sum_{j=1}^{n_\boxplus}\sum_{k=1}^{n_\boxplus} a_\circ[i,j,k]\cdot a_\mathcal{P}[i,j,k]\right) \tag{3.13}$$

The result can be improved by computing volume-percental values of the interval $[0, 1]$ based on the intersection of the patch with the voxels $a_{\mathcal{P}}$. Than the same improvement has to be applied for computing $a_{\circ}$, which can be achieved e.g. using sub-sampling.

Using such a naïve approach intrinsically assumes, that the surface patch has no complex shape like undercuttings. From practical experience it is know that this assumption holds for most $\mathbf{p}_i$. Cases not holding this assumption are determined by an angle $\geq 90°$ between $\mathbf{n}_i$ and the normals of each face $\mathbf{t}_j \in \mathcal{P}_i$. These rather rare cases will be covered by an alternative design of our filter. An error estimation is given in Chapter 5.

Regular Sampling of an Irregular Surface Patch

As choosing a sampling interval like $n_{\boxplus}/r$ for a numerical solution, relates to the number of faces of $\mathcal{P}_i$, the relationship to the number of measuring points connecting triangles, is of importance. When r is chosen for a sphere in similar manner than choosing r of the disk B_r^2, than a certain number of faces and vertices is expected within the ball B_r. There were approximately 5 line segments and almost the same number of points within $B_r^2 \in \mathbb{R}^2$ for the example in the previous section and a rough guess is that the number of points has to be squared for vertices in $\mathbb{R}^3$. Meshing a number of n vertices regularly organized in squares means that the number of faces will be twice the number of vertices, because

$$|\mathcal{L}_v| = n^2 \quad \text{and} \quad |\mathcal{L}_f| = \frac{n^2 - 1}{2} \tag{3.14}$$

For a triangulated surface the relation between numbers of vertices, faces and edges can be precisely determined using the *Euler characteristic*:

$$\chi(\mathcal{M}_2) = |\mathcal{L}_v| - \frac{|\mathcal{L}_e| + |\mathcal{L}_{\partial e}|}{2} + |\mathcal{L}_f| \tag{3.15}$$

The term in the middle of this equation is a consequence of $\mathcal{L}_e$ having twice the number of edges, because of the distinction by orientation – except the edges along the boundary of a mesh stored in $\mathcal{L}_{\partial e}$. Note that this equation may give an improper result, when non-manifold edges are present in $\mathcal{L}_e$. Assuming no holes within the surface patch consisting of faces $\mathbf{f}_{\mathcal{P}_i}$, $|\mathcal{L}_{\partial e}|$ becomes 0 and $|\mathcal{L}_v|$ can be solved using Equation 2.14:

$$\chi(\mathcal{M}_2) = |\mathcal{L}_v| - \frac{3 \cdot |\mathbf{f}_{\mathcal{P}_i}|}{2} + |\mathbf{f}_{\mathcal{P}_i}| \quad \Rightarrow \quad |\mathcal{L}_v| = \chi(\mathcal{M}_2) + \frac{|\mathbf{f}_{\mathcal{P}_i}|}{2} \tag{3.16}$$

However, the majority of 3D-models is homeomorphic to a convex polyhedron we assume $\chi(\mathcal{M}_2)$ equals 2. Following *Shannon's theorem* we have to choose twice the number of vertices evenly distributed on a patch of, sampled by an array of size $n_{\boxplus}^2$. Using random samples of $\mathcal{P}_i$ the sampling size $n_{\boxplus}$ can be approximated:

$$n_{\boxplus} \gtrapprox 2 \cdot \sqrt{|\mathbf{p}_{\mathcal{P}_i}|} \quad \equiv \quad n_{\boxplus} \gtrapprox 2 \cdot \sqrt{2 + \frac{|\mathbf{f}_{\mathcal{P}_i}|}{2}} \tag{3.17}$$

Equation 3.17 means that $n_{\boxplus}$ has to be chosen $\gtrapprox 14.4$, expecting ≈ 100 faces per sphere regarding the initial example. In this equation the factor of 2 from the *Shannon's theorem* is preceding, while the addition of 2 – of the *Euler characteristic* – contributes only for patches with a small number of faces.

Transforming the Filter Function

With an increasing r, the number of faces to be quantized into voxels increases with the power of two for flat surfaces – it will increase more for arbitrary surfaces. Even when the root to extracted for estimation of $n_{\boxplus}$, the number of voxels increases by the power of three, which quickly leads to performance and memory issues. To avoid these problems, the intrinsic assumption of a minor number of patches $\mathcal{P}_i$ of complex shape was used to design a new filter, which was inspired by the painting *La persistència de la memòria* by the artist Salvador Dalí [ES95]. Within these painting, melting watches are shown, which still show the time as their function value, while the process of melting changes their shape.

As the idea of melting corresponds to a vertical transformation, the sphere can be transformed, when it is given as parametrized function $f_r(x, y)$ as shown in Equation 3.18. The function $f_r^+(x, y)$ states the upper half and $f_r^-(x, y)$ states the lower half of the sphere. This allows to compute a melted sphere described by $f_r^{\oplus}(x, y)$ enclosing the same volume as $f_r(x, y)$, because the melted lower half of the sphere $f_r^{\ominus}$ becomes 0. Therefore it can be stored as two-dimensional filter mask in an array similar to those used for processing of 2D-images. Equation 3.18 assumes the center of the sphere to be located already at the origin $(0, 0, 0)^T$ and not at $\mathbf{p}_i$.

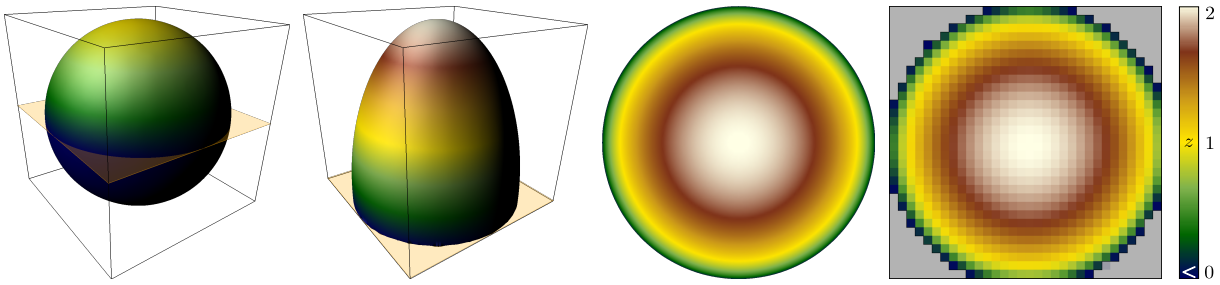
$$f_r(x,y) := \begin{cases} f_r^+(x,y) = +\sqrt{r^2 - x^2 - y^2} \\ f_r^-(x,y) = -\sqrt{r^2 - x^2 - y^2} \end{cases} \xrightarrow{+f_r^+(x,y)} \begin{array}{rcl} f_r^{\oplus}(x,y) & = & 2\sqrt{r^2 - x^2 - y^2} \\ f_r^{\ominus}(x,y) & = & 0 \end{array} \tag{3.18}$$

This transformed and parametrized sphere is stored in a two-dimensional array, where each cell represents a value of $f_r^{\oplus}(x, y)$. Figure 3.6 illustrates the transformation of the sphere to an array, which will be used as filtermask. The color of the cell shows $f_r^{\oplus}(x, y)$, which is normalized to a radius of 1 in voxel space. The gray cells are outside the spherical neighborhood holding no real value. Figure 3.6d is a visualization of the filtermask stored as an array of size $n_{\boxplus}^2$. In other terms we can call this array the heightmap representing $f_r^{\oplus}(x, y)$. The array related to the function f is defined as:

$$f^{\oplus}[j,k] := f_r^{\oplus}(x_j, y_k) \text{ for } j,k \in \{1,\ldots,n_{\boxplus}\} \text{ and } r = \frac{n_{\boxplus}}{2} \tag{3.19}$$

As arrays are typically accessed by a single index, an alternative definition is:

$$f^{\oplus}[j] := f^{\oplus}(x_l, y_m) \text{ with } m = l \bmod n_{\boxplus} \text{ and } l = \frac{j-m}{n_{\boxplus}} \text{ for } j \in \{1,\ldots,n_{\boxplus}^2\} \tag{3.20}$$



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

Figure 3.6: Transformation of (a) the sphere $f_r(x, y)$ with $r = 1$ to (b) $f_1^{\oplus}(x, y)$ and $f_1^{\ominus}(x, y) = 0$. (c) Top-view of the $f_1^{\oplus}(x, y)$, which is (d) rastered into an array $f^{\oplus}[j, k]$ with e.g. 32^2 cells. The gray colored cells have no real value, because these are outside the sphere. (e) Colorramp for z.

The computation of $f^{+}[j, k]$ and $f^{\oplus}[j, k]$ as filtermasks is shown in Algorithm 1. Notice the different brackets for arrays to distinguish them from mathematical functions. The arrow $\leftarrow$ means the assignment of a value to a storage cell, which can be a variable or the cell of an array.

Algorithm 1 Compute the filtermask f^{+} and $f^{\oplus}$

```
for j = 1 to n_⊞ do
    x ← (j − n_⊞/2)/n_⊞
    for k = 1 to n_⊞ do
        y ← (k − n_⊞/2)/n_⊞
        f⁺[j, k] ← √(1 − x² − y²)
        f^⊕[j, k] ← 2 · f⁺[j, k]
    end for
end for
```

In consequence the local patch $\mathcal{P}_i$ has to undergo the same transformation as the sphere. The first step has to translate the vertices of $\mathcal{P}_i$ by $-\mathbf{p}_i$, which moves the center of the patch to the origin $\mathbf{o}$. As second step the patch has to be rotated so that $\mathfrak{n}_i$ equals the orientation of the z-axis $\mathfrak{z}$, which can be done by computing the angle ϕ for rotation about an axis defined by $\mathbf{p}_i$ and a direction $\mathfrak{r}$ using the inner product $\langle \cdot, \cdot \rangle$

$$\phi = \arccos \langle \hat{\mathfrak{n}}_i, \mathfrak{z} \rangle \quad , \quad \mathfrak{r} = \mathfrak{n}_i \times \mathfrak{z} = (u, v, w)^T \quad \text{with} \quad \mathfrak{z} = (0, 0, 1)^T \tag{3.21}$$

This leads to a series of affine transformations, which can be combined within a 4×4 matrix of homogeneous coordinates $\mathbf{A}_{0z_i}$. This combined matrix is multiplied with each vertex $\mathbf{p}_i \in \mathcal{P}_i$

$$\mathbf{p}_i' = \mathbf{p}_i \mathbf{A}_{0z_i} = \mathbf{p}_i \mathbf{T}_{xz}^{-1} \mathbf{T}_z^{-1} \mathbf{R}_z(\phi) \mathbf{T}_z \mathbf{T}_{xz} \mathbf{T}_{\mathbf{p}_i} \tag{3.22}$$

where $\mathbf{T}_{\mathbf{p}_i}$ translates $\mathbf{p}_i$ into the origin and $\mathbf{T}_z \mathbf{T}_{xz}$ rotates $\mathbf{r}$ to the z-axis to perform the rotation about the z-axis with $\mathbf{R}_z(\phi)$.

The transformation matrices required for Equation 3.22 are defined by ϕ and the elements of $\mathfrak{n}_i$ and $\mathfrak{r}$:

$$\mathbf{T}_{\mathbf{p}_i} = \begin{pmatrix} 1 & 0 & 0 & -x_i \\ 0 & 1 & 0 & -y_i \\ 0 & 0 & 1 & -z_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{T}_z = \begin{pmatrix} w/|\mathfrak{r}| & 0 & -\sqrt{u^2+v^2}/|\mathfrak{r}| & 0 \\ 0 & 1 & 0 & 0 \\ \sqrt{u^2+v^2}/|\mathfrak{r}| & 0 & w/|\mathfrak{r}| & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.23}$$

$$\mathbf{R}_z(\phi) = \begin{pmatrix} cos(\phi) & -sin(\phi) & 0 & 0 \\ sin(\phi) & cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{T}_{xz} = \begin{pmatrix} u/\sqrt{u^2+v^2} & v/\sqrt{u^2+v^2} & 0 & 0 \\ -v/\sqrt{u^2+v^2} & u/\sqrt{u^2+v^2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.24}$$

When $|\mathfrak{r}| = |\hat{\mathfrak{r}}| = 1$ the final transformation matrix becomes:

$$\mathbf{A}_{0z_i} = \begin{pmatrix} u^2+(v^2+w^2)cos(\phi) & uv(1-cos(\phi))-w\sin(\phi) & uw(1-cos(\phi))-v\sin(\phi) & -x_i \\ uv(1-cos(\phi))-w\sin(\phi) & v^2+(u^2+w^2)\cos(\phi) & vw(1-cos(\phi))-u\sin(\phi) & -y_i \\ uw(1-cos(\phi))-v\sin(\phi) & vw(1-cos(\phi))-u\sin(\phi) & w^2+(u^2+v^2)\cos(\phi) & -z_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.25}$$

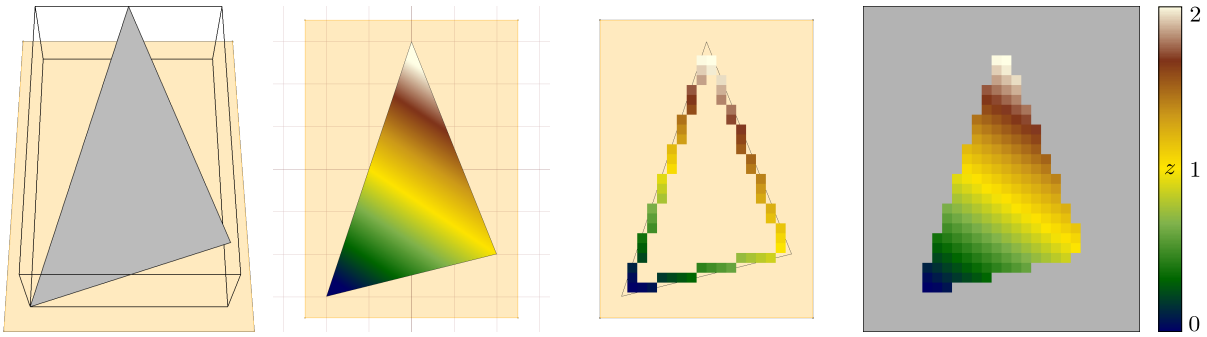
which can be implemented using a small number of commands and therefore rapidly computed for each vertex of each patch $\mathcal{P}_i$.

Having the patch centered at the origin and aligned to the z-axis, we could treat $\mathcal{P}_i$ like a parametrized surface and shift each vertex by adding $\sqrt{r^2-x^2-y^2}$ to its position similar to Equation 3.18. This is problematic for an implementation, because faces intersecting the sphere having one or two vertices outside the spherical neighborhood will get an imaginary z-coordinate whenever $x^2+y^2 > r^2$.

A solution would be to compute new vertices for these faces within the same plane, which are located inside the sphere. This leads to a computationally expensive case-by-case analysis of all the possibilities of the intersections of an arbitrary triangle with a sphere. To avoid the analysis of cases, the triangles are rasterized as heightmap $\mathcal{P}_i[j,k]$ into an array having the same size $n_{\boxplus}^2$ as the filtermask.

Figure 3.7 demonstrates the process of rasterization of a triangle $\mathbf{t} \in \mathcal{P}_i$: First the edges are rasterized into the heightmap using [Bre65]. Afterwards the values for the cells within the triangle are interpolated. Anti-aliasing of the edges [For85] is not required as the adjacent triangles will automatically fill the adjacent cells of $\mathcal{P}_i$ and it unnecessarily decreases computing performance.

As the surfaces $\mathcal{M}_2$ can have borders, the array $\mathcal{P}_i[j,k]$ can contain cells with no value. Like for any other filter the treatment of borders along areas of missing data is important for robustness. In the domain of image processing in 2D these borders are typically the borders of an array representing an image and can be treated as shown in [Jäh05], pp. 49. For the volume computation using the heightmap of $\mathcal{P}_i$ the borders of its array are of no concern as the surface patch is centered and scaled to fit into the array. Figure 3.8a shows the heightmap of a synthetic surface patch. This is achieved using the bounding box



(a) **(b)** **(c)** **(d)** **(e)**

Figure 3.7: Rasterization of a triangle $\mathbf{t} := \{(-2,-3,0)^T, (2,-2,1)^T, (0,3,2)^T\}$ shown (a) with bounding box and xy-plane and (b) its top-view colored by according to the colorramp in (e). Edges rastered into the array $\mathcal{P}_i[j,k]$ are shown in (c), which are used for (d) interpolating the cells in-between. Gray means no value.

defined by the TNB frame for this approach using a single radius (scale). When there are triangles missing in the patch itself, we get arbitrary areas of missing data within the array and along its borders.

To treat areas enclosed by borders we have to detect them first. This is achieved by initializing the cells of $\mathcal{P}_i[j,k]$ with `NaN`, i.e. "not a number". Rasterizing the triangles of $\mathcal{P}_i$ will fill all cells covered by triangles with a valid number, which will leave missing areas tagged as `NaN`. An example is shown in Figure 3.8b, where data is missing within the array, along the border and at a corner. These three cases are treated by interpolation:

1. Interpolate z for each row from left to right.
2. Interpolate the column along the border from top to bottom. Afterwards use this column to interpolate from left to right.
3. Compute the average of z of the corner using the closest cell left and the closest cell above the corner. This average value is used to interpolate the column along the border. Finally the values of the missing rows are filled.

The last case works analog for the other corners. Practically an implementation will treat the corners first; the columns at the left and right borders second; and finally the rows. Figure 3.8d shows the absolute error of the interpolated cells, which has a relative maximum of 7.7% with respect to the radius of the sphere. The error typically increases towards the corners of the array. As we use a cylindrical neighborhood the areas next to the corners have the least influence on our filter result.

Having the surface patch as heightmap, the next step is to compute the transformation $\mathcal{P}'_i[j,k] \leftarrow \mathcal{P}_i[j,k]$ using $f^{\oplus}[j,k]$ as it is shown in Algorithm 2. This algorithm as well as its improved version is designed to handle surface patches with and without border treatment. Border treatment is more relevant for meshes with lower resolution, while it can cost unnecessary computing time for high resolution meshes without gaining accuracy.

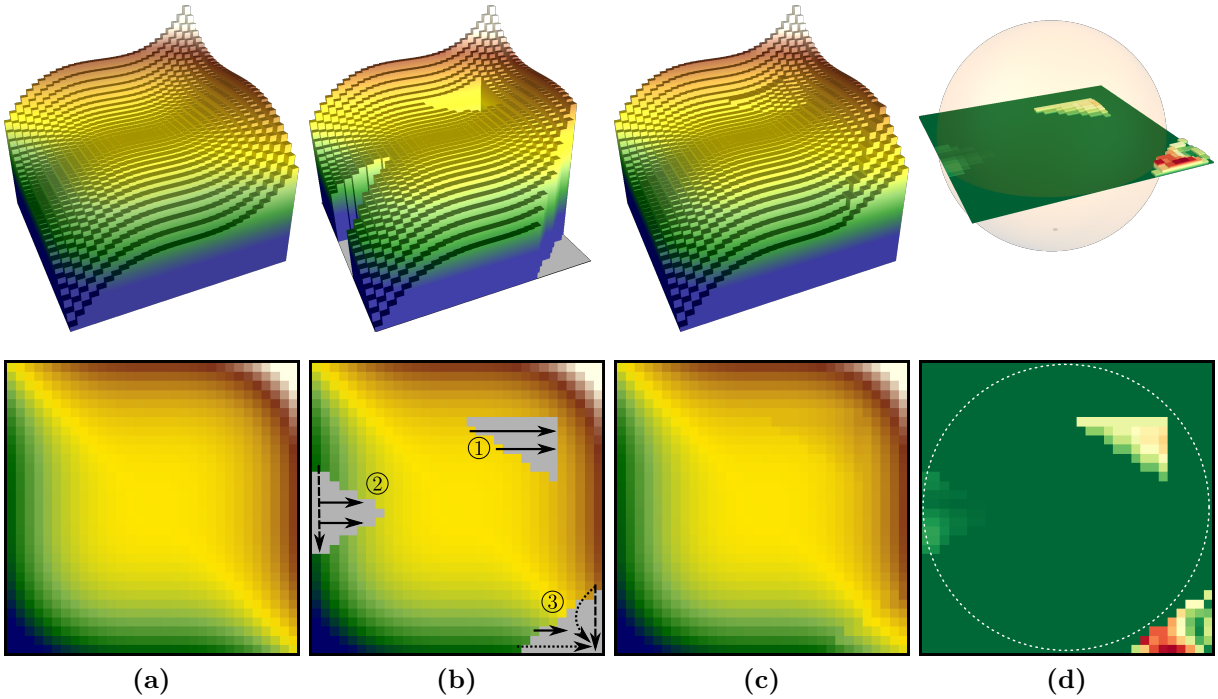


Figure 3.8: Synthetic surface patch rastered as heightmap (a) complete and (b) with parts (triangles) missing. The array (c) with interpolated cells and their (d) absolute error. The intersection with the sphere shows the relevance of these errors for further computing.

The if-statement in Algorithm 2 tags parts of $\mathcal{P}_i$, which are below the spherical neighborhood. This is important for the numerical integration of the intersection of $\mathcal{P}'_i[j,k]$ and $f^{\oplus}[j,k]$ as these parts have to be skipped for integration. As basic mathematical operations are much faster to compute than access to the memory, the performance can be improved by adding the intersection and the integration to Algorithm 2. This is achieved by adding the minimum of the arrays $\mathcal{P}'_i[j,k]$ and $f^{\oplus}[j,k]$ as shown in Algorithm 3: The principle and the result of transforming $\mathcal{P}_i$ with Algorithm 3 is shown in Figure 3.9.

Algorithm 2 Transform a surface patch $\mathcal{P}'_i[j,k] \leftarrow \mathcal{P}_i[j,k]$

```
for j = 1 to n_⊞ do
    for k = 1 to n_⊞ do
        P'_i[j,k] ← P_i[j,k] + f^⊕[j,k]/2
        if P'_i[j,k] < 0 then
            P_i[j,k] is below the sphere
        end if
    end for
end for
```

As the integral invariant $\hat{V}_r(\mathbf{p}_i)$ itself is the only result of interest, the algorithm can be simplified and its performance improved by skipping the computation of $\mathcal{P}'_i$ and $\mathcal{P}''_i$. Therefore f_r^+ is used as the only filtermask. Additionally we can dissolve the $min(\dots)$

Algorithm 3 Computing $V_r(\mathbf{p}_i)$ of the intersection between sphere and surface

$V_r(\mathbf{p}_i) \leftarrow 0$
for $j = 1$ to $n_{\boxplus}$ **do**
  for $k = 1$ to $n_{\boxplus}$ **do**
    $\mathcal{P}'_i[j,k] \leftarrow \mathcal{P}_i[j,k] + f^{\oplus}[j,k]$
    if $\mathcal{P}'_i[j,k] < 0$ **then**
      $\mathcal{P}'_i[j,k] = 0$
    end if
    $\mathcal{P}''_i[j,k] \leftarrow min(\mathcal{P}'_i[j,k], f^{\oplus}[j,k])$
    $V_r(\mathbf{p}_i) \leftarrow V_r(\mathbf{p}_i) + \mathcal{P}''_i[j,k]$
  end for
end for

operation by using "$\not\geq$" instead of "<". This will not only skip unnecessary computing operations below the sphere in programming languages like *C++*, it will also automatically avoid unnecessary calculations outside the sphere for imaginary entries of $\mathcal{P}_i[j]$ – see Section A.2 (page 159).

These performance relevant improvements are shown in Algorithm 4. Regarding real world data, the Algorithm 3 is 100 to 1000 times faster than a naïve implementation, while the improvement steps leading to Algorithm 4 increase the performance by an additional factor of 10 to 20. Additionally the algorithm contains the normalization of $V_r(\mathbf{p}_i)$ to $\hat{V}_r(\mathbf{p}_i)$. Note that $|\bar{B}_r|$ is the volume for a half of the sphere and it depends only on the radius. Therefore it is pre-computed for all $\mathbf{p}_i$. Figure 3.10 shows the result for $\hat{V}_r$ with $r = 1\,mm$ for the synthetic wedge and the part of the triangular mesh from Figure 3.2a (on page 44).

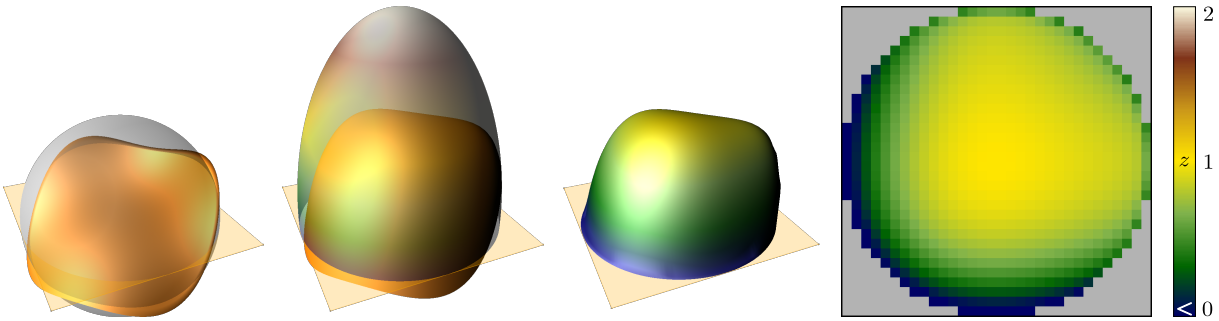
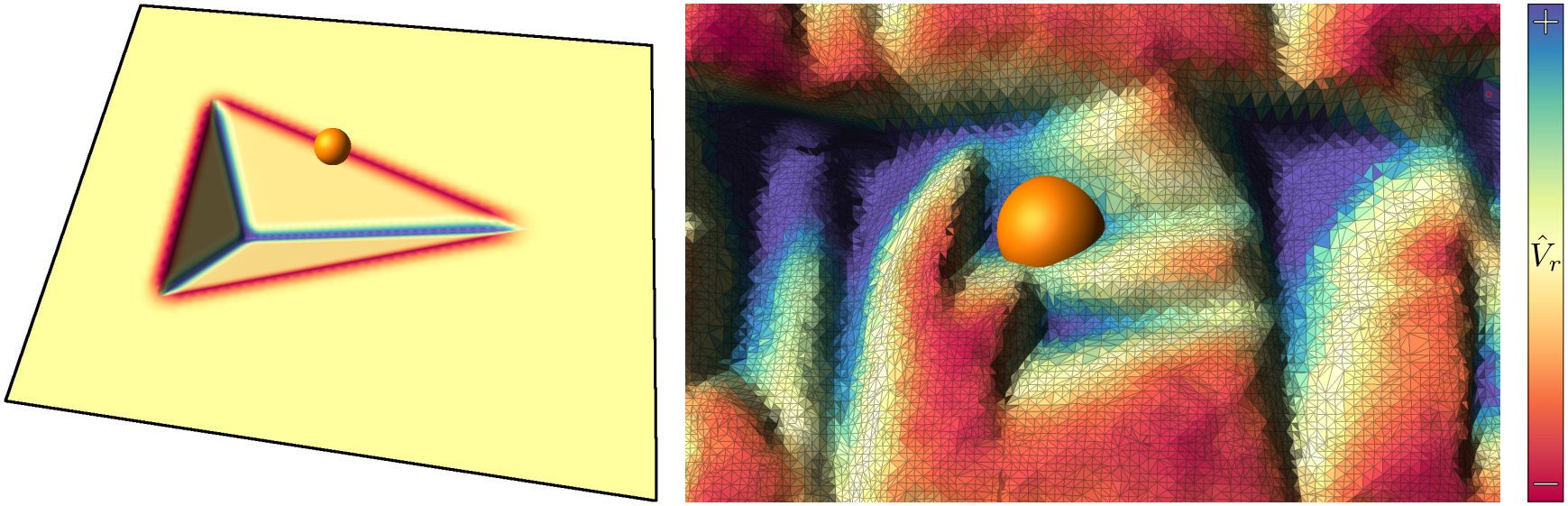


Figure 3.9: Visualization of Algorithm 3 to compute $V_{r=1}$ at $(0,0,0)^T$ for $\mathcal{P}_2 = (x^3 + y^3)/2$. (a) Sphere and the cylindric surface patch of $\mathcal{P}_2$. (b) Both are transformed using $f_r^{\oplus}$. (c) Result after $min(\dots)$ stored in the (d) array $\mathcal{P}''_2$ with e.g. 32^2 cells. (e) Colorramp for z.

Algorithm 4 Faster computation of $V_r(\mathbf{p}_i)$ including normalization to $\hat{V}_r(\mathbf{p}_i)\ \forall\ \mathbf{p}_i$

$|\widetilde{B}_{\boxplus}| \leftarrow \sum_{j=1}^{n_{\boxplus}^2} f_r^+[j] \approx \frac{2}{3}\pi r^3$ ▷ Numeric approximation of half of the sphere's volume
for $\mathbf{p}_i \in \mathcal{L}_v$ **do**
  $V_r(\mathbf{p}_i) \leftarrow 0$
  for $j = 1$ to $n_{\boxplus}^2$ **do**
    if $\mathcal{P}_i[j] \not> -f_r^+[j]$ **then**
      continue
    end if
    if $\mathcal{P}_i[j] > f_r^+[j]$ **then**
      $V_r(\mathbf{p}_i) \leftarrow V_r(\mathbf{p}_i) + 2 \cdot f_r^+[j]$
    else
      $V_r(\mathbf{p}_i) \leftarrow V_r(\mathbf{p}_i) + \mathcal{P}_i[j] + f_r^+[j]$
    end if
  end for
  $\hat{V}_r(\mathbf{p}_i) \leftarrow \frac{V_r(\mathbf{p}_i)}{|\widetilde{B}_{\boxplus}|} - 1$ ▷ Normalization to $]-1, +1[$
end for



(a) (b) (c)

Figure 3.10: $\hat{V}_r$ computed using spheres having $r = 1\,mm$ for (a) the synthetic wedge and (b) real wedges acquired with a 3D-scanner. (c) Colorramp for $\hat{V}_r$: negative values in red correspond to convex areas. Concave regions become positive shown in blue. For a better perception of the surface a simple light model was used leading to darkened colors.

Volume Integral Invariants for $r \to \infty$ and $r \to 0$

The previously shown algorithms provide means to estimate a curvature, which can be used to determine features with a certain size. If analyzing the noise of a mesh or its general curvature properties as well as determining the maximum meaningful value of r for local filtering, it is necessary to compute the volume integral for $r \to 0$ and $r \to \infty$. This subsection does not apply Equation 3.5 for normalization to simplify the following equations, while an implementation has to maintain the normalization to $]-1,1[$.

In case $r \to \infty$ the ball $B(r)$ will grow steadily using increasing amounts of the volume enclosed by the mesh. At a radius $\mathtt{r}_{max_i}$ the ball will have enclosed the complete mesh and its volume. Therefore $V_{r \geq \mathtt{r}_{max_i}}$ is constant while $|B_r(\mathbf{p}_i)| \to \infty$ is leading to

$$\lim_{r\to\infty} \hat{V}_r(\mathbf{p}_i) = \lim_{r\to\infty} \frac{V_r(\mathbf{p}_i)}{|B_r(\mathbf{p}_i)|} = 0 \tag{3.26}$$

Because $r \geq \mathtt{r}_{max_i}$ leads to a continuously decreasing $\hat{V}_r(\mathbf{p}_i)$ by $1/r^3$, the upper boundary for r as a parameter for filtering at a point $\mathbf{p}_i$ can be given as $\mathtt{r}_{max_i}$. Furthermore the minimum of $\mathtt{r}_{max_i}$ is of interest, as it is the upper boundary for r, where the filter can have a meaningful result for a given mesh. This boundary $\mathtt{R}_{max}$ can be computed by

$$\mathtt{R}_{max} = \min_i(\mathtt{r}_{max_i}) = \min_i \left(\max_j \left(\left| \begin{pmatrix} x_j \\ y_j \\ z_j \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \right| \right) \right) \quad \forall \quad i,j \in \{1,\ldots,|\mathcal{L}_v|\} \tag{3.27}$$

An implementation of Equation 3.27 for real world data should make use of e.g. *Binary Space Partitioning* (BSP) [FKN80] like octrees, otherwise the computation time will increase by $O(n^2)$ with $n = |\mathcal{L}_v|$. As Equation 3.27 does not require any connectivity information i.e. faces, it can also be applied on point clouds and therefore it is suitable for a wider range of input data provided e.g. by TLS 3D-scanners.

When $r \to 0$, the ball will shrink until it contains only $\mathbf{p}_i$ and parts of the faces of its 1-ring (see Section 2.2, page 29). This means that for a radius $\mathtt{r}_{min_i} \geq r > 0$, the volume $V_{r \leq \mathtt{r}_{min_i}}$ can be expressed using a constant:

$$V_{r \leq \mathtt{r}_{min_i}} = const\, \frac{4}{3}\pi r^3 = const\, |B_r| \tag{3.28}$$

This allows for computing $\hat{V}_{r\to 0}(\mathbf{p}_i)$ using *l'Hôpital's rule*:

$$\hat{V}_{r\to 0}(\mathbf{p}_i) = \lim_{r\to 0} \hat{V}_r(\mathbf{p}_i) = \lim_{r\to 0} \frac{V_r(\mathbf{p}_i)}{|B_r(\mathbf{p}_i)|} = \frac{\lim\limits_{r\to 0} V_{r \leq \mathtt{r}_{min_i}}}{\lim\limits_{r\to 0} |B_r|} = const \quad \text{with} \quad 0 < r \leq \mathtt{r}_{min_i} \tag{3.29}$$

because nominator and denominator tend to 0, but their (third) derivatives are constant:

$$V'''_{r \leq \mathtt{r}_{min_i}} = const \cdot (4\pi r^2)'' = const \cdot (8\pi r)' = const \cdot 8\pi \quad \text{and} \quad |B_r|''' = 8\pi \tag{3.30}$$

Determining $\min_i(\mathtt{r}_{min_i})$ as the lower boundary $\mathtt{R}_{min}$ for r used as filter parameter for a mesh can be computed analog to Equation 3.27 using:

$$\mathtt{R}_{min} = \min_i \left(\min_{j \neq i} \left(\left| \begin{pmatrix} x_j \\ y_j \\ z_j \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \right| \right) \right) \quad \forall \quad i, j \in \{1, \ldots, |\mathcal{L}_v|\} \tag{3.31}$$

While the above equation can also be computed without connectivity information, a much less naïve implementation for a mesh should make use of the 1-ring shown in Section 2.2 on page 29. Such a 1-ring contains faces $\mathring{\mathbf{t}}_i$ sharing the vertex $\mathbf{p}_i$, which is used to determine the edges $\mathbf{e}_j(i)$ connected to $\mathbf{p}_i$:

$$\mathtt{R}_{min} = \min_i \left(|\mathbf{e}_j(i)| \right) \tag{3.32}$$

Therefore no BSP is required and the number of computing operations is dramatically minimized, because having high-resolution implies:

$$\mathring{\mathbf{t}}_i \subset \mathcal{L}_f \quad \text{with} \quad |\mathring{\mathbf{t}}_i| \ll |\mathcal{L}_f| \tag{3.33}$$

For an implementation of Equation 3.32, the orientation of the edges of the triangles can be neglected. Therefore the edges between $\mathbf{p}_i$ and all other vertices of the 1-ring have to be used, which is typically provided by the algorithms managing a mesh. This avoids doubling the number of computations as each directed edge exists twice in $\mathcal{L}_e$, when there is no border as defined in Equation 2.16.

The radii shown in the previous equations are typical and computable for each dataset and can be ordered:

$$0 < \mathtt{R}_{min} \leq \mathtt{r}_{min_i} \leq r \leq \mathtt{R}_{max} \leq \mathtt{r}_{max_i} < \infty \tag{3.34}$$

where $\mathtt{R}_{min}$ reflects the resolution provided by a 3D-scanner and $\mathtt{R}_{max}$ corresponds to the overall size of the acquired object. The distribution of $\mathtt{r}_{min_i}$ is also an acquisition related indicator: e.g. a large scattering of values for $\mathtt{r}_{min_i}$ often means a mesh simplification for data reduction. Choosing $r \to \mathtt{R}_{max}$ leads to a classificator of the object's shape. Meaningfull results for filtering using the integral invariant on real data can only be computed within $[\mathtt{R}_{min}, \mathtt{R}_{max}]$.

Having these meaningful boundary values for r, the volume

$$V_{\mathtt{R}_{min}}(\mathbf{p}_i) \equiv V_{r \leq \mathtt{r}_{min_i}} \quad \forall \quad i \in \{1, \ldots, |\mathcal{L}_v|\} \tag{3.35}$$

has to be computed to determine the actual value of $\hat{V}_{r \to 0}(\mathbf{p}_i)$ e.g. for noise detection. The algorithm to compute $\mathcal{P}_i[j,k]$ can be used for this purpose as it is, because it can be applied to the lists of triangles $\mathring{\mathbf{t}}_i$ of the 1-rings and the radius of $\mathtt{R}_{min}$. In consequence the Algorithm 4 can also be used without further modification to compute $V_{\mathtt{R}_{min}}(\mathbf{p}_i)$. The computation of $V_{\mathtt{R}_{max}}(\mathbf{p}_i)$ can be computed in an analog manner.

3.3 Surface and Line based Integral Invariants

Computing the normalized volume integral invariant is one means to compute characteristic values for surface features, which may not be sufficient for finding specific features on $\mathcal{M}_2$. The reason is, that computing $\hat{V}_r(\mathbf{p}_i)$ is a projection of a local property of a surface patch to single value. This means, that patches with different shape can and do map on the same value. As volume integration relates to mean curvature, the incrementation of the feature space using a different projection is required.

Figure 3.11 shows two different surface patches having the same volume integral invariant $\hat{V}_{r=2}(\mathbf{p}_w) = \hat{V}_{r=2}(\mathbf{p}_p) = 0.47$ due to the different depth of the impressions: The depth for the pyramid shaped impression at $\mathbf{p}_p$ is 0.695, while the wedge shaped impression at $\mathbf{p}_w$ has a depth of 1. These two examples represent a synthetic variant for a wedge shape impression $\mathcal{P}_w$ as expected for cuneiform characters, while $\mathbf{p}_p$ in a rather flat $\mathcal{P}_p$ relates to measurement outliers (noise). The distinction of these patches requires at an additional feature metric.

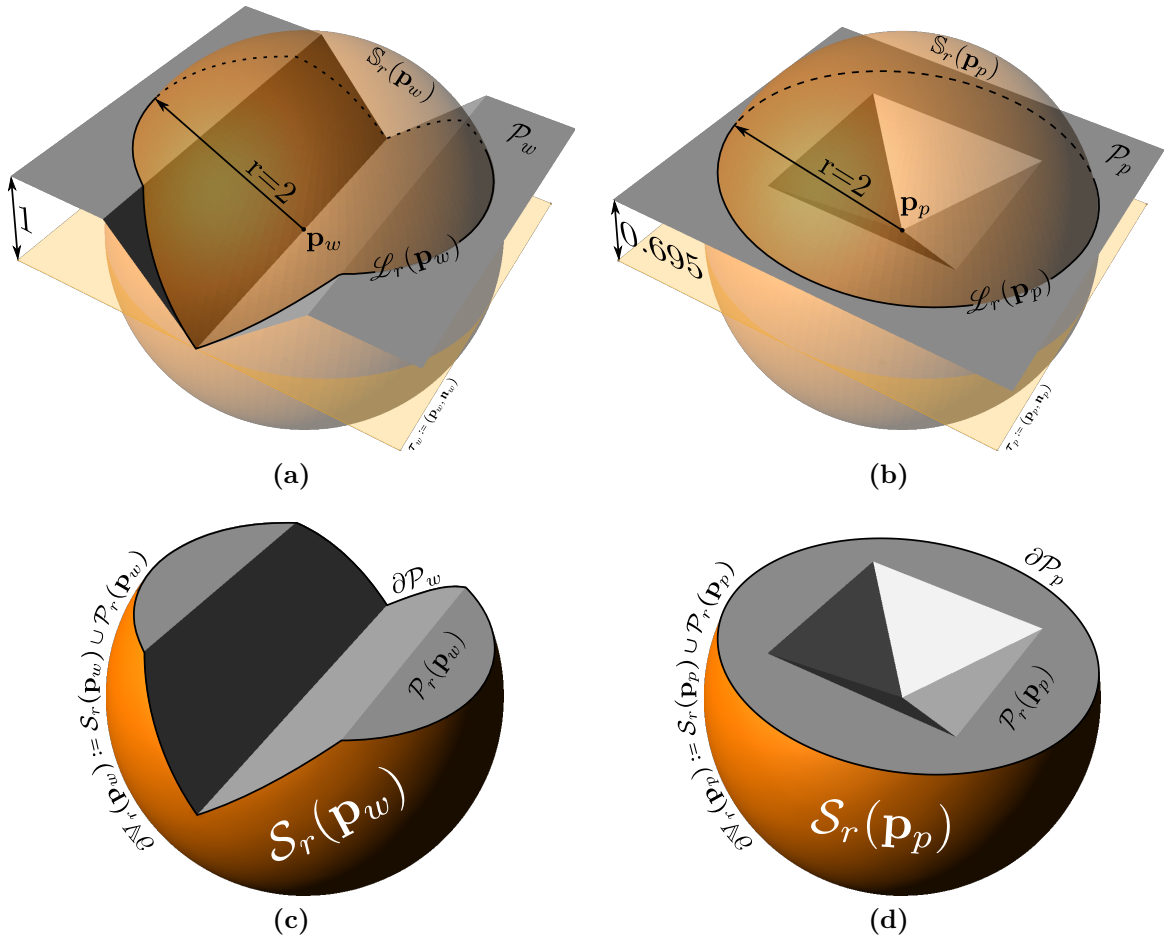


Figure 3.11: Two different surface patches $\mathcal{P}_{i=w,p}$ having the same volume integral invariant: (a,c) for a deep wedge at $\mathbf{p}_w$ and (b,d) a shallow pyramid impression at $\mathbf{p}_p$. (a,b) Intersection of the 2×2 patches with the sphere $B_{r=2}(\mathbf{p}_i)$, resulting in (c,d) the border surface $\partial\mathbb{V}_r(\mathbf{p}_i)$ and the surfaces' border lines $\partial\mathcal{P}_i = \partial\mathcal{S}_i$.

As shown in Figure 3.11d and Figure 3.11c, the borders ∂V_r along the integrated volumes is the union of two surfaces, which can be used for this purpose. The first surface is essentially the patch $\mathcal{P}_i$ already used in previous sections. The subset $\mathcal{S}_r$of the sphere's surface $\mathbb{S}_r$ – defined in Equation 3.1c – is the second surface enclosing the volume:

$$\mathcal{S}_r(\mathbf{p}_i) \coloneqq \mathbb{S}_r(\mathbf{p}_i) \cap \mathbf{1}_{D_\mathcal{M}} \tag{3.36}$$

Note that Equation 3.36 is related to Equation 3.4 leading to the volume based integral invariant V_r, when the definite integral using the ball as interval is formulated as intersection with the manifold's indicator function:

$$\mathbb{V}_r(\mathbf{p}_i) \coloneqq B_r(\mathbf{p}_i) \cap \mathbf{1}_{D_\mathcal{M}} \tag{3.37}$$

The same modus operandi using $\mathcal{M}_2$ instead of its indicator function leads to two additional manifolds. The first manifold is a surface patch $\mathcal{P}_r$ similar to $\mathcal{P}_i$ defined in Equation 3.10 on page 47 for the numeric solution of V_r:

$$\mathcal{P}_r(\mathbf{p}_i) \coloneqq \mathcal{M}_2 \cap B_r(\mathbf{p}_i) \tag{3.38}$$

The second manifold is a 1D-dimensional curve $\mathscr{L}_r$ embedded in $\mathbb{R}^3$ defined as:

$$\mathscr{L}_r(\mathbf{p}_i) \coloneqq \mathcal{M}_2 \cap \mathbb{S}_r(\mathbf{p}_i) \tag{3.39}$$

Furthermore this 1D-dimensional manifold is the shared border of the surface based integral invariants:

$$\mathscr{L}_r = \partial\mathcal{S}_r = \partial\mathcal{P}_r \tag{3.40}$$

In this and the following sections these surfaces $\mathcal{P}_r$, $\mathcal{S}_r$ and their border line $\mathscr{L}_r$ are used to compute further integral invariants acting as additional feature metrics.

Let $\partial\mathbb{V}_r$ be the border of the volume $\mathbb{V}_r$ leading to the integral invariant V_r, then this border can be defined as union of the surface $\mathcal{S}_r$ and the top surface $\mathcal{P}_r$:

$$\partial\mathbb{V}_r(\mathbf{p}_i) \coloneqq \mathcal{S}_r(\mathbf{p}_i) \cup \mathcal{P}_r(\mathbf{p}_i) \tag{3.41}$$

In short this equation becomes $\partial\mathbb{V}_r \coloneqq \mathcal{S}_r \cup \mathcal{P}_r$ leading to a sum of surface based integral invariants:

$$|\partial\mathbb{V}_r| = |\mathcal{S}_r| + |\mathcal{P}_r| \tag{3.42}$$

having e.g. mm^2 as unit for real world data, which can be normalized using the half of a sphere in accordance with the normalization of the volume integral invariant. This means a normalization to a disk B_r^2 for $\mathcal{P}_r$, because it is the top surface – leaving the surface of half of a sphere's surface for normalization of $\mathcal{S}_r$ to a surface based integral invariant:

$$\hat{S}_r = \frac{|\mathcal{S}_r|}{\frac{1}{2}|\mathbb{S}_r|} - 1 = \frac{|\mathcal{S}_r|}{2\pi r^2} - 1 \quad \text{and} \quad \hat{P}_r = \frac{|\mathcal{P}_r|}{|B_r^2|} = \frac{|\mathcal{P}_r|}{\pi r^2} \tag{3.43}$$

Because $\mathcal{S}_r \subset \mathbb{S}_r$, the interval of the normalized integral invariant $\hat{S}_r$ is limited to $]-1,+1[$. In contrast $\mathcal{P}_r$ is subset of an arbitrary surface, its absolute value can reach

$+\infty$, because fractal shapes cannot be excluded in general. Therefore $\hat{P}_r$ has an interval of $]0, +\infty[$. For real world radii $r \gg 0$, $\hat{P}_r$ and $\hat{B}_r$ should not be summarized, because analog to Equation 3.26 the surface integral invariant $\hat{B}_r$ for a finite mesh decreases continuously by $1/r^2$ for $r \geq \mathtt{R}_{max}$ to

$$\lim_{r\to\infty} \hat{S}_r = \frac{\lim_{r\to\infty} |\mathcal{S}_r|}{2\pi r^2} - 1 = -1 \tag{3.44}$$

resulting in

$$\lim_{r\to\infty} \widehat{|\partial V_r|} = \lim_{r\to\infty} \hat{S}_r + \lim_{r\to\infty} \hat{P}_r = \lim_{r\to\infty} \hat{P}_r \ . \tag{3.45}$$

Regarding curvature, the surface based integral invariant $P_r = |\mathcal{P}_r|$ is related to *Gaussian curvature*, because

$$|\mathcal{P}_r| = 2\pi r^2 - \pi\kappa_G r^3 + O(r^4) \tag{3.46}$$

for $r \ll 1$ as shown in Equation 20 of [PWHY09] and proven by [HT03]. From a practical point of view $\hat{P}_r$ is an indicator for surface roughness, when it is used in conjunction with $\hat{V}_r$, which gives the overall shape of a surface patch. as it leads to a more meaningful value describing a surface than $\hat{S}_r$. Having a surface consisting of triangles the numeric solution of the normalized surface integral invariant $\hat{P}_r$ is

$$\hat{P}_r = \frac{|P_r|}{|B_r^2|} = \frac{1}{|B_r^2|} \sum_j A_B(\mathbf{t}_j) \tag{3.47}$$

where A_B is the area of the part of a triangle inside a ball

$$A_B(\mathbf{t}_j) = A_B(\mathbf{t}_j, \mathbf{p}_i, r) = |\, \mathbf{t}_j \cap B_r(\mathbf{p}_i)\,| \tag{3.48}$$

Because of Equation 3.45 and 3.46, computing $\hat{P}_r$ is of greater interest, Therefore the next step is to design an algorithm for this surface based integral invariant for meshes. Given the fact that

$$\partial\mathcal{P}_r = \partial\mathcal{S}_r \quad \rightarrow \quad |\partial\mathcal{P}_r| = |\partial\mathcal{S}_r| \tag{3.49}$$

computing the normalized length

$$\hat{L}_r = \frac{|\partial\mathcal{P}_r|}{|\mathbb{S}_r^1|} = \frac{|\partial\mathcal{P}_r|}{2\pi r} \tag{3.50}$$

instead of $\hat{B}_r$ will provide a related line based integral invariant L_r, requiring almost no extra computing steps as shown in the next Section.

3.3.1 Fast Estimation for Semi-Regular Dense Data

Assuming that $\mathcal{M}_2$ is regular, i.e. consisting ideally of congruent equilateral triangles, means that all edges have the same length $|\mathbf{e}_{\equiv}|$. Than the surface integral invariant can be estimated as:

$$\hat{P}_r(\mathbf{p}_i) \approx \frac{1}{\pi r^2}\sum_{j=1}^{n} A(\mathbf{t}_j) = \frac{1}{\pi r^2} n \frac{\sqrt{3}}{4}|\mathbf{e}_{\equiv}|^2 \;\forall\; \mathbf{t}_j \in \mathcal{P}_i \text{ with } j = \{1,\ldots,n\} \text{ and } |\mathbf{e}_{\equiv}| \ll r \qquad (3.51)$$

In short terms the $\hat{P}_r(\mathbf{p}_i)$ can be determined very quickly by counting the faces of $\mathcal{P}_i$. Note that this requires the faces $\mathbf{t}_j$ to be within a spherical neighborhood, otherwise n will count faces outside $B_r(\mathbf{p}_i)$ as shown in Figure 3.5.

A fast estimator for $\hat{L}_r(\mathbf{p}_i)$ for a regular $\mathcal{M}_2$ is the number of faces of $\mathcal{P}_i$ intersecting $\mathbb{S}_r(\mathbf{p}_i)$. For $r \gg |\mathbf{e}_{\equiv}|$ a good approximation is the number of faces having an edge $\mathbf{e}_{kl}$ intersecting the sphere. As $\mathbf{p}_i$ is the center of B_r the faces $\mathbf{t}_{\partial i}$ along $\mathscr{L}_r(\mathbf{p}_i)$ can be determined:

$$\mathbf{t}_{\partial i} := \{\mathbf{p}_A, \mathbf{p}_B, \mathbf{p}_C\} \quad \exists\; k \neq l \;\text{ with }\; |\mathbf{p}_k - \mathbf{p}_i|^2 < r^2 \leq |\mathbf{p}_l - \mathbf{p}_i|^2 \;,\; k,l \in \{A,B,C\} \qquad (3.52)$$

Then the number $\tilde{L}_r$ of faces $\mathbf{t}_{\partial i}$ approximates $\hat{L}_r(\mathbf{p}_i)$. Note that this estimation will not determine all faces intersecting the sphere like the gray triangle in Figure 3.12b. However these faces contribute only a very small amount of their area to $P_r(\mathbf{p}_i)$ and make no practical difference for an estimation using $\tilde{L}_r$ as the adjacent face shown in orange color, having one vertex inside the sphere will be counted anyway. Figure 3.12a shows the five cases of edge positions in respect to the sphere, to be considered for a precise computation of $\hat{P}_r$ and $\hat{L}_r$ for each edge $\mathbf{e}_{kl}$ of the triangles along $\mathscr{L}_r(\mathbf{p}_i)$.

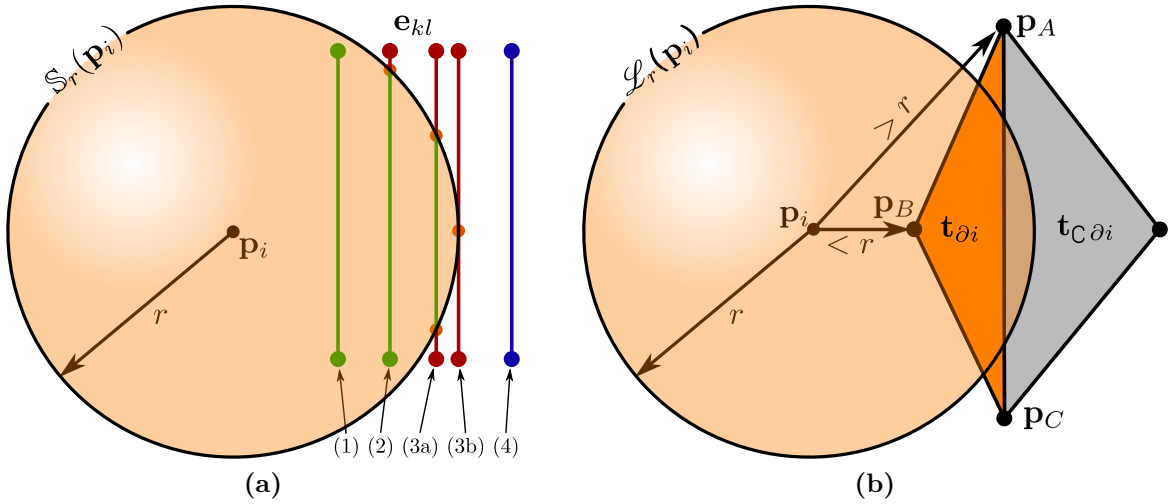


Figure 3.12: Intersection of the triangle's edges with a sphere: (a) The five cases of possible edge placements. (b) Case (3) showing the triangle $\mathbf{t}_{\complement\,\partial i}$ not taken into account by Equation 3.52 in gray, while the triangle $\mathbf{t}_{\partial i}$ in orange color is accounted for estimating $\hat{L}_r$.

3.3.2 Efficent and Precise Computation for Irregular Data

The principles for estimating the area based and line based integral invariants by counting from the previous section is extended to a precise computation in this section. Therefore the presented methods work for any given mesh with arbitrary triangles requiring almost no extra performance. This is achieved by determining the intersection of the triangles $\mathbf{t}_j$ with the sphere $B_r(\mathbf{p}_i)$ resulting in planar polygon with up to three circle segments as shown in Figure 3.13. These circle segments are also located in a triangle's plane and each segment shares two intersecting points with the polygon.

The most common cases of intersections between triangles and a sphere within acquired meshes are shown in Figure 3.13a. Figure 3.13b shows the case with the maximum number of six intersections per face. The case shown in Figure 3.13c can be neglected in case $r \gg |\mathbf{e}_{AB}|$. As the sphere is always centered at one of the $\mathcal{M}_2$ vertices $\mathbf{p}_i$ and the faces are in a n-ring neighborhood this case practically never occurs. Note that this case exists in theory and for highly degenerated meshes having a high curvature at $\mathbf{p}_i$ surrounded by faces of different size.

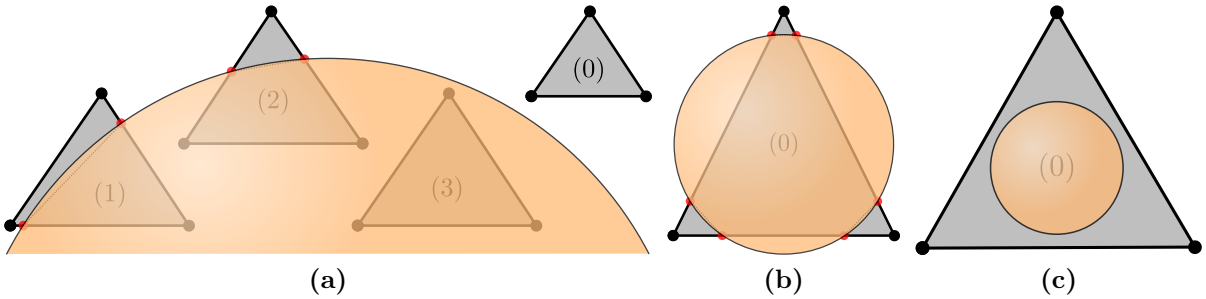


Figure 3.13: Sphere-triangle intersection: (a) Most commonly there are two points of intersection between the triangle's edges and the sphere, when 1 or 2 vertices of the triangle are inside the sphere. Triangles having 0 or 3 vertices inside the sphere are typically inside or outside the sphere. (b) Maximum of six possible points of intersection between the edges and the sphere. (c) Type of intersection only existing for extremely degenerated 3D-models.

Comparing r with $|\mathbf{p}_k - \mathbf{p}_i|$ in Equation 3.52 allows to determine if a triangle's vertex is inside or outside the ball. The triangles $\mathbf{t}_{\mathsf{C}\,\partial i}$ determined using this Equation fulfill a sufficient condition for a triangle's intersection with B. However the triangle $\mathbf{t}_{\mathsf{C}\,\partial i}$ in Figure 3.12b shows an intersection, where this condition is not fulfilled. Figure 3.13 shows further types of intersection, which have to be covered for a case-by-case analysis to determine a necessary condition. The case shown in Figure 3.13c can be neglected, because it only appears in extremely degenerated meshes. This leads to a less complex necessary condition using the points of intersection. Additionally the points of intersections have only to be computed unless all vertices are inside the ball.

These points of intersection $\mathbf{p}_{s_{1,2}}$ are found along the edges and are implicitly given by a triangle's definition. Each edge is defined by a pair of sorted position vectors $\mathbf{p}_{e_{1,2}}$ using the triangle's orientation. As an edge is a subset of an infinite line, the points of intersection between line and sphere are only of relevance, when they are within the

subset. The line can be described as

$$\mathbf{p}_s = \lambda\left(\mathbf{p}_{e_2} - \mathbf{p}_{e_1}\right) + \mathbf{p}_{e_1} = \lambda\mathbf{k} + \mathbf{d} \tag{3.53}$$

with

$$\lambda = \begin{cases} [0,1] & \mathbf{p}_s \text{ is a point along the edge} \\ 0 & \mathbf{p}_s = \mathbf{p}_{e_1} \\ 1 & \mathbf{p}_s = \mathbf{p}_{e_2} \\ \text{otherwise} & \end{cases} \tag{3.54}$$

This line is intersected with the sphere given by the position of its center $\mathbf{p}_i$ and the radius r described by:

$$\left(\mathbf{p}_s - \mathbf{p}_i\right)^2 = r^2 \tag{3.55}$$

Substituting the equation for the line (Eq. 3.53) in the equation of the sphere (Eq. 3.55) leads to

$$a\lambda^2 + b\lambda + c = 0 \tag{3.56}$$

with

$$a = |\mathbf{k}|^2 \tag{3.57}$$
$$b = 2 \cdot \langle \mathbf{k}, (\mathbf{d} - \mathbf{p}_i) \rangle \tag{3.58}$$
$$c = |\mathbf{p}_i|^2 + |\mathbf{d}|^2 - 2 \cdot \langle \mathbf{p}_i, \mathbf{d} \rangle - r^2 \tag{3.59}$$

Using the vectors' elements the Equations 3.57, 3.58 and 3.59 become:

$$a = (x_{e_2} - x_{e_1})^2 + (y_{e_2} - y_{e_1})^2 + (z_{e_2} - z_{e_1})^2 \tag{3.60}$$
$$b = 2\left((x_{e_2} - x_{e_1})(x_{e_1} - x_i) + (y_{e_2} - y_{e_1})(y_{e_1} - y_i) + (z_{e_2} - z_{e_1})(z_{e_1} - z_i)\right) \tag{3.61}$$
$$c = x_i^2 + y_i^2 + z_i^2 + x_{e_1}^2 + y_{e_1}^2 + z_{e_1}^2 - 2\left(x_i x_{e_1} + y_i y_{e_1} + z_i z_{e_1}\right) - r^2 \tag{3.62}$$

The solutions of the Quadratic equation with $\lambda_1 \leq \lambda_2$ are:

$$\lambda_1 = \frac{-b - \sqrt{\Delta}}{2a} \quad , \quad \lambda_2 = \frac{-b + \sqrt{\Delta}}{2a} \quad \text{with the discriminant } \Delta = b^2 - 4ac \tag{3.63}$$

As the discriminant is used to extract the root having two different signs, it is used to predetermine irrelevant and special cases:

$$\Delta = \begin{cases} < 0 & \nexists\, \mathbf{p}_{s_{1,2}} & \Rightarrow \text{ no intersection at all} \\ = 0 & \exists\, \mathbf{p}_{s_1} = \mathbf{p}_{s_2} & \Rightarrow \text{ the line is tangent} \\ > 0 & \exists\, \mathbf{p}_{s_1} \neq \mathbf{p}_{s_2} & \Rightarrow \text{ the line intersects the sphere} \end{cases} \tag{3.64}$$

In case $\lambda_{1,2} = [0,1]$ the sorted points of intersection $\mathbf{p}_{s_{1,2}}$ are computed using Equation 3.53 and returned by a function `IntersectLineSphere(`$\mathbf{p}_1$`,`$\mathbf{p}_2$`,`$\mathbf{p}_i$`,`r`)`. In case the edge is tangent this function must not return two identical points. Note that an alternative formulation of Equation 3.63 is required to achieve numerical stability as shown in [PTVF92].

The previous equations for lines and edges can be used as an alternative to Equation 3.52 by determining λ_C of the closest point along the line and its distance to sphere's center. This closest point can be computed using the inner product and substitution of Equation 3.53:

$$\langle \mathbf{p}_i - \mathbf{p}_{e_1} - \lambda_C \left(\mathbf{p}_{e_2} - \mathbf{p}_{e_1} \right), \left(\mathbf{p}_{e_2} - \mathbf{p}_{e_1} \right) \rangle = 0 \tag{3.65}$$

The solution using the points coordinates is:

$$\lambda_C = \frac{(x_i - x_{e_1})(x_{e_2} - x_{e_1}) + (y_i - y_{e_1})(y_{e_2} - y_{e_1}) + (z_i - z_{e_1})(z_{e_2} - z_{e_1})}{(x_{e_2} - x_{e_1})(x_{e_2} - x_{e_1}) + (y_{e_2} - y_{e_1})(y_{e_2} - y_{e_1}) + (z_{e_2} - z_{e_1})(z_{e_2} - z_{e_1})} \tag{3.66}$$

This alternative overcomes the drawback of using Equation 3.52 to pre-determine candidates for intersection, because it does account for the degenerated case shown in Figure 3.13c and for triangles like $\mathbf{t}_{\mathsf{C}\partial i}$ shown in Figure 3.12b.

The sorted points of intersections and the vertices of the triangle inside or along the sphere are used as ordered list $\mathcal{L}_s$ describing the orientated polygon. This polygon is flat and located along the plane $\boldsymbol{\tau}_j$ defined by the triangle $\mathbf{t}_j$ and its normalized normal $\hat{\mathbf{n}}_j$ in *Hesse normal form* (HNF):

$$\boldsymbol{\tau}_j = (\hat{n}_{j_x}, \hat{n}_{j_y}, \hat{n}_{j_z}, w_j) \quad \text{with} \quad w_j = \langle \mathbf{p}_k, \hat{\mathbf{n}}_j \rangle \quad \text{and} \quad \mathbf{p}_k \in \mathbf{t}_j \tag{3.67}$$

The intersection of $\boldsymbol{\tau}_j$ with $\mathbb{S}_r(\mathbf{p}_i)$ is a circle $\mathbb{S}^1_{r_s}(\mathbf{p}_\tau)$. As r_s is typically smaller than r it is called small circle from now on. In case r_s becomes r it is also referred to as great circle within literature. Important examples for great circles e.g. from geography, are the Equator and the Prime meridian together with the 180th meridian, which defines the International Date Line. However, great circles are special cases of small circles. Therefore using the formulae for small circles covers all cases of intersection. The center $\mathbf{p}_\tau$ of the small circle is the orthogonal project of $\mathbf{p}_i$ having the distance

$$d_j = \langle \mathbf{p}_i, \hat{\mathbf{n}}_j \rangle - w_j = \hat{n}_{j_x} x_i + \hat{n}_{j_y} y_i + \hat{n}_{j_z} z_i - w_j \ , \tag{3.68}$$

which allows to compute

$$r_s = \sqrt{r^2 - d_j^2} \tag{3.69}$$

and

$$\mathbf{p}_\tau = \mathbf{p}_i + d_j \cdot \hat{\mathbf{n}}_j \ . \tag{3.70}$$

Note that $r = d_j$ means a tangent plane and $r > d_j$ means no intersection at all. The small circle $\mathbb{S}^1_{r_s}(\mathbf{p}_\tau)$ and a pair points of intersection $(\mathbf{p}_{s_1}, \mathbf{p}_{s_2})$ of an intersecting edge define a circle segment. This circle segment has an area

$$A_C(r_s, \alpha) = \frac{r_s^2}{2} \cdot (\alpha - \sin\alpha) \tag{3.71}$$

where the angle α is

$$\alpha = \measuredangle(\mathbf{p}_{s_2}, \mathbf{p}_\tau, \mathbf{p}_{s_1}) = \arccos\left(\frac{\langle \mathbf{u}, \mathbf{v} \rangle}{|\mathbf{u}||\mathbf{v}|} \right) \quad \text{with} \quad \mathbf{u} = \mathbf{p}_{s_1} - \mathbf{p}_\tau \quad \text{and} \quad \mathbf{v} = \mathbf{p}_{s_2} - \mathbf{p}_\tau \tag{3.72}$$

Finally the area of the polygon itself has to be computed using the points stored in $\mathcal{L}_s$. As these points are located in the plane $\boldsymbol{\tau}_j$ an approach is to project the points in $\mathbb{R}^3$ into the two-dimensional coordinate system of the plane. The result is a polygon $\mathcal{L}_s^2$ embedded in $\mathbb{R}^2$. Than the polygon can be decomposed into (two-dimensional) triangles $\mathbf{t}_k^2$. Therefore each triangle consists of an arbitrary point $\mathbf{p}_a^2 = (x_a, y_a)^T$ and pairs of points $\mathbf{p}_k^2$ and $\mathbf{p}_{k+1}^2$ of the edges of the polygon:

$$\mathbf{t}_k^2 := \{\mathbf{p}_a^2, \mathbf{p}_k^2, \mathbf{p}_{k+1}^2\} \tag{3.73}$$

Note that $k+1$ has to be modulo $|\mathcal{L}_s^2|$, which means $k+1 \equiv 1$ for $k = |\mathcal{L}_s|$. This ensures, that the edge connecting the last point of the polygon and its first point, is properly taken into account. As the polygon is simple i.e. has no self intersections, the sum of the signed areas of the triangles is the area of the polygon. Choosing $\mathbf{p}_a^2 = (0,0)^T$ the area of the polygon is computed by:

$$A(\mathcal{L}_s^2) = \sum_{k=1}^{n} A(\mathbf{t}_k^2) = \frac{1}{2}\sum_{k=1}^{n} (x_k y_{k+1} - x_{k+1} y_k) \quad \text{and} \quad n = |\mathcal{L}_s^2| \;\;,\;\; n+1 \equiv 1 \tag{3.74}$$

Figure 3.14 shows this principle, which works for convex and concave polygons.

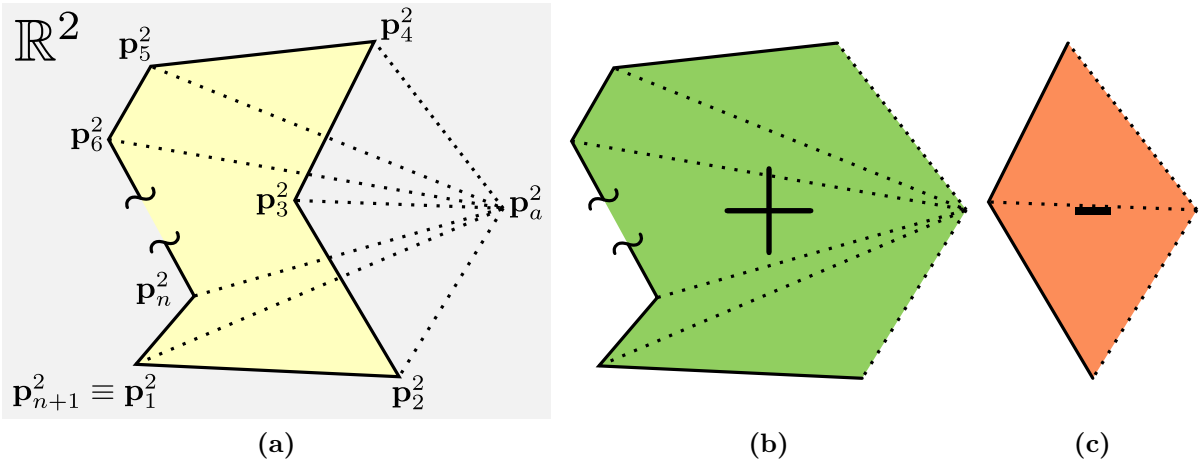


Figure 3.14: Area of a simple polygon in $\mathbb{R}^2$ decomposed into triangles. (a) Ordered points $\mathbf{p}_k^2$ describing the polygon $\mathcal{L}_s^2$. The arbitrary point $\mathbf{p}_a^2$ decomposes the polygon into triangles (b) with positive area and (c) with negative area.

As projecting the points $\mathbf{p}_k \rightarrow \mathbf{p}_k^2$ costs computing time, the concept of decomposing the polygon in $\mathbb{R}^2$ can be extended to $\mathbb{R}^3$. Therefore the points of $\mathcal{L}_s$ stay untouched. Than the arbitrary point $\mathbf{p}_a$ has to be in $\mathbb{R}^3$, but it does not have to be within the plane $\boldsymbol{\tau}_j$. Figure 3.15a shows $\mathcal{L}_s \in \boldsymbol{\tau}_j$ and $\mathbf{p}_a \in \mathbb{R}^3$. This figure also shows the (three-dimensional) triangles

$$\mathbf{t}_k := \{\mathbf{p}_a, \mathbf{p}_k, \mathbf{p}_{k+1}\} \tag{3.75}$$

as dotted lines, which have an area of:

$$A(\mathbf{t}_k) = \frac{1}{2}\,|(\mathbf{p}_k - \mathbf{p}_a) \times (\mathbf{p}_{k+1} - \mathbf{p}_a)| \tag{3.76}$$

The sum of the areas of Equation 3.76 does not equal the area of the polygon, but the signed sum of triangles using the closest point of $\boldsymbol{\tau}_j$ to $\mathbf{p}_a$ is. These triangles are shown in Figure 3.15b. One solution would be to compute and use this closest point, but instead it is more efficient to use the fact that the inner product of normalized vectors is the *cosinus* of the angle α between them. This leads to the computing the areas of the triangles within the plane using the normals of the triangles and the normal of the plane:

$$A(\mathbf{t}_k) = \frac{1}{2}\langle \hat{\mathbf{n}}_j, (\mathbf{p}_k - \mathbf{p}_a) \times (\mathbf{p}_{k+1} - \mathbf{p}_a) \rangle \tag{3.77}$$

When $\mathbf{p}_a$ is chosen as $(0,0,0)^T$, Equation 3.77 can be reduced to:

$$A(\mathbf{t}_k) = \frac{1}{2}\langle \hat{\mathbf{n}}_j, \mathbf{p}_k \times \mathbf{p}_{k+1} \rangle \tag{3.78}$$

Then the polygon area can be computed as sum of triangle normals of $2A(\mathbf{t}_k)$ length, having an area of $A(\mathbf{t}_k)\cos\alpha_k$ by:

$$A(\mathcal{L}_s) = \sum_{k=1}^{n} A(\mathbf{t}_k) = \frac{1}{2}\langle \hat{\mathbf{n}}_j, \sum_{k=1}^{n} (\mathbf{p}_k \times \mathbf{p}_{k+1}) \rangle \quad \text{and} \quad n = |\mathcal{L}_s| \;\; , \;\; n+1 \equiv 1 \tag{3.79}$$

Note that this equation does not require any arbitrary point nor a computational expensive projection of the points of $\mathcal{L}_s$ to $\mathbb{R}^2$ using a transformation matrix. This equation is valid convex and concave polygons as long as they are planar and simple – similar to Equation 3.74.

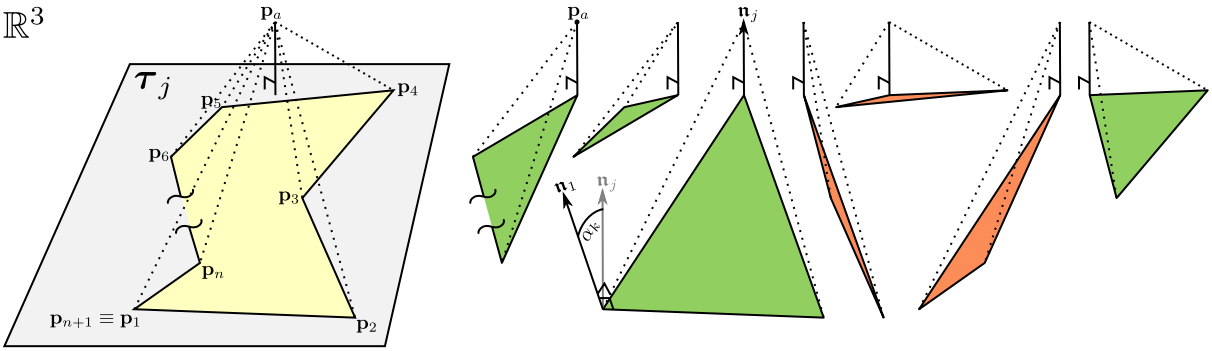


Figure 3.15: A simple polygon in $\mathbb{R}^3$ (a) decomposed into arbitrary triangles using an arbitrary point $\mathbf{p}_a$ and (b) their signed areas $A(\mathbf{t}_k)\cos\alpha_k$, where α_k is the angle between the normal $\hat{\mathbf{n}}_j$ of the plane $\boldsymbol{\tau}_j$ and the triangles' normals.

The Algorithm 5 to 7 show the computation of the area $A_B(\mathbf{t}_j)$ for triangles intersected by a sphere B_r as sum of $A(\mathcal{L}_s)$ and the areas of the circle segments. This Algorithm can be used on all triangles, but for performance reasons a priori knowledge should be incooperated to avoid unnecessary calls of `IntersectLineSphere(`$\mathbf{p}_m$`,`$\mathbf{p}_l$`,`$\mathbf{p}_i$`,`r`)`. Such knowledge can be derived from neighborhood relations as shown later in Section 4.1.

Within the Algorithm a list $\mathcal{L}_s$ of pairs is used to describe the polygon resulting from the intersection. The first value of the pairs contains points of intersections as well as points of the triangle within the sphere and maintains the order given by the edges, which are directional. The second entry of the pair is a reference to the edge a point belongs to. These pairs of points and edge references are stored at the end of the list using `append`. The pairs can be accessed using similar to arrays e.g. $\mathcal{L}_s[k]$ will return the k-th pair. Accessing the k-th point is achieved by $\mathcal{L}_s[k].\mathbf{p}$ and the edge reference can be accessed using $\mathcal{L}_s[k].e$.

The length $|\mathcal{L}_s|$ of the list can not exceed six entries as shown in Figure 3.13b. This is true even there are four possible calls of `append` for each of the three edges, because they are mutually exclusive due to comparison with $<$ instead of $\leq$ before adding $\mathbf{p}_m$ and $\mathbf{p}_l$. A length of 0 means that the triangle is outside the sphere as there are no points inside the sphere nor any intersection. In this case there is no area and the Algorithm returns 0. If there is only 1 point in the list means that a numeric error occurred, but it is safe to return 0 as there will be no relevant area. Generally the list $\mathcal{L}_s$ has to contain an even number of pairs, when computed correctly.

Having $|\mathcal{L}_s| = 2$ occurs in cases as shown in Figure 3.12b as triangle $\mathbf{t}_{\mathbb{C}\,\partial i}$, where all vertices are outside the sphere, but the triangle is partially inside. In case r_s is smaller than the radius of the triangle's incircle, it is possible that the larger segment of the circle is enclosed having the complementary area $r_s^2(\pi - (\alpha - \sin\alpha)/2)$. However this case is practically irrelevant. The overall algorithm to compute the surface area $A_B(\mathbf{t}_j)$ used to compute the integral invariant $\hat{P}_r$ consists of three parts:

Algorithm 5 Part 1: Prepare polygon to compute $A_B(\mathbf{t}_j)$

```
1:  function A_B( t, p_i, r )
2:      L_s ← ∅                          ▷ empty list for six pairs of points and edge references
3:      for (m, l) ← {(A,B),(B,C),(C,A)} do                                  ▷ all edges
4:          if |p_m − p_i| < r then          ▷ the first vertex of the edge is inside the sphere
5:              L_s.append( p_m, m )                 ▷ insert point and edge reference into list
6:          end if
7:          (p_s1, p_s2) ← IntersectLineSphere(p_m, p_l, p_i, r)
8:          if ∃p_s1 then
9:              L_s.append( p_s1, m )                ▷ insert point and edge reference into list
10:         end if
11:         if ∃p_s2 then
12:             L_s.append( p_s2, m )                ▷ insert point and edge reference into list
13:         end if
14:         if |p_l − p_i| < r then          ▷ the second vertex of the edge is inside the sphere
15:             L_s.append( p_l, m )                 ▷ insert point and edge reference into list
16:         end if
17:     end for
```

Algorithm 6 Part 2: Special case of a circle segment $A_B \leftarrow A_C$

18: **if** $|\mathcal{L}_s| < 2$ **then** ▷ no intersection and no point outside ⇒ $\mathbf{t}_j$ outside B_r
19: **return** 0 ▷ . . . means zero area
20: **end if**
21: $d_j \leftarrow \langle \mathbf{p}_i, \hat{\mathbf{n}}_j \rangle - w_j$ ▷ distance of $\mathbf{p}_i$ to the triangle's plane $\boldsymbol{\tau}_j = (\hat{\mathbf{n}}_j, w_j)$
22: $r_s \leftarrow \sqrt{r_2 - d_j^2}$ ▷ radius of the small circle of the sphere
23: $\mathbf{p}_\tau \leftarrow \mathbf{p}_i + d_j \cdot \hat{\mathbf{n}}_j$ ▷ center of the small circle
24: **if** $|\mathcal{L}_s| = 2$ **then** ▷ two points of intersection and no point of the triangle inside
25: $\alpha \leftarrow \measuredangle(\mathcal{L}_s[1].\mathbf{p}, \mathbf{p}_\tau, \mathcal{L}_s[2].\mathbf{p})$
26: **return** $A_C(r_s, \alpha)$ ▷ return the area of a circle segment
27: **end if**

Algorithm 7 Part 3: Areas of sub-triangles and circle segments for $A_B(\mathbf{t}_j)$

28: $A_S \leftarrow 0$ ▷ variable accumulating the circle segments' areas
29: $\mathbf{A}_P \leftarrow (0,0,0)^T$ ▷ vector used to compute the polygon area
30: **for** $k \leftarrow [2, |\mathcal{L}_s|]$ **do** ▷ all points of intersection
31: $\mathbf{p}_1 \leftarrow \mathcal{L}_s[k-1].\mathbf{p}$ ▷ first point of a polygon's edge
32: $\mathbf{p}_2 \leftarrow \mathcal{L}_s[k].\mathbf{p}$ ▷ second point of a polygon's edge
33: **if** $\mathbf{p}_1 \neq \mathbf{p}_2$ **then** ▷ there is an area to compute
34: $\mathbf{A}_P \leftarrow \mathbf{A}_P + (\mathbf{p}_1 \times \mathbf{p}_2)$
35: **if** $\mathcal{L}_s[k-1].e \neq \mathcal{L}_s[k].e$ **then** ▷ change of edges ⇒
36: $\alpha \leftarrow \measuredangle(\mathbf{p}_1, \mathbf{p}_\tau, \mathbf{p}_2)$ ▷ compute angle describing the circle segement
37: $A_S \leftarrow A_S + r_s^2 \cdot (\alpha - \sin\alpha)$ ▷ add circle segement's area
38: **end if**
39: **end if**
40: **end for**
41: $\mathbf{A}_P \leftarrow \mathbf{A}_P + (\mathbf{p}_2 \times \mathcal{L}_s[1].\mathbf{p})$ ▷ closing edge of the polygon – $\mathbf{p}_2 \equiv \mathcal{L}_s[|\mathcal{L}_s|].\mathbf{p}$
42: $A_B \leftarrow \frac{1}{2}(A_S + \langle \hat{\mathbf{n}}_j, \mathbf{A}_P \rangle)$ ▷ area of the circle segments plus the area of the polygon
43: **return** A_B ▷ return the area $P_r = |\mathcal{P}_r|$
44: **end function**

The Algorithm 7 (part 3) additionally provides the angle α, which can be used to compute the arc length of the circle segment. The sum of the arc lengths of the intersecting triangles $\mathbf{t}_j \rightsquigarrow \mathscr{L}_r$ and the integral invariant $\hat{L}_r$ is computed by

$$\hat{L}_r = \frac{L_r}{|\mathbb{S}_r^1|} = \sum_j \frac{L_B(\mathbf{t}_j)}{|\mathbb{S}_r^1|} = \sum_j \frac{\pi r_{s_j} \alpha_j}{\pi r} = \frac{1}{r} \sum_j \alpha_j r_{s_j} \quad , \quad \alpha \text{ in radiant} \tag{3.80}$$

Therefore Algorithm 8 is an extended version of part 3 of the overall algorithm, which additionally computes the run length $L_r = |\mathscr{L}_r|$ of the line of intersection required for the integral invariant $\hat{L}_r$ by computing $L_B(\mathbf{t}_j) = |\mathbf{t}_j \cap \mathbb{S}_r(\mathbf{p}_i)|$. This is achieved with a minimum of additional computing operations. Part 1 and part 2 require no changes.

Algorithm 8 Part 3: including the run length L_B of circle segments for L_r

$L_B \leftarrow 0$ ▷ **variable accumulating the circle segments' run length**
$A_S \leftarrow 0$
$\mathbf{A}_P \leftarrow (0,0,0)^T$
for $k \leftarrow [2, |\mathcal{L}_s|]$ **do**
    $\mathbf{p}_1 \leftarrow \mathcal{L}_s[k-1].\mathbf{p}$
    $\mathbf{p}_2 \leftarrow \mathcal{L}_s[k].\mathbf{p}$
    if $\mathbf{p}_1 \neq \mathbf{p}_2$ **then**
        $\mathbf{A}_P \leftarrow \mathbf{A}_P + (\mathbf{p}_1 \times \mathbf{p}_2)$
        if $\mathcal{L}_s[k-1].e \neq \mathcal{L}_s[k].e$ **then** ▷ change of edges ⇒
            $\alpha \leftarrow \measuredangle(\mathbf{p}_1, \mathbf{p}_\tau, \mathbf{p}_2)$
            $A_S \leftarrow A_S + r_s^2 \cdot (\alpha - \sin\alpha)$
            $L_B \leftarrow L_B + \alpha r_s$ ▷ **add circle segement's run length**
        end if
    end if
end for
$\mathbf{A}_P \leftarrow \mathbf{A}_P + (\mathbf{p}_2 \times \mathcal{L}_s[1].\mathbf{p})$
$A_B \leftarrow \frac{1}{2}(A_S + \langle \hat{\mathbf{n}}_j, \mathbf{A}_P \rangle)$
return A_B, L_B ▷ return the area $P_r = |\mathcal{P}_r|$ **and the length** $L_r = |\mathscr{L}_r|$

3.4 Volume Integral Invariant Computation using the Divergence Theorem

The previous sections show three different integral invariants, computed using two different approaches. The first approach uses an image processing inspired method to compute the volume integral invariant using a numeric solution with a regular grid for the indicator functions of the volume. In contrast the second approach for the surface and line based integral invariants relates to vector algebra, which requires less computing operation providing results less prone to numeric errors. Therefore this section shows the precise computation of integral invariants and an algorithm computing all integral invariants at once.

The extension from Algorithm 7 to Algorithm 8 in Section 3.3.2 already introduced the concept of computing the surface integral invariant $\hat{P}_r$ and the integral invariant of the intersecting line $\hat{L}_r$ at the same time by adding just one additional calculation. Therefore a combined algorithm for all three integral invariants requires either to compute $\hat{P}_r$ and $\hat{L}_r$ using a numeric solution like previously shown for $\hat{V}_r$ or – vice versa – a solution for $\hat{V}_r$ using vector algebra. The latter is the better choice, because converting the unstructured mesh to a regular grid requires the additional parameter $n_{\boxplus}$ for rasterization. Furthermore this parameter introduces quantification errors, when it is chosen to small or it costs unnecessary computing time, when it is chosen to large.

The presented approach using numeric integration of $\hat{V}_r$ is essentially adding the volume of cuboids, which are computed from the mesh's triangles $\mathbf{t}_j$ as shown in Figure 3.7 on page 53. These are right rectangular prisms having squared and equal sized bases aligned along the xy-plane as shown in Figure 3.8. On the other hand side there is Equation 2.9 on page 27 for computing the volume of a mesh, which was derived using divergence theorem. The resulting Equation 2.10 is a sum of prisms, where the non-zero coordinate of the balance points is their height and area is half of the normal vectors length.

As the Algorithm 5, 6 and 8 already provides the area of a triangle or its intersection, the equation for a mesh volume can be adapted to compute the volume $|\mathbb{V}_r(\mathbf{p}_i)| = |B_r(\mathbf{p}_i) \cap \mathbf{1}_{D_\mathcal{M}}|$ leading to $V_r(\mathbf{p}_i)$. Lets begin with the volume (of the mesh) using the xy-plane:

$$V_\mathcal{M} = \sum_j A(\mathbf{t}_j) \begin{pmatrix} 0 \\ 0 \\ s_{z_j} \end{pmatrix} \hat{\mathbf{n}}_j = \frac{1}{2} \sum_j \begin{pmatrix} 0 \\ 0 \\ s_{z_j} \end{pmatrix} \mathbf{n}_j \text{ with } \mathbf{s}_j = \begin{pmatrix} s_{x_j} \\ s_{y_j} \\ s_{z_j} \end{pmatrix} = \frac{\mathbf{p}_{A_j} + \mathbf{p}_{B_j} + \mathbf{p}_{C_j}}{3} \tag{3.81}$$

The z-coordinate s_{z_j} of the balance point $\mathbf{s}_j$ is the distance to the xy-plane $\boldsymbol{\tau}_{xy} = (0, 0, 1, 0) = (\boldsymbol{\mathfrak{z}}, 0)$. Because this distance is the height h_j of a 3-sided prism with $\mathbf{t}_j$ as base, Equation 3.81 becomes:

$$V_\mathcal{M} = \sum_j A(\mathbf{t}_j)\, h_j\, \langle \hat{\mathbf{n}}_j, \boldsymbol{\mathfrak{z}} \rangle = \frac{1}{2} \sum_j \langle \mathbf{n}_j, h_j\, \boldsymbol{\mathfrak{z}} \rangle \text{ with } \boldsymbol{\mathfrak{z}} = (0, 0, 1)^T \tag{3.82}$$

Then the parts of this equation can be exchanged as follows using the TNB frame as reference:

- The list of faces $\mathcal{L}_f$ is reduced to those of $\mathcal{P}_i$ – reducing computational complexity.
- $A(\mathbf{t}_j)$ becomes the area $A_B(\mathbf{t}_j)$ of triangles completely and partially inside the ball.
- The reference plane $\boldsymbol{\tau}_{xy}$ is exchanged by $\boldsymbol{\tau}_i = (\hat{\mathbf{n}}_i, d_i)$ by replacing $\boldsymbol{\mathfrak{z}}$ with $\hat{\mathbf{n}}_i$ and using the patch's central vertex $\mathbf{p}_i$ to determine d_i.
- The height h_j becomes the distance of each triangle's center of gravity $\mathbf{s}_j$ to $\boldsymbol{\tau}_i$. This includes the circle segments introduced by the triangles $\mathbf{t}_{\mathsf{C}\,\partial i}$ along $\mathbb{S}_r(\mathbf{p}_i)$.

These changes lead to a signed volume

$$V_{\mathcal{P}} = |\mathbb{V}_{\mathcal{P}}| = \sum_j A_B(\mathbf{t}_j)\; h_j\; \langle \hat{\mathbf{n}}_j, \hat{\mathbf{n}}_i \rangle \quad \text{with} \quad h_j = \langle \mathbf{s}_j, \hat{\mathbf{n}}_i \rangle - d_i \quad \text{and} \quad \mathbf{t}_j \in \mathcal{P}_i \tag{3.83}$$

This volume accounts for the volume between $\mathcal{P}_i$ and $\boldsymbol{\tau}_i$, which is enclosed by a sphere as shown in the schematic in Figure 3.16a using a two-dimensional slice of the sphere and the patch. Therefore $\mathbb{V}_{\mathcal{P}}$ is a subset of $\mathbb{V}_r$ described by a set of prisms denoted as $\mathscr{L} \times \mathbf{n}_i$, because the prisms' bases are enclosed by $\mathscr{L} = \partial\mathcal{P}$. To compute the integral invariant $V_r = |\mathbb{V}_r|$ two more volumina enclosed by the sphere ∂B are required. First the signed volume outside the set of prisms has to be computed:

$$V_{\mathscr{L}} = |\mathbb{V}_{\mathscr{L}}| = \int_{B_r(\mathbf{p}_i)} \mathbf{1}_{\mathscr{L}} \quad \text{and} \quad \mathbf{1}_{\mathscr{L}} = \begin{cases} 1 & \text{outside the prism set } \mathscr{L} \times \mathbf{n}_i \\ 0 & \text{otherwise} \end{cases} \tag{3.84}$$

The scheme of $V_{\mathscr{L}}$ is shown in Figure 3.16b. Secondly the half of the ball's volume – shown in Figure 3.16c – has to be added to compensate for the signed volumes. The union $\mathbb{V}_r$ of the volumina is shown in Figure 3.16d and the sum of these volumina are the volume integral invariant:

$$V_r = V_{\mathcal{P}} + V_{\mathscr{L}} + \frac{|B_r|}{2} \tag{3.85}$$

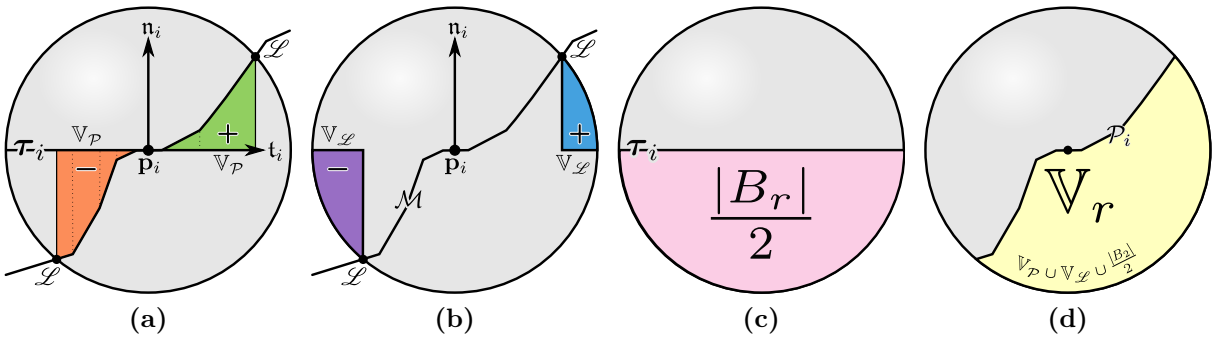


Figure 3.16: Volumes computed using (a) the divergence theorem, (b) outside the prism set $\mathscr{L} \times \mathbf{n}_i$ within the ball and (c) half of the sphere, which add up to (d) the volume $\mathbb{V}_r \rightsquigarrow V_r$.

Having Equation 3.85 leads to two extensions of the overall algorithm from the previous section. The first extension regards to Equation 3.83 and requires the implementation of the center of gravity, which is the sum of all of the centers of gravities of the circle segments and the sub-triangles weighted by the corresponding areas. The formulae is deducted in the same manner as Equation 3.79 on page 68 and it is solution very similar, which means that intermediate results of the overall algorithm can be reused an the number of extra computation can be kept to a minimum.

The second and major extension concerns the indicator function shown in Equation 3.84, which consists of sectors of prisms having different ellipsis as base. These prisms can be approximately by prisms having a circular base, because this leads to a sum of sectors of napkin rings, which are spheres with a cylindric hole as shown in Figure 3.17.

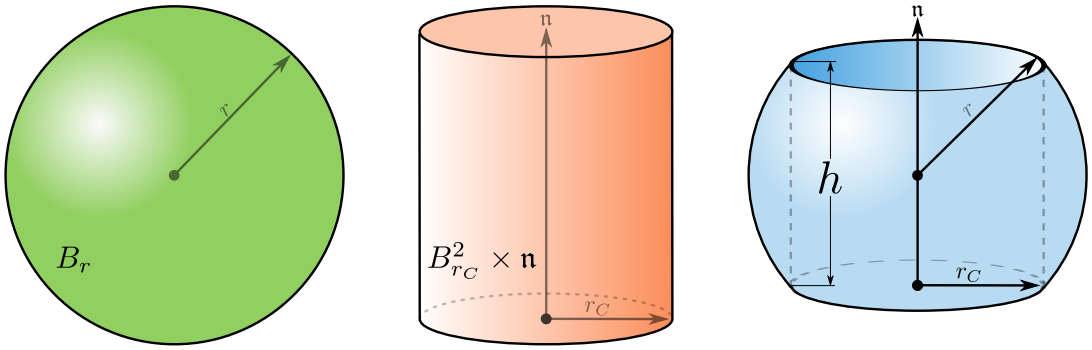


Figure 3.17: Napkin ring of height h: Subtraction of cylinder $B^2_{r_C} \times \mathfrak{n}$ from a sphere B_r.

The volume V_{Napkin} of a napkin ring depends only on its height:

$$V_{\text{Napkin}} = \left| B_r \setminus \left(B^2_{r_C} \times \mathfrak{n} \right) \right| = \frac{\pi h^3}{6} \tag{3.86}$$

and therefore the volume of a napkin ring's sector having an angle α is:

$$V_{\text{Napkin}}(h, \alpha) = \frac{\pi h^3}{6} \cdot \frac{\alpha}{2\pi} = \frac{h^3 \alpha}{12} \tag{3.87}$$

The orthogonal projections $\mathbf{p}^{\tau}_{s_{1,2}}$ of the points of intersection $\mathbf{p}_{s_{1,2}}$ onto the plane $\boldsymbol{\tau}_i := (\mathbf{p}_i, \mathfrak{n}_i)$ allow to compute the angle α:

$$\alpha = |\alpha_1 - \alpha_2| = \arccos \frac{\langle \mathbf{p}^{\tau}_{s_1} - \mathbf{p}_i, \mathbf{p}^{\tau}_{s_2} - \mathbf{p}_i \rangle}{|\mathbf{p}^{\tau}_{s_1} - \mathbf{p}_i| |\mathbf{p}^{\tau}_{s_2} - \mathbf{p}_i|} \tag{3.88}$$

As the distance between a point of intersection $\mathbf{p}_s$ and its projection $\mathbf{p}^{\tau}_s$ is half of the height $h(\mathbf{p}_s) = 2 \cdot |\mathbf{p}^{\tau}_s - \mathbf{p}_s|$ of a napkin ring, the two heights of the two $\mathbf{p}_s$ can be used as a simple estimator:

$$V_{\mathscr{L}}(\mathbf{p}_{s_1}, \mathbf{p}_{s_2}) \approx \frac{h^3(\mathbf{p}_{s_1}) + h^3(\mathbf{p}_{s_2})}{24} \alpha \tag{3.89}$$

Because $h(\mathbf{p}_s)$ depends on the distance $r_p = |\mathbf{p}_s^\tau - \mathbf{p}_i|$, $h(\mathbf{p}_s)$ becomes $h(r_p)$ using the parameter form of Equation 3.89:

$$V_{\mathscr{L}}(r_p) = \frac{1}{12} \int_{\alpha_1}^{\alpha_2} h^3(r_p)\, d\alpha \tag{3.90}$$

The next step is to determine the function to compute the height $h(r_p)$. Therefore Figure 3.18 shows the schematic for an intersection of an arbitrary plane, defined by an intersecting triangle. As mentioned before, the intersecting line is a small circle of of radius $r_s \leq r$. A subset of this circle is the intersection of the triangle, which defines the interval $[\alpha_1, \alpha_2]$ as shown in Figure 3.18b.

The projection of the small circle $\mathbb{S}^1_{r_s}$ is an ellipse, which has a semi-major axis of r_s and a semi-minor axis of $r_t = r_s \cdot \langle \hat{\mathbf{n}}_i, \hat{\mathbf{n}}_j \rangle = r_s \cos\beta$. Equation 3.68 on page 66 provides the distance d_j of the small circle's center to $\mathbf{p}_i$. Then the distance between the ellipse's center and $\mathbf{p}_i$ is $r_m = d_j \cdot \sin\beta$. For further computations only r_m is of relevance, while the rotation about $\mathbf{p}_i$ can be neglected. Therefore the points of the ellipse in the 2D-projection are computed using the angle θ as parameter:

$$\mathbf{p}_s(\theta) = \begin{pmatrix} & & r_s \cos\theta \\ r_m & + & r_t \sin\theta \end{pmatrix} \tag{3.91}$$

The distance of these points to the center of the ellipse is:

$$r_e(\theta) = \sqrt{(r_s \cos\theta)^2 + (r_t \sin\theta)^2} \tag{3.92}$$

Because the ellipse center has an offset of $(0, r_m)^T$ from $\mathbf{p}_i$, which is the origin of the plane's coordinate system, the distance of $\mathbf{p}_s$ to $\mathbf{p}_i$ is:

$$\begin{aligned} r_p(\theta) &= \sqrt{(r_s \cos\theta)^2 + (r_m + r_t \sin\theta)^2} \\ &= \sqrt{(r_s \cos\theta)^2 + (d_j \sin\beta + r_s \langle \hat{\mathbf{n}}_i, \hat{\mathbf{n}}_j \rangle \sin\theta)^2} \end{aligned} \tag{3.93}$$

Then the distance between a point $\mathbf{p}_s$ along $\mathbb{S}^1_{r_s}$ and its projection $\mathbf{p}_s^\tau$ to $\boldsymbol{\tau}_i$ is the first cathetus length of a right triangle with a hypotenuse of length r and the second cathetus having a length of $r_p(\theta)$:

$$h(r_p) = h(r_p(\theta)) := \sqrt{r^2 - r_p^2(\theta)} = h(\theta) \tag{3.94}$$

This leads to the height of the napkin ring:

$$h(\theta) = 2\sqrt{r^2 - (r_s \cos\theta)^2 - (d_j \sin\beta + r_s \langle \hat{\mathbf{n}}_i, \hat{\mathbf{n}}_j \rangle \sin\theta)^2} \tag{3.95}$$

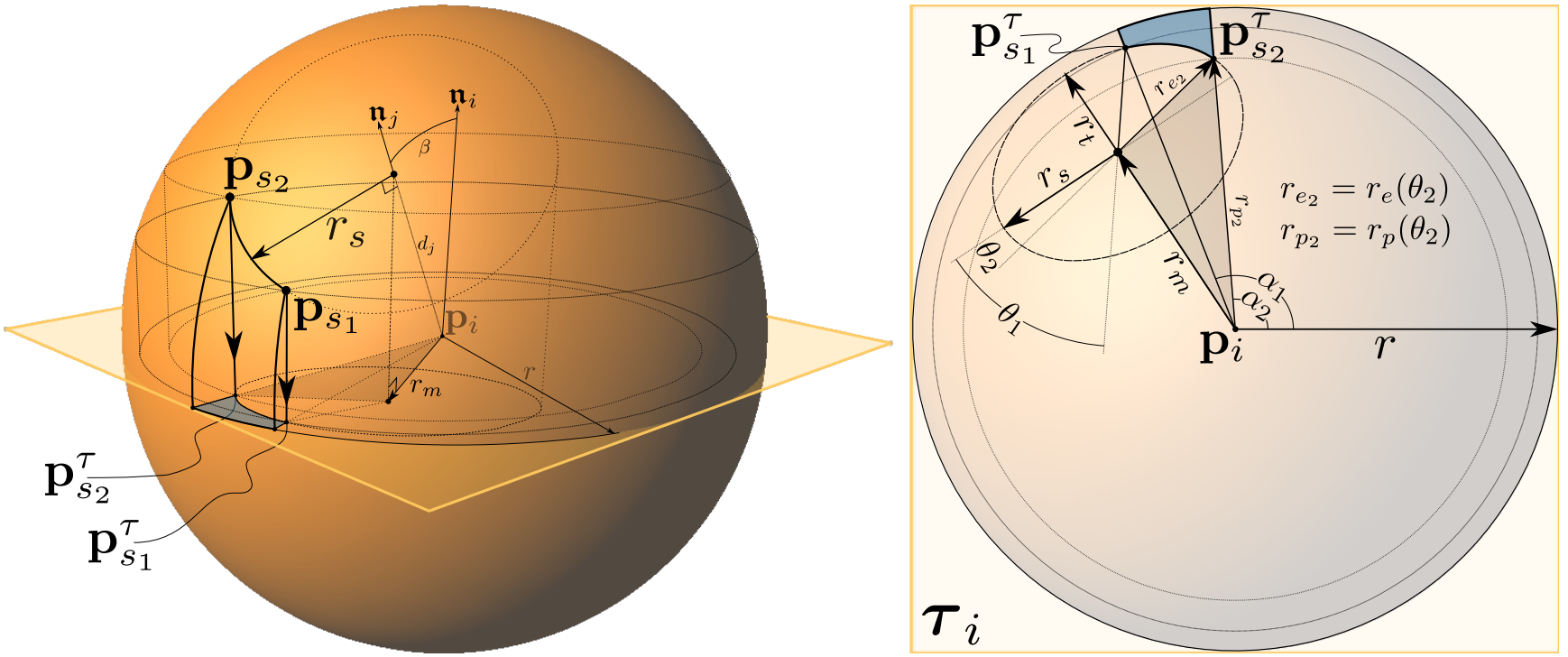


Figure 3.18: Schematic for computing $V_{\mathscr{L}}$: (a) Perspective view and (b) top view on the plane $\boldsymbol{\tau}_i$ and the projection of the small circle $\mathbb{S}^1_{r_s}$, which becomes an ellipse.

Because of the triangle defined by r_m, $r_e(\theta)$ and $r_p(\theta)$, the interval $[\alpha_1, \alpha_2]$ can be substituted by $[\theta_1, \theta_2]$ using the law of sines:

$$\frac{r_p(\theta)}{\cos\theta} = \frac{r_e(\theta)}{\sin\alpha} \tag{3.96}$$

Note that sine becomes the cosine, because the angle between r_m and $r_e(\theta)$ actually is $\theta + \pi/2$ as shown in Figure 3.18b. In similar manner α can be substituted in Equation 3.90 by θ:

$$V_{\mathscr{L}}(\theta_1, \theta_2) = \frac{1}{12} \int_{\theta_1}^{\theta_2} \left(2\sqrt{r^2 - r_p^2(\theta)}\right)^3 \, d\theta \tag{3.97}$$

With

$$c_\beta = \sin\left(\arccos\langle \hat{\mathbf{n}}_i, \hat{\mathbf{n}}_j \rangle\right) \tag{3.98}$$

and

$$e_\beta = \langle \hat{\mathbf{n}}_i, \hat{\mathbf{n}}_j \rangle \tag{3.99}$$

the volume can be computed by:

$$V_{\mathscr{L}}(\theta_1, \theta_2) = \frac{2}{3} \int_{\theta_1}^{\theta_2} \left(r^2 - (r_s \cos\theta)^2 - (c_\beta d_j + e_\beta r_s \sin\theta)^2\right)^{\frac{3}{2}} \, d\theta \tag{3.100}$$

Note that the interval $[\theta_1, \theta_2]$ can actually consist of up to three intervals, depending on the number of times the triangle intersects the sphere as shown in Figure 3.13 on page 64.

Normalizing the small circle's radius shown in Equation 3.69 on page 66 is achieved with:

$$\hat{d}_j = \frac{d_j}{r} \quad \Rightarrow \quad \hat{r}_s = \frac{r_s}{r} = \sqrt{1 - \hat{d}_j^2} \quad \text{and} \quad \hat{d}_j, \hat{r}_s \in [0, 1] \tag{3.101}$$

This allows to pre-compute terms used to compute the normalized volume

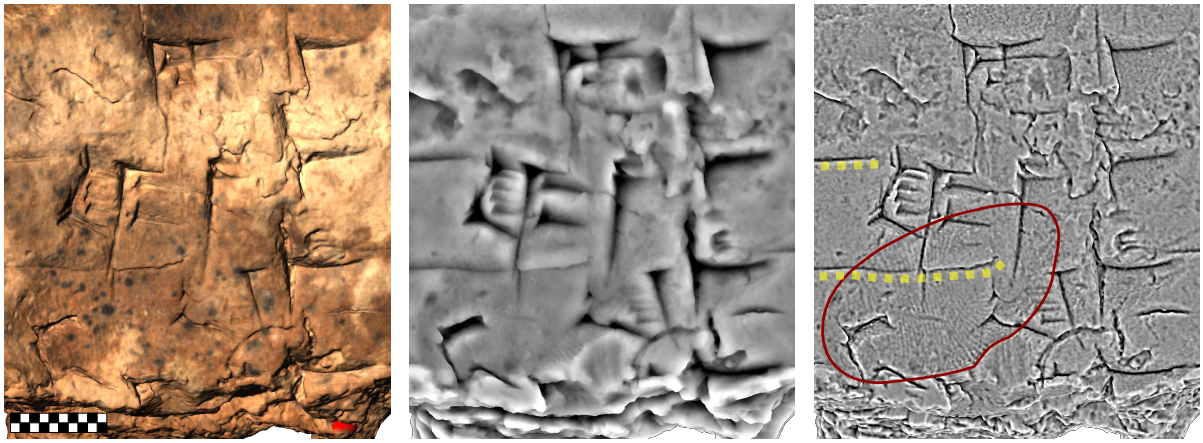
$$\hat{V}_{\mathscr{L}}(\theta_1, \theta_2) = \frac{2}{3} \int_{\theta_1}^{\theta_2} \left(1 - (\hat{r}_s \cos\theta)^2 - (c_\beta \hat{d}_j + e_\beta \hat{r}_s \sin\theta)^2\right)^{\frac{3}{2}} \, d\theta \tag{3.102}$$

To extend the Algorithm 8 on page 71 the Equations 3.96 and 3.99 have to be implemented introducing only a minor number of additional operations and a lookup table for pre-computed volumes using Equation 3.102.

3.5 Multiple Scales and Feature Vectors

Having different types of integral invariants as shown in the previous section, allows to distinguish different features of similar size. However features in general and wedges om cuneiform tablets are of different size. Figure 3.19 shows an example, where the width of the wedges is between $1\,mm$ and $4\,mm$. This figure shows a subset of three lines of cuneiform characters – the fourth line is partially broken off. Traces of two thin lines for horizontal ruling separate the lines and helped the ancient scholar to align the characters.

The volume integral invariant $\hat{V}_2$ (short for $\hat{V}_{r=2\,mm}$) in Figure 3.19b shows the larger wedges in dark color (concave) and the ridges of the fracture in light color (convex). In contrast the lines for ruling, the smaller wedges as well as small fractures of the tablet are more distinctive in Figure 3.19c using $\hat{V}_{0.4}$. Additionally a fingerprint becomes visible for both images computed using integral invariants. For the larger radius it appears as elliptic shape of darker color, which represents the concave shape of the impression left by the tip of the finger. The ridges of a fingerprint become visible for the smaller radius as their width is approximately $0.5\,mm$.



(a) **(b)** **(c)**

Figure 3.19: 3D-model of tablet W 20248,10 of the Heidelberg *Uruk-Warka* collection: Visualization of details in an area of 40 by $45\,mm$ using (a) the acquired color and virtual illumination; the volume integral invariant $\hat{V}_r$ computed (b) with $r = 2\,mm$ and (c) $r = 0.4\,mm$. The elliptic line encloses a fingerprint. The dotted lines mark the (ancient) lines for horizontal ruling. (b,c) Dark colors mean concave; medium gray flat; and bright colors convex areas.

The use and the outcomes of using two scales is already a clear indicator for the importance of using multiple scales. This is even more important as both the figures are not practicable to document the epigraphic evidence of cuneiform tablets, because all the details have to be present in one image. Additionally Figure 3.19b appears to be blurred, which is hindering the character recognition for the human expert as well as for an automated system. Vice versa Figure 3.19c contains a tremendous amount of surface details, where most of it is noise and has to be discarded.

The consequent step was to adopt the integral invariant method(s) as *Multi Scale Integral Invariant* (MSII) filters to extract relevant details i.e. features. Therefore the principle of this approach is to use multiple spheres $B_r(\mathbf{p}_i)$ with different radii r centered at the same vertex $\mathbf{p}_i$. Having concentric spheres in multiple scales we get a discrete, real-valued feature function for each of the integral invariants. These functions are shortly called feature vectors. Let n be the number of scales and $\mathcal{M}_2$ is the surface of a 3D-model, than there are volume based feature vectors:

$$\begin{aligned} \mathfrak{V} : \mathcal{M}_2 &\longrightarrow]-1,+1[^n \quad \equiv \quad]0\%,100\%[^n \\ \mathbf{p}_i &\longmapsto \mathfrak{V}_i = (\mathfrak{v}_1,\ldots,\mathfrak{v}_n)^T \qquad \text{with } \mathfrak{v}_j = \hat{V}_{r_j}(\mathbf{p}_i) \text{ and } 1 \le j \le n \end{aligned} \tag{3.103}$$

and surf based feature vectors:

$$\begin{aligned} \mathfrak{P} : \mathcal{M}_2 &\longrightarrow]0,+\infty[^n \\ \mathbf{p}_i &\longmapsto \mathfrak{P}_i = (\mathfrak{p}_1,\ldots,\mathfrak{p}_n)^T \qquad \text{with } \mathfrak{p}_j = \hat{P}_{r_j}(\mathbf{p}_i) \text{ and } 1 \le j \le n \end{aligned} \tag{3.104}$$

and line based feature vectors:

$$\begin{aligned} \mathfrak{L} : \mathcal{M}_2 &\longrightarrow]0,+\infty[^n \\ \mathbf{p}_i &\longmapsto \mathfrak{L}_i = (\mathfrak{l}_1,\ldots,\mathfrak{l}_n)^T \qquad \text{with } \mathfrak{l}_j = \hat{L}_{r_j}(\mathbf{p}_i) \text{ and } 1 \le j \le n \end{aligned} \tag{3.105}$$

computed using the concentric spheres $B_{r_j}(\mathbf{p}_i)$. The radius r_1 has to be $\ge \mathtt{R}_{min}$ and $r_n \le \mathtt{R}_{max}$ as shown in Equation 3.34 on page 59. The radii are typically uniformly spread and r_1 is equivalent to a radius of 0, when $r_1 \le \mathtt{R}_{min}$.

Figure 3.20 shows the volume, surface and line based integral invariants for the synthetic surfaces of Figure 3.11 on page 60. As the synthetic surfaces are 4 units wide and long, the radii for the 5 scales illustrating this example are: 0.4, 0.8, 1.2, 1.6 and 2. Therefore the feature vectors at the central point $\mathbf{p}_w$ of the wedge and $\mathbf{p}_w$ of the pyramid are:

$$\mathfrak{V}_w = (0.50, 0.50, 0.50, 0.49, 0.43)^T \qquad \mathfrak{V}_p = (0.53, 0.53, 0.53, 0.50, 0.43)^T \tag{3.106}$$

$$\mathfrak{P}_w = (1.00, 1.00, 1.00, 1.02, 1.05)^T \qquad \mathfrak{P}_p = (0.88, 0.88, 0.88, 0.92, 0.95)^T \tag{3.107}$$

$$\mathfrak{L}_w = (1.00, 1.00, 1.00, 1.01, 1.03)^T \qquad \mathfrak{L}_p = (0.88, 0.88, 0.88, 0.90, 0.94)^T \tag{3.108}$$

For the volume based feature vectors the percentile interval was chosen, because it shows that $\hat{V}_r$ along the base of an ideal wedge has to be 3/4, while $\hat{P}_r$ and $\hat{L}_r$ are 1. Both examples have the same value for the first three elements of each feature vector, because the enclosed shape is identical. Inverting the surface changes only the result for the volume based feature vectors, which become negative:

$$\begin{aligned} \mathfrak{V}_w &= (-0.50, -0.50, -0.50, -0.49, -0.43)^T \qquad \text{for the wedge and} \\ \mathfrak{V}_p &= (-0.53, -0.53, -0.53, -0.50, -0.43)^T \qquad \text{for the pyramid impression.} \end{aligned} \tag{3.109}$$

This means concavities and convexities can only be distinguished using the volume integral.

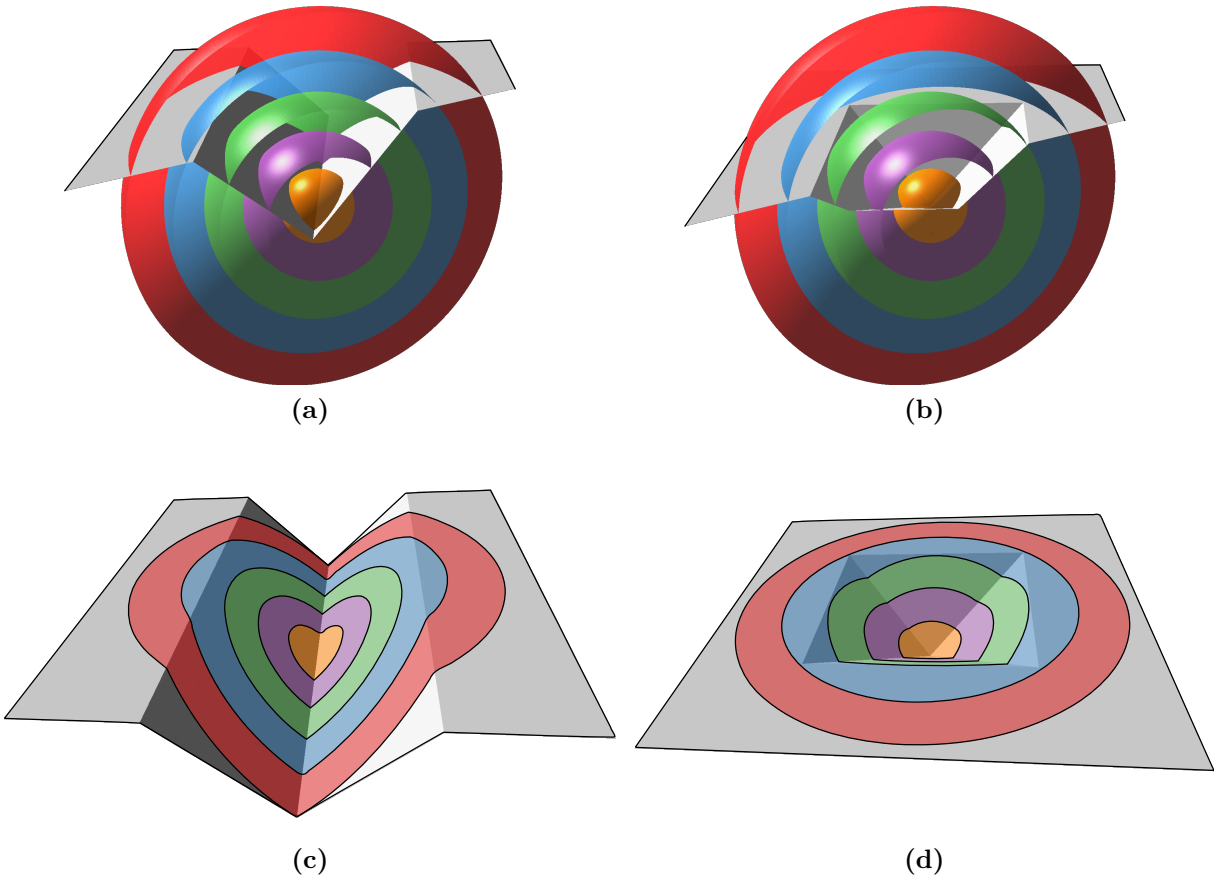


Figure 3.20: Multi-Scale Integral Invariants (MSII) for 5 scales using (a,b) the volume below the surface, (c,d) the surface area and the length of their border lines.

A classic multi-scale representation for features in signal processing is the representation as a pyramid. It is computed by repeated processing and subsampling an image or a signal i.e. a feature vector. This is efficient in terms of computing time and memory usage for data with a regular sampling interval. As meshes are irregular and pyramids introduce unflexible multigrid data structures, it is more practicable to compute the integral invariants for each point of the surface in each scale. The immediate benefit is a straight forward mapping of a single scale using the elements of the feature vectors as shown in Figure 3.19b and 3.19c using a grayscale color map to visualize the elements $\mathfrak{v}_{2mm}$ and $\mathfrak{v}_{0.4mm}$ of all feature vectors $\mathfrak{V}$ holding volume integral invariants.

This grayscale color map is actually a false-color map, despite its relatively natural appearance for the eye of a human beholder. The reason is that any color mapping to the surface computed by a function like an integral invariant is not the true color of an object, which is shown in Figure 3.19a.

3.5.1 Feature Metrics

Having feature vectors for all vertices describing the surface, the next step is to decide which vertices are relevant and belong to characters or other areas of interest. Therefore feature vectors can be interpreted as points in an n-dimensional space a classification can be done by using a metric in feature space. Let $\mathfrak{F}_k$ be a representative point for a feature then e.g. the Euclidean distance $|\mathfrak{F}_i - \mathfrak{F}_k|_2$ to the feature vectors of the points $\mathbf{p}_i$ becomes a metric for features. Such a feature vector

$$\mathfrak{F} = (\mathfrak{f}_1, \ldots, \mathfrak{f}_n)^T \tag{3.110}$$

can be one of those shown in Equation 3.103, 3.104 and 3.105 and n is the number of scales. Concatenating these vectors is generally not recommended as their feature spaces can be of different size. This is reflected by the interval of the elements of the feature vectors, which are either in $]-1,+1[$ or in $]0,+\infty[$.

Looking at the feature vectors as a function (or signal) depending on the radii of the spheres, a classification in Fourier space using *Fast Fourier Transform* (FFT) can be applied as done in signal processing. FFT is used in many applications – e.g. it is used in [PWHY09] in an approach to estimate surface integral invariants. However there are various other means to map the high-dimensional points in feature space down to a single meaningful value per vertex $\mathbf{p}_i$, which than can be used either to segment the surface or for feature visualization using false-color texture maps.

The *GigaMesh* software framework – described in the next chapter – provides the following methods using feature vectors $\mathfrak{F}_i$ as an input, where some methods require at least one reference vector $\mathfrak{F}_k$:

- Manhattan distance: $|\mathfrak{F}_i - \mathfrak{F}_k|_1$
- Euclidean distance: $|\mathfrak{F}_i - \mathfrak{F}_k|_2$ (commonly: $|\mathfrak{F}_i - \mathfrak{F}_k|$)
- Correlation R_{fg} of a signal: $f \equiv \mathfrak{F}_i$ and $g \equiv \mathfrak{F}_k$
- Auto-correlation R_{ff} of a signal: $f \equiv \mathfrak{F}_i$ [Bra65]

Successful experiments with 3rd party tools like *MathWorks Matlab* have been conducted:

- *Fast Fourier Transform* (FFT) [CT65]
- *Wavelet Transformation* (WT) [Ric53]

Neglecting the signal/function characteristic of the feature vectors, the feature space could additionally be segmented using:

- Clustering methods based on persistent homology [RML12] based on [CGOS11, EH08].

All these methods vary in computational complexity and numbers of parameters as well as robustness. Depending on the object, satisfying segmentation could be achieved with at least one of the methods. Surprisingly even the Manhattan distance, which is computed using only addition and subtraction:

$$|\mathfrak{F}_i - \mathfrak{F}_k|_1 = \sum_{j=1}^{n} |\mathfrak{f}_{i_j} - \mathfrak{f}_{k_j}| \tag{3.111}$$

could provide at least helpful visualizations for the human experts for a notable quota of real-world 3D-models. However the Manhattan and Euclidean distances are not suitable for the whole variety of man-made and partially destroyed objects. FFT and WT are computational costly, introduce extra parameters and their result is another feature space, which requires to choose and parametrize an additional function to compute a metric. Therefore the correlation functions R are a suitable trade-off as shown in the next section.

3.5.2 Convolution of Feature Vectors

To stick to the paradigm of keeping the user-set parameters as low as possible, the autocorrelation function R_{ff} is a reasonable starting point, because it does not require any reference $\mathfrak{F}_k$. Furthermore the wedges of cuneiform characters – like many other man-made surface details – have some kind of symmetry and as R_{ff} is known to response with high values for symmetric $\mathfrak{F}_i$ and low values for non-symmetric $\mathfrak{F}_i$, The result of the autocorrelation $R_i \equiv R_{ff}$ per vertex $\mathbf{p}_i$ can be visualized similar to the visualizations of single scales as shown in Figure 3.19. Additionally it acts as a basis for extraction of the cuneiform characters in the next chapter.

The auto-correlation function R_{ff} for a continuous function f is the convolution $*$ with the complex conjugate $\bar{f}$:

$$R_{ff} = f * \bar{f} = \int_{-\infty}^{+\infty} f(\tau)\bar{f}(\tau - t)d\tau = \int_{-\infty}^{+\infty} \bar{f}(\tau)f(\tau + t)d\tau \tag{3.112}$$

As the feature vectors are real, $\bar{f}$ can be substituted by f and as $\mathfrak{F}_i$ are discrete, this leads to the convolution of a discrete signal by itself, i.e. auto-correlation of feature vectors:

$$R_i = \sum_l \mathbf{R}_i(l) \tag{3.113}$$

with

$$\mathbf{R}(l) = (\mathfrak{F} * \mathfrak{F})(l) = \sum_{j\in\mathbb{Z}} \mathfrak{f}_j \mathfrak{f}_{l-j} \quad \text{with} \quad \mathfrak{f}_j \equiv 0 \text{ for } j \leq 1 \wedge j \geq n \tag{3.114}$$

$\mathbf{R}_i(l)$ has $2n$ non-zero elements for n even and $2n-1$ non-zero elements for n odd.

$\mathbf{R}_i(l)$ in Equation 3.114 means that the feature vector is understood as signal extended with zeros to both directions. The signal

$$\ldots, 0, 0, 0, \mathfrak{f}_1, \ldots, \mathfrak{f}_n, 0, 0, 0, \ldots$$

is convoluted with itself:

$$\begin{array}{r}
\ldots, \mathfrak{f}_1, \ldots, \mathfrak{f}_n, 0, 0, 0, 0, 0, 0, \ldots \\
\ldots \\
\ldots, 0, 0, \mathfrak{f}_1, \ldots, \mathfrak{f}_n, 0, 0, 0, 0, \ldots \\
\ldots, 0, 0, 0, \mathfrak{f}_1, \ldots, \mathfrak{f}_n, 0, 0, 0, \ldots \\
\ldots, 0, 0, 0, 0, \mathfrak{f}_1, \ldots, \mathfrak{f}_n, 0, 0, \ldots \\
\ldots \\
\ldots, 0, 0, 0, 0, 0, 0, \mathfrak{f}_1, \ldots, \mathfrak{f}_n, \ldots
\end{array}$$

Computing R_i for each $\mathbf{p}_i$ results in a function value, which becomes small for the inflection points along the surface $\mathcal{M}_2$ and are shown as dark colors in Figure 3.21b. These inflection points align along the border of the wedges as contour lines. Figure 3.21a shows a the surface in true color.



(a)



(b)

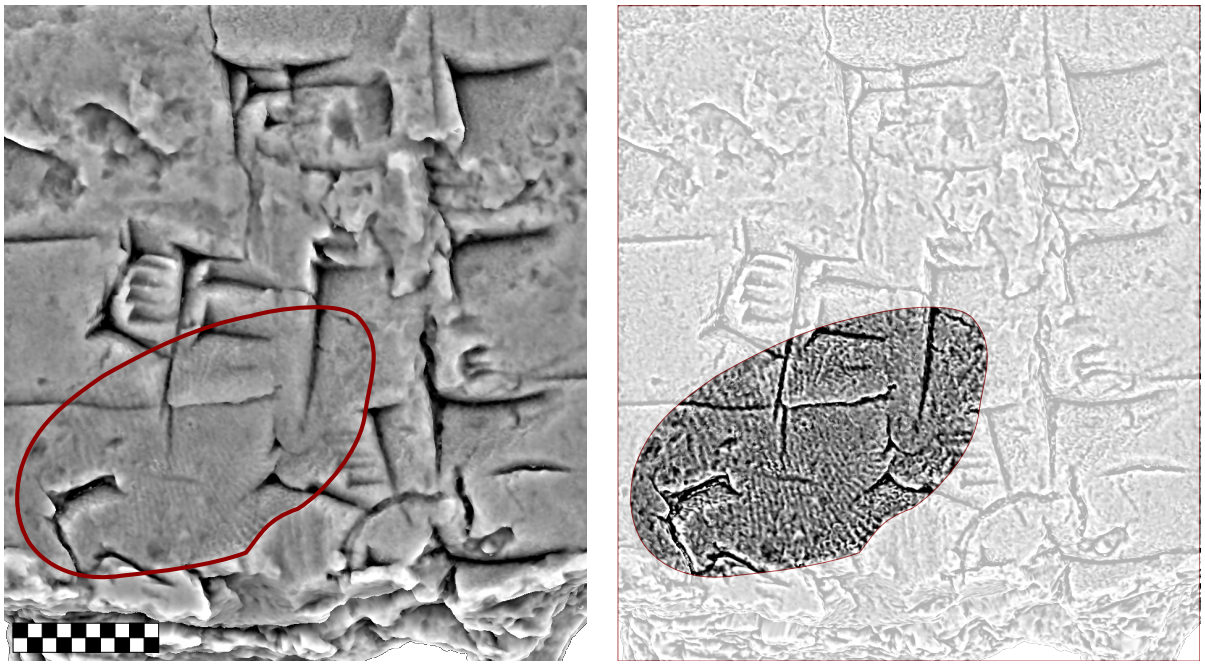
Figure 3.21: Four lines with omina from the shaded 3D model W 20430,101, *Uruka-Warka Sammlung* (a) with acquired (true color) texture map and (b) with contourlines computed using autocorrelation R_i represented by a grayscale (false color).

Figure 3.21b shows the drawback of using autocorrelation, because vertices outside the contour lines have the same R_i as the vertices inside the wedges. To overcome this drawback at least on reference point $\mathfrak{F}_k$ in th feature space has to be given. Therefore the complex conjugate $\bar{f}$ of R_{ff} hast to be exchanged by $g \equiv \mathfrak{F}_k$. This leads from autocorrelation to the (cross)-correlation $R_{i,k}$ based on:

$$R_{fg} = f * g \equiv \bar{f}(-t) * g(t) \quad \text{and} \quad R_{fg} = \int_{-\infty}^{+\infty} g(\tau) f(t-\tau) d\tau \tag{3.115}$$

The ideal reference of a wedge imprinted with a rectangular stylus is ideally $\mathfrak{F}_k = (0.5, \ldots, 0.5)^T$. $\mathfrak{F}_k$ actually varies from tablet to tablet, because a stylus is never perfect, the impressions are neither, the acquisition distorts the wedges and dirt does also distorts the surface. Practical experiments have shown that choosing already one feature vector manually selected of the bottom of one of the wedges, improves the result significantly. Choosing multiple references $\mathfrak{F}_k$ and selecting an average or minimum of multiple $R_{i,k}$ leads to minor improvements of the results.

Figure 3.22a shows $R_{i,k}$ visualized as grayscale for $r_n = 2\,mm$. In this figure the wedges are more distinctive and the overall shape as well as the ridges of the fingerprint become visible at the same time, which is an important improvement to the initial Figure 3.19 using the same scales. Figure 3.22b was computed with focus for further improving the visibilty of the fingerprint using $r_n = 0.5\,mm$.



(a) **(b)**

Figure 3.22: Detail of the 3D-model of tablet W 20248,10 of the *Heidelberg Uruk-Warka* collection: (a) $R_{i,k}$ for $r_n = 2\,mm$ and (b) enhancing the fingerprint using $r_n = 0.5\,mm$. Dark colors means concave; medium gray flat; and bright colors convex areas.

3.5.3 Combining Metrics

An alternative to choosing multiple references $\mathfrak{F}_k$ is to use different metrics to compensate for their specific drawbacks. The difficulty of combining methods is the different nature of the limits and distributions of the values of the metrics shown in Section 3.5.1. One can argue that such differences can be leveled out by normalizing the result for each metric and the application of a weighting function. In the end this typically leads to an expert system with a series of parameters, which are difficult to understand and to configure for an user.

However, not all metrics are difficult to combine – especially not, when they are related. This is the case for the (cross-)correlation shown in Equation 3.115 and autocorrelation shown in Equation 3.113. The difference of these equations is basically the choice of input data, while the computing steps are identical, when implemented. Furthermore this means that the range of these functions results are identical, while only their meaning is different: one provides the outlines of the characters, while the other can be used to determine which areas are inside or outside.

This is shown in Figure 3.23, where a cuneiform tablet is shown with illumination and a solid color as well as without illumination using false color maps for correlation and autocorrelation:



(a) **(b)** **(c)**

Figure 3.23: Cuneiform tablet of the HOS found 1975 in Assur depicting a "*Königsinschrift*": (a) Illuminated 3D-model with plain gray color and metallic surface properties to enhance the contrast. (b) Grayscale color map visualizing the correlation $R_{i,k}$ with $\mathfrak{V}_k$ selected at the deepest point of an impression of a wedge. (c) Black-red-yellow-white color map depicting the autocorrelation R_i (short for $R_{i,i}$), which emphasis the outlines of cuneiform characters. (b,c) Dark colors mean high auto- and cross-correlation.

Having the autocorrelation R_{ff} returning value within the same range as R_{fg}, an average can be computed by:

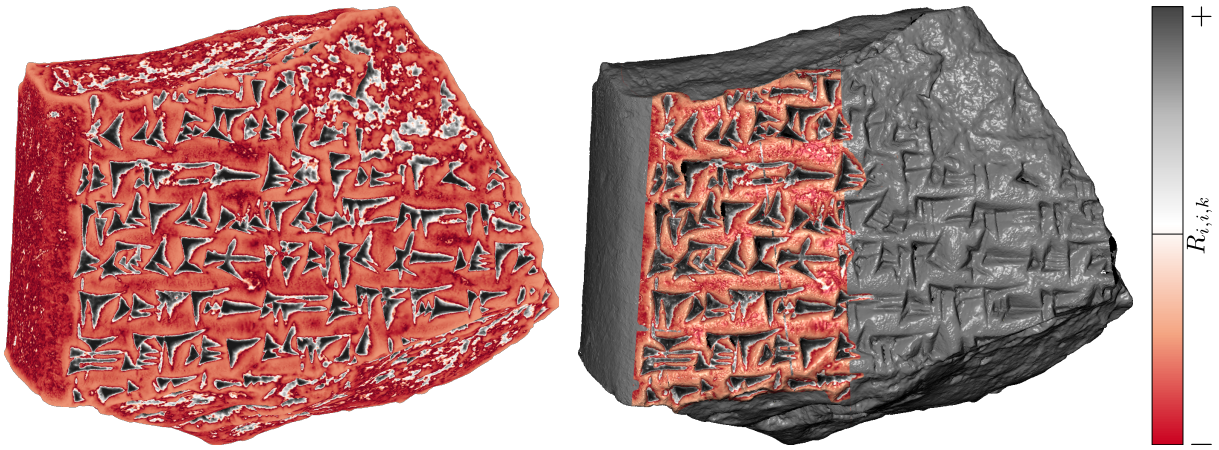
$$R_{ffg} = \frac{R_{ff} + R_{fg}}{2} \tag{3.116}$$

which is another metric leading to a combined metric for the feature vectors of $\mathbf{p}_i$:

$$R_{i,i,k} = R_{i,i} + R_{i,k} \tag{3.117}$$

withe the feature vector $\mathfrak{F}_k$ as reference. The division by 2 can be discarded as it has no influence for visualization or segmentation of $R_{i,i,k}$. Furthermore this combination has linear complexity $O(n)$, which practically means doubling the computing time of the correlation. In terms of absolute time, computing $R_{i,i,k}$ requires only a few seconds, even for the largest high-resolution data sets of cuneiform tablets.

The result for $R_{i,i,k}$ using the multi-scale volume integral invariant $\mathfrak{V}_i$ is shown in Figure 3.24. In Figure 3.24a the tablet is visualized using a red-white-black color map shown in Figure 3.24c, where red colors mark areas outside the characters. Along the contour lines is a clear contrast between the areas outside and inside the wedges, where areas outside are always in red color and wedges contain the full range of $R_{i,i,k}$ represented by white to black color:



(a) **(b)** **(c)**

Figure 3.24: 3D-Model of the HOS "*Königsinschrift*" visualized (a) using $R_{i,i,k}$ without illumination and (b) partially textured with $R_{i,i,k}$ using virtual light and metallic surface properties for better contrast. (c) Color map, where red colors mark areas outside of the wedges. Areas of the tablet within the characters show the full range of colors from white to black.

The careful observer will notice that there are areas along the fracture of tablet, which contain a distribution of values – respectively colors – of $R_{i,i,k}$ very similar to those of the wedges. These can be removed during connected components labeling using the multi-scale surface integral invariant $\mathfrak{P}_i$, which is shown in the first section of the next chapter.

3.6 Summary

This chapter shows the adaption of integral invariants for discrete data sets by introducing their concept on 1D-curves embedded in $\mathbb{R}^2$. The concept is extended to 2D-manifolds $\mathcal{M}_2$ embedded in $\mathbb{R}^3$ using regular rasterization of local surface patches $\mathcal{P}_i \subset \mathcal{M}_2$. As a result, an image processing inspired approach to estimate the volume integral invariants $\hat{V}_r$ for $\mathcal{M}_2$ is shown. The *Dalí* inspired regular filter mask used in this work, can be used to compute $\hat{V}_r$ on 2D-image data having an underlying regular grid, like range-images or *Digital Terrain Models* (DTMs) for future work.

As the ball B_r is used to compute $\hat{V}_r$ leading to a relative of the mean curvature, the surface based integral invariant $\hat{P}_r$ related to Gaussian curvature is investigated using the sphere $\mathbb{S}_r$. It is required to distinguish shapes introduced by noise from those actually being part of a cuneiform character. Additionally an efficient algorithm to compute $\hat{P}_r$ and the line based integral invariant $\hat{L}_r$ is proposed. $\hat{L}_r$ is a remote relative to the Gaussian curvature, similar to the second surface integral invariant $\hat{S}_r$, which is the remotely related of the mean curvature. These four integral invariants provide distinguished filter responses for surface features.

Afterwards a vector algebra inspired method to compute $\hat{V}_r$ is shown, which utilizes the Divergence theorem. This approach is part of a combined algorithm efficiently using the intermediate results of the computation of $\hat{P}_r$ and $\hat{L}_r$. Remarkable for this algorithm using the irregular data as it is, is the fact that it is not limited to $\mathcal{M}_2$ embedded in $\mathbb{R}^3$, because the dimension has no influence on the equations used within the proposed algorithm. This enables classification of surface features far beyond geometry by using additional information, like the acquired color of an object or its surface reflectance. Both are already provided by state-of-the-art 3D-scanners – especially those using the principle of *Time-Of-Flight* (TOF).

Because the character's basic elements and additional features like lines for ruling exists in different sizes, the multi-scale computation of integral invariants is shown. The result of this Multi-Scale Integral Invariant (MSII) filtering are high-dimensional feature vectors $\mathfrak{F}_i$. These vectors can be interpreted as signals and the application of methods like Wavelet Transformation can be applied to compute feature metrics. An approach of a combined metric using the cross-correlation and auto-correlation for improved feature representation is presented. Such combinations allow to visualize large shapes like the *Winkelhaken* within the same representation as the tiny ridges of fingerprints. The multi-scale extension requires no more than one more parameter, which can be easily selected by an user by choosing one point of the surface having a reference vector $\mathfrak{F}_k$.

Having a robustly computed feature value using MSII filtering for each point $\mathbf{p}_i$ of the surface, the next step – shown in the next Chapter – is to extract these features and reduce the high-resolution data to a meaningful vector representation as introduced in Section 1.2.

Chapter 4

Feature Extraction embedded in the GigaMesh ⊛ Software Framework

This chapter provides methods for extraction of features like characters or fingerprints. In other terms this means a massive reduction of the amount of data of an acquired 3D-model down to areas and lines of interest, which provide a meaningful and minimal description of a feature.

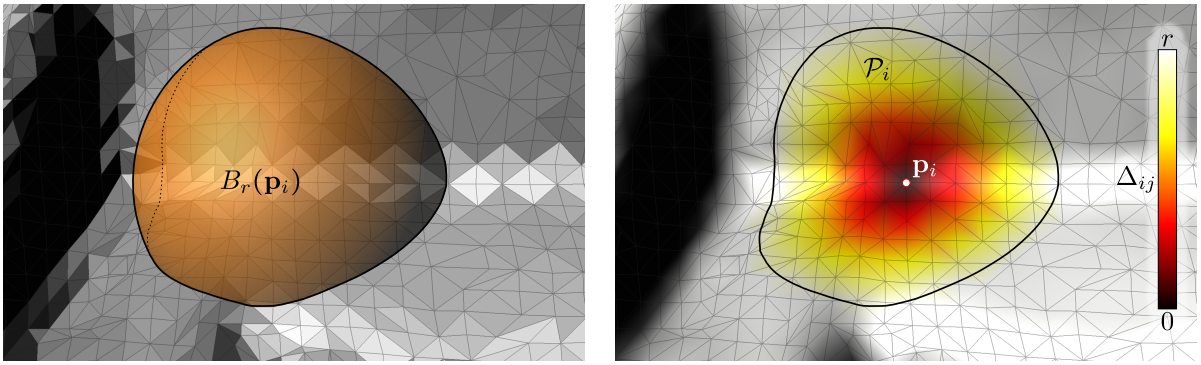
The areas of interest are determined using the connectivity provided by a mesh and the feature spaces introduced in the previous chapter. Therefore the points of the mesh are classified using the metrics applicable to feature vectors. Afterwards the local neighborhood along the surface of the classified points is analyzed and points of the same class and being neighbors become connected components. The outlines of such a components are determined with sub-triangle accuracy, because they contain even more points of interest than an area. In case of cuneiform characters three quarters of these points are located along an area's border.

The outline's points of interest are detected using a curvature estimator and two integral invariants for 1-manifolds in $\mathbb{R}^3$. These integral invariants for polygonal lines are used detect sharp corners and their angles. Afterwards a Voronoi inspired approach is applied for a robust skeletonization exclusively connecting points of extremal values of the feature metrics, e.g. maxima of a correlation. To fully extended the branches of a skeleton to the border of an area of interest, the corners detected along an outline are used as leaf nodes. Connecting the nodes is performed within the manifold using geodesic distances.

Finally the embedment of the methods within the *GigaMesh* framework is shown. It is designed for a layered, modular and efficient use of the algorithms presented within this thesis. The base layer of this framework includes detecting and repairing irregularities as well as parallel computing and a strategy for an efficient memory usage. Together with two additional layers for rendering high-resolution 3D-models and a *Graphical User Interface* (GUI) allows *GigaMesh* to be adapted for further application. This chapter shows results for synthetic data for testing the modules, while the results for real-world data are shown in Chapter 5.

4.1 Determination of Local Neighborhoods and Feature Outlines

Previous chapters assume a surface patch $\mathcal{P}_i$, which contains a local subset of a mesh within a ball $B_r(\mathbf{p}_i)$. Such a patch is shown in Figure 4.1, where its vertices $\mathbf{p}_j$ are colored by the Euclidean distance $\Delta_{ij} = |\mathbf{p}_j - \mathbf{p}_i|_2$.



(a) **(b)**

Figure 4.1: Surface patch $\mathcal{P}_i$ for computing integral invariants (a) enclosed by a ball $B_r(\mathbf{p}_i)$ and (b) its vertices colored by their Euclidean distance $\Delta_{ij} = |\mathbf{p}_j - \mathbf{p}_i|_2$ within $\mathcal{P}_i$.

The Euclidean distance introduces a complexity of $O(n^2)$, which is impractical for real world data. The common solution is the use of *Binary Space Partitioning* (BSP) [Thi87], which is typically an octree for 3D-data or a quadtree [FKN80] in case of 2D-images. The octree generally exhibits the $O(\log n)$ complexity, which is virtually constant having a reasonable depth of the tree.

As integral invariants require neighbors determined by the parameters of a surface, means that the points' distances measured by their Euclidean distances are not necessarily neighbors [MCH+06]. Therefore BSP will add vertices and triangles to $\mathcal{P}_i$ corrupting the integral invariants. This is shown for a synthetic example for a 1D-curve shaped like a bottle in $\mathbb{R}^2$ in Figure 4.2. Even for this simple example, both the Euclidean distance Δ_{ij} and the BSP contain parts of the curve located at the bottle neck, which are within the radius r, but are not neighbors in the sense of the parametrization by ϕ of the curve.

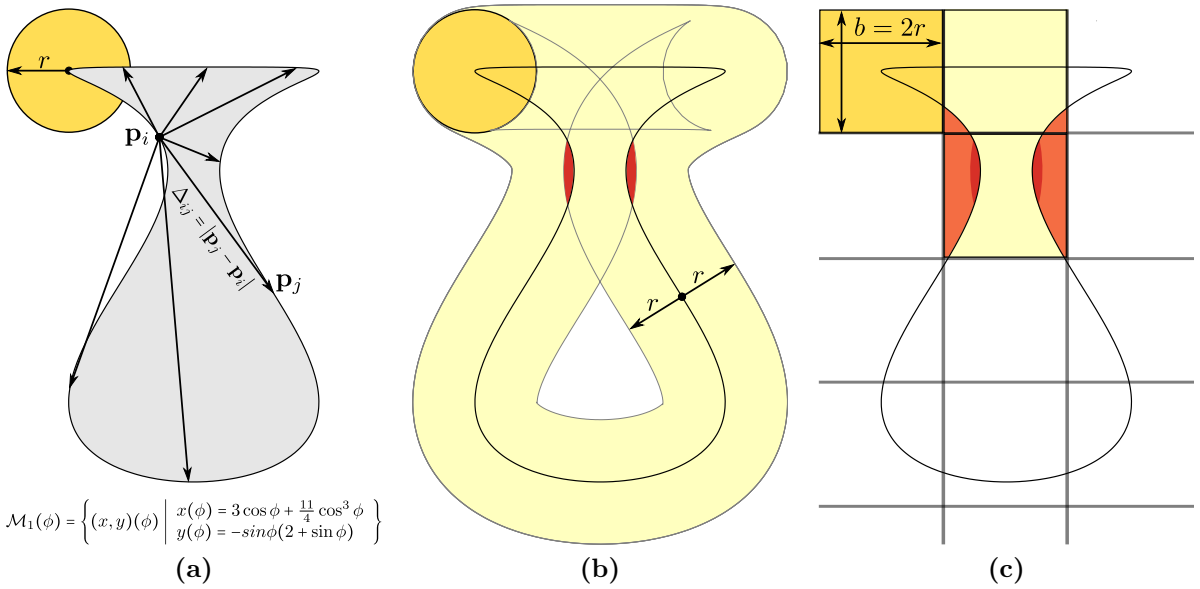


Figure 4.2: A curve $\mathcal{M}_1$ embedded in $\mathbb{R}^2$: (a) Determination of the closest neighbors $\mathbf{p}_j$ using the Euclidean distance $\Delta_{ij} \leq r$. (b) Neighbors determined using Binary Space Partitioning. The parts of the curve next to the red marked areas are wrongly tagged as neighbors by these methods, leading to incorrect results computing an integral invariant with radius r.

Combining both methods is not an alternative, because the wrongly chosen points using Δ_{ij} will remain as they are a subset of the wrong points of the BSP as shown in Figure 4.2c. One can argue that using a smaller space partitioning with $b < 2r$ and $b \to 0$ can prevent wrong points, but this leads to an increased depth of the tree, which dramatically reduces performance and a bad choice of r or a complex shape will break the system anyway.

As the parameters nor the function describing acquired surfaces is not known and virtually impossible to determine for a whole mesh, the neighborhood relation given by the edges of the triangle has to be used. For regular meshes this can be done using the k-ring neighbors, but not for irregular meshes as shown in Figure 2.6c on page 29.

Therefore an algorithm is required, which starts at a seed point $\mathbf{p}_i$ and searches along the triangulated surface until no more vertices $\mathbf{p}_j$ with $\Delta_{ij} \leq r$ are found. This leads to the so-called marching front algorithms shown in the next section.

Marching Front: Recursive — Iterative

Marching front algorithms as e.g. proposed in [NK02] provide the means to find neighbors within the geodesic distance. A remotely related method used to compute octrees, is the marching cubes algorithm [LC87], which marches through grid structures and e.g. segment volume data [MSS84]. Therefore a similar algorithm is required, which marches along the irregular triangular grid until the boundary of the ball B_r is reached.

Using the 1-ring in a recursive manner is the naïve approach, which works only on small neighborhoods, because the memory usage increases by the power of two. This approach is shown in Algorithm 9, where $\mathcal{P}_i$`.insert` adds faces to the globally accessible patch and prevents duplicates. The faces $\mathring{\mathbf{t}}_j$ of the 1-ring about $\mathbf{p}_j$ are determined according to Section 2.2 on page 29.

Algorithm 9 Marching front – Recursively using the 1-ring

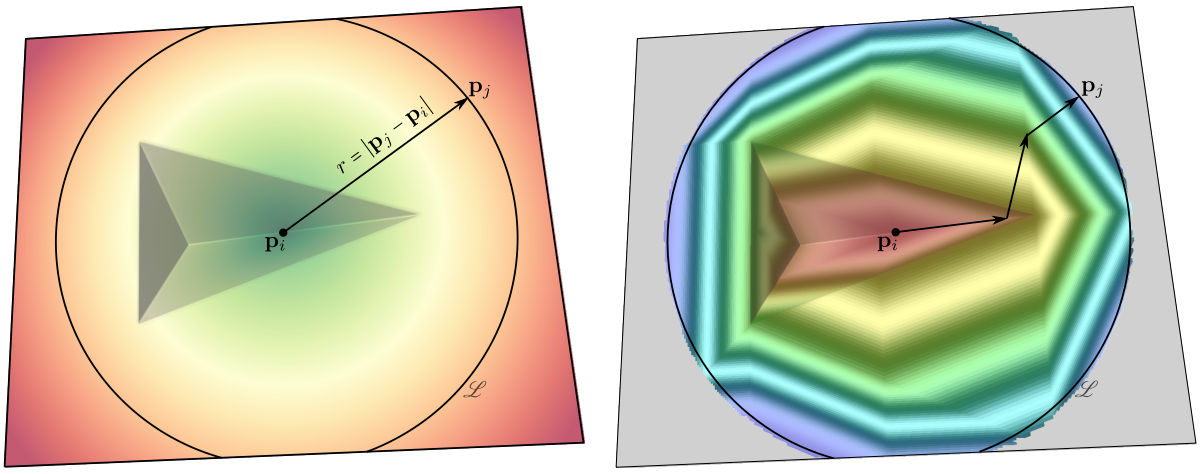
```
 1: function MARCHINGFRONT( p_j, p_i, r )          ▷ p_i and r define the ball B
 2:     L_v̊ ← p(t̊_j) \ p_i                          ▷ fetch all vertices of the 1-ring, excluding p_j
 3:     for p_k ∈ L_v̊ do                             ▷ for all 1-ring vertices
 4:         if |p_k − p_i| ≤ r then                  ▷ compute the distance to the sphere's center
 5:             P_i.insert( t̊_k )                   ▷ add the triangles of the vertex to the surface patch
 6:             P_i.insert( MARCHINGFRONT( p_k, p_i, r ) )          ▷ recursive call
 7:         end if
 8:     end for
 9:     return P_i
10: end function
```

The practical alternative to the recursive Algorithm 9 is achieved by unrolling the recursion using an explicit stack 𝌆 of vertices organized as list $\mathcal{L}_{𝌆}$. The vertices of a list $\mathcal{L}_{\mathring{v}}$ are put on top of the stack using $\mathcal{L}_{𝌆}$`.push`, while $\mathcal{L}_{𝌆}$`.pop` retrieves the vertex on top of the stack. For performance reasons it is recommended to avoid duplicate vertices within the stack. This has to happen, when a push is performed and can be efficiently implemented by using bit-arrays as shown in Section 4.4.1 on page 118. Algorithm 10 shows the computation of $\mathcal{P}_i$ without a recursion. The stack $\mathcal{L}_{𝌆}$ holds the different stages of the marching front, while $\mathcal{P}_i$ is determined.

Figure 4.3 shows a visualization of the application of the marching front algorithm. Figure 4.3a represents the Euclidean distance $|\mathbf{p}_j - \mathbf{p}_i|$ as well as the intersection $\mathscr{L}$ of the sphere $B_r(\mathbf{p}_i)$ with the surface. The false colors in Figure 4.3b denotes the number of the iteration step of Algorithm 10, when a face is added to $\mathcal{P}_i$. Faces required for computing the integral invariants along $\mathscr{L}$ and partially outside the sphere are shown in color.

Algorithm 10 Marching front using an explicit stack $\mathcal{L}_{\equiv}$ to determine $\mathcal{P}_i$

```
1:  function MARCHINGFRONT( p_i, r )                    ▷ p_i and r define the ball B
2:      L_≡.push( p_i )                                 ▷ initialize the explicit stack with the seed vertex
3:      while L_≡ ≠ ∅ do                                ▷ as long as the stack is not empty
4:          p_j ← L_≡.pop()                             ▷ fetch the vertex on top of the stack
5:          if |p_j − p_i| ≤ r then                     ▷ compute the distance to the sphere's center
6:              L_v̊ ← p(t̊_j) ∖ p_j                      ▷ fetch all vertices of the 1-ring, excluding p_j
7:              L_≡.push( L_v̊ )                         ▷ add new vertices to the stack
8:              P_i.insert( t̊_j )                       ▷ add the triangles of the 1-ring to the surface patch
9:          end if
10:     end while
11:     return P_i
12: end function
```



(a) **(b)**

Figure 4.3: Synthetic cuneus with false colors representing (a) the Euclidean distance (b) k-ring based distance (bright to dark means 5000 iteration steps) between the vertices $\mathbf{p}_j$ and the seed vertex $\mathbf{p}_i$ of an application of the marching front algorithm.

4.1.1 Connected Components Labeling

Having the feature vectors $\mathfrak{F}_i$ computed using Multi-Scale Integral Invariants, which are mapped to a meaningful function values $f(\mathbf{p}_i)$ per vertex $\mathbf{p}_i$ allows to group vertices belonging to the same feature i.e. a cuneiform characters. Such feature related groupings are called connected regions [Jäh05]. Each group of vertices gets a unique id $l_{\mathcal{C}}$ by:

- assigning an integer valued tag $l_{\mathcal{C}} \in \mathbb{Z}^+$ to each vertex, which is equivalent to
- adding the vertices to a list $\mathcal{C}_l$ – short for $\mathcal{C}_{l_{\mathcal{C}}}$.

These groups are called labels, which is short for connected components labels. The process of assigning labels is shortly refereed to as labeling.

A connected component $\mathcal{C}_l$ is determined by the labeling Algorithm 11, which is an adapted version of the efficient marching front Algorithm 10:

- r is replaced by the threshold $\mathfrak{t}_{\mathbf{p}}$ and
- $|\mathbf{p}_j - \mathbf{p}_i|$ is replaced by the generic function value $f(\mathbf{p}_j)$.

The algorithm can be initialized with any arbitrary vertex $\mathbf{p}_i$. Vertices are added to the marching front as long as their function value does no exceed their threshold ($f(\mathbf{p}_j) < \mathfrak{t}_{\mathbf{p}}$). In case a vertex with a function value exceeding the threshold is encountered, the vertex is assigned ($\hookleftarrow$) to the background $\mathcal{C}_\emptyset$ i.e. no area of interest. The term background has its origins in processing of 2D-images, where pixels can be part of a background like the sky in a photograph.

Algorithm 11 Determine a connected component $\mathcal{C}_l$ with a seed vertex $\mathbf{p}_i$

Require: $\exists\, f(\mathbf{p}_i) \,\forall\, \mathbf{p}_i \in \mathcal{L}_v$ ▷ function values have to be present

1: **function** LABEL($\mathbf{p}_i$, $\mathfrak{t}_{\mathbf{p}}$) ▷ $\mathfrak{t}_{\mathbf{p}}$ is the stop criterion for the marching front
2: $\mathcal{C}_l \leftarrow \emptyset$ ▷ initialize a new connected component
3: $\mathcal{L}_{\text{▤}}$.push($\mathbf{p}_i$) ▷ initialize the explicit stack with the seed vertex
4: **while** $\mathcal{L}_{\text{▤}} \neq \emptyset$ **do** ▷ as long as the stack is not empty
5: $\mathbf{p}_j \leftarrow \mathcal{L}_{\text{▤}}$.pop() ▷ fetch the vertex on top of the stack
6: **if** $f(\mathbf{p}_j) \leq \mathfrak{t}_{\mathbf{p}}$ **then** ▷ compare the vertex function value with the threshold
7: $\mathcal{C}_l \hookleftarrow \mathbf{p}_j$ ▷ assign the label to a vertex
8: $\mathcal{L}_{\text{▤}}$.push($\mathbf{p}_k$), $\mathbf{p}_{k\neq j} \in \mathring{\mathbf{t}}_j$ ▷ add 1-ring vertices to the stack/front
9: **else**
10: $\mathcal{C}_\emptyset \hookleftarrow \mathbf{p}_j$ ▷ tag the vertex as background $\mathcal{C}_\emptyset$
11: **end if**
12: **end while**
13: **return** $\mathcal{C}_l$ ▷ connected components group of vertices
14: **end function**

In general there is more than one connected component per $\mathcal{M}_2$ leading to Algorithm 12, which determines the list of connected components including the background:

$$\mathcal{L}_C = \{\mathcal{C}_\emptyset, \mathcal{C}_1, \ldots, \mathcal{C}_m\} \quad \text{with} \quad \mathcal{M}_2 = \mathcal{C}_\emptyset \cup \bigcup_{l=1}^{m} \mathcal{C}_l \tag{4.1}$$

Algorithm 11 assigns all vertices $\mathbf{p}_i \in \mathcal{L}_v$ to a connected component, when $f(\mathbf{p}_i) \leq \mathfrak{t}_\mathbf{p}$ i.e. $\mathbf{p}_i$ is part of an area of interest. In case this condition is not met, the vertex is assigned to the background $\mathcal{C}_\emptyset$. A modular use of the labeling algorithm is ensured, because it relies on the pre-computed $f(\mathbf{p}_i)$. Therefore the labeling does not depend on any specific function, which is typically computed using a feature metric. The conditional branch in line 5 of Algorithm 12 prevents calls of Algorithm 11 causing unnecessary computational expensive initialization and processing of the marching front before the vertex $\mathbf{p}_i$ is assigned to $\mathcal{C}_\emptyset$.

Algorithm 12 Determine the connected components $\mathcal{L}_C$ of a triangular mesh

Require: $\exists\, f(\mathbf{p}_i) \,\forall\, \mathbf{p}_i \in \mathcal{L}_v$ ▷ function values have to be present

```
 1: function LABELING( M_2, t_p )
 2:     L_C ← {C_∅} ∨ C_∅ ← ∅          ▷ remove labels and background tags from all vertices
 3:     for i = [1, |L_v|] do
 4:         if p_i ∉ L_C then            ▷ find a seed vertex not labeled nor background
 5:             if f(p_i) ≤ t_p then     ▷ having a f(p_i) below the threshold
 6:                 L_C ↩ LABEL( p_i, t_p )   ▷ determine and add an area of interest
 7:             else
 8:                 C_∅ ↩ p_i            ▷ tag the vertex as background C_∅
 9:             end if
10:         end if
11:     end for
12: end function
```

Figure 4.4 shows the result of the labeling, where gray color denotes the background $\mathcal{C}_\emptyset$ i.e. no areas of interest. Figure 4.4a depicts the iterative marching front used in Algorithm 11. The number of iterations is shown as colorramp from white to black, which appear as distorted concentric circles having an arbitrary seed vertex as center. Figure 4.4b visualizes the label number l by assigning one color per connected component. For better contrast only the 11 colors were used using $l \mod 11$ and therefore each color occurs repeatingly. A combined visualization with additional virtual illumination is shown in Figure 4.4c.

A closer investigation of Figure 4.4 shows that there is a minor number of labels in the fractured area of the tablet, which are no characters. In contrast the blank areas of the tablet on the left hand side contains virtually none of these false positive errors. The reason for these errors strongly relates to Section 3.3 on page 60, which shows that additional integral invariants improves the determination between noise and features.

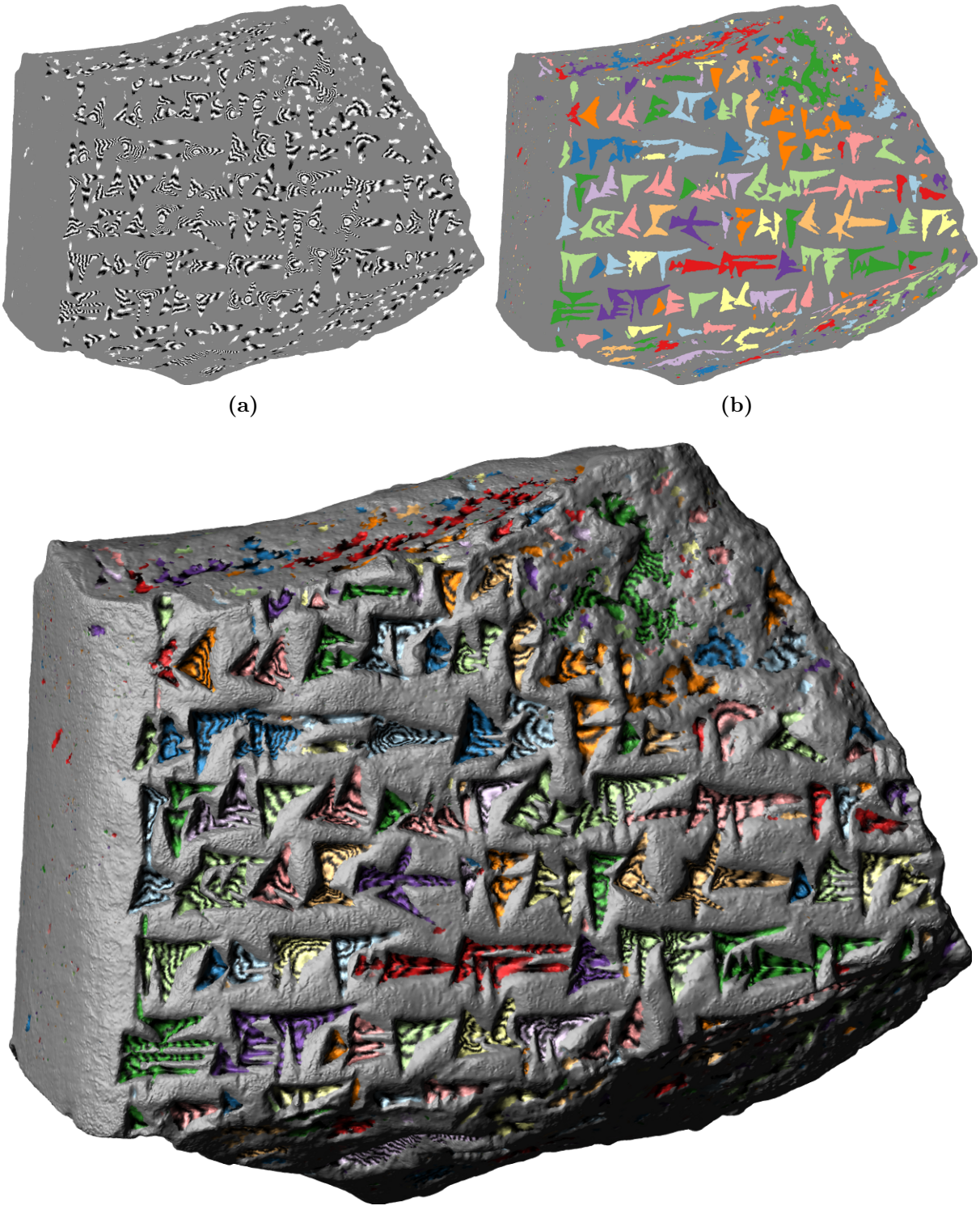


Figure 4.4: Connected components labels $\mathcal{C}_l$ of a cuneiform tablet. Areas in gray color mean background $\mathcal{C}_\emptyset$ i.e. no area of interest. (a) Marching front: black to white means 500 iteration steps of Algorithm 12. (b) Visualization of the label number $l_\mathcal{C}$ repeatingly using 11 colors for better contrast. (c) Combined visualization of (a) and (b) with virtual illumination.

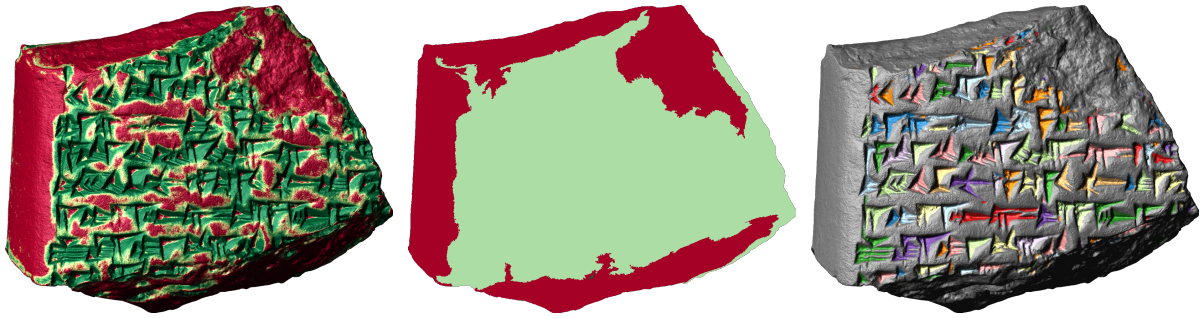
Eliminating False Positives using Connected Component Masks

False positive components can generally be discard using statistics about their area $|\mathcal{C}|$, boundary length $|\partial\mathcal{C}|$, eccentricity and other properties related to their shape. For example the areas of the components detected at the smooth side of tablet is much smaller than the smallest wedge. However in case of the false positives along the fracture there is no generic set of parameters for such statistical properties. Therefore the surface integral invariant $\hat{P}$ is incorporated to exclude areas falsely labeled as areas of interest using the volume integral invariant $\hat{V}$. This is achieved by:

1. computing the correlation of $\mathfrak{P}_i$ to the selected reference vertex $\mathbf{p}_k$ of a character,
2. labeling the areas with low correlation and
3. selecting the largest areas to discard
4. the false positive components, which were previously determined using $\hat{V}$.

Figure 4.5a shows the result of the correlation $R_{i,k} = \mathfrak{P}_i * \mathfrak{P}_k$, where areas of high correlation are shown in green. The areas in red color have a low correlation as well as $\hat{P} \to 0$, because the surface integral invariant – a relative of the Gaussian curvature – becomes zero for the flat areas as well as for the randomly curled surface of the fracture.

Figure 4.5b shows the mask derived by choosing the distinctively larger connected components i.e. areas without characters. The connected components determined based on $\hat{V}$ shown Figure 4.4c within the green areas are kept, while those in the red areas are discarded. The result is depicted in Figure 4.5c having labels only for areas containing characters. Note that the mask does not contain the large edges having a non-zero mean curvature. However this is no drawback as these convex edges were not part of an area of interest in the initial labeling.



(a) **(b)** **(c)**

Figure 4.5: Cuneiform tablet (a) colored using correlation of surface based feature vectors, where red means low correlation and green high correlation respectively a low and high values of mean curvature. (b) Derived mask using the distinctively largest connected components. (c) Result of the application of the mask, discarding labeled areas containing no characters.

Having the areas of interest determined, allows to determine the components of the feature i.e. the wedges of the cuneiform character as shown in the next section.

4.1.2 Areas of Interest – Feature Boundaries

Because three quarter of the information required to decode a character is located along the boundary $\partial\mathcal{C}$ of a labeled area, processing these boundaries is highly relevant for character extraction. As an ideal cuneus has the shape of a tetrahedron, the three corners of its base are supposed to be along $\partial\mathcal{C}$. Such an outline is a polygonal piecewise linear curve as it is derived from the piecewise flat manifold $\mathcal{M}_2$. These curves $\mathcal{M}_2$ are called polygonal chains or polylines, which is short for polygonal line.

The simplest solution to compute these outlines is to remove all vertices and faces part of the background $\mathcal{C}_\emptyset$, which lets the labeled areas remain. The result is a new mesh with boundaries equal to the outlines. Then the methods shown in Section 2.2 on page 28ff can be applied for their determination. Due to the irregular structure of acquired 3D-models, the arbitrary triangles have jagged edge polylines as a consequence using such an approach.

These jagged edge outlines are avoided by interpolating additional vertices for the polylines along the triangles edges. This leads to improved polylines, which are de-facto an isoline along the threshold $\mathfrak{t}_\mathbf{p}$ of the vertices' function values $f(\mathbf{p}_i)$. Having pairs of vertices $\mathbf{p}_A$ and $\mathbf{p}_B$ of the edges of triangles with

$$f(\mathbf{p}_A) < \mathfrak{t}_\mathbf{p} < f(\mathbf{p}_B) \quad \text{or} \quad f(\mathbf{p}_A) > \mathfrak{t}_\mathbf{p} > f(\mathbf{p}_B) \tag{4.2}$$

This criteria is implemented as follows:

$$\left(f(\mathbf{p}_A) - \mathfrak{t}_\mathbf{p}\right)\left(f(\mathbf{p}_B) - \mathfrak{t}_\mathbf{p}\right) = \begin{cases} < 0 & \text{the triangle is along } \partial\mathcal{C} \\ otherwise \end{cases} \tag{4.3}$$

Then the interpolated vertices $\bar{\mathbf{p}}_k$ of the polylines are

$$\bar{\mathbf{p}}_k = \lambda\left(\mathbf{p}_B - \mathbf{p}_A\right) + \mathbf{p}_A \;\text{ with }\; \lambda = \frac{\mathfrak{t}_\mathbf{p} - f(\mathbf{p}_A)}{f(\mathbf{p}_B) - f(\mathbf{p}_A)} \;\text{ and }\; \lambda =]0,1[\tag{4.4}$$

Because tracing and properly connecting the interpolated vertices along the triangles is done with an adapted marching front algorithm, where $\bar{\mathbf{p}}_k = \mathbf{p}_A$, $\bar{\mathbf{p}}_k = \mathbf{p}_B$ and $f(\mathbf{p}_A) = \mathfrak{t}_\mathbf{p} = f(\mathbf{p}_B)$ are special cases. These cases have to be handled correctly for a robust implementation to compute correct contour lines. The case $\bar{\mathbf{p}}_k = \mathbf{p}_A = \mathbf{p}_B$ can only occur for faces with zero area, which can be excluded and detected in advance as shown on page 32 in Section 2.2.

The adaption of the marching front algorithm is as follows. Instead if vertex, the seed is a face $\mathbf{t}_i$ having an edge fulfilling the criteria shown in Equation 4.2. Than the adjacent face along this edge becomes the seed for the next iteration. Generally the marching front has to stop, when it returns to the initial seed face $\mathbf{t}_i$, because for manifolds having no holes, the polylines are closed curves. However due to holes the marching front has to trace the isoline a starting a second time from $\mathbf{t}_i$ in opposite direction, when a border of the mesh is reached.

As the faces of the mesh have an orientation, this orientation should be maintained and inherited to the polyline, because it is useful for further processing and visualization of the feature boundaries. Generally the adjacent faces along the orientated edges with the criteria of Equation 4.2 are used to iteratively trace and compute the polylines' vertices. In case of $\bar{\mathbf{p}}_k = \mathbf{p}_A$ or $\bar{\mathbf{p}}_k = \mathbf{p}_B$ the 1-ring of the vertex equaling $\bar{\mathbf{p}}_k$ has to be used to find the next adjacent face.

Figure 4.6 shows the final and intermediate results of processing a detail of a cuneiform tablet. Figure 4.6a is the virtually illuminated mesh used to compute the MSII filter result stored as function value $f(\mathbf{p}_i)$, which represents the correlation to a character in Figure 4.6b. Figure 4.6c shows the connected components surrounded by the adjacent irregular faces determined using the 1-ring. These faces are traced by the adapted marching front to compute the contour line – isoline along the threshold $\mathfrak{t}_\mathbf{p}$ – shown in Figure 4.6d. The figures showing the labeled areas are slightly shaded, because the labels and polylines are still embedded in $\mathbb{R}^3$ for further analysis as shown in the next Section.

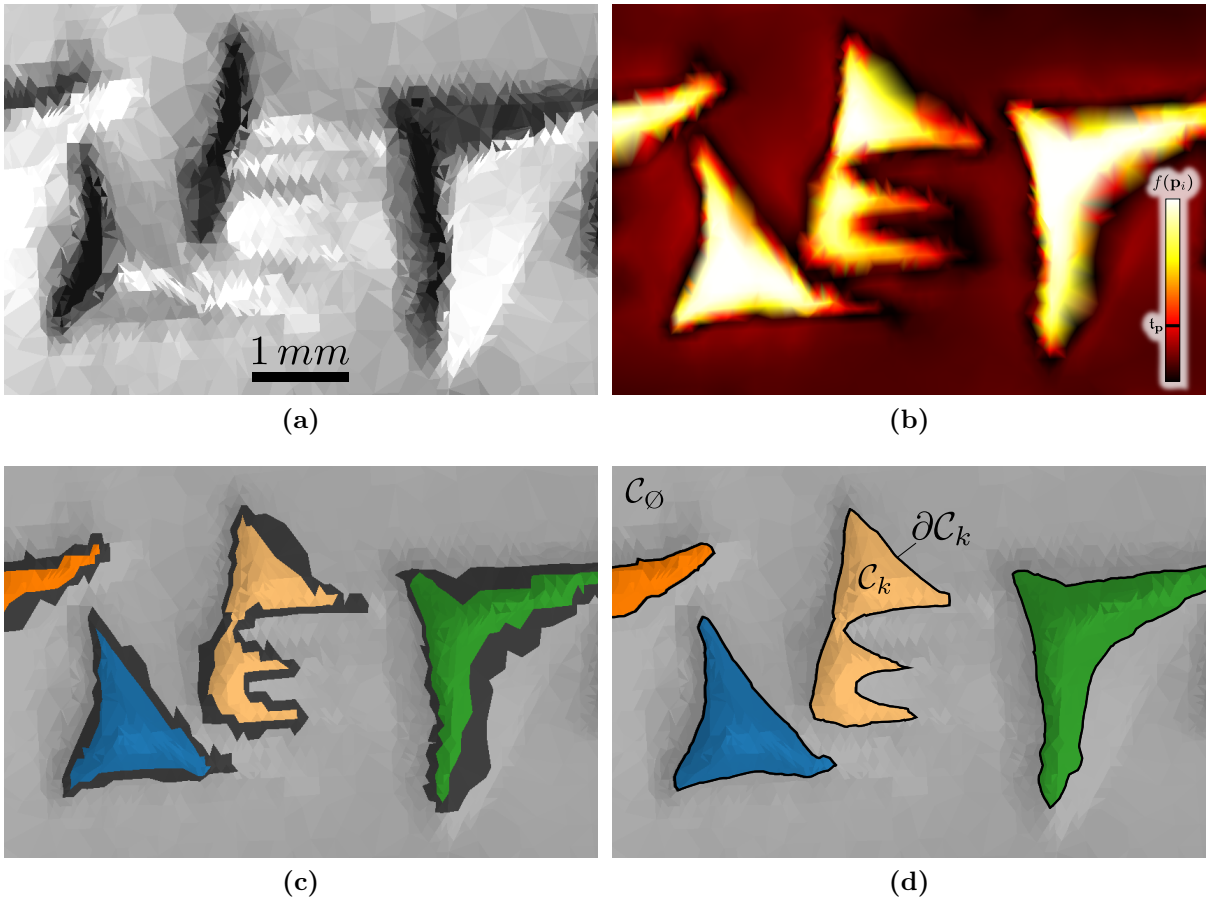


Figure 4.6: Detail of the cuneiform tablet HOS G10 with (a) virtual illumination using flat shading to show the mesh's triangles, (b) result of the MSII filtering, where bright colors mean high correlation $f(\mathbf{p}_i)$ to a feature, which is used to compute (c) the connected components $\mathcal{C}_k$ shown in color. The dark colored area surrounding the labels are the irregular faces in the 1-ring neighborhood, which are used to trace the (d) contour lines $\partial\mathcal{C}_k$ represented as polyline.

4.2 Detecting Characteristic Points in Polygonal Lines

The previous section provides a polyline $\mathbf{l}_k \equiv \partial\mathcal{C}_k$, which is a 1D-curve embedded in $\mathbb{R}^3$ as well as it is embedded in the surface $\mathcal{M}_2$. Three quarter of the characteristic points of the cuneiform characters are located along this one-dimensional manifold. These points have distinct curvature maxima of the outlines, which have to be distinguished from other maxima due to noise. Additionally there are other extremal points, which were formed by imprinting adjacent and intersecting wedges.

In terms of data structure, a polygonal line is related to the definition of the faces $\mathbf{t}$ in Equation 2.5 on page 26. Therefore such a line is a sorted list of points $\{\mathbf{p}_A, \mathbf{p}_B, \ldots, \mathbf{p}_N\}$. A closed polyline means that the first point $\mathbf{p}_A$ of the polygonal chain is also its last point $\mathbf{p}_A \equiv \mathbf{p}_N$. This assumption does not hold for real world data due to missing data (holes) and therefore $\mathbf{p}_A$ is stored as $\mathbf{p}_N$ to distinguish between open and closed polygonal lines.

In good practice of software engineering the data structure and processing methods of polylines as shown in this chapter can be re-used to e.g. analyze borders of the mesh or outlines of areas of interest determined by other algorithms. This can lead to polygonal lines with branches, which have to be represented accordingly by trees instead of lists. However the presented principles to compute curvature related measures to determine features remain the same.

These polygonal lines for a 3D-model are stored in a list $\mathcal{L}_l$. Each of the vertices computed using orientated edges, inherit the normal of the face(s) an edge belongs to. This normal is required for three purposes, which are used and shown in the next sections:

- proper rendering of an outlines visibility, when displayed together with the mesh,
- projection/unwrapping of characters from all sides of cuneiform tablets and
- signed curvature estimation, required to determine between convex and concave points of interest.

This means that a polyline $\mathbf{l}$ actually holds pairs of vectors for position and orientation of $\mathcal{M}_2$:

$$\mathbf{l} = \{(\mathbf{p}_A, \mathbf{n}_A), (\mathbf{p}_B, \mathbf{n}_B), \ldots, (\mathbf{p}_N, \mathbf{n}_N)\} \tag{4.5}$$

Figure 4.7 shows a polygonal line outlining three parallel wedges, previously shown together with other examples of outlines in Figure 4.6. Figure 4.7b shows a detail of the tip of the lower wedge, denoted as $\mathbf{p}_V$ having adjacent vertices $\mathbf{p}_U$ and $\mathbf{p}_W$. The normals of the orientated faces $\mathbf{t}_i$ and $\mathbf{t}_j$ are shown, which are used to compute an average normal $\mathbf{n}_V = (\mathbf{n}_i + \mathbf{n}_j)/2$ for $\mathbf{p}_V$. Furthermore all five points of interest $\{\mathbf{p}_E, \mathbf{p}_K, \mathbf{p}_M, \mathbf{p}_Q, \mathbf{p}_V\} \in \mathbf{l}_k$, which have to be detected along the polygonal line are marked in Figure 4.7a.

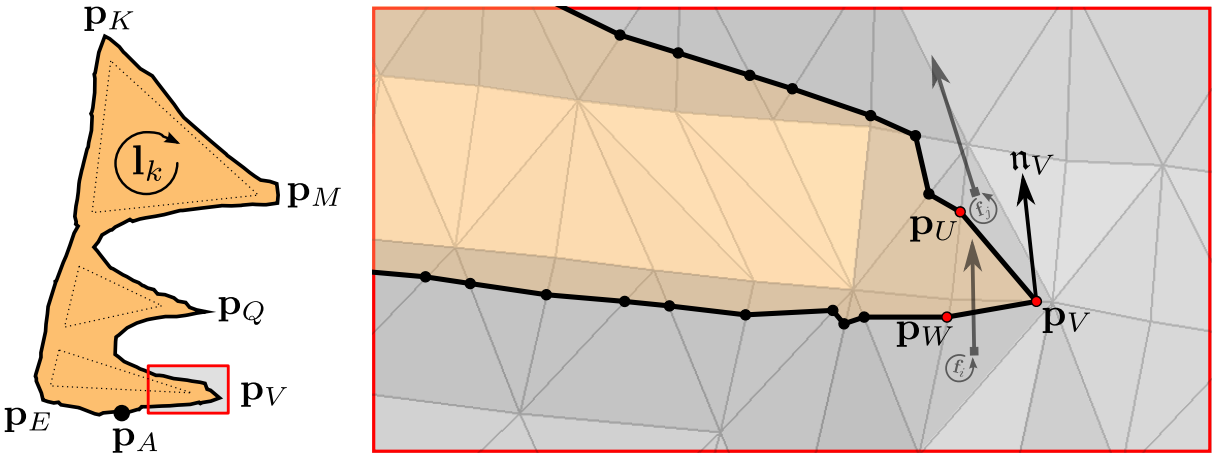


Figure 4.7: Three adjacent parallel wedges with horizontal orientation enclosed by (a) a polygonal line $\mathbf{l}_k$ having the five points of interest denoted as $\{\mathbf{p}_E, \mathbf{p}_K, \mathbf{p}_M, \mathbf{p}_Q, \mathbf{p}_V\}$. $\mathbf{p}_A$ marks the first and last point of the closed polygon. (b) Detail of $\mathbf{p}_V$ having a normal $\mathbf{n}_V$ inherited of the faces $\mathbf{t}_i$ and $\mathbf{t}_j$ used to compute the polygonal line.

4.2.1 Signed Curvature Estimation

Per-se the curvature of a polygon is infinite at its corners and zero in between the vertices. This applies to the computed outlines as well as for the expected outlines of the wedges. Therefore the points of the computed outlines have to be matched to the corners of the wedges. The curvature $\tilde{\kappa}$ for a point $\mathbf{p}_V$ with adjacent points $\mathbf{p}_U$ and $\mathbf{p}_W$ – as shown in Figure 4.7b – is estimated as

$$\tilde{\kappa}(\mathbf{p}_V) = \frac{(|\mathbf{p}_{UV}| + |\mathbf{p}_{VW}|)\,|\mathbf{n}_{UVW}|}{|\mathbf{p}_{UV}||\mathbf{p}_{VW}||\mathbf{p}_{UW}|} \quad \text{with} \quad \mathbf{n}_{UVW} = \mathbf{p}_{UV} \times \mathbf{p}_{VW} \tag{4.6}$$

$$\text{and} \quad \mathbf{p}_{UV} = \mathbf{p}_V - \mathbf{p}_U \;,\; \mathbf{p}_{VW} = \mathbf{p}_W - \mathbf{p}_V \;\text{ and }\; \mathbf{p}_{UW} = \mathbf{p}_W - \mathbf{p}_U \;.$$

The result is an unsigned curvature estimator $\tilde{\kappa} \geq 0$, which does not allow to distinguish between convex and concave corners of the polygon. As all the points of interest are convex corners it is necessary to compute a signed curvature.

Because each vertex $\mathbf{p}_V$ of the polyline is accompanied by a normal $\mathbf{n}_V$, a sign can be assigned to $\tilde{\kappa}$ by comparison with the normal used in Equation 4.6:

$$\overset{\pm}{\kappa}(\mathbf{p}_V) \leftarrow \begin{cases} -\tilde{\kappa}(\mathbf{p}_V) & \langle \mathbf{n}_V, \mathbf{p}_{UV} \times \mathbf{p}_{VW} \rangle < 0 \\ +\tilde{\kappa}(\mathbf{p}_V) & otherwise \end{cases} \tag{4.7}$$

The result for the polygon shown in Figure 4.7b using Equation 4.6 and 4.7 is shown in Figure 4.8a. As expected $\overset{\pm}{\kappa}_V$ is described by a discrete series of peaks along the run-length of the polygon, which merely correspond to 5 positive peaks and 2 negative peaks of an outline of the three parallel wedges.

Smoothing the peaks of $\overset{\pm}{\kappa}$ along the run-length of the polygonal chain with a Gaussian kernel, where d_{max} is the maximum run-length $d_{\mathbf{l}}(\cdot,\cdot)$ of the vertices $\mathbf{p}_{V+i}$ to $\mathbf{p}_V$

$$\bar{\kappa}(\mathbf{p}_V) = \frac{1}{\sqrt{2\pi}} \sum_i \overset{\pm}{\kappa}(\mathbf{p}_{V+i})\, e^{\tau} \quad \text{with} \quad \tau = -\frac{9}{2}\left(\frac{d_{\mathbf{l}}(\mathbf{p}_V, \mathbf{p}_{V+i})}{d_{max}}\right)^2 \tag{4.8}$$

leads to feasible curvatures values for all vertices allowing for detection of points of interest.

In Equation 4.8 only the neighboring vertices $\mathbf{p}_{V+i} \in \mathbf{l}$ with $d_{\mathbf{l}} \leq d_{max}$ are used to compute $\bar{\kappa}$. This prevents a computational complexity of $O(n^2)$, where $n = |\mathbf{l}|$ is the cardinality of the points describing the polyline. The equation additional implies a normal distribution $\mathcal{N}(\mu, \sigma^2)$ centered at $\mathbf{p}_4$ by a mean value $\mu = 0$, narrowed by a variance $\sigma^2 = 1/3$ and being normalized by d_{max}.

The improved results are shown in Figure 4.8b for $d_{max} = 0.5\,mm$ and for $d_{max} = 1\,mm$ in Figure 4.8c.

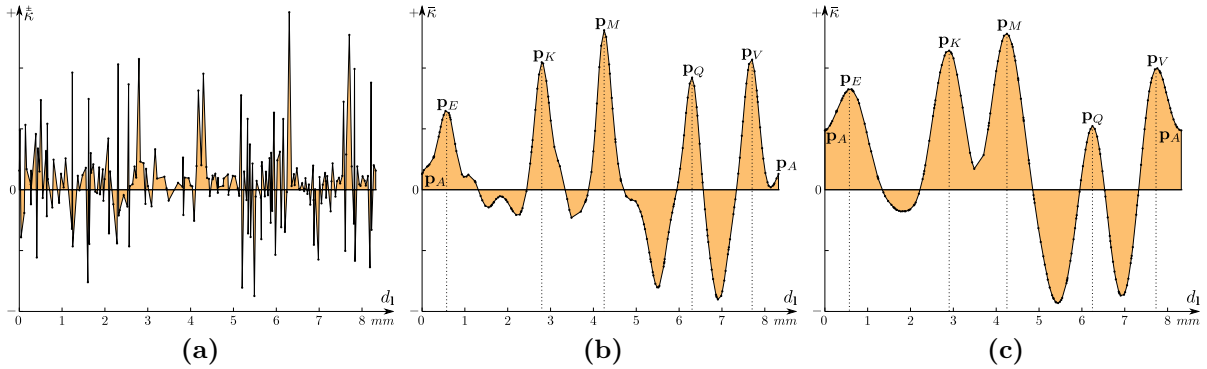


Figure 4.8: Estimated signed curvature (a) using Equation 4.7. (b) Smoothed signed curvature $\bar{\kappa}$ using a Gaussian kernel with a width of $0.5\,mm$ and (c) $1\,mm$. The first vertex of $\mathbf{l}_k$ is $\mathbf{p}_A$, while points of interested are annotated as $\mathbf{p}_B$ to $\mathbf{p}_V$. These points are located at the positive maxima of $\bar{\kappa}$. The x-axis shows the run-length $d_{\mathbf{l}}(\mathbf{p}_A, \mathbf{p}_i) \forall \mathbf{p}_i \in \mathbf{l}_k$.

The meaning of the curves describing the signed curvature $\bar{\kappa}$ become clear, when they are plotted as *OpenGL* polygon strip along the mesh. Such an *OpenGL* visualization is shown in Figure 4.9 for the previous' sections example for cuneiform outlines.

The width d_{max} and the variance σ^2 for smoothing has to be chosen heuristically. These choices have a strong influence on the success of detecting the points of interest. Therefore the next section shows an alternate approach using two integer invariants having a single parameter r, which is determinable using the *Shannon theorem* shown in Section 2.1 on page 19.

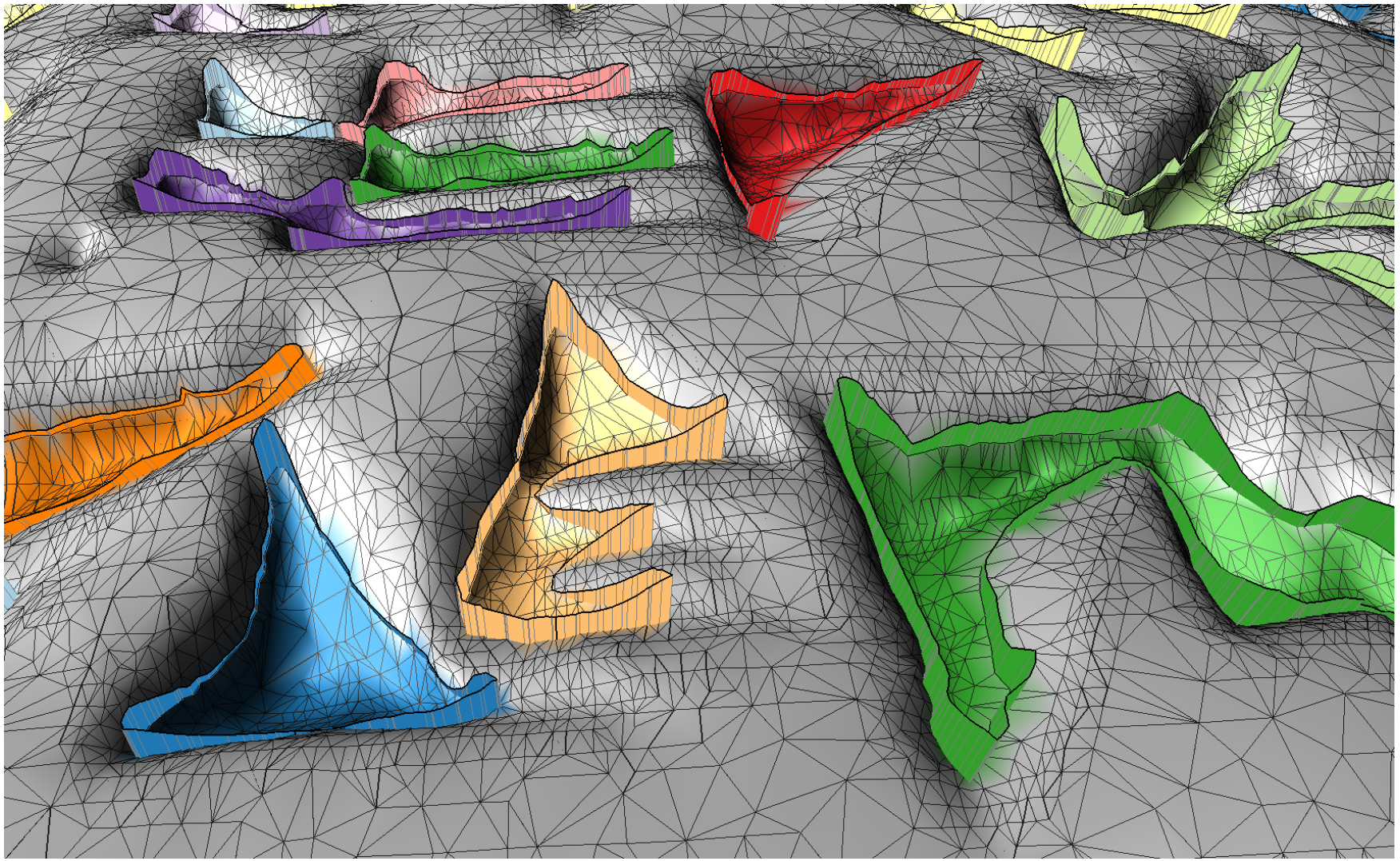


Figure 4.9: *OpenGL* visualization using a polygon strip orthogonal to the mesh representing the signed and smoothed curvature $\bar{\kappa}$. The polygon strips are shifted upwards by the minimal (negative) value of $\bar{\kappa}$ for visualization of all function values $\bar{\kappa}(\mathbf{p})$.

4.2.2 Sharp Corner Detection using Integral Invariants

Using curvature estimation and smoothing as shown in the previous section, has drawbacks like choosing the width of the Gaussian kernel and unwanted extrema, e.g. the positive maximum between $\mathbf{p}_E$ and $\mathbf{p}_K$. Furthermore the estimated location of the points of interest are prune to shifting depending on the kernel width.

An alternative and more robust approach is to apply integral invariants for curves $\mathcal{M}_1$ embedded in $\mathbb{R}^3$. Section 3.1 introduces an integral invariant $\hat{A}_r(\mathbf{p}_i)$ in Equation 3.3 on page 43 for $\mathcal{M}_1$ embedded in $\mathbb{R}^2$, which does not exist in $\mathbb{R}^3$. However there are two other integral invariants, which can be computed for 1D-curves in $\mathbb{R}^2$, $\mathbb{R}^3$ and $\mathbb{R}^{>1}$ in general. The first integral invariant is the run-length within the ball:

$$Q_r = |\mathcal{Q}_r| \quad \text{with} \quad \mathcal{Q}_r = \mathbf{l} \cap B_r^{>1}(\mathbf{p}_i) \tag{4.9}$$

and is related to Gaussian curvature in similar manner than P_r for $\mathcal{M}_2$ in $\mathbb{R}^3$. The second integral invariant is computed using the two points of intersection of

$$\mathscr{M}_r = \mathbf{l} \cap \mathbb{S}_r^{>0}(\mathbf{p}_i) = \{\mathbf{p}_u, \mathbf{p}_v\} \tag{4.10}$$

because $M_r = |\mathscr{M}_r|$ is an angle related to curvature

$$M_r \rightsquigarrow \alpha_\kappa = \arccos\left(\frac{\langle \mathbf{u}, \mathbf{v}\rangle}{|\mathbf{u}||\mathbf{v}|}\right) \text{ with } \mathbf{u} = \mathbf{p}_i - \mathbf{p}_u \text{ and } \mathbf{v} = \mathbf{p}_v - \mathbf{p}_i \tag{4.11}$$

related to L_r for $\mathcal{M}_2$ in $\mathbb{R}^3$. Therefore the limits of the ball/sphere's radius r have to be within the interval $[\mathtt{R}_{min}, \mathtt{R}_{max}]$ as shown in Section 3.2 on pages 58ff.

These two integral invariants – and their relatives – have a very different behavior, when they are applied to a sharp edge, which is any edge with $\beta < \pi/2$: Let $\mathbf{p}_C$ be a corner with an angle β as shown in Figure 4.10a, than the integral invariant $\hat{Q}$ is computed as:

$$\hat{Q}(\mathbf{p}_i) = \frac{\hat{Q}^+(\mathbf{p}_i) + \hat{Q}^-(\mathbf{p}_i)}{2} = \frac{1}{2}\left(\frac{Q^+(\mathbf{p}_i)}{r} + \frac{Q^-(\mathbf{p}_i)}{r}\right) \tag{4.12}$$

where "+" denotes the segments in forward direction of $\mathbf{l}$ and "−" in backward direction. Therefore $\hat{Q}^+(d)$ is a function of (filter response to) the sphere being moved along $\mathbf{l}$ towards $\mathbf{p}_C$:

$$Q^+(d) = d + e(d) \text{ with } d = |\mathbf{p}_C - \mathbf{p}_i| \tag{4.13}$$

where

$$e(d) = \frac{1}{2}\left(-p + \sqrt{p^2 - 4q}\right) \text{ with } p = 2d\cos\beta \text{ and } q = d^2 - r^2 \tag{4.14}$$

shown in Figure 4.10b for $\beta = [0, \pi]$.

Leading to the second filter response

$$\alpha_\kappa(d) = \arccos\frac{a(d)}{r} = \arccos\frac{d + e(d)\cos\beta}{r} \tag{4.15}$$

shown in Figure 4.10c for $\beta = [0, \pi]$, which can be normalized to $[0, 1]$ by π. In this ideal case the response for $\hat{Q}(d)$ and $\alpha_\kappa(d)$ between $\mathbf{p}_C$ and $\mathbf{p}_D$ is the same mirrored result. Due to normalization, the radius has no further influence on $\hat{Q}$ nor α_κ.

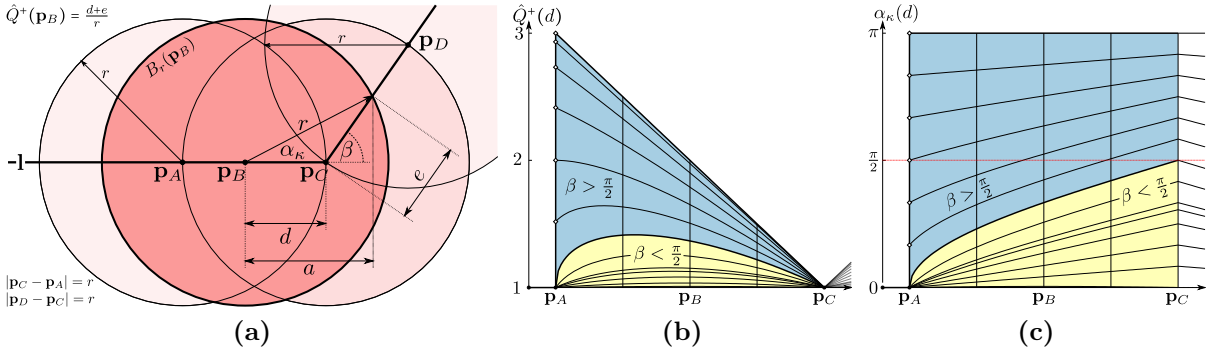


Figure 4.10: Ideal polyline $\mathbf{l}$ with (a) corner at $\mathbf{p}_C$ and an angle β. Integral invariants (b) $\hat{Q}^+(d)$ and (c) $\alpha_\kappa(d)$ computed for $\beta = [0, \pi]$ along $\mathbf{l}$ between $\mathbf{p}_A$ and $\mathbf{p}_C$. Sharp corners with $\beta > \pi/2$ shown in blue color have a step response.

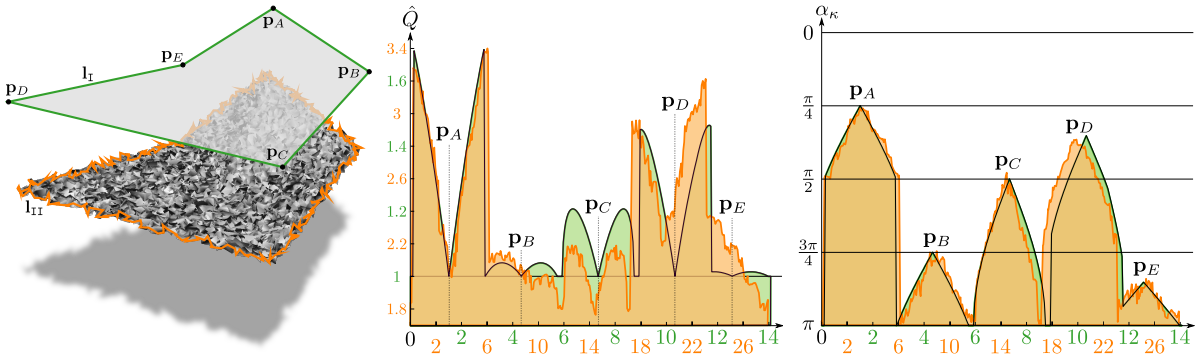
For closed polylines, $\hat{Q}^+$, $\hat{Q}^-$ and $\hat{Q}$ have an interval of $[1, +\infty]$. In case of real world data, these integrals can be computed for open polylines resulting in an interval of $[0, +\infty]$. Values of $[0, 1]$ are expected at the begin and end of an open polyline. The other case for $\hat{Q} < 1$ is, when $r > \mathtt{R}_{max}$.

As Figure 4.10b suggests, $\hat{Q}^+$ and $\hat{Q}^-$ are expected to be smaller than 3, which means $\beta = \pi$ for an ideal polyline. Vice versa one can expect $\hat{Q} \approx 1$ if there is any straight line segment present. The second interesting characteristic shown in Figure 4.10 is the fact, that corners with an angle $\beta > \pi/2$ have a very step response for both integral invariants. This means that each corner can be detected using these very distinct responses, having amplitudes corresponding to β. Furthermore these step responses are located exactly at the distance r to the corner in each direction along the polyline.

For practical application, where straight lines and well defined corners have to be found, an interval $[\ll 1, \gg 2]$ for $\hat{Q}(\mathbf{l}) = \hat{Q}(\mathbf{p}_i) \forall \mathbf{p}_i \in \mathbf{l}$ is expected as a consequence of noise. The same applies for the directional integral invariants $\hat{Q}^+$ and $\hat{Q}^-$, which have an interval $[\ll 1, \gg 3]$, when noise is present. This includes any other distortion introduced e.g. by numerical errors due to interpolating the polyline in previous processing steps. Therefore noise can be canceled by scaling $\hat{Q}(\mathbf{l})$ by making an educated guess using the distribution of values outside the expected interval.

Figure 4.11 shows an ideal polyline $\mathbf{l}_\mathtt{I}$ in green color computed from an ideal surface patch. This patch has five corners including the distinct angles of $\pi/4$, $\pi/2$ and $3\pi/4$. The patch's vertices are heavily distorted by random shifting leading to a strongly jagged outline $\mathbf{l}_\mathtt{II}$ (in orange). It has to be stressed, that estimating the curvature of $\mathbf{l}_\mathtt{II}$ as shown in the previous section has no solution allowing to determine any of the original corners.

In this example the values of $\hat{Q}(\mathbf{l}_{\text{II}})$ are in $[1.8, 3.4]$. Because the 10% quantile of $\hat{Q}(\mathbf{l}_{\text{II}})$ is ≈ 2, scaling $\hat{Q}(\mathbf{l}_{\text{II}})$ by $1/2$ leads to a result having only minor differences to its ideal counterpart $\hat{Q}(\mathbf{l}_{\text{I}})$. The same applies to $\alpha_\kappa(\mathbf{l}_{\text{II}})$, where only the run-length has to be scaled. Computing the scaling factor using the run-length of $\mathbf{l}_{\text{I}}$ and $\mathbf{l}_{\text{II}}$ yields an even better result, but it is neglected as such ground truth does typically not exist for real world data.



(a) **(b)** **(c)**

Figure 4.11: Influence of noise on $\hat{Q}$ and α_κ: (a) The ideal outline $\mathbf{l}_{\text{I}}$ (green) is degenerated using vertex displacement to $\mathbf{l}_{\text{II}}$ (orange). (b) $\hat{Q}(\mathbf{l}_{\text{II}})$ scaled down using $1/2$ as an educated guess (10% quantile). For comparison $\hat{Q}(\mathbf{l}_{\text{I}})$ for the ideal polyline is shown. (c) α_κ for $\mathbf{l}_{\text{I}}$ and $\mathbf{l}_{\text{II}}$, where only the run-length had to be scaled by $1/2$.

Because $r \lessapprox |\mathbf{p}_E - \mathbf{p}_D|$ was chosen, the dull corner $\mathbf{p}_E$ is practically detectable only in $\alpha_\kappa(\mathbf{l}_{\text{II}})$ and not in $\hat{Q}(\mathbf{l}_{\text{II}})$. This shows that α_κ is the more robust measure against noise.

An additional property of α_κ can be observed in Figure 4.11c: The values for α_κ at the distinctive corners $\mathbf{p}_j$ are equal to β:

$$\lim_{\mathbf{p}_i \to \mathbf{p}_j} \alpha_\kappa(\mathbf{p}_i) = \beta \quad \text{for} \quad \mathbf{p}_j \in \{\mathbf{p}_A, \mathbf{p}_B, \mathbf{p}_C, \mathbf{p}_D, \mathbf{p}_E\} \tag{4.16}$$

Because the distance to a corner is $|\mathbf{p}_i - \mathbf{p}_j| = d$ as shown in Equation 4.15 and with the definition of $e(d)$ in Equation 4.14 shown in Figure 4.10c, the proof is:

$$\lim_{d \to 0} \left(\arccos \frac{d + e(d) \cos \beta}{r} \right) = \beta \quad \text{with} \quad \lim_{d \to 0} e(d) = r \tag{4.17}$$

Therefore the y-axis in Figure 4.11c shows β at the maxima of $\alpha_\kappa(\mathbf{p}_i)$ and these maxima are located exactly at the correct position along the polylines.

Application of Q_r and α_κ to Cuneiform Outlines

The previous section shows the properties for integral invariants of 1D-manifolds on synthetic data leading to robust corner detection. Therefore the example of the cuneiform characters previously used for to show the curvature estimator $\bar{\kappa}$ allows for a direct comparison with integral invariants. The polyline $\mathbf{l}_k$ and three vertices illustrating the integral invariants are shown in Figure 4.14.

The computed values for $\alpha_\kappa(\mathbf{l}_k)$ are shown in Figure 4.12 for three scales of r – similar to the normalization parameter d_{max} for $\bar{\kappa}$. These scales are: $\mathtt{R}_{min}$, $0.5\,mm$ and $1\,mm$. In contrast to $\bar{\kappa}$, the maxima of α_κ are not shifted, when r is changed.

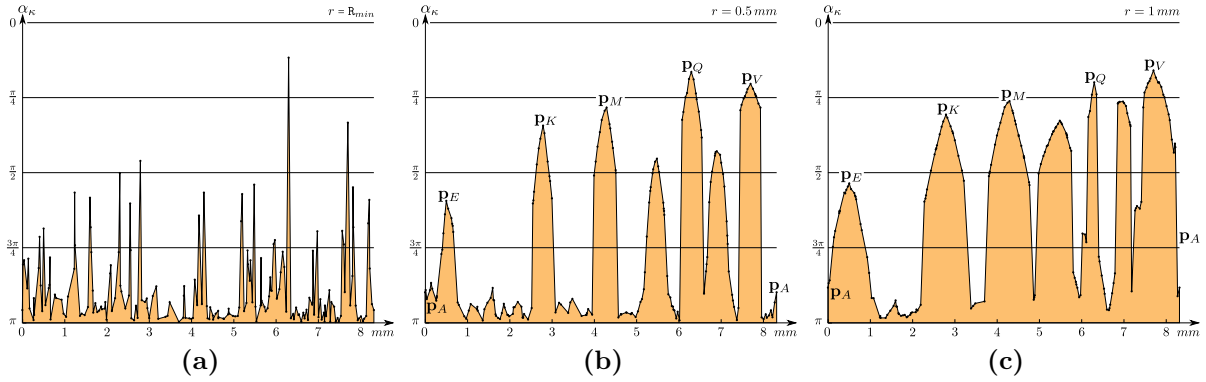


Figure 4.12: Integral invariant $\alpha_\kappa(\mathbf{l}_k)$ computed for the radii (a) $\mathtt{R}_{min} = 0.18\,mm$ (b) $0.5\,mm$ and (c) $1\,mm$. The first and last vertex of $\mathbf{l}_k$ is $\mathbf{p}_A$. Points of interested are marked as $\mathbf{p}_B$ to $\mathbf{p}_V$. The x-axis shows the run-length along $\mathbf{l}_k$.

Figure 4.13 shows the directional run-length integral invariants $\hat{Q}^+$, $\hat{Q}^-$ and their average $\hat{Q}$ as shown in Equation 4.12. The steep filter responses important for corner detection are canceled for $\hat{Q}$, while they are detectable for $\hat{Q}^{+,-}$. The maxima of dull corners like $\mathbf{p}_E$ become more distinct, for larger values of r. Additionally the height of the maxima are virtually constant for distinct corners like $\mathbf{p}_K$, $\mathbf{p}_M$ and $\mathbf{p}_V$.

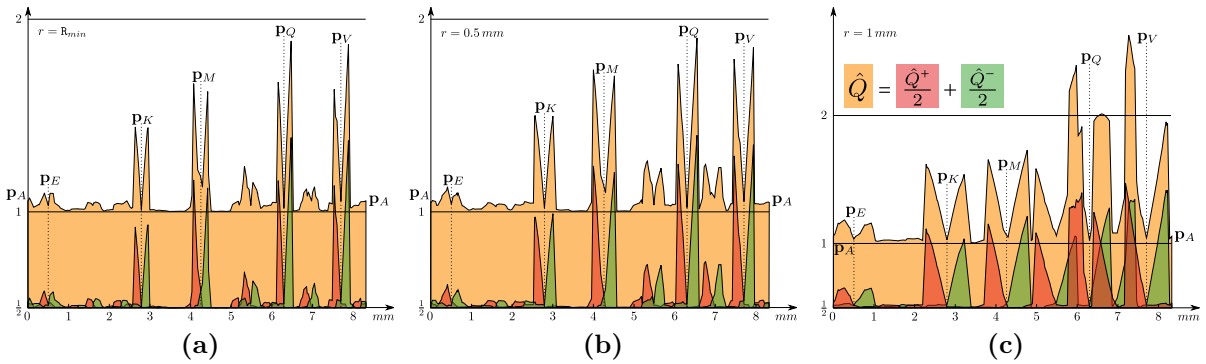
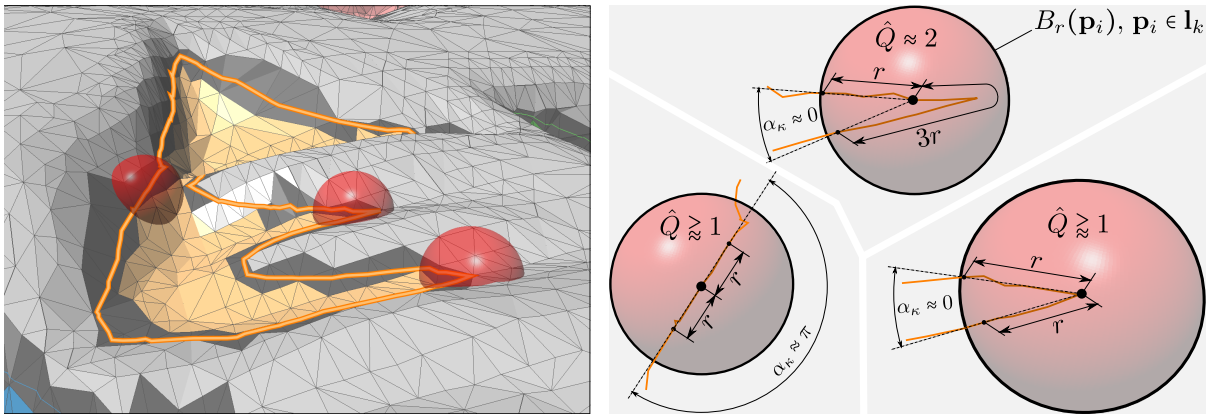


Figure 4.13: Integral invariants $\hat{Q}^{+,-}(\mathbf{l}_k)/2$ in red and green and their sum $\hat{Q}$ for the scales r (a) $\mathtt{R}_{min}$ (b) $0.5\,mm$ and (c) $1\,mm$. The first and last vertex of $\mathbf{l}_k$ is $\mathbf{p}_A$. Points of interested are marked as $\mathbf{p}_B$ to $\mathbf{p}_V$. The x-axis shows the run-length along $\mathbf{l}_k$.

Figure 4.14a shows the polyline $\mathbf{l}_k$ used in previous examples as well as three selected vertices representative for lines and corners of the outlines of cuneiform characters. Figure 4.14b shows these selected vertices and their corresponding integral invariants: $\alpha_\kappa \approx 0$ and $\alpha_\kappa \approx \pi$ having $\hat{Q} \gtrapprox 1$ and $\hat{Q} \approx 2$.



(a) **(b)**

Figure 4.14: Cuneiform outline $\mathbf{l}_k$ (orange) (a) embedded in $\mathcal{M}_2$ with three selected vertices and spheres/balls used to compute the integral invariants. (b) Line segments (black) within the spheres having the extrema $\alpha_\kappa \gtrapprox 0$ and $\alpha_\kappa \approx \pi$ as well as the extrema $\hat{Q} \gtrapprox 1$ and $\hat{Q} \approx 2$.

Summary – List of Points of Interest $\mathcal{L}_\oplus$

This sections shows methods to determine points of interest by computing extrema of curvature measures for 1D-manifolds embedded in $\mathbb{R}^3$. The first method is an estimator $\tilde{\kappa}$, which has to be smoothed to achieve reasonable values $\bar{\kappa}$ for detecting points of interest. Additionally this estimator has a sign indicating concave and convex corners, because a polyline $\mathbf{l} = \partial\mathcal{C}$ is typically embedded in $\mathcal{M}_2$.

The other methods are integral invariants: α_κ is an angle between the two points of intersection of $\mathbf{l}$ with $\mathbb{S}(\mathbf{p}_i)$ and $\mathbf{p}_i$. The maxima of $\alpha_\kappa(\mathbf{l})$ are located at sharp corners and α_κ is equal to the angle β of a corner. The intersection of $\mathbf{l}$ and B_r leads to subset of the polyline having a run-length Q_r. This run-length is an integral invariant, which can be split into two directional run-length based integral invariants Q_r^+ and Q_r^-. The latter have less canceling effects than their average Q_r and therefore lead to better results for corner detection. Furthermore $\hat{Q}$ allows for estimation of the amount of noise distorting a polyline, when a-priori knowledge of the polyline is given.

The greatest benefit is achieved, when all methods are used together. Therefore the points of interest, i.e. sharp corners for cuneiform characters are stored as list of vertices $\mathcal{L}_\oplus$ used to determine the characters' skeletons in the next section.

4.3 Extraction of Skeletons – Decoding Character Elements

The next important step of character extraction is the skeletonization of the connected components shown in previous sections. Therefore the extremal points of the feature function's values $f(\mathbf{p}_i)$ within a connected component $\mathcal{C}$ becomes the list of vertices $\mathcal{L}_\circledast \subset \mathcal{C}$. This list $\mathcal{L}_\circledast$ typically contains points of interest like the deepest point of a wedge, points shared by adjacent wedges and points of wedge intersections.

In case of using correlation based functions like $f(\mathbf{p}_i) \leftarrow R_{i,i,k}$ of Equation 3.117 on page 85, the points of interest are located at the local maxima of $f(\mathbf{p}_i)$. These points can be determined using the 1-ring $\mathring{\mathbf{t}}_i$ and assigned ($\hookleftarrow$) to $\mathcal{L}_\circledast$:

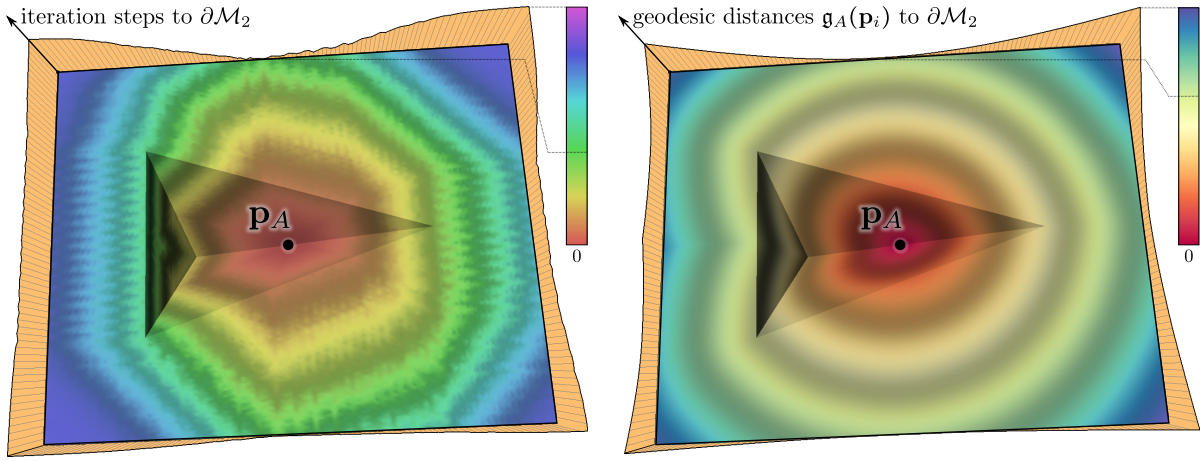
$$f(\mathbf{p}_i) > f(\mathbf{p}_k) \;\forall\; \mathbf{p}_{k\neq i} \in \mathring{\mathbf{t}}_i \begin{cases} \text{true} & \mathcal{L}_\circledast \hookleftarrow \mathbf{p}_i \\ \text{false} \end{cases} \quad \text{and} \quad \mathbf{p}_i \in \mathcal{C} \tag{4.18}$$

The 1-ring is defined in Section 2.2 on page 29.

Then the computation of the lines connecting the points of interest is related to Voronoi-diagrams. The points in $\mathcal{L}_\circledast$ act as seed vertices to compute a skeleton for an area of interest $\mathcal{C}$. In contrast to the skeleton computed using the classic Voronoi-algorithm, the points of interest in $\mathcal{L}_\circledcirc$ located along the border $\partial\mathcal{C}$ are used as the skeleton's end nodes, which are always connected to the nearest seed vertices of $\mathcal{L}_\circledast$. This eliminates false positive branches of the skeletons; it ensures that branches to end nodes have proper length as the branch always ends at the outline and not within the area; and it enables to connect the end nodes of non-convex points of $\partial\mathcal{C}$.

Connecting nearest points within $\mathcal{M}_2$ means tracing the connection between points along the surface. The marching front algorithm is an iterative tracing algorithm shown in Algorithm 10 on page 93. Additionally computing the geodesic distance is a necessity, otherwise the marching front advances arbitrarily, because the vertices are iteratively traced in the arbitrary order of $\mathcal{L}_v$. The arbitrary nature of the marching front is shown in Figure 4.15a in contrast to its extension by ordering the vertices by their geodesic distance shown in 4.15b. Even for this perfectly ordered synthetic data-set the distances – measured in iteration steps – from the seed vertex $\mathbf{p}_A$ to the border of $\mathcal{M}_2$ is unevenly distributed. The visualization of the geodesic distances $\mathfrak{g}_i(\mathbf{p}_A)$ between $\mathbf{p}_A$ and all other vertices $\mathbf{p}_i$ results in a pattern of concentric circles along the surface.

Computing the geodesic distance exponentially increases computing time. However it is possible to limit the computational complexity to $O(n\log(n))$ by approximating the geodesic distance [NK02, Mar06]. In this approach the front is always advanced at the edge $\mathbf{e}_{AB}$ along the marching front with vertices having the shortest geodesic distance. Then the adjacent face $\mathbf{t}_j$ is chosen, which has a vertex $\mathbf{p}_C$ in 1-ring distance to the front. To estimate the geodesic distance $\mathfrak{g}_i(\mathbf{p}_C)$ a triangle $\widetilde{\mathbf{t}} := \{\mathbf{p}_A, \widetilde{\mathbf{p}}_D, \mathbf{p}_B\}$ is assumed, where $\widetilde{\mathbf{p}}_D$ corresponds to the seed vertex $\mathbf{p}_i$. Therefore the lengths of the edges are $|\mathbf{e}_{AD}| = \mathfrak{g}_i(\mathbf{p}_A)$ and $|\mathbf{e}_{DB}| = \mathfrak{g}_i(\mathbf{p}_B)$.



(a) **(b)**

Figure 4.15: Synthetic cuneus with false colors representing (a) the iteration steps from $\mathbf{p}_A$ and (b) the geodesic distances $\mathfrak{g}_A(\mathbf{p}_i)$ to the mesh's border. The curves plotted orthogonal to the surfaces are the distances between $\mathbf{p}_A$ and $\partial\mathcal{M}_2$ using the corresponding metric.

Using the angles of the triangle $\mathbf{t}_j$ at $\mathbf{p}_A$ and $\mathbf{p}_B$:

$$\alpha = \arccos\langle \hat{\mathbf{e}}_{CA}, \hat{\mathbf{e}}_{AB} \rangle \qquad \beta = \arccos\langle \hat{\mathbf{e}}_{AB}, \hat{\mathbf{e}}_{BC} \rangle \tag{4.19}$$

and the angles of $\tilde{\mathbf{t}}$:

$$\tilde{\alpha} = \arccos\left(\frac{|\mathbf{e}_{AB}|^2 + \mathfrak{g}_i^2(\mathbf{p}_A) - \mathfrak{g}_i^2(\mathbf{p}_B)}{2\mathfrak{g}_i(\mathbf{p}_A)|\mathbf{e}_{AB}|}\right) \quad \tilde{\beta} = \arccos\left(\frac{\mathfrak{g}_i^2(\mathbf{p}_B) + |\mathbf{e}_{AB}|^2 - \mathfrak{g}_i^2(\mathbf{p}_A)}{2|\mathbf{e}_{AB}|\mathfrak{g}_i(\mathbf{p}_B)}\right) \tag{4.20}$$

the geodesic distance is estimated:

$$\mathfrak{g}_i(\mathbf{p}_C) = \begin{cases} \mathfrak{g}_i(\mathbf{p}_A) + |\mathbf{e}_{CA}| & \text{when } (\alpha + \tilde{\alpha}) \geq \pi \\ \mathfrak{g}_i(\mathbf{p}_B) + |\mathbf{e}_{BC}| & \text{when } (\beta + \tilde{\beta}) \geq \pi \\ |\mathbf{e}_{CD}| & \text{otherwise} \end{cases} \tag{4.21}$$

with

$$\begin{aligned} |\mathbf{e}_{CD}| &= \sqrt{|\mathbf{e}_{CA}|^2 + \mathfrak{g}_i^2(\mathbf{p}_A) - 2|\mathbf{e}_{CA}|\mathfrak{g}_i(\mathbf{p}_A)\cos(\alpha + \tilde{\alpha})} \\ &\equiv \sqrt{|\mathbf{e}_{BC}|^2 + \mathfrak{g}_i^2(\mathbf{p}_B) - 2|\mathbf{e}_{BC}|\mathfrak{g}_i(\mathbf{p}_B)\cos(\beta + \tilde{\beta})} \end{aligned} \tag{4.22}$$

As $\mathbf{p}_C$ is typically in 1-ring distance of other edges of the marching front, its geodesic distance is computed more than once. Therefore only the smallest value of $\mathfrak{g}_i(\mathbf{p}_C)$ is kept for further computation, because only the length of the shortest path between $\mathbf{p}_i$ and $\mathbf{p}_C$ is of relevance. Typically the geodesic distance is computed three times for each vertex.

Precautions have to be implemented, because of accumulating errors from the estimation as well as numeric errors. These errors lead to edge lengths of $\tilde{\mathbf{t}}$, which do not fulfill the triangle inequality. This can be detected using the result of the fractions in Equation 4.20 as intermediate results. In case the result of a fraction is $1+\epsilon$ or $-(1+\epsilon)$ the angle $\tilde{\alpha}$ respectively $\tilde{\beta}$ has to be set accordingly to $\lesssim \pi$ or $\gtrsim 0$.

Exchanging the explicit vertex stack $\mathcal{L}_{\equiv}$ of Algorithm 10 on page 93 to a heap handling the edges of the marching front ordering them by $\min(\mathfrak{g}_i(\mathbf{p}_A), \mathfrak{g}_i(\mathbf{p}_B))$ allows to compute $\mathfrak{g}$ using previous equations. Additionally this extended algorithm inherits the labeling capability as shown in Figure 4.16a. This is practical as excluded areas like the background $\mathcal{C}_\emptyset$ are treated identical to missing parts of the surface.

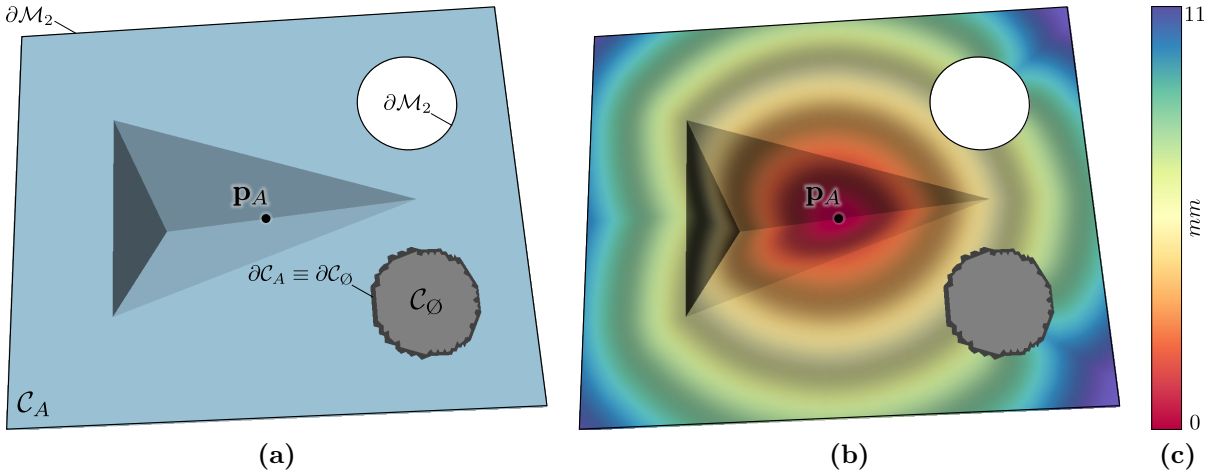


Figure 4.16: Synthetic cuneus (a) labeled and (b) colored by mapping $f(\mathbf{p}_i) \leftarrow \mathfrak{g}_A(\mathbf{p}_i)$ to a (c) color ramp. Labels and geodesic distance are determined at the same time by the improved marching front.

As a useful option e.g. as weight for other methods, the geodesic distance can be assigned as function value. $f(\mathbf{p}_i) \leftarrow \mathfrak{g}_A(\mathbf{p}_i)$ is used in Figure 4.16b by a color mapping method of the *GigaMesh* framework as shown in the Section 4.4.

Multi-Seed Initialization of the Marching Front

Using multiple seed vertices for the marching front finally enables to compute the skeletons of the areas of interest. This is achieved by initializing multiple seed vertices at the same time by adding the edges of the 1-ring neighborhood of the seeds $\mathbf{p}_i$ to the heap. The edges including the seed vertices are not added to the heap as they are de-facto behind the initial state of the marching front. Therefore their edge lengths (euclidean distances) are the initial geodesic distances:

$$\mathring{\mathfrak{g}}_i(\mathbf{p}_A) = |\mathbf{e}_{iA}| = |\mathbf{p}_A - \mathbf{p}_i| \;\forall\; \mathbf{p}_A \in \mathring{\mathbf{t}}_i \tag{4.23}$$

leading to the trivial, but required initialization $\mathfrak{g}_i(\mathbf{p}_i) = 0$ of the seed vertices.

A Voronoi-diagram is achieved, when the edges in the heap contain a reference unique to the seed vertex. This can be the seed's index i or a sequentially generated label id l, which is passed along when the front advances. In case an implementation uses pointers to vertices the label id can be immediately set – determining connected components $\mathcal{C}$. Figure 4.17a shows the Voronoi-diagram and the geodesic shortest paths between the vertices defining the wedge. Figure 4.17b visualizes the geodesic distances.

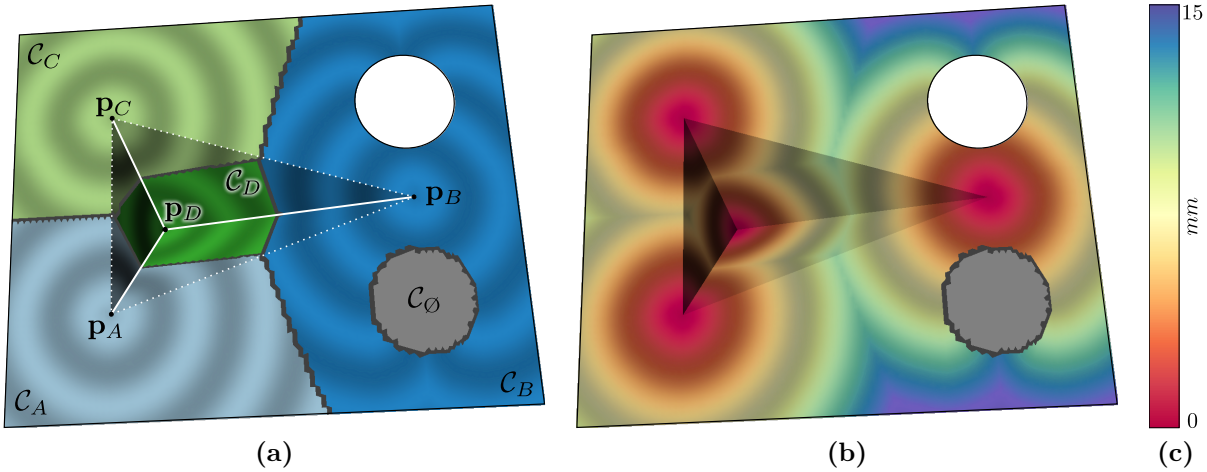


Figure 4.17: Synthetic cuneus with false colors representing (a) the Euclidean distance (b) k-ring based distance (bright to dark means 5000 iteration steps) between the vertices $\mathbf{p}_j$ and the seed vertex $\mathbf{p}_i$ of an application of the marching front algorithm.

Prioritized Line Tracing

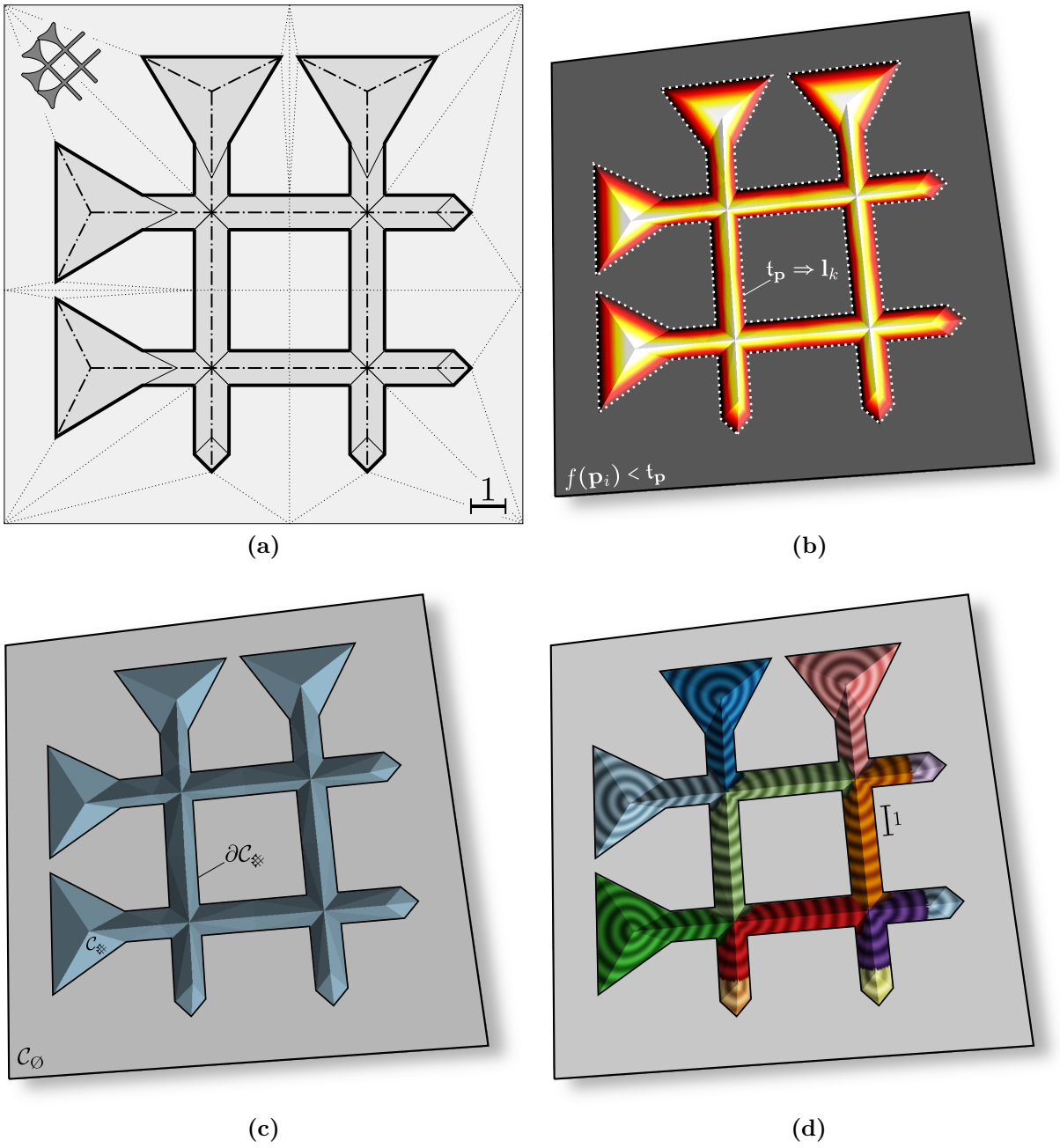
Adding prioritization is a necessity, because skeletonization using Voronoi-diagrams treats the surface equally. This leads to skeletons of characters, which do not correspond to a manual drawing. The reason is that an expert draws in the direction of a wedge and therefore in the same order as the scribe. This applies to other applications dealing with recognition of hand-writings. For example: lines drawn with ink have a decreasing width, which reflects the direction a line was drawn.

Let the first point of a line or wedge be called head – the last point is called tail and let the $f(\mathbf{p}_i)$ represent the correlation $R_{i,j}$ to an ideal head. Then the seed vertices of the candidate points of interest in $\mathcal{L}_{\circledast} \in \mathcal{C}$ can be sorted descending. This allows to start tracing line segments from the heads to the tails within an area of interest:

1. Initialize the first vertex $\mathbf{p}_i$ from $\mathcal{L}_{\circledast}$ and remove it from the list.
2. Advance the marching front using $\mathfrak{g}$ until a vertex $\mathbf{p}_j \in \mathcal{L}_{\circledast}$ or $\partial\mathcal{C}$ is reached.
3. If a vertex $\mathbf{p}_j$ is found, store the directed line segment $\mathbf{e}_{ij} \coloneqq \{\mathbf{p}_i, \mathbf{p}_j\}$.
4. Repeat until all vertices of $\mathcal{C}$ are processed.

The result is a skeleton very similar to a manual drawing including the proper orientation of the lines. It is important that prioritized feature line tracing is applied to each $\mathcal{C} \in \mathcal{L}_C$ at a time, otherwise the fronts of unrelated areas will stop and advance not according to the intended sequence. Finally the points of interest of $\mathbf{l} \equiv \partial\mathcal{C}$ have to be connected with the nearest $\mathbf{p}_j \in \mathcal{L}_{\circledast}$. These line segments to the skeleton's end nodes in $\mathbf{l}$ can be determined using sub-labeling $\mathcal{L}_{\circledast}$.

Figure 4.18 shows the prioritized line tracing for a synthetic 3D-model of the cuneiform sign 𒅆 known as *kaskal*, which consists of four intersecting wedges. Because of its symmetry and multiple intersections of wedges, it is an ideal test-case for prioritized line tracing. The 2D-sketch with the skeleton as dashed-dotted line are shown in Figure 4.18a. While the skeleton requires 16 vertices to describe the wedges and 4 vertices for the intersection, the 3D-model generated of the sketch requires 60 vertices. This number can be reduced down to 48 by using 1 vertex instead of 4 for each of the cuneus' tips. As *kaskal* is not the complexest character known in *Assyriology*, it stresses the *assyriolgists'* demand for high-resolution acquisition and robust processing methods.
Figure 4.18c shows the area of interest $\mathcal{C}_{𒅆}$ surrounded by the background $\mathcal{C}_{\emptyset}$ determined using the function values $f(\mathbf{p}_i)$ visualized in Figure 4.18b with the dotted isoline of the threshold $\mathfrak{t}_{\mathbf{p}}$. The sub-labeled $\mathcal{C}_{𒅆}$ is shown in Figure 4.18d, where the brightness of the sub-labels varies using the geodesic distance to $\mathcal{L}_{\circledast} \subset \mathcal{C}_{𒅆}$. The results for the selected wedges of the example for $\mathcal{C}_k$ of HOS G10 are shown in Figure 5.4 on page 131. Further results for cuneiform tablets are shown in Section 5.2 and Section 5.3. The faces and vertices defining the synthetic 3D-model of *kaskal* is shown in Section A.3 on page 160.



(a) **(b)**

(c) **(d)**

Figure 4.18: Synthetic 3D-model of the cuneiform character *kaskal*: (a) 2D-sketch with the skeleton as dashed-dotted lines. (b) the function values $f(\mathbf{p}_i) \leftarrow R_{i,k}$ and the isoline along the threshold $\mathfrak{t}_\mathbf{p}$. (b) Surface $\mathcal{M}_2$ segmented by $\mathfrak{t}_\mathbf{p}$ into the background $\mathcal{C}_\emptyset$ and the area of interest $\mathcal{C}_\#$. (d) Sub-labeled area showing the advances of the marching front using $\mathfrak{g}$ from the wedges' heads to tails.

4.4 Filter and Visualization Framework

The design of all of the previously shown methods always includes modularity and low computational complexity, because their implementation as algorithms are part of a processing pipeline. This pipeline has to extract meaningful shapes in reasonable time out of imperfect 3D-models consisting of tremendous amounts of vertices connected by triangular faces. Section 2.2 shows the pitfalls for algorithms introduced by common file formats like *Stanford Polygon* (PLY). These pitfalls are prevented by the different algorithms to compute different Multi-Scale Integral Invariants (MSII). Altogether allows to compute feature vectors per vertex without manipulating the acquired surface and therefore preventing the introduction of additional artificial artifacts.

Therefore an example for a basic MSII filtering pipeline is shown in Figure 4.19, as it is used in Section 3.5 to visualize fingerprints and characters on cuneiform tablets. The building bricks of this pipeline compute the feature vectors $\mathfrak{F}_i$ and function values $f(\mathbf{p}_i)$ for the vertices $\mathbf{p}_i \in \mathcal{L}_v$. The reference vector $\mathfrak{F}_k$ can be set using a-priori knowledge e.g. using $0.5 \forall \mathfrak{v} \in \mathfrak{V}_k$ for wedge shapes. Using a *Graphical User Interface* (GUI) is a user-friendly alternative to choose a reference index k from pin-pointing a reference vertex $\mathbf{p}_k$ for the convolution $R_{i,k}$. Finally false colors are assigned to the vertices depending on their function values to render a raster image using OpenGL.

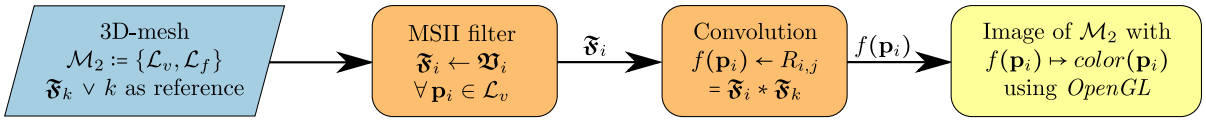


Figure 4.19: Pipeline for feature visualization of a surface $\mathcal{M}_2$ defined by a list of vertices $\mathcal{L}_v$ connected by faces $\mathcal{L}_f$ using the volume based integral invariant $\hat{V}_r$. Therefore the generic feature vector $\mathfrak{F} \leftarrow \mathfrak{V} := (\dots, \hat{V}_r, \dots)$. The reference vector $\mathfrak{F}_k$ for the correlation $R_{i,k}$ is either based on a-priori knowledge or selected using a GUI.

The pipeline becomes significantly longer, when a vector representation is required using skeletons and outlines of connected components $\mathcal{C}$. Figure 4.20 shows this workflow using the combined convolution $R_{i,i,j}$ and a threshold $\mathfrak{t}_\mathbf{p}$ to determine the labels $\mathcal{C}_k$ as shown in the previous section. This includes the determination of points of interest $\mathcal{L}_\circledast \subset \mathcal{C}_k$ and $\mathcal{L}_\textcircled{1} \subset \partial\mathcal{C}_k$. A similar pipeline is common in (2D) image processing and pattern recognition [RP66]. However, to discard false-positive $\mathcal{C}$ as shown in Section 4.1.1 on page 97 adds the surface based integral invariant filter; its convolution and labeling; and a binary operation for excluding areas of non-interest.

Having four or more building bricks – each of them providing variations of methods – requires a framework to easily arrange and configure a pipeline for a given task. This enables the use of the presented methods for very different applications to extract any kind of feature. Therefore all the algorithms of this thesis have been collected in the *GigaMesh* framework named after a suggestion by Susanne Krömker: The name *GigaMesh* indicates the capability to process meshes of large size and adding the letter "l" leads to the name of the ancient king *Gilgamesh* described in an epic found on cuneiform tablets [Mau05].

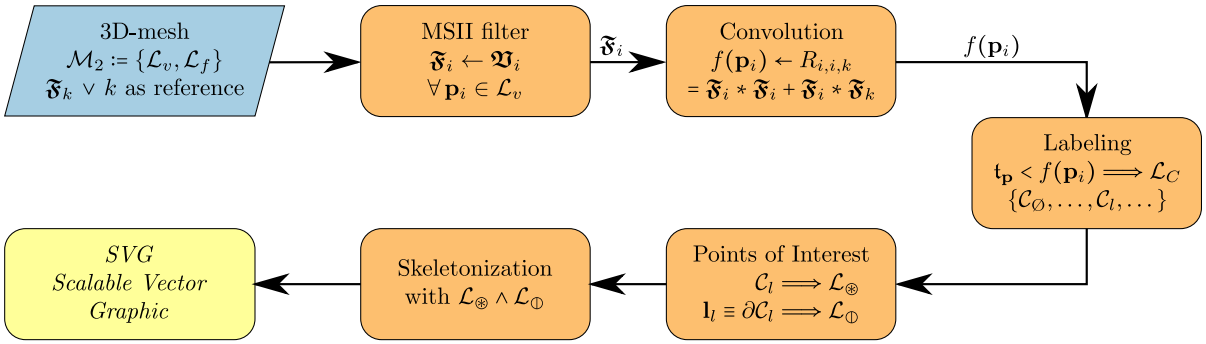


Figure 4.20: Extended pipeline of Figure 4.19 for computing a vector drawing of cuneiform characters. Thresholding by $\mathfrak{t}_\mathbf{p}$ results in areas of interest $\mathcal{C}_l$ and their boundaries $\partial\mathcal{C}_l$. $\mathcal{L}_\circledast \subset \mathcal{C}_l$ and $\mathcal{L}_\textcircled{1} \subset \partial\mathcal{C}_l$ are the points of interest defining the elements of a character.

To process large meshes, the framework's classes for geometric primitives are tailored to minimize memory usage and maximize performance as shown in the next Section. It can be used with a GUI enabling user interaction on desktop computers and it can be used in text terminals for highly automated tasks on compute servers. This is achieved by a layered concept shown in Section 4.4.2.

4.4.1 Memory Management and Parallel Processing

Algorithms for huge data-sets are strongly influenced by the two main properties of computers main memory: it is available only in a limited quantity and the access via the memory bus is dramatically slower than computing operations. Additionally memory access by parallel processes has further influence on the design of algorithms and their performance. Depending on the type of data additional overhead is introduced by controlling access to the shared memory. *GigaMesh* uses *semaphores* [PH12] to control the memory access for the parallelized algorithms. For example data being read-only can be accessed in the same manner as serial processing.

Parallel processing of the Multi-Scale Integral Invariants was easily enabled, because the memory access is read-only and the feature vectors $\mathfrak{F}_i$ as well as the resulting function value $f(\mathbf{p}_i)$ are written only once and have no influence on the results computed for any other vertex $\mathbf{p}_{j\neq i}$. As computing the MSII's is the computational costliest part of the processing pipeline the performance increases linear by the number of processors available. Therefore all the methods presented in Chapter 3 are parallelized within the *GigaMesh* framework using *POSIX-threads* [But97]. The marching front derivatives used throughout this chapter can be used by parallel processes without modifications and are so-called thread-safe.

This leaves the question how many properties and intermediate results for a geometric primitive gives the optimal balance between computing time, time for memory access and memory usage. The arising discussion is often suppressed citing *Moore's law* [Moo65], which states that steadily more and more computing resources become available. Interpreting this law for project planing is a good practice to make an efficient use of resources

available, when the project is finished and not those when it begins. The illicit interpretation is that one just has to wait to solve the question about memory usage and computing power, which will let one to wait forever, because *Moore's law* applies to the data-gathering devices, like a 3D-scanner, too.

Memory Usage per Geometric Primitive

Having in mind that *Moore's law* will change the absolute numbers of the following estimation, let us compute the maximum amount of memory allowed per geometric primitive using the 3D-models and computers available of the time, when this thesis was written. The 3D-models acquired in the *Vorderasiatische Abteilung der Berliner Museen* (VAM) for the *Heidelberg Assyriology* have $|\mathcal{L}_v| \approx [5, 10] \cdot 10^6$ vertices. Larger objects' 3D-models like the relief of *Scheschonq I.*[1] [Feu81] of the *Heidelberg Egyptology* have $|\mathcal{L}_v| \approx 25 \cdot 10^6$ vertices despite using a larger FOV of the *Breuckmann* 3D-Scanner resulting in a lower spatial resolution than used for the much smaller cuneiform tablets.

Regarding the *Euler characteristic* shown in Equation 3.15 on page 49 the total number of explicitly given primitives for 3D-models is:

$$|\mathcal{L}_\mathcal{M}| = |\mathcal{L}_v| + |\mathcal{L}_f| \approx 3 \cdot |\mathcal{L}_v| \tag{4.24}$$

Using implicitly given primitives like edges doubles $|\mathcal{L}_\mathcal{M}|$ as there are three orientated edges per face. To prevent the long access times of secondary memory (hard-disk drives) either $75 \cdot 10^6$ or $150 \cdot 10^6$ primitives should fit within a computer's primary memory. State-of-the-art desktop computers and entry level servers typically have 24 *GigaByte (*1024^3 *byte)* (GB) of primary memory, which leaves 343 byte per explicit primitive respectively 171 byte for all three types of primitives. Therefore only explicit primitives are used within *GigaMesh*.

Because on today's 64-bit system the smallest block of memory for any stored value is 8 byte leads to a maximum of 42 storable values per vertex and face. Practically this number is even lower as several 8 byte blocks are deducted due to the overhead of a computer's *Operating System* (OS). For example the internal identification of *C++* classes requires at least 8 byte adding 8 byte, when it uses *virtual* methods [Str00].

Summarizing these considerations about memory usage lead to a rule of thumb: For each acquired point of a surface a total of 1 000 bytes of memory are allocated to store the vertex's position, color, surface normal, feature vector, adjacent faces, etc. In even shorter terms: millions of measuring points in $\mathcal{L}_v$ require $|\mathcal{L}_v|$ *GigaByte* of primary memory.

[1] Thanks to Dr. Dina Faltings

Memory Optimization with Bit Arrays as Sequence Containers

As a consequence of using less than 343 byte per primitive, even well-established sequence containers like the *C++* `std::vector` have to be avoided and arrays preferred, because already an empty container requires e.g. twice the memory of an array for a position vector. The `boolean` data-type has the least memory usage as it will allocate 63 unused bits. Data types like `char` and `short integers` have a better usage and may be automatically optimized, when compiled. `Double` precision floating point numbers and `long integers` are the only ones using 64 bits at a time. Therefore *GigaMesh* internally uses `double` precision for elements of vectors like the position $\mathbf{p}$.

In theory bits acting as flags could be avoided. Nevertheless they are useful to explicitly tag properties of primitives, which are computational expensive to determine e.g. tagging singular vertices as shown in Section 2.2 on page 29. Gathering the list of face $\mathring{\mathbf{t}}_i$ requires to search all faces $\mathbf{t}_j \in \mathcal{L}_f$ for a reference to $\mathbf{p}_i$. Algorithms requiring this information for all vertices lead to:

$$O(|\mathcal{L}_v| \cdot |\mathcal{L}_f|) \approx O(2 \cdot |\mathcal{L}_v|^2) \equiv O(n^2) \tag{4.25}$$

An alternative would be to store this information in an array of pointers, having an arbitrary length. This requires a minimum of 2 bytes for the pointer to the array and an `integer` for its length. The actual size of the array depends on the number of faces a vertex belongs to. Having typically ≈ 6 faces in a 1-ring means ≈ 8 extra bytes per vertex to determine a vertex's singularity tag, which can be represented by just 1 bit. Having 343 bytes, the use of such an array wastes $\approx 3\%$ of memory, while the single bit in a bit array requires a fraction of a thousandth of the available memory: $1/(343 \cdot 8) = 0.4‰$. Additionally the computational complexity for the example of determining singularity in Equation 4.25 is reduced to $O(n)$ having the flag pre-determined for all vertices $\mathbf{p}_i \in \mathcal{L}_v$.

Therefore each primitive class has one 64 bit general purpose array as a member, which allocates the smallest block of memory possible. For the algorithms represented in this thesis 10 bits of this bit array were used leaving plenty of space for future extensions. As these bits are merely intermediate flags, they are sufficiently covered in the *GigaMesh*'s documentation. The introduction of bit arrays to the *GigaMesh* framework was a major optimization for memory usage. The following sections shows the additional performance optimization, which was achieved by exchanging the well-established sequence containers with bit arrays.

Performance Optimization using Bit Arrays

Exchanging sequence containers like the *C++* `std::set` with bit arrays leads to an additional dramatic performance improvement: Practically analyzing the actual time to compute MSIIs lead to the conclusion, that a vast fraction of the total processing time is contributed by the the marching front Algorithm 10 on page 93. At a first glance this algorithm appears to have no room for improvement, because its complexity cannot further be reduced. However computational complexity is not the only factor determining computing time.

Closer investigation showed that most of the time was spent by the sequence containers' operations `push`, `pop` and `insert` used for the stack $\mathcal{L}_{\text{▤}}$ and the surface patch $\mathcal{P}_i$. As sequence containers add overhead in terms of memory the same is expected regarding computing time. Because these containers typically wrap around arrays the overhead for performance is virtually not existing compared to using an array. The low performance is due to resizing the array, when elements are added and removed, which requires time consuming interaction between the processor and the primary memory.

Therefore using the vertices' bit arrays to actually tag them `visited` instead of organizing them in a list dramatically reduces computing time and at the same time it compromises parallel processing. To ensure thread-safety, the bit representing the `visited` tag has to be moved out of the vertex class into a bit array accessible to one thread at a time. This replaces the sequence container of $\mathcal{P}_i$ by a bit array $\mathtt{P}_i$ having $|\mathcal{L}_v|$ bits to decide if a vertex of $\mathcal{L}_v$ was `visited` or not. $\mathtt{P}_i$ is represented by $\lceil |\mathcal{L}_v|/64 \rceil$ blocks $\mathtt{p}_k$ of memory e.g. using an *unsigned long integer*.

Accessing the `visited` flag of a vertex $\mathbf{p}_j$ is achieved by fetching memory block $\mathtt{p}_k$ with an index $k = \lfloor j/64 \rfloor$. Then the bit with index m of a reference block `r` has to be set:

$$\mathtt{r} \leftarrow 2^m \quad \text{with} \quad m = j - k \tag{4.26}$$

and compared using the logical operator *or*:

$$\mathtt{visited} = \mathtt{p}_k \vee \mathtt{r} \tag{4.27}$$

The fastest implementation of 2^m is to set the least significant bit of the block `r` and apply a left-shift. Using *C++* the block `r` can be reduced and the previous two equations become:

$$\mathtt{p[k]} \ \mathtt{|=} \ \mathtt{1} \ \mathtt{<<} \ \mathtt{m} \quad \text{with} \quad \mathtt{k} = \mathtt{floor}(j/\mathtt{64.0f}) \quad \text{and} \quad \mathtt{m} = j - \mathtt{k} \tag{4.28}$$

Furthermore all the lists keeping track of a binary property of a geometric primitive can be exchanged by a bit array – including the stack $\mathcal{L}_{\text{▤}}$. Using two bit arrays for $\mathcal{P}_i$ and $\mathcal{L}_{\text{▤}}$ for the marching front increased the computing speed by a factor in the hundreds and thousands. Additionally all the marching front algorithm's derivatives performance benefit from using bit arrays.

4.4.2 Layered Concept for Data Handling, Filtering and User Interface

The modular processing pipeline allows for adaption of the workflow for feature extraction depending on the application. As modules like the *MSII filter* provides several variants to compute integral invariants leads to a large number of combinations. Hard-coding all these combinations is as impracticable as asking an user to implement their own pipeline. The compromise is to provide an user interface including visualization of intermediate results.

Therefore *GigaMesh* got a base layer of classes and methods as described previously, which can be used to assemble a tool-chain for command line use on compute servers. To give a visual feedback of final and intermediate results an *OpenGL* layer was added. On top of this second layer a GUI using *Trolltech*'s *Qt*, because it is available on all major platforms under the GNU *General Public License* (GPL) – an *Open Source* license.

Layer I: Processing Methods

This layer contains all the algorithms to compute integral invariants as well as means to determine irregularities shown in Section 2.2. In spite of the integral invariant algorithms being robust against irregularities and missing data (holes), the results improve, when faulty parts of a mesh are repaired. A mesh is typically repaired in a first step by removing faulty primitives.

Removing primitives of a mesh is related to the morphological operation erosion. Section 2.2 on pages 25ff. shows different types primitives to be eroded, because they are considered irregular. Each type of irregularity requires a different erosion method. The application of an erosion method can introduce new irregularities e.g. removing a non-manifold edge can lead to a singular vertex. Another common case is the removal of such singular vertices, which leads to new singular vertices within the 1-ring neighborhood.

This means that the erosion methods have to be applied repeatingly and their order has an influence on the amount of removed primitives. As this number has to be kept to a minimum the *Layer I* of *GigaMesh* provides an highly automated mesh polishing algorithm, which fulfills this demand. Erosion additional leads to a segmentation of the mesh $\mathcal{M}_2$ into connected components $\mathcal{C}$, which are either a meaningful part of the object or noise. The parts introduced by measuring errors are typically smaller than the meaningful components.

Therefore the connected components labeling as shown in Section 4.1.1 on pages 94ff. is applied and the surface area $|\mathcal{C}_l|$ of each connected $\mathcal{C}_l$ is computed. Then the components are sorted by their area:

$$|\mathcal{C}_1| \leq |\mathcal{C}_2| \leq \cdots \leq |\mathcal{C}_n| \tag{4.29}$$

Next the index i is determined as there typically exists:

$$|\mathcal{C}_i| \lll |\mathcal{C}_{i+1}| \tag{4.30}$$

and leads to a threshold $\mathfrak{t}_{\mathcal{C}} > |\mathcal{C}_i|$ to remove small surface components introduced by noise, which are shortly denoted as labels $\mathcal{C}_{i<t}$.

Removing singular vertices has to applied before removing $\mathcal{C}_{i<t}$, because the erosion of singularities typically increases the number of labels $\mathcal{C}_{i<t}$. The same can happen, when faces along non-manifold edges are removed. Before these are removed, sticky faces have be removed as very first, because they introduce a sub-set of non-manifold edges. This leads to a sequence of erosion operations, which has to be repeated until the number of vertices $|\mathcal{L}_v|$ and faces $|\mathcal{L}_f|$ of a 3D-model does not decrease anymore:

1. Remove sticky faces and face with zero area shown on page 32.
2. Remove non-manifold faces shown on page 30.
3. Label all vertices as shown using Algorithm 12 on page 95 $\rightsquigarrow \{\dots, \mathcal{C}_l, \dots\} \wedge \nexists \mathcal{C}_\emptyset$ without considering the function value $f(\mathbf{p}_i)$, e.g. using $\mathfrak{t}_{\mathbf{p}} = +\infty$.
4. Compute the area $|\mathcal{C}_l|$ of all labels $\mathcal{C}_l$ using Equation 2.7 on page 27.
5. Remove all labels $\mathcal{C}_{i<t}$ having an area less than $\mathfrak{t}_{\mathcal{C}}$.
6. Remove solo vertices $\mathcal{L}_{v_s}$ shown in Equation 2.20 on page 28.
7. Remove singular vertices – shown on page 29.

Afterwards the eroded mesh becomes a so-called clean mesh. Note that in most of the cases of clean meshes, there will be holes $\partial\mathcal{M}_2$. Therefore these holes have to be filled, which can be achieved by dilation as shown in [Lie03]. Due to numeric errors the dilation can re-introduce faces with zero area and other irregularities, the erosion and dilation have to be repeatingly applied to a mesh until no more holes are present:

1. Erode the mesh until it is clean as shown above.
2. Determine a list of boundaries $\partial\mathcal{M}_2 \coloneqq \bigcup_{i=1}^{n} \partial\mathcal{C}_i$ of the holes.
3. Remove boundaries $\partial\mathcal{C}_i$ of large connected components with $|\mathcal{C}_i| > \mathfrak{t}_{\mathcal{C}}$ from the list.
4. Fill the remaining holes $\partial\mathcal{C}_{i>t}$ within the list.

The result is a so-called perfect mesh, which provides the best numeric results in respect to the quality of an acquired data-set. Because even a perfect mesh will have boundaries $\partial\mathcal{M}_2$ the border treatment of the algorithms to compute integral invariants stays relevant for robust processing. However, when obeying the described order, it is guaranteed that the remaining boundaries are kept as far as possible from the areas of interest.

Layer II: Rendering with OpenGL

The *Open Graphics Language* (OpenGL) is the natural choice for visualizing meshes, because the immediate rendering commands accept the elements of the list of vertices $\mathcal{L}_v$, the list of faces $\mathcal{L}_f$ and additional information like normals and colors. Because OpenGL has a strong bias towards synthetic data as used in *Computer Aided Design* (CAD) and games the immediate rendering commands have an extremely low performance, when large numbers of elements have to be drawn. One solution would be to implement *Level of Detail* (LOD) methods, which adds computational complexity to the mesh handling and increases memory usage. This can be avoided by using the OpenGL's *Vertex Buffer Objects* (VBOs) [MB05] designed for larger data-sets.

The use of VBOs allows for uploading the triangular mesh's data (vertices, faces, colors, normals) for non-immediate-mode rendering. Therefore their use has a substantial performance benefit, because the data is located in the video device's memory. Additional the VBOs are bound to an array allowing for incremental updates, which enables efficient updates as well as a modular use.

The majority of the *GigaMesh*'s algorithms operate on one out of the three main components describing a mesh:

- Geometric properties represented by position vectors $\mathbf{p}_i$ and normal vectors $\mathbf{n}_i$,
- Connectivity representing the faces $\mathbf{t}_j$ and
- Color used to map function values, labels or 3D-model's true-color.

Two VBOs are used for the first two components. The three different types of information represented by color are stored in another three VBOs. The VBOs for colors could realized by a single VBO, but as these arrays add only four byte per vertex for red; green; blue; and transparency, it is performance-wise more convenient to separate them. The use of VBOs allows an user to inspect even the largest 3D-models with a reasonable number of frames per second.

Because not all information present for a mesh has to be uploaded to the graphics card for visualization and the card's efficient memory organization, a graphics card with 1 GB of memory is a good match for a computer having 24 GB of main memory. This can be illustrated by the same example of today's large meshes as shown on page 117. The example mesh having $25 \cdot 10^6$ vertices requiring a total upload of $1000 \cdot 10^6$ bytes to the video device's memory:

- Three 4-byte `float` numbers for the vertices' positions:
  $25 \cdot 10^6 \times 3 \times 4 = 300 \cdot 10^6$ bytes.
- Three 4-byte `integer` numbers for describing the faces:
  $50 \cdot 10^6 \times 3 \times 4 = 600 \cdot 10^6$ bytes.
- Four 1-byte `short` numbers for the color and transparency:
  $25 \cdot 10^6 \times 4 \times 1 = 100 \cdot 10^6$ bytes.

The handling of the VBOs is event-driven, which means that buffers are created in response to user actions and destructed when the underlying data changes. Because data changes occur mostly on user actions a destruction of a buffer typically triggers an immediate (re-)construction. However binding the destruction to data changes instead of user actions automatically ensures that the visualization is always representing the correct state of a mesh, because the visualization methods are forced to construct the required VBO, when it is not present.

An essential part of visualizing numeric results stored in the multi-purpose variable $f(\mathbf{p}_i)$ is the choice of the color ramp. The simplest representation of these function values is linear mapping of the values to a gray scale. Adding color allows to show much more details. For example the so-called *hot* map using the colors red and yellow between black and white reveals much more feature details, when used with correlation of MSII feature vectors.

Another common color ramp is based on the *hue, saturation and value* (HSV) color model, which is easy to implement, but often very inconvenient to interpret. This arises the question about proper color schemes, which are easy to interpret and maintain a sufficient amount of details within the areas of interest. Furthermore the color choice can get more difficult, when colors have to be e.g. printer friendly or colorblind safe.

Visualization techniques in modern cartography offer schemes for such purposes. Quoting [Bre94] there are three groups of schemes:

- Sequential schemes are suited to ordered data that progress from low to high. Lightness steps dominate the look of these schemes, with light colors for low data values to dark colors for high data values.
- Diverging schemes put equal emphasis on mid-range critical values and extremes at both ends of the data range. The critical class or break in the middle of the legend is emphasized with light colors and low and high extremes are emphasized with dark colors that have contrasting hues.
- Qualitative schemes do not imply magnitude differences between legend classes, and hues are used to create the primary visual differences between classes. Qualitative schemes are best suited to representing nominal or categorical data.

Examples for these groups of schemes are available via the so-called *ColorBrewer* in print [BHH03] and via the Internet[2]. The *ColorBrewer* schemes used for the majority of figures in this thesis are: *Spectral*, *RdGy*, *RdYlGn* for $f(\mathbf{p}_i)$ and *Paired* for connected components $\mathcal{C}_k$ together with an additional light gray color representing the background $\mathcal{C}_\emptyset$. For height related values of $f(\mathbf{p}_i) \leftarrow z$ a color ramp typically used for depicting ranges of elevation in geographic maps is used in Figures 3.6 to 3.9, 5.3d on pages 51–55, 129. Such color ramps are also known as *Hypsometric tints*.

[2] http://colorbrewer.org – last checked 29th February 2012

Layer III: Graphical User Interface

Automation of the workflow to process specific objects in large quantities like cuneiform tablets is achieved using the first and second layer. A GUI allows for altering and experimenting with the workflow for unique objects and new applications. This requires a dynamic interface to follow and influence the data-flow. In other terms a technique similar to event-driven programming using signals is the best choice. This technique is offered by libraries such as *Trolltech*'s *Qt* [BS08], GTK+ [Kra07] or *Boost* [Sch11]. *Qt* was chosen for *GigaMesh*, because the included and therefore highly *Integrated development environment* (IDE) allowed for rapidly prototyping the GUI.

As the lower layers of *GigaMesh* provide all the means for an event-driven GUI, the menu structure provided to the user reflects the processing pipeline shown in Figure 4.20 extended by controls for the visualization itself and means for further inspection of the results:

1. Import of meshes $\mathcal{M}_2$ including pre-computed feature vectors $\mathfrak{V}_i$, function values $f(\mathbf{p}_i)$ and label $l_\mathcal{C} \rightsquigarrow \mathcal{L}_C$ related data. Intrinsic support for exporting the same data.
2. Visualization options to separately inspect geometric primitives as well as projection settings of the virtual camera.
3. Selection of primitives for partially processing and inspecting. The selection can be done using properties like $f(\mathbf{p}_i)$ or $\mathcal{C}_l \in \mathcal{L}_C$ as well as arbitrary choices by the user.
4. Erosion and dilation methods extended by the possibility to split the mesh and/or apply transformation operations e.g. Equation 3.24 on page 52.
5. Compute methods for MSII shown in Chapter 3 and metrics for feature vectors as shown in Section 3.5.1 on page 81.
6. Determination of areas of interest $\mathcal{C}_l$ using either a user selection or an automated selection by function value(s). Seed vertices can be limited to suppress false-positives.
7. Surface related metrics like the Geodesic distances $\mathfrak{g}$, Euclidean distances to primitives, and the determination of extremal values of $f(\mathbf{p}_i)$.
8. Additional methods: unwrapping [BKMK10] using developable and non-developable surfaces e.g. as shown in Figure B.14 on page 178.
9. Selection of color schemes, which is assisted by a Histogram plot and configurable color mapping e.g: linear, equalized, logarithmic and cut-off using the quantile.
10. Export of true-to-scale high-resolution 2D raster images computed using tiled rendering. Additionally vector drawings can be added as an overlay.

All these methods are found from left to right within the main menu on top of the GUI. The export of the results is found together with the import options as `File` menu is found on the left hand side in accordance to the principle layout common for all software packages. Sub-modules of algorithms like the different erosion methods used to clean a mesh are found in sub-menus.

4.5 Summary and the Framework's Overall Workflow

This chapter is about "putting it all together". First the vertices are put into connected components representing the areas of interest. This is achieved by computing the geodesic neighborhood using the marching front algorithm. In other terms the manifold is segmented into labeled areas related to characters. False-positives are neglected using masks determined by additional types of integral invariants. All areas without label id are considered as background $\mathcal{C}_\emptyset$ and discarded from further processing.

Labeling
$\mathfrak{t}_\mathbf{p} < f(\mathbf{p}_i) \Longrightarrow \mathcal{L}_C$
$\{\mathcal{C}_\emptyset, \ldots, \mathcal{C}_l, \ldots\}$

As second step to extract meaningful data out of the high-resolution data, the points of interest defining a character are determined: The borders of the areas of interest are analyzed for extremal points, because three quarter of the points defining a wedge are located along its border. This is achieved by adapting integral invariants for 1D-manifolds $\mathcal{M}_1$ in $\mathbb{R}^3$, which is a robust detection of sharp corners with an angle less than $\pi/2$. Furthermore a signed curvature estimator is shown to distinguish between concave and convex corners.

Points of Interest
$\mathcal{C}_l \Longrightarrow \mathcal{L}_\circledast$
$\mathbf{l}_l \equiv \partial\mathcal{C}_l \Longrightarrow \mathcal{L}_\textcircled{1}$

In the third step the extremal points of the surface's function value, representing the points of interest within the areas of interest are determined. Together with the points of the borders, these are used to reduce the areas to skeletons using a Voronoi inspired algorithm: The marching front and the geodesic distance is used to sub-segment the areas $\mathcal{C}_l$. The points of interest within an area are connected to each other. In contrast to the traditional Voronoi algorithm, the seed vertices are introduced by their priority i.e. correlation to a wedge and the end-nodes of the skeleton are extended to the points of interest along the border.

Skeletonization
with $\mathcal{L}_\circledast \wedge \mathcal{L}_\textcircled{1}$

Finally all the methods are organized in a processing pipeline. Its minimalistic version relies on the methods shown in Chapter 3 and allow for feature visualizations. This pipeline is extended by the previous three steps to extract a meaningful vector representation. All the required methods – having several variations – are collected in the base layer of the *GigaMesh* framework, which is designed for efficient use of a computer's primary memory as well as for performance, e.g. by using parallel computing. Furthermore erosion methods were added to remove faulty primitives introduced by measuring errors and a dilation method to reconstruct missing parts of the surface. The middle layer handles the visualization using OpenGL allowing to apply various color schemes for the different compute methods applicable to a surface. A *Graphical User Interface* (GUI) is provided by the top layer, which allows for exploration of additional variants of the processing pipeline.

Erosion
Removal of Irregularities

Dilation
Filling Holes $\partial\mathcal{C}_i \subset \partial\mathcal{M}_2$ with $|\mathcal{C}_i| < \mathfrak{t}_\mathcal{C}$

Repair methods.

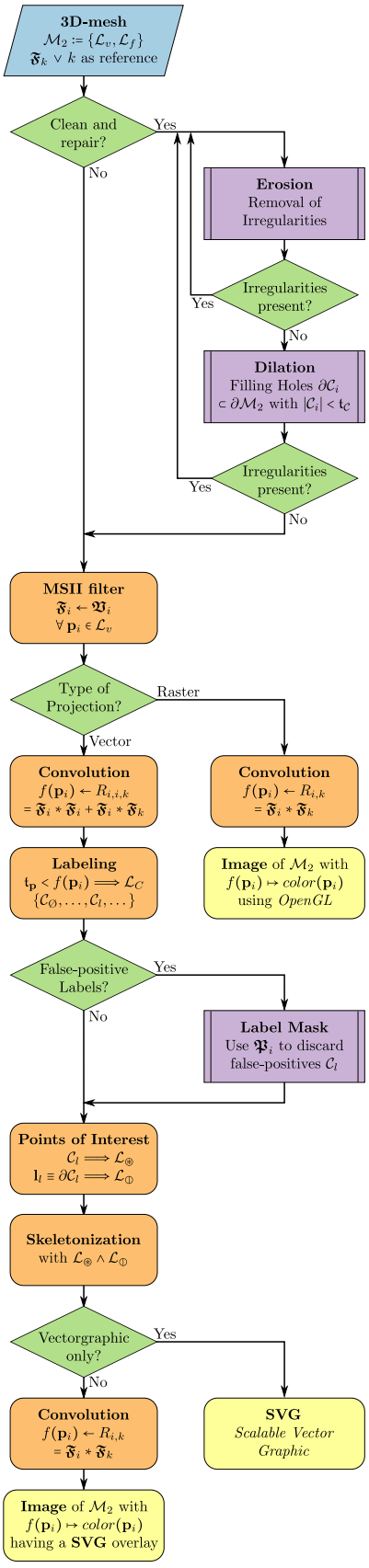
The next page shows the overall workflow using the *GigaMesh* framework. Results of processing 3D-models of cuneiform tablets and the extraction of characters are shown in the next chapter.

The Figure on the right hand-side shows a detailed version of the processing pipeline, which includes the minimal pipeline shown in Figure 4.19 and its extended version shown in Figure 4.20. Before the Multi-Scale Integral Invariants are computed it is possible to automatically repair an acquired mesh lowering the numeric errors.

Depending on the task either raster images for an autopsy are rendered or – following the extended pipeline – vector drawings are computed. The latter allows to add label mask(s) e.g. using the surface based integral invariant $\mathfrak{P}_i$ to discard false-positives as shown on page 97. Afterwards the list of points of interest $\mathcal{L}_{\textcircled{\scriptsize I}}$ along the polyline $\mathbf{l}_l$ is determined. This list contains the end-nodes of the skeleton having the vertices with extremal values in $\mathcal{L}_{\textcircled{\scriptsize I}}$ as seeds ordered by their function values.

Finally the vector graphics can be exported as an overlay in a *Scalable Vector Graphics* (SVG) file for a raster image using a 2D-projection. The projection is typically, but not limited to be true-to-scale and orthographic. Objects with rotational symmetry can be unwrapped first using either a sphere or a cone as supporting surface. Methods independent of the workflow, which can be applied at any given time like the projection are not shown in this figure. Such combined graphics are equivalent to manual drawings with underlying photographs.

For further digital processing the skeletons, outlines and points of interest can be exported in 2D and 3D, e.g. for *Optical Character Recognition* (OCR) or paleographic analysis. This leads to a representation using splines and a normalized minimal representation. Both are shown in the next chapter.



GigaMesh flowchart.

Chapter 5

Results

In the previous section the areas of interest i.e. determination of connected components is leading to the final step of character extraction. Therefore the points of interest $\mathcal{L}_{\textcircled{\scriptsize I}}$ within a component and along its border are connected. The result is an abstract representation: The points $\mathcal{L}_{\textcircled{\scriptsize I}}$ of the border are used to describe the characters with parametric curves. *Scalable Vector Graphicss* (SVGs) were chosen, because these documents are described by the *eXtensible Markup Language* (XML) [Ray03] and the choice of *Bézier splines* is intrinsic as these parametric curves are included in the SVG's standard [Eis02]. SVG is an open standard and it is provided by *World Wide Web Consortium* (W3C).

While the $\mathcal{L}_{\textcircled{\scriptsize I}}$ are predefined points ("knots" or "cusp-nodes") of the spline, the control points are determined using the points of interest $\mathcal{L}_{\circledast}$ within a connected component. This is achieved using a method inspired by *Delaunay triangulation* and the geometric dual *Voronoi diagrams* to connect $\mathcal{L}_{\circledast}$ to $\mathcal{L}_{\textcircled{\scriptsize I}}$. The resulting skeleton and its branching nodes are the control points of the spline. The *Bézier splines* are an overlay to the raster images similar to manual vector drawings having a photograph as base layer. Computing the raster images is shown in the first section of this chapter.

Because the aim of character extraction is *Optical Character Recognition* (OCR), an even more minimal and normalized representation is required and shown. It consist of two pairs of vectors. The major pair of vectors describe the position of a wedge and its principal direction. The minor pair describe the head of a wedge, which is of importance to distinguish between a the two principle types of wedges: the common cunei (e.g. ⊢) and the *Winkelhaken* (e.g. ◀).

Finally the accuracy and performance of the proposed algorithms to determine integral invariants is evaluated. The accuracy with respect to multiple scales is shown for the surface based integral invariant and the volume based integral invariant alike. Additionally this survey is conducted using meshes of the same object having different resolutions. Because the *GigaMesh* framework is tailored to process large quantities of real world data, its overall performance of the mesh-processing pipeline is shown. This overall performance relates to a computational complexity of less than $O(n_v \log n_v)$, which was measured for the hundreds of processed objects having large numbers of n_v vertices.

5.1 MSII Feature Visualization – Raster Images

Having a robust algorithm to compute Multi-Scale Integral Invariants for triangulated irregular meshes is the first step to extract features like imprinted characters on arbitrary surfaces. As feature extraction depends on the MSII filter responses, visualization of these responses as colors to the surface enables rapid verification by an autopsy. The simplest means using the *GigaMesh* framework is to assign e.g. an integral invariant for each scale to the general purpose variable, which is mapped to a color-ramp. Six examples of such raster images with such mappings are shown in Figure 5.1, which are orthographic projections of a cuneiform tablet.

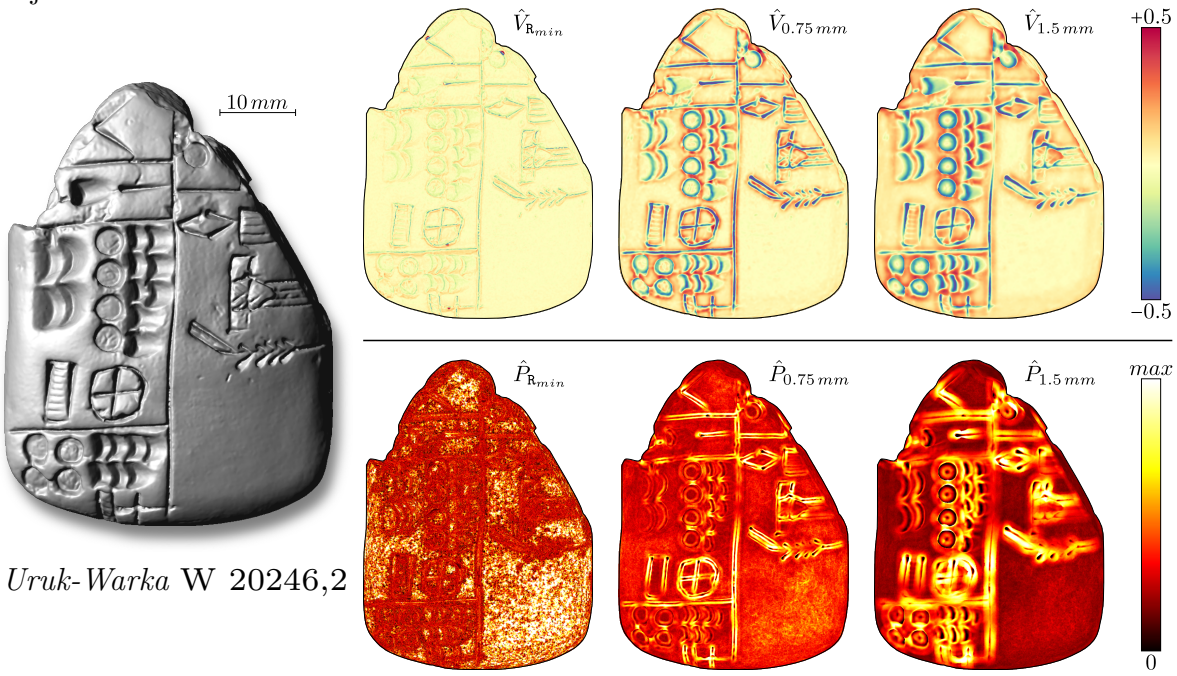


Figure 5.1: Orthographic projections of (top) three scales of the volume based integral invariant $\hat{V}_r$ and (bottom) the surface based integral invariant $\hat{P}_r$ computed for a cuneiform tablet.

The result of orthographic projections for each side of the bounding box of an object is the so-called *fat-cross* as shown in Figure 5.2. A total of 53 tablets were acquired within a week in the *Vorderasiatische Abteilung der Berliner Museen* (VAM) for the *Assur* research project of the *Heidelberger Akademie der Wissenschaften* in 2010. Afterwards MSII filtering was applied and *fat-crosses* using a gray-scale were rendered for an internal corpora. Further 166 tablets were acquired, processed and rendered for a second corpus in 2011, which required less than three man-months. All these tablets will be published in the *Keilschrifttexte aus Assur literarischen Inhalts* (KAL). Selected examples are shown in Appendix B.

Additionally to orthographic projections for the *fat-cross* raster images, two unwrapping methods are available within *GigaMesh* for objects with spherical and conical shapes including cylinders. As cylinder seals were of great importance to verify documents of the ancient Middle East in the past, and still are in the presence, the combination of MSII filtering and unwrapping is a superior method for their analysis.

Figure 5.2 shows a cylinder seal made of glass acquired in the VAM having the *Assur Fundnummer* (Ass.) no. 20417 – a photograph is shown in [Ped97]. Note that characters on seals are mirrored and only the imprint of the seal shows the characters in their proper orientation.

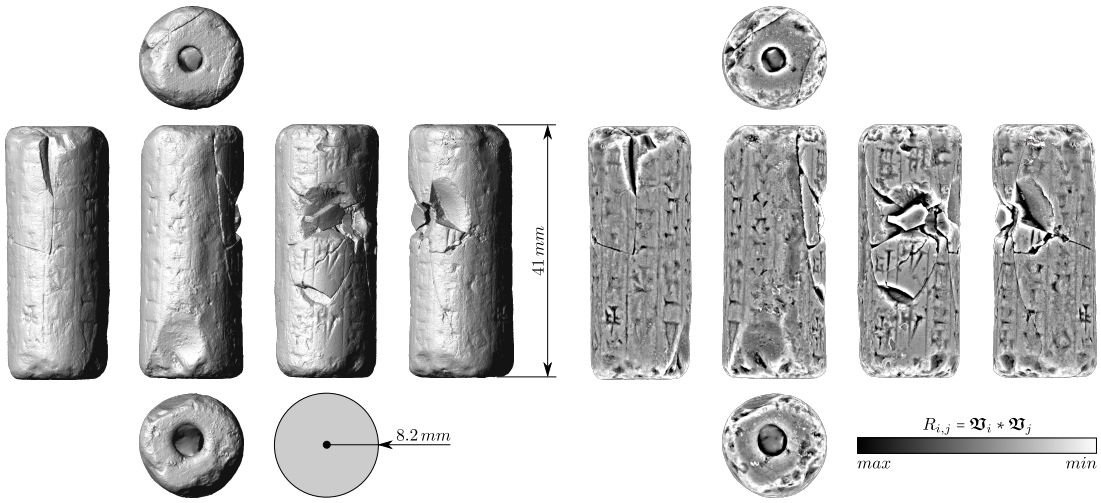


Figure 5.2: 3D-model of the cylinder seal VAM Ass. no. 20417, made of glass: (a) Visualized using virtual illumination and (b) volume based integral invariant filtering. The views computed orthogonal to the seal's bounding box are arranged as *fat-cross*.

Figure 5.3 shows an isometric view of the seal and the ideal cylinder, the virtually illuminated unwrapping, the unwrapped MSII filter result and a height map. The height map of an unwrapping is equivalent to the Hausdorff distance between the original 3D-model and an ideal cone used for unwrapping. Furthermore the surface normals of the unwrapping were inverted, otherwise the characters on a seal appear mirrored as it happens for 3D-models of molds. Figure B.14 on page 178 shows an additional example for unwrapping a clay nail, which has the shape of a cone.

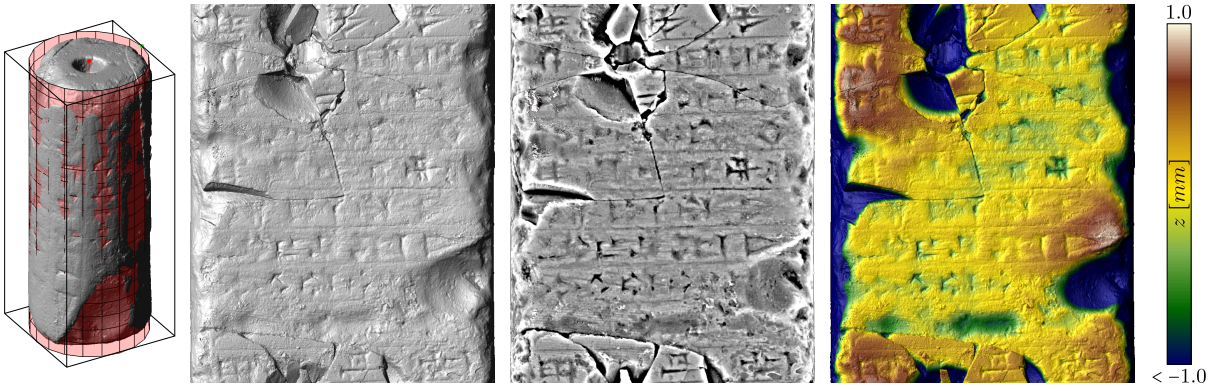


Figure 5.3: Cylinder seal VAM Ass. no. 20417: (a) Isometric view including the ideal cylinder used to determine (b) the unwrapped surface (shown with virtual illumination), (c) the MSII filter response and (d) a height map equivalent to the Hausdorff distance between the cylinder and the seal.

5.2 Geometric Character Representation – Vector Drawings

After autopsy on raster images, the next step is to compute a geometric representation of the characters. This representation has to be a subset of the mesh using distinctive points for a geometric exact position, while it has to discard unnecessary details distractive for reading a character. The polyline for the border is typically an isoline of a feature metric and the polylines of the skeleton are geodesic paths. Both are a geometric accurate representations of a character, which are replaced by an abstract line i.e. a parametric curve.

Bézier splines were chosen, because they are supported by the standardized and open file format *Scalable Vector Graphics* (SVG), which is used for manual drawings and automated drawings. This specific type of splines is used in the programming language *Metafont* to define vector fonts which are used for typesetting in TEX [Píš04]. An additional example is the LATEX related programming language *tikz* for generation of vector graphics using segments of cubic *Bézier splines*. The representation of synthetic cuneiform sign *kaskal* in *tikz* is shown in Appendix A.4 on pages 162ff.

The position vectors and direction vectors defining such splines are determined based on the geometry provided by the mesh's areas of interests $\mathcal{C}$. It has to be stressed that the mesh is used to compute the splines in $\mathbb{R}^3$ and not the (2D) raster images. Thereby the properties of the mesh's primitives are taken into account. The character's distinctive points are located along the border $\partial\mathcal{C}$ having a subset of points of interest $\mathcal{L}_{\textcircled{\scriptsize !}}$. Additional points are stored in $\mathcal{L}_{\circledast} \subset \mathcal{C}$ providing further spatial information i.e. depicting the location of a wedge's head. The points of $\mathcal{L}_{\textcircled{\scriptsize !}}$ have to be connected with those of $\mathcal{L}_{\circledast}$ as all these points are in a context. Connecting the points of $\mathcal{L}_{\textcircled{\scriptsize !}} \cup \mathcal{L}_{\circledast}$ is achieved by prioritized line tracing inspired by Voronoi-diagrams as shown in Section 4.3 on pages 113ff. The result are polylines describing the skeleton additional to the polylines of the outline.

Figure 5.4a shows the Voronoi inspired diagram of the sub-segmentation of areas of interest. The resulting skeleton is shown in Figure 5.4b, where $\mathbf{p}_1$ is one example for a branching node within $\mathcal{C}$ connected with the end-nodes $\mathbf{p}_K, \mathbf{p}_M$ located along $\partial\mathcal{C}$. The point $\mathbf{p}_2$ of the wedge in the middle is connected its three nearest neighbors $\mathbf{p}_E$, $\mathbf{p}_Q$ and $\mathbf{p}_1$. It must not be connected to the fourth neighbor $\mathbf{p}_V$, because a wedge is described by a total of four points.

As $\mathbf{p}_V$ is a tip of wedge, an additional branching-node $\tilde{\mathbf{p}}_3$ is introduced, because the lowest of the three parallel wedges has no seed vertex for skeletonization. Therefore $\mathbf{p}_2$ has to be connected to the closest geodesic path of the skeleton. This is achieved by backtracking the geodesic distance shown in Figure 5.4a. To avoid a connection to a point close to $\mathbf{p}_2$, the geodesic distance is computed using a weight function. Typically the volume integral invariant filter responses are used as weight, which enforces tracing lines along the vales and avoids lines crossing the ridges. The point where the backtracked line reaches the skeleton became the additional branching node $\tilde{\mathbf{p}}_3$.

Figure 5.4 shows two methods to determine splines as abstract representations. To illustrate these two methods the points $\mathbf{p}_1$, $\mathbf{p}_K$ and $\mathbf{p}_M$ shown in Figure 5.4b are used. The detected sharp corners in $\mathcal{L}_{\textcircled{\scriptsize 1}}$ become so-called cusp nodes of the splines. This means that these points have two tangent vectors $\mathbf{t}$ defining smooth parts of the curve left and right of such a node. The angle β between these vectors defines how sharp the corner becomes. Therefore Figure 5.4c shows the result for $\beta = 0$ using the directional vector between a skeleton's node and a branching node, which are $\mathbf{t}_{1K} = \mathbf{p}_1 - \mathbf{p}_K$ and $\mathbf{t}_{1L} = \mathbf{p}_1 - \mathbf{p}_L$ for the annotated points.

An alternative is shown in Figure 5.4d, where closest points like $\mathbf{p}_L$ of the polyline's segment $\mathbf{p}_K\mathbf{p}_M$ become so-called smooth nodes. The two tangents of such a smooth node have opposing directions and are parallel to the segments end-nodes: $\mathbf{t}_{KL} = -\mathbf{t}_{ML}$ and $\mathbf{t}_{KL} = \lambda\,(\mathbf{p}_K - \mathbf{p}_M)$, where $0 < \lambda < 2/|\mathbf{p}_K - \mathbf{p}_M|$. The length λ of the tangents depends on the orientation and length of $\mathbf{t}_{1K}$ and $\mathbf{t}_{1L}$ to prevent self-intersections of the spline.

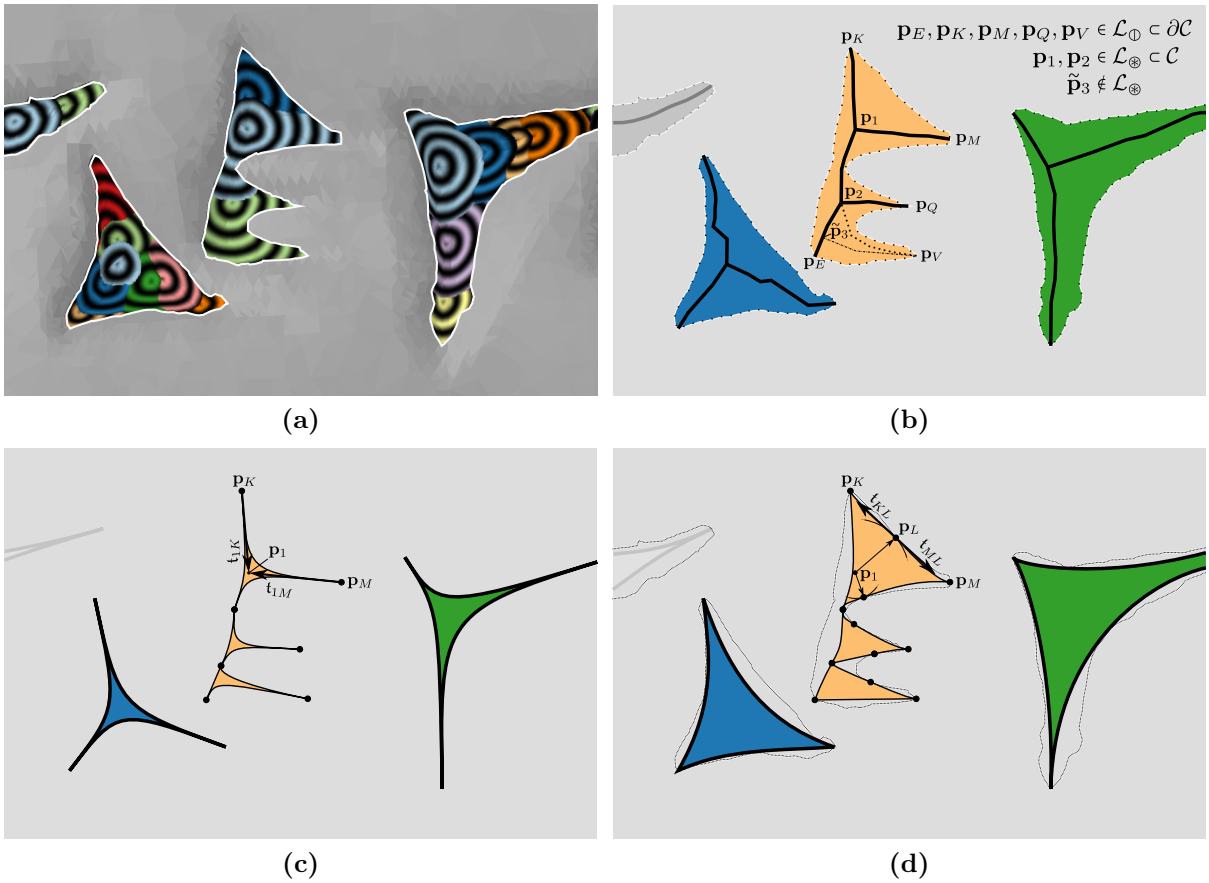


Figure 5.4: Detail of the cuneiform tablet HOS G10: (a) areas of interest sub-segmented using prioritized line tracing having (b) skeletons and points of interest for branching in $\mathcal{L}_{\circledast}$ and end-nodes in $\mathcal{L}_{\textcircled{\scriptsize 1}}$. *Bézier splines* using (c) only cusp nodes like $\mathbf{p}_K, \mathbf{p}_M$ with an angle $\beta = 0$ and (d) using additional smooth nodes like $\mathbf{p}_L$ having tangent vectors in opposing direction.

The improved results using weighted geodesic distances are shown in Figure 5.5 for the cuneiform character *Ka* introduced in Figure 1.6 on page 9. This figure additionally shows the 3D-model of the tablet and the visualization of the volume integral as well as the labeled areas of interest.

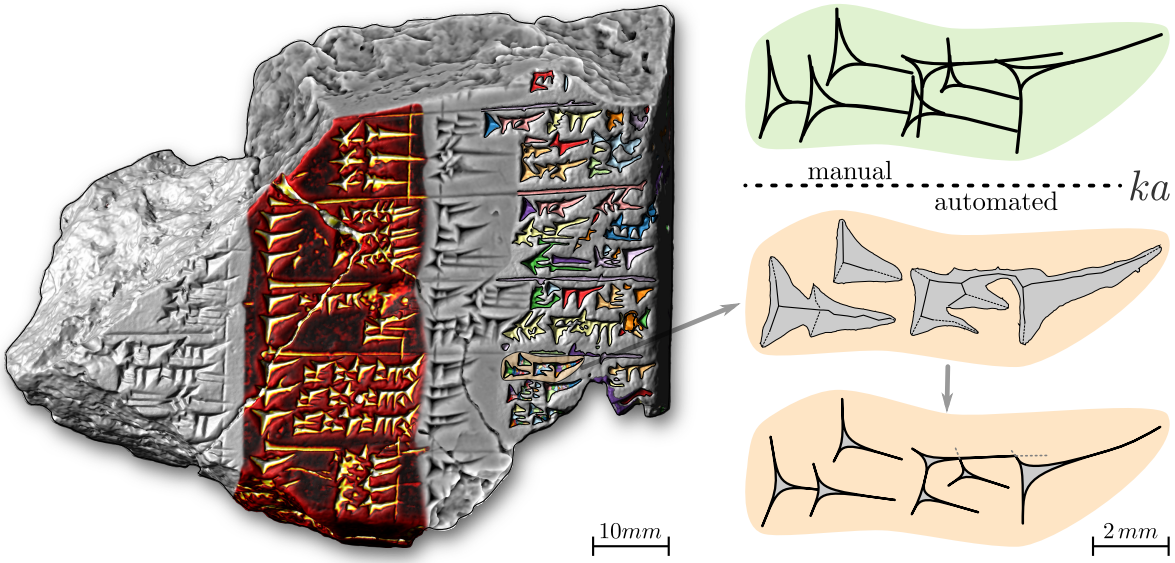


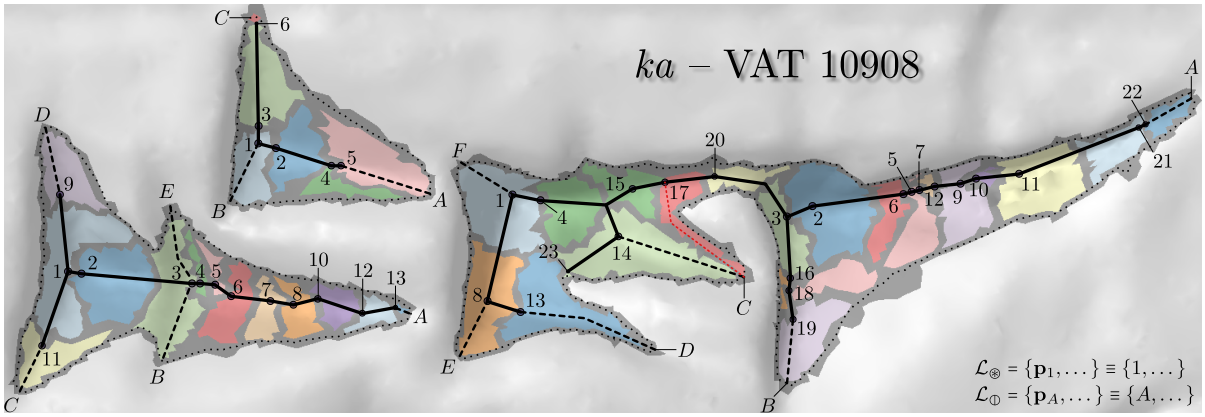
Figure 5.5: 3D-model of the tablet VAT 10908 acquired in the VAM. From left to right: virtual illuminated monochrome surface; combined correlation $R_{i,i,k}$ in shades of red and yellow; correlation $R_{i,k}$ as gray-scale without illumination; areas of interest colored by label id; and the skeleton used to compute the spline representation in comparison with the manual drawing.

The details of the character are shown in Figure 5.6: The skeleton computed without weights is shown in Figure 5.6a. Figure 5.6b shows the improved results and the parts of the skeleton being more robust against false connections. The volume integral invariant was used as weight and normalized to $[0, 1]$ shown in Figure 5.6c, where 0 relates to valleys (dark colors) and 1 to ridges (bright colors).

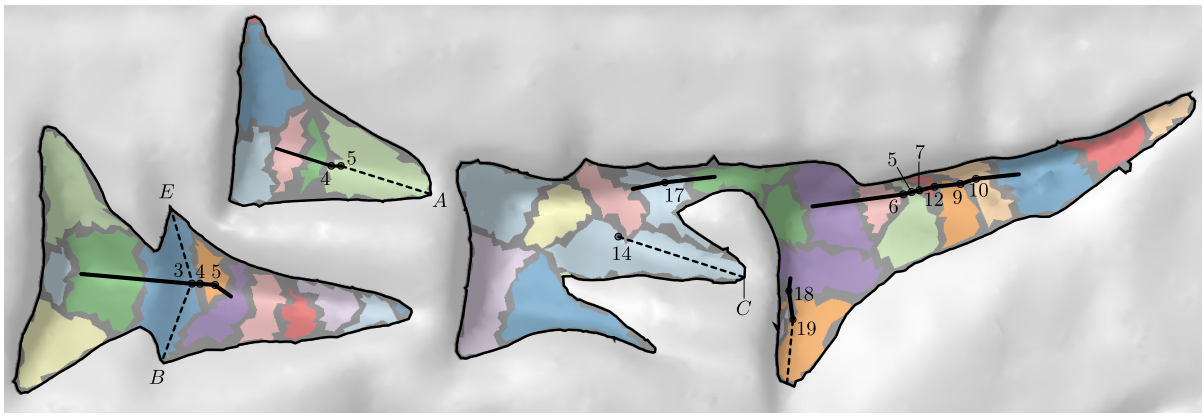
Having *Bézier splines* makes the automatically computed SVG drawing interchangeable with the manual drawings, requiring only minor corrections to fulfill an application-specific specification. These corrections can be comfortably applied using *Open Source* tools like *Inkscape*[1] as well as by using well-known proprietary software packages.

As cuneiform script is a handwriting, the representation as splines contains information like curvature radii and tilting parameters, which are suitable for paleographic research. A representation without splines can be computed in case of *Optical Character Recognition* (OCR) is the main aim. This normalized minimal representation is shown in the next section.

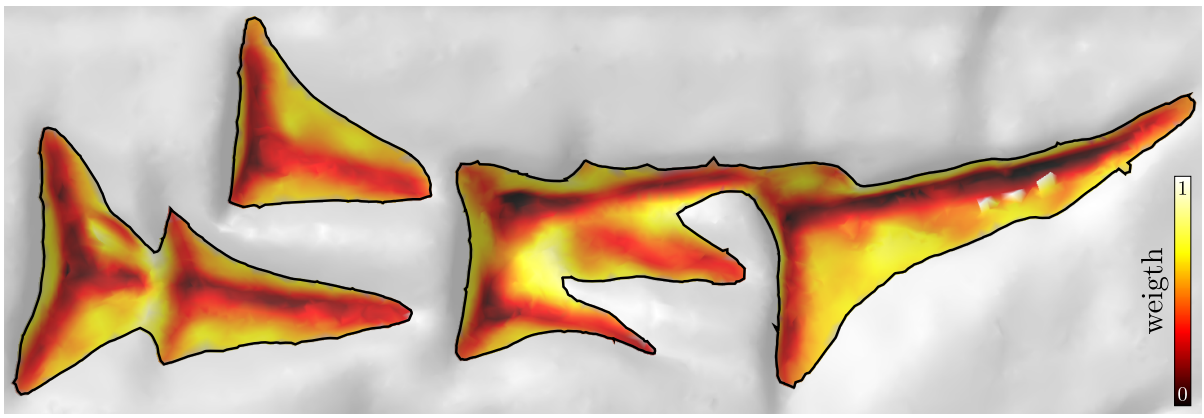
[1] http://inkscape.org – last checked 9th April 2012



(a)



(b)



(c)

Figure 5.6: Automated vector drawing of cuneiform character *ka* of tablet VAT 10908 computed (a) using the prioritized line tracing. (b) Improved skeleton computed with weighted geodesic distances using (c) the volume integral invariant to enforce geodesic paths within the vales having lower valued weights then the ridges.

5.3 Normalized Minimal Vector Representation

This section shows the extraction of cuneiform characters as minimal sets of vectors suitable for *Optical Character Recognition* (OCR). Such a representation is crucial for search engines, compact storage and machine processes like machine translation and text mining. Figure 1.7 on page 9 is a sketch of the principle behind such a representation. It is extended using the *Bézier splines* shown in the previous section, because these splines already include the necessary vectors for positions and directions.

The minimal vector representation of a wedge is derived from $\mathcal{L}_{\circledast}$ and $\mathcal{L}_{\textcircled{\scriptsize I}}$, which are the same lists as used to determine the splines. Then the representation consists of:

- one position vector $\mathbf{p}_{\circledast}$ located in the center of a wedge's head,
- one vector $\mathbf{t}_{\textcircled{\scriptsize I}}$ describing the principal direction of a wedge from $\mathbf{p}_{\circledast}$ to its tip and
- two vectors $\mathbf{t}_{\textcircled{\scriptsize A}}$, $\mathbf{t}_{\textcircled{\scriptsize B}}$ giving the directions from $\mathbf{p}_{\circledast}$ to the corner points of the head.

The angles between the principal direction and $\mathbf{t}_{\textcircled{\scriptsize A}}$, $\mathbf{t}_{\textcircled{\scriptsize B}}$ have to be

$$\arccos\langle\hat{\mathbf{t}}_{\textcircled{\scriptsize I}}, \hat{\mathbf{t}}_{\textcircled{\scriptsize A}}\rangle > \frac{\pi}{2} \quad \wedge \quad \arccos\langle\hat{\mathbf{t}}_{\textcircled{\scriptsize I}}, \hat{\mathbf{t}}_{\textcircled{\scriptsize B}}\rangle > \frac{\pi}{2} \quad . \tag{5.1}$$

Furthermore $\mathbf{t}_{\textcircled{\scriptsize A}}$ and $\mathbf{t}_{\textcircled{\scriptsize B}}$ have to have opposite directions in respect to $\mathbf{t}_{\textcircled{\scriptsize I}}$, which means

$$\arccos\langle\hat{\mathbf{n}}_{\textcircled{\scriptsize A}}, \hat{\mathbf{n}}_{\textcircled{\scriptsize B}}\rangle > \frac{\pi}{2} \quad \text{with} \quad \mathbf{n}_{\textcircled{\scriptsize A}} = \mathbf{t}_{\textcircled{\scriptsize I}} \times \mathbf{t}_{\textcircled{\scriptsize A}} \quad , \quad \mathbf{n}_{\textcircled{\scriptsize B}} = \mathbf{t}_{\textcircled{\scriptsize I}} \times \mathbf{t}_{\textcircled{\scriptsize B}} \tag{5.2}$$

has to be fulfilled. In short terms a wedge's minimalistic outline is a triangle $\{\mathbf{t}_{\textcircled{\scriptsize I}}, \mathbf{t}_{\textcircled{\scriptsize A}}, \mathbf{t}_{\textcircled{\scriptsize B}}\}$, which has to enclose the origin $\mathbf{o}$ – respectively $\mathbf{p}_{\circledast}$ has to be enclosed by the triangle $\{\mathbf{t}_{\textcircled{\scriptsize I}} + \mathbf{p}_{\circledast}, \mathbf{t}_{\textcircled{\scriptsize A}} + \mathbf{p}_{\circledast}, \mathbf{t}_{\textcircled{\scriptsize B}} + \mathbf{p}_{\circledast}\}$.

All vectors together are a wedge's minimal description stored in a set with four entries:

$$\{\mathbf{p}_{\circledast}, \mathbf{t}_{\textcircled{\scriptsize I}}, \mathbf{t}_{\textcircled{\scriptsize A}}, \mathbf{t}_{\textcircled{\scriptsize B}}\} \quad \text{where} \quad \mathbf{p}_{\circledast} \in \mathcal{L}_{\circledast} \quad , \quad \mathbf{t} = \mathbf{p} - \mathbf{p}_{\circledast} \quad \text{and} \quad \mathbf{p} \in \mathcal{L}_{\textcircled{\scriptsize I}} \tag{5.3}$$

Several of such sets are used to describe a cuneiform character. Normalization is achieved using the height of the line a character belongs to as scaling factor. For OCR the character's width is typically used to fit into a unit squares. The following example neglects such an unproportional scaling for better illustration.

The top row of Figure 5.7 shows the textbook depiction, the manual drawing and the automated drawing of the cuneiform character *ka*. Both drawings are based on tablet VAT 10908 shown in Figures 1.6 and 5.5 on pages 9 and 132. This character consists of seven wedges in these three variants. The number of sets is often constant for a characters regardless of the time and place a text was written i.e. the possible variants of a character. On the other hand the orientation and relative positions of the wedges are less constant as cuneiform script is a handwriting.

The bottom row of Figure 5.7 shows the seven quadruples of vectors describing each of the three variants of *ka*. The vectors for direction are shown as arrows. By definition these vectors share the same initial point, while their terminal points are equal to the cusp nodes of the splines. The related quadruples are shown with the same color.

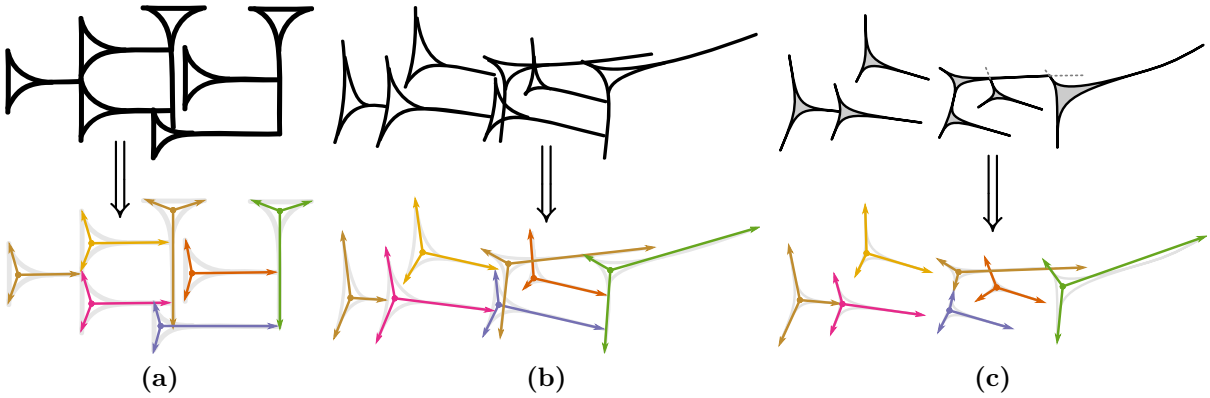


Figure 5.7: Cuneiform character *ka*. Top row: spline representation of the (a) textbook depiction, (b) manual drawing and (c) automated drawing of tablet VAT 10908. Bottom row: minimal representation using a quadruple of vectors for each of the character's wedges.

The three scaled and merged minimal representations of *ka* are shown in Figure 5.8a. Application of scaling the direction vectors and translation of the position vectors without changing the topology leads to Figure 5.8b, which clearly indicates the relation of the clustered wedges.

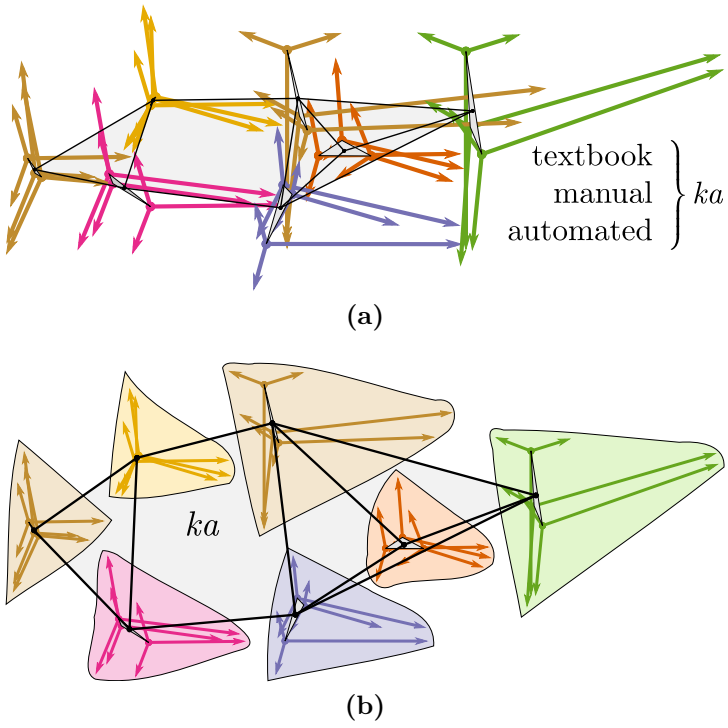


Figure 5.8: Scaled and merged minimal representations of (a) the character *ka* and (b) the same representation with minor transformations showing clusters of related wedges.

The assyriolgists' differentiation between the common wedge and the *Winkelhaken* (shown on page 7) can be coded as ratio between the lengths of the directional vectors:

$$\text{type of wedge} \leftarrow \begin{cases} \text{common wedge, e.g. } \curlyvee \text{ or } \rightarrowtail & \left(|\mathbf{t}_{①}| \gg |\mathbf{t}_{Ⓐ}|\right) \wedge \left(|\mathbf{t}_{①}| \gg |\mathbf{t}_{Ⓑ}|\right) \\ \textit{Winkelhaken}\text{, e.g. } \blacktriangleleft \text{ or } \blacktriangleleft & \left(|\mathbf{t}_{①}| < |\mathbf{t}_{Ⓐ}|\right) \wedge \left(|\mathbf{t}_{①}| < |\mathbf{t}_{Ⓑ}|\right) \\ \text{other shape} & \text{otherwise} \end{cases} \tag{5.4}$$

The sets of vectors in $\mathbb{R}^2$ as well as in $\mathbb{R}^3$ can be concatenated to points of higher dimension as used for OCR. Therefore the 3D-acquisition of tablets and their vectorization is important to establish a corpus of normalized minimal vector representations. This corpora can include manual vector drawings as well. It has to be noted that characters within manual drawings are subject to transformations, which are introduced by the interpretation or drawing styles of todays draftspersons.

Concatenating the sets describing the wedges of a character leads to a point (position vector) in a high-dimensional space. An application of such a high dimensional representation is the interpolation of characters. Figure 5.9 shows examples of interpolated characters, where the three high-dimensional points of the variants of *ka* were used to interpolate further variants. Finally these interpolated high-dimensional points were back-projected to $\mathbb{R}^2$ and their spline representation were computed.

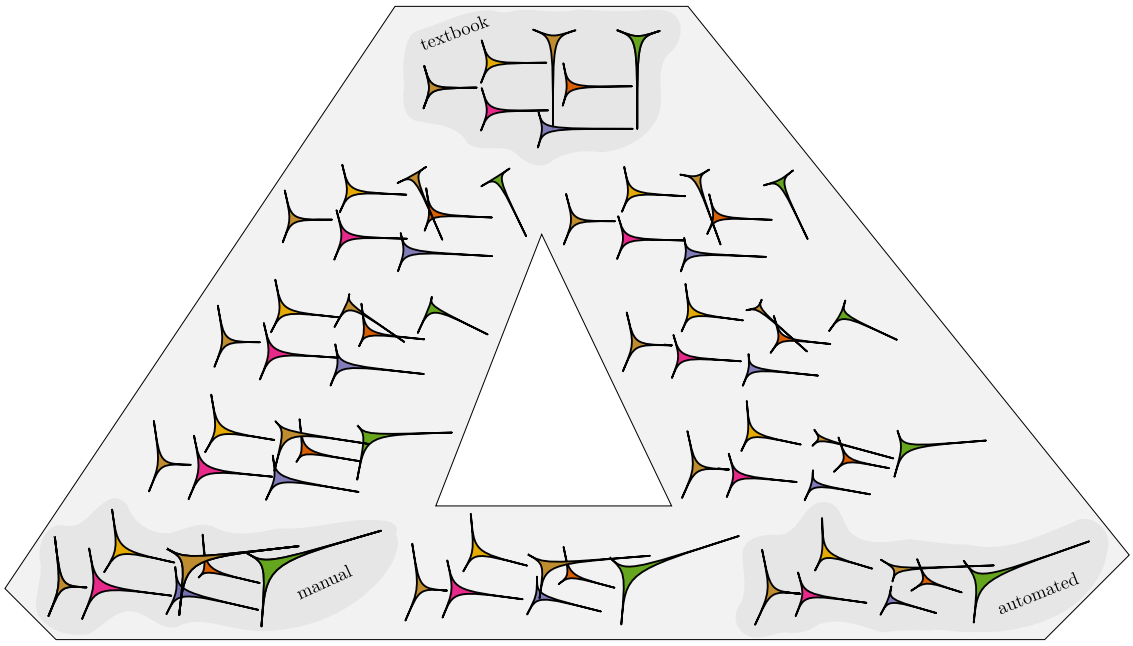


Figure 5.9: Interpolated character variants between the textbook depiction, manual drawing and automated drawing of the character *ka* using the quadruples of vectors.

The next section shows the performance and accuracy evaluation of the algorithms leading to the extraction and representation of characters using vectors, splines and raster images. Within this evaluation the algorithms for computing the Multi-Scale Integral Invariants are investigated, because they are essential for character extraction and are the most computationally expensive modules of the *GigaMesh* framework.

5.4 Evaluation of Performance and Accuracy

Processing large quantities of high-resolution irregular meshes requires a good balance of reasonable computing time in respect to accuracy. Therefore this section shows the performance and errors to be expected using the algorithms to compute and estimate the Multi-Scale Integral Invariants.

This includes the determination of truncation errors and overall computational complexity of the algorithms presented in this thesis and available in the *GigaMesh* framework. The focus of this section is on the MSII filtering, as these algorithms virtually account for all the performance and quality of the extraction of character representations.

The first section shows the accuracy and performance for the surface based integral invariant for multiple scales as well as for different resolutions. A similar analysis is shown for the volume based integral invariants in the second section. Finally the performance and computational complexity for the processing pipeline computing both integral invariants are shown for a representative set of more than three hundred 3D-models processed with *GigaMesh*. Appendix B shows an additional selection of the variety of processed objects.

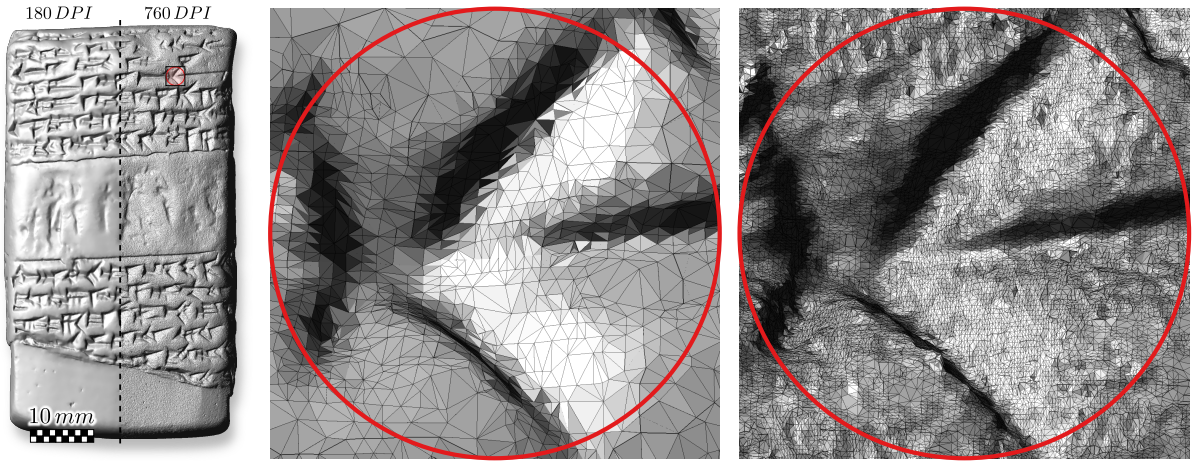
5.4.1 Computing and Estimating the Surface based Integral Invariant

3D-scanners typically provide fixed spatial resolutions for acquisition. To be on the safe side it is common practice to oversample an acquired surface, because there exists a vast number of reduction algorithms. The latter are not always applicable, because features have to be detected in very different scales.

Therefore this sections demonstrates the effects on computing time and estimation errors on tablet HOS G10 using two 3D-models having $3.7 \cdot 10^6$ vertices and $6.6 \cdot 10^6$ vertices. The low resolution model has an absolute resolution of approximately 180 DPI, while the high resolution variant has approximately 760 DPI. Computing DPI as measure related to spatial resolution for surfaces is shown in Section 2.1 on page 21.

Figure 5.10a shows the tablet having a low resolution for the left half and high resolution for the right half. Choosing the radius for the integral invariants is canonical using the *Shannon theorem* and [Köt08]. Therefore the radius computed has to be $1.5\,mm$ corresponding to the width of the larger wedges. This tablet also contains smaller characters, which are not part of the cuneiform script. The Figure 5.10b and Figure 5.10c show the same $3 \times 3\,mm$ detail of the surface for the low and high resolution.

For this tablet four scales of the surface integral invariant P were computed for illustration purposes. Practical experiments have shown that 8 to 16 scales give the best results in terms of accuracy and performance. Except for this section, all the multi-scale based results shown in this thesis have 16 scales. Using 2^n scales has the benefit of faster and simpler decomposition for Wavelet Transformation, which is one of the feature metrics introduced in Section 3.5.1 on page 81.



(a) (b) (c)

Figure 5.10: Tablet HOS G10. (a) The left half shows the 3D-model with $3.7 \cdot 10^6$ vertices $\approx 180\ DPI$ and the right half shows the 3D-model having $6.6 \cdot 10^6$ vertices $\approx 760\ DPI$. (b) $3 \times 3\,mm$ detail of the low resolution 3D-model and (c) the same detail in high resolution.

Let s be the scale for computing the normalized integral invariant $\hat{P}_{r \cdot s}$ with $r = 1.5\,mm$ being the largest scale using $s = [0.0, 1.0]$. Then the error of the estimation algorithm proposed in Equation 3.51 on page 63 and the exact solution using Algorithm 7 on page 70 is computed and shown in Table 5.1. This table apportions the mean error and standard deviation for all four scales computed for the low resolution and high resolution 3D-model.

$\hat{P}_{r \cdot s}$	$180\,DPI$		$760\,DPI$	
Scale s	Mean error [%]	Standard deviation [%]	Mean error [%]	Standard deviation [%]
1.00	13.054	4.511	2.951	1.495
0.75	17.393	6.478	3.944	1.913
0.50	25.054	10.514	5.895	2.710
0.25	50.870	24.962	11.838	5.316

Table 5.1: Mean error and standard deviation of estimating $\hat{P}_{r \cdot s}$ compared to the precise solution, computed for four scales s applied to the 3D-models of HOS G10 having different resolutions.

The results shown in Table 5.1 are plotted in Figure 5.11, The colors represent the scales, depicted as disks $B_{r \cdot s}$ scaled by s in Figure 5.11b. The lines in the middle represent the mean error, while the lines on above and below represent the standard deviation. As expected the error and the deviation decreases by the power of two for larger scales for both resolutions of the tablet. Furthermore the error decreases by the power of two for higher resolution 3D-model for scales of the same size.

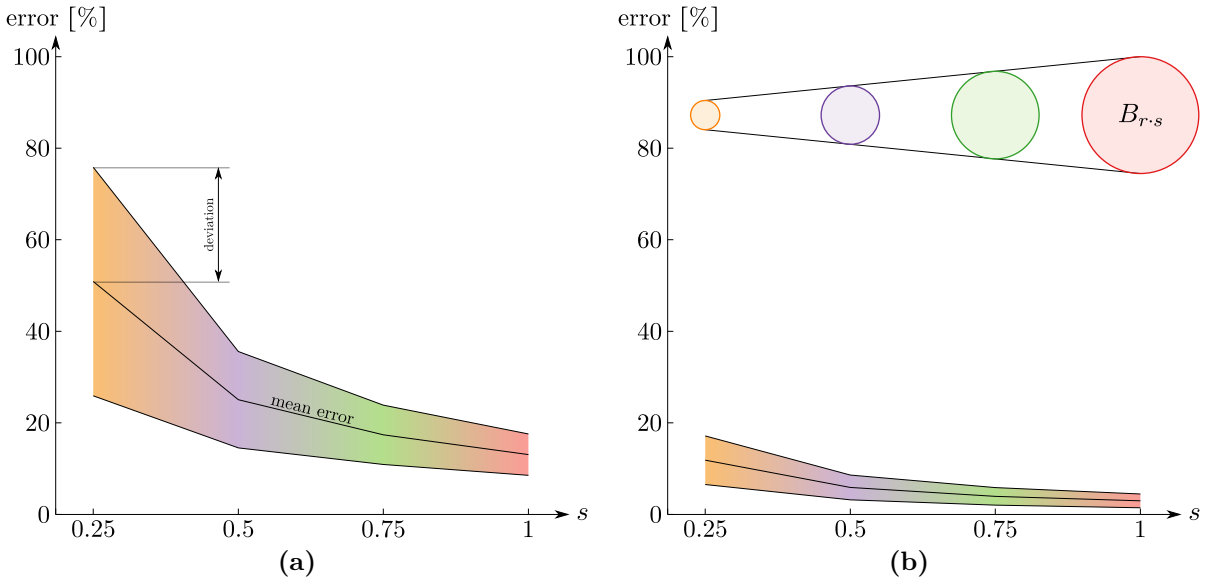


Figure 5.11: Mean error and standard deviation of estimating $\hat{P}_{r\cdot s}$ compared to the precise solution, computed for four scales s applied to the 3D-models of HOS G10 (a) for low resolution and (b) for high resolution.

Figure 5.12 on the next page shows the integral invariants for the largest and smallest scale ($s = 0.25, 1$) as color mapped to the *ColorBrewer's Spectral* color-ramp. Therein blue colors mean $\hat{P}_{r\cdot s}$ tend towards zero, red colors mean values tend towards $+\infty$ and mean values are shown in yellow. Additionally the density in vertices/mm^2 is shown, because depending on the unknown algorithms within the 3D-scanner's acquisition and processing methods, the distribution of the density of vertices along the surface is not constant. Furthermore this density accounts for strong variations of the error and it varies in respect to the resolution of the 3D-model. High densities are shown in red colors and low densities in green colors.

Regarding performance the estimation algorithm is approximately ten times faster regardless of the scale and the number of triangles to processed within the ball B_r. The reason is an identical computational complexity, which is $O(n^2)$ and n being the number of triangles. The increased computing time is accounted only by the increased number of computation steps in Algorithm 7 compared to those required to implement Equation 3.51. Therefore it is recommended to use the precise algorithm unless the distribution of the density of vertices is constant.

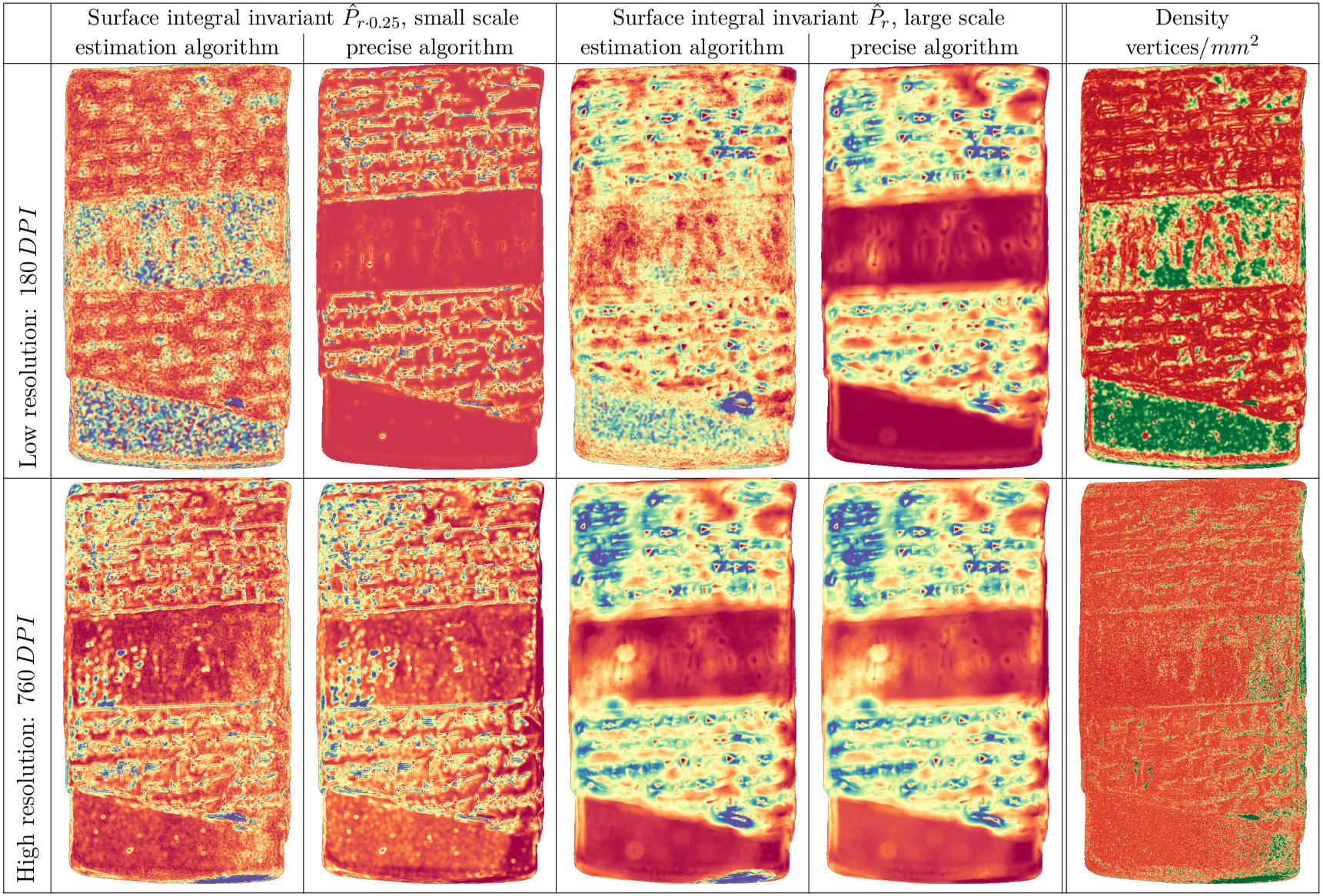


Figure 5.12: Surface integral invariant $\hat{P}_{r\cdot s}$ for a small scale ($r/4$) and a large scale ($r = 1.5\,mm$). Areas with larger estimation errors correlate with areas having a low density show in green color in the right column.

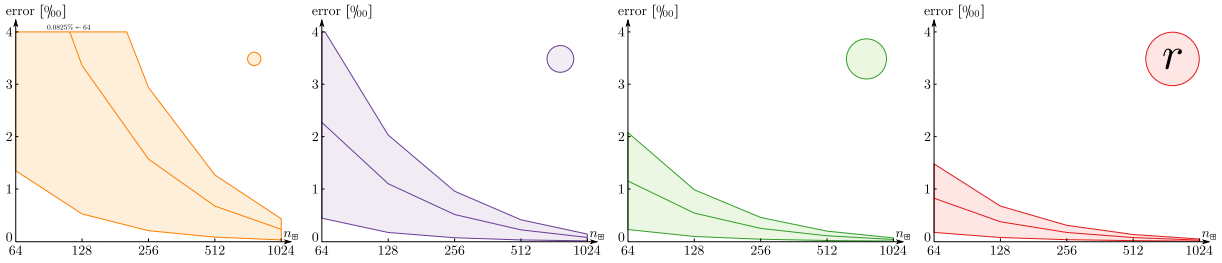
5.4.2 Determining the Volume based Integral Invariant Feature Space

The volume integral invariant $\hat{V}$ can be determined using either a regularly sampled voxel grid or using the divergence theorem. Regular sampling is shown in Section 3.2 and leads to an estimator, where $n_{\boxplus}$ is the sampling interval for a ball's radius r. Realizing an algorithm with the divergence theorem shown in Section 3.4 results in a precise computation, which can be combined with the algorithm for computing the surface based integral invariant $\hat{P}$.

Therefore the divergence theorem based algorithm is expected to have a computational complexity of $O(n^2)$, while processing voxels per-se has $O(n_r^3)$ – n_r being the number of triangles within the ball B_r. However the voxel based algorithm is optimized using the idea of melting spheres applying a local transformation shown in Equation 3.18 on page 50. Then the voxel grid was replaced by a regular two-dimensional grid of prisms with a square base.

As a consequence processing these prisms has a complexity of $O(n_{\boxplus}^2)$ and $n_{\boxplus}$ is typically smaller than n_r. Nevertheless the number of processing operations and their execution time as well as the algorithms' initializations contribute to actual computing time t_Δ. The relative computing time between estimation by regular sampling and computation using the divergence theorem is shown in the right column of Table 5.2 and Table 5.3 for the low and high resolution example of HOS G10.

Furthermore Table 5.2 and Table 5.3 show the mean error and the standard deviation of the estimation compared to the results of the precise divergence based algorithm. The results for the estimator increase with finer sampling using larger values of $n_{\boxplus}$, while the performance rapidly decreases. Figure 5.13 shows the plots of the errors for all four scales estimated using $n_{\boxplus} = 64$. This example has the largest error, which is less than 6.8 ‰ for all scales and both resolutions.



(a) $s = 0.25$ **(b)** $s = 0.5$ **(c)** $s = 0,75$ **(d)** $s = 1$

Figure 5.13: Comparison of estimating and computing $\hat{V}_{r,s}$ for HOS G10 having $180\,DPI$. Mean error and standard deviation depending on the quantization by $n_{\boxplus} = 64$ and the four scales s of the ball $B_{r,s}$ with $r = 1.5\,mm$.

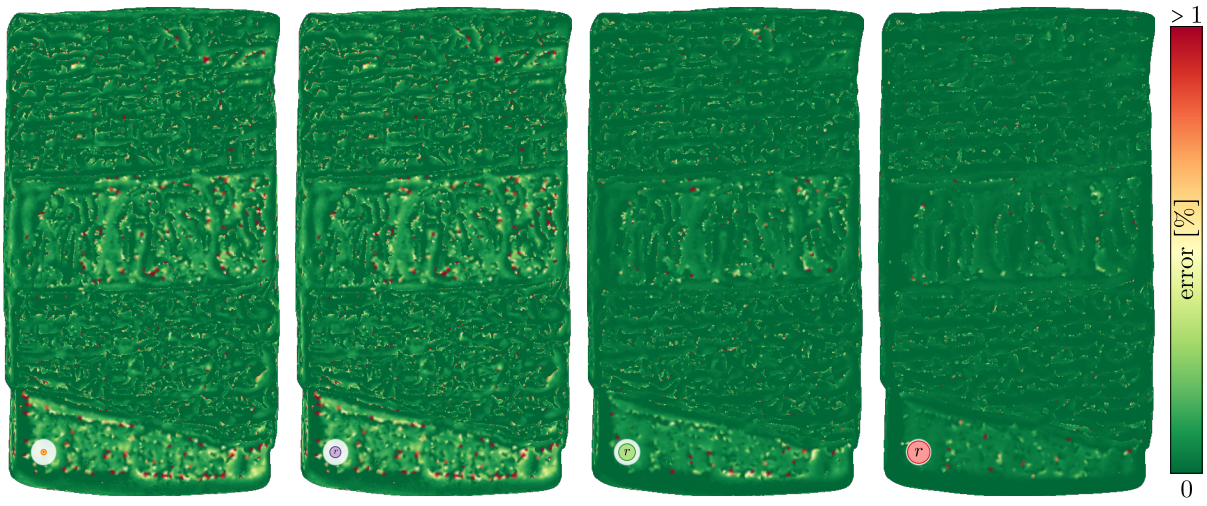
$\hat{V}_{r\cdot s}$	mean error [‰]				standard deviation [‰]				Comp.
$n_{\boxplus}$	$s = 1$	0.75	0.5	0.25	$s = 1$	0.75	0.5	0.25	time, rel.
64	0.825	1.151	2.271	6.802	0.654	0.925	1.829	5.455	0.9
128	0.374	0.539	1.100	3.357	0.300	0.446	0.930	2.833	1.7
256	0.170	0.248	0.512	1.571	0.137	0.208	0.444	1.367	4.7
512	0.072	0.106	0.220	0.675	0.058	0.089	0.192	0.594	16.2
1024	0.024	0.036	0.075	0.229	0.020	0.030	0.065	0.202	61.3

Table 5.2: Comparison of estimating and computing $\hat{V}_{r\cdot s}$ for HOS G10 having 180 DPI. Mean error and standard deviation depending on the quantization by $n_{\boxplus}$ and the balls' scales s. The right column is the factor for computing time of the estimation to the exact computation.

$\hat{V}_{r\cdot s}$	mean error [‰]				standard deviation [‰]				Comp.
$n_{\boxplus}$	$s = 1$	0.75	0.5	0.25	$s = 1$	0.75	0.5	0.25	time, rel.
64	0.774	0.854	1.549	4.362	0.744	0.736	1.261	3.589	1.0
128	0.351	0.375	0.701	2.084	0.360	0.338	0.627	1.876	1.1
256	0.158	0.172	0.338	1.031	0.169	0.158	0.321	1.001	1.5
512	0.067	0.075	0.150	0.457	0.073	0.069	0.147	0.467	2.8
1024	0.022	0.025	0.051	0.154	0.024	0.023	0.050	0.161	7.5

Table 5.3: Comparison of estimating and computing $\hat{V}_{r\cdot s}$ for HOS G10 having 760 DPI. Mean error and standard deviation depending on the quantization by $n_{\boxplus}$ and the balls' scales s. The right column is the factor for computing time of the estimation to the exact computation.

Figure 5.14 shows the mean error of the estimated $\hat{V}_{r\cdot s}(\mathbf{p}_i)$ for each vertex $\mathbf{p}_i$ of the low resolution 3D-model for four scales s and $n_{\boxplus} = 64$. This is the worst case of results using the estimation having an error below 1% percent, which applies to virtually any given mesh. This means that the *Dalí* inspired estimation algorithm is very robust.



(a) $s = 0.25$ **(b)** $s = 0.5$ **(c)** $s = 0.75$ **(d)** $s = 1$

Figure 5.14: Mean error of the estimated $\hat{V}_{r\cdot s}(\mathbf{p}_i)$ for each vertex $\mathbf{p}_i$ of the 3D-model having 180 DPI for $n_{\boxplus} = 64$ and four scales s from (a) to (d).

Figure 5.15 shows the *Euclidean norm* of the feature vectors $f(\mathbf{p}_i) \leftarrow |\mathfrak{V}_i|$ mapped to a color ramp. Each element of a feature vector $\mathfrak{v}_j \leftarrow \hat{V}_{r \cdot s}(\mathbf{p}_i)$ represents the j-th scale s and was estimated using $n_{\boxplus} = 64$. As the estimation error is less than one percent there is no visible difference for results with larger values of $n_{\boxplus}$. The visible differences between these two results are more distinct details, which depend predominantly on the spatial resolution.

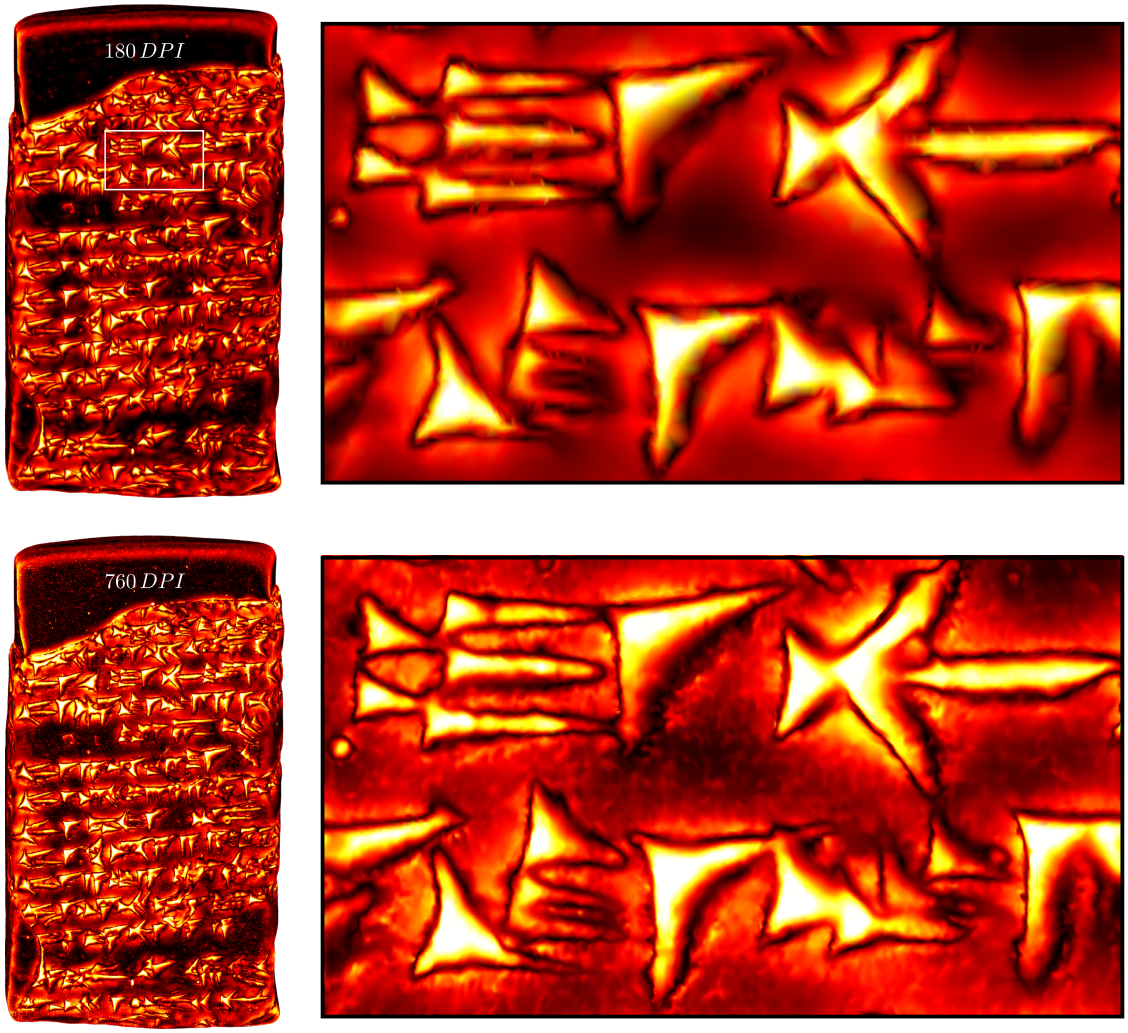


Figure 5.15: *Euclidean norm* $|\mathfrak{V}_i|$ of the feature vectors computed using $\hat{V}_{r \cdot s}$ with $r = 1.5\,mm$ and $n_{\boxplus} = 64$ for the scales $s \in \{0.25, 0.5, 0.75, 1\}$. The top row shows the low resolution 3D-model and the bottom row the variant with high resolution. Bright colors mean large values of $|\mathfrak{V}_i|$.

Therefore both the estimator and the precise method yield comparable results for the application shown throughout this thesis. However, the choice for one of the methods depends on structure of the input data. Processing regular grids like *Digital Terrain Models* (DTMs) will gain dramatic performance increases using the estimator, while processing irregular grids with high curvatures will strongly benefit from precise computation of the volume integral invariant.

5.4.3 Combined Calculation of the Multi-Scale Integral Invariants

Chapter 3 proposes an algorithm for combined calculation of the three integral invariants $\hat{V}$, $\hat{P}$ and $\hat{L}$. These invariants are typically to be computed in multiple scales. The result are the feature vectors $\mathfrak{V}$,$\mathfrak{P}$ and $\mathfrak{L}$ computed for balls and spheres of radius $r \cdot s$. The radius r relates to the size of the largest feature with the scales $s = [0, 1]$ as shown in the previous section.

For cuneiform tablets the number of scales is typically 16, while the largest wedges in general require the radius $r = 2\,mm$. To achieve comparable results, both the number of scales and the radius were set to these de-facto default values. The tablets acquired for the HAW's *Assur Forschungsstelle* were divided into two groups. The first group contains tablets of smaller size and less detail, while the second group contains large tablets with larger characters. Therefore the tablets' 3D-models in both groups consist of one to ten millions of vertices.

Most of the tablets are actually fragments and it is common knowledge in *Assyriology* that fragments can be joined within a museum collection and even with objects found in other museums. Therefore the fractured sides were acquired with the 3D-scanner to enable virtual joints of the "3D-puzzle" with fragments distributed all over the world. An example for this puzzle is the conjoint of tablets no. 9927 and no. 9970 shown in Figure 5.16. The joint of these two fragments – denoted as 99(27,70) in Table 5.4 – was identified by their textual content and they have been manually reassembled.

For virtually all 3D-models of cuneiform tablets, the following two tables show the results for representative samples regarding performance indicating computational complexity. The acquired area $A_{\mathcal{M}}$ is shown in mm^2. For estimating the resolution

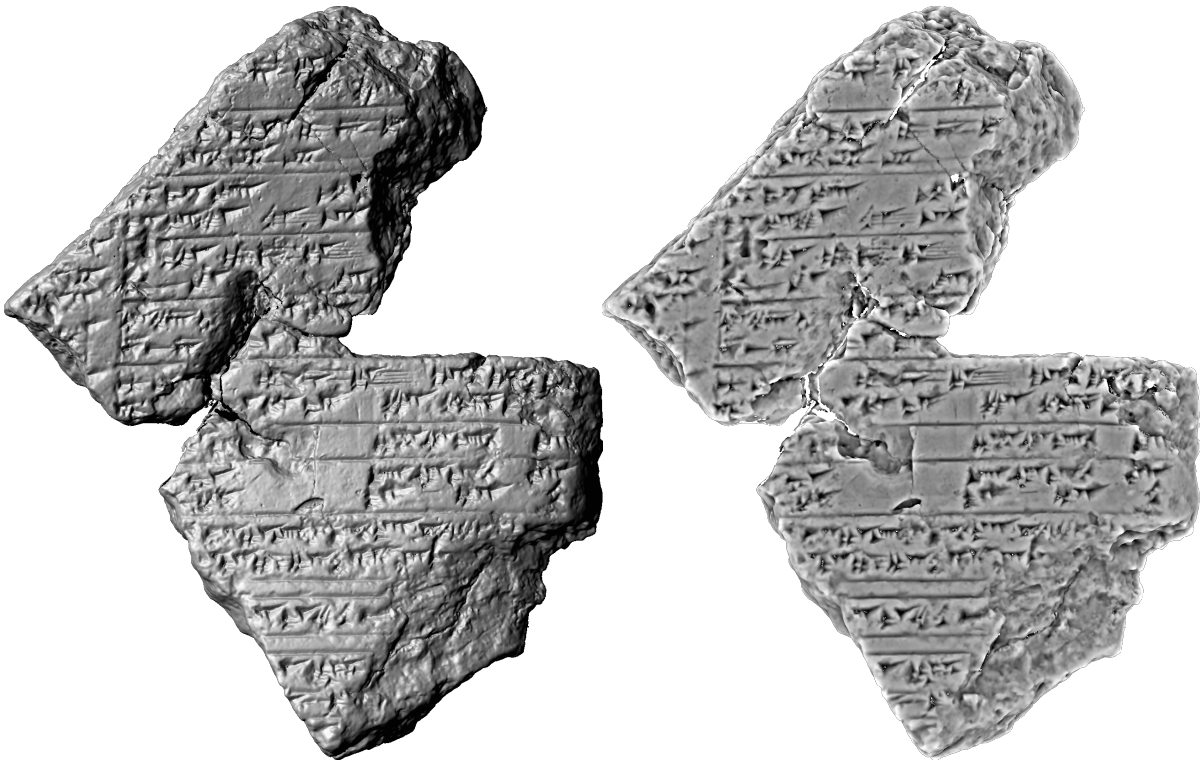
$$R_{\mathcal{M}} = \sqrt{\frac{|\mathcal{L}_v|}{A_{\mathcal{M}}}} \tag{5.5}$$

of the 3D-model the area $A_{\mathcal{M}}$ is expressd in $inch^2$. This estimation is related to the vertex density distribution shown in the previous section. While the vertex density is typically computed in mm^{-2}, the square root of the average in *Dots Per Inch* (DPI) allows a coarse comparison with 2D-acquisition methods as shown in Section 1.1 on pages 3ff.

Table 5.4 shows the results for a group of 27 tablets having less detail and size, which were acquired with a resolution of $\approx 200\,DPI$. Table 5.5 shows 26 examples with greater details, which were acquired having a higher resolution of $\approx 1200\,DPI$. The 53 tablets in these tables are a representative subset of the total of 166 tablets acquired in the VAM. Both tables are sorted by ascending numbers of vertices $n_v = |\mathcal{L}_v|$.

The number of vertices n_v and processing time t_Δ per 3D-model are normalized to $\hat{n}_v$ and $\hat{t}_\Delta$ using the tables' smallest tablets (first entry, with bold numbers). Therefore the right column shows

$$\tilde{t}(\hat{n}_v) = \hat{n}_v \cdot \log \hat{n}_v \tag{5.6}$$



(a) (b)

Figure 5.16: Conjoined tablets VAT no. 9927 and 9970 acquired in the *Vorderasiatische Abteilung der Berliner Museen* (VAM). 3D-model scaled to 95% of the original size (a) with virtual illumination and (b) with a gray-scale texture map computed using Multi-Scale Integral Invariants.

indicating an assumed normalized processing time related to a computational complexity of $O(n \cdot \log n)$.

For 3D-models with larger numbers of vertices, $\hat{t}_\Delta$ becomes smaller than $\widetilde{t}(\hat{n}_v)$. The break-even points are marked with horizontal lines within the tables. There are cases before and after the break-even point, which do not fulfill this rule. The reason is the resolution $R_\mathcal{M}$ respectively the vertex density distribution, which has an influence on the computing time. However, the differences between $\widetilde{t}(\hat{n}_v)$ and $\hat{t}_\Delta$ are minor and can be neglected in the big picture.

Tablet VAT no.	Vertices $n_v = \lvert\mathcal{L}_v\rvert$	$A_{\mathcal{M}}$ $[mm^2]$	$R_{\mathcal{M}}$ $[DPI]$	t_Δ $[min]$	$\hat{n}_v = \frac{n_v}{217\,611}$	$\hat{t}_\Delta = \frac{t_\Delta}{3} - 1$	$n_v \cdot \log n_v$
9971d	**217 611**	1 318	165	**3**	**1**	**0**	0
9950b	322 543	1 633	198	5	1.5	0.7	0.3
9971c	503 064	2 857	176	7	2.3	1.5	0.8
10974	959 415	5 243	183	14	4.4	3.9	2.8
10373	1 011 984	4 881	207	16	4.7	4.5	3.2
9970a	1 057 459	5 900	179	15	4.9	4.1	3.4
10212	1 110 145	6 167	180	16	5.1	4.3	3.6
10590	1 295 314	7 051	184	19	6	5.6	4.7
9950a	1 318 122	6 613	199	21	6.1	6.1	4.8
9884	1 337 580	7 288	184	20	6.1	5.7	4.8
9970b	1 386 117	7 752	179	20	6.4	5.9	5.2
9971b	1 455 375	8 113	179	21	6.7	6.2	5.5
10949	1 491 973	7 783	192	23	6.9	6.9	5.8
11113	1 620 977	9 089	178	24	7.4	7.1	6.4
11066	1 667 636	9 712	172	24	7.7	7	6.8
10272	1 950 168	10 567	185	30	9	9.1	8.6
99(27,70)	2 295 734	12 281	187	34	10.5	**10.7**	**10.7**
10330	2 476 170	15 316	162	34	11.4	10.7	12
9971a	2 646 410	14 194	186	41	12.2	12.9	13.3
10594	2 777 839	13 880	200	44	12.8	14	14.2
10338	3 818 765	20 122	190	60	17.5	19.4	21.8
10297	4 131 210	22 992	180	61	19	19.6	24.3
13825	4 489 628	21 271	211	79	20.6	25.7	27.1
10297b	5 007 454	28 296	177	73	23	24	31.3
10432	5 022 501	29 250	172	75	23.1	24.5	31.5
9543	5 454 936	28 679	190	86	25.1	28.4	35.1
9728	9 897 424	43 291	229	177	45.5	59.3	75.4

Table 5.4: Compute times t_Δ for combined determination of $\mathfrak{V}$,$\mathfrak{P}$ and $\mathfrak{L}$ with $r = 2\,mm$ and 16 scales s. The 27 large tablets were acquired with a resolution of $\approx 200\,DPI$. The conjoined tablets no. 9927 and no. 9970 are by chance the break-even point, where the performance decrease is less than $n_v \cdot \log n_v$.

Tablet VAT no.	Vertices $n_v = \lvert\mathcal{L}_v\rvert$	$A_{\mathcal{M}}$ $[mm^2]$	$R_{\mathcal{M}}$ $[DPI]$	t_Δ $[min]$	$\hat{n}_v \approx \frac{n_v}{1.4 \cdot 10^6}$	$\hat{t}_\Delta \approx \frac{t_\Delta}{105}$	$n_v \cdot \log n_v$
9970c	**1 364 686**	1 078	1266	**105**	**1**	**0**	0
11706	1 633 532	1 191	1372	136	1.2	0.3	0.1
10569g	2 155 847	1 780	1211	142	1.6	0.4	0.3
9456	2 596 745	2 099	1237	160	1.9	0.5	0.5
11267	2 364 788	1 932	1224	162	1.7	0.5	0.4
9450	2 754 072	2 138	1288	187	2	0.8	0.6
10569d	3 190 604	2 744	1163	201	2.3	0.9	0.8
9457	3 187 844	2 563	1244	206	2.3	1	0.8
10569b	2 913 221	2 293	1270	209	2.1	1	0.7
9970d	3 624 437	3 132	1157	221	2.7	1.1	1.2
10569c	3 326 430	2 547	1306	264	2.4	1.5	0.9
11234	4 061 074	3 235	1255	272	3	1.6	1.4
10569f	4 328 340	3 477	1245	293	3.2	1.8	1.6
10569	4 158 863	3 268	1273	294	3	1.8	1.4
11583	4 866 538	4 098	1188	303	3.6	1.9	2
11278	5 140 600	4 220	1218	332	3.8	2.2	2.2
11380	4 837 864	3 727	1298	353	3.5	2.4	1.9
10077	5 295 236	4 077	1299	372	3.9	2.5	2.3
11558	6 026 297	5 113	1179	382	4.4	2.7	2.8
11520	6 598 858	5 425	1216	427	4.8	3.1	3.3
10569h	6 040 665	4 554	1326	465	4.4	3.4	2.8
11403	7 002 755	5 606	1249	474	5.1	3.5	3.6
11274	8 775 083	7 432	1181	577	6.4	4.5	5.2
11308	9 931 764	8 114	1224	665	7.3	5.3	6.3
10569e	8 598 434	6 520	1319	694	6.3	5.6	5
11503	9 940 620	7 728	1286	722	7.3	5.9	6.3

Table 5.5: Compute times t_Δ for combined determination of $\mathfrak{V}$,$\mathfrak{P}$ and $\mathfrak{L}$ with $r = 2\,mm$ and 16 scales s. The 26 small tablets were acquired with a resolution of $\approx 1200\,DPI$. The break-even point, where the performance decrease is less than $n_v \cdot \log n_v$ is marked with two horizontal lines.

5.5 Summary

Summarizing this chapter shows twofold results. The first result concerns curvature related representations for the application to cuneiform tablets. The readability of the highly complex and detailed cuneiform script becomes dramatically improved on raster images of 3D-models by mapping a feature metric like $f(\mathbf{p}_i) \leftarrow R_{i,j}$ for each vertex $\mathbf{p}_i$ of the mesh to a color-ramp. Furthermore automatically computed vector drawings – outlines and skeletons – are added as an overlay. These automated vector drawings are highly similar and compatible to manual vector drawings having photographs as base layer.

Reducing the vector drawings to a minimal and normalized vector representation is a reduction of the irregular high-resolution 3D-mesh to a meaningful set of vectors for each wedge. These sets of vectors consist of one vector for a wedge's position and one vector for the principal direction. The additional two vectors for minor directions encode the type of wedge (⊢ vs ◀). Interpolation of hand-written cuneiform characters is shown in Figure 5.9 on page 136 using this representation, which is additionally usable for *Optical Character Recognition* (OCR), e.g. using *Support Vector Machines* (SVMs) [CV95, SV99]. This becomes even more evident considering the data reduction by a factor of more than a thousand. One representative example is the 3D-model of tablet VAT 10908, which has $1.3 \cdot 10^6$ vertices. This vast number of vertices was reduced to 1680 meaningful points, which describe the wedges represented as 420 sets of vectors.

The second result concerns the performance and accuracy of computing and estimating the different Multi-Scale Integral Invariants. Regarding accuracy the precise (and combined algorithm) is generally preferable, while the estimators are simpler to implement and require less computing operations. Furthermore the estimators' accuracy increases with the square of the scale's size, i.e. radius of the balls and spheres used to compute integral invariants. Therefore the estimators are supposed to have big performance gains for less irregular 3D-models like *Digital Terrain Models* (DTMs).

Finally the overall performance was shown for a vast amount of cuneiform tablets in various sizes, indicating an overall computational complexity of less than $O(n_v \cdot \log n_v)$. for 3D-models with large numbers of n_v vertices.

Chapter 6

Conclusion and Outlook

The work presented in this thesis concern methods and algorithms to estimate and compute integral invariants, which are curvature related measures. These measures are computed in multiple scales resulting in a high dimensional feature space. Calculations and segmentation of this vertex related space leads to extraction of features. This extraction is shown for cuneiform characters, a specific and challenging application due to the complex nature of this writing system. These characters are located on arbitrary surfaces captured with a 3D-scanner as high-resolution irregular meshes consisting of vast amounts of vertices and triangles. For extraction as well as for visualization a mesh processing pipeline is proposed, which follows the principles of pattern recognition and the image processing pipeline.

The main part of this work consists of the development of new and efficient algorithms for computing Multi-Scale Integral Invariants. Because of the difficulty of determining an analytic solution and the fact that the meshes are not represented in a parametric form lead to two sets of fast estimators for the volume and surface based integral invariants $\hat{V}$ and $\hat{P}$. These estimation methods are extended to a combined algorithm providing a precise solution.

The volume integral $\hat{V}$ relates to mean curvature being robust against noise, while the surface integral $\hat{P}$ is a relative to the *Gaussian curvature*. At the same time $\hat{P}$ is an indicator for surface roughness i.e. noise. The combined algorithm includes the computation of a third integral invariant $\hat{L}$, which is line based and is an additional measure to determine noise and differ it from shape. Furthermore a forth integral invariant $\hat{S}$ is introduced, which is robust against noise and relates to $\hat{V}$.

These four integral invariants for the 2D-manifold represented by a triangular mesh in $\mathbb{R}^3$ are the results of the intersection of a ball B or sphere $\mathbb{S}$ with the manifold $\mathcal{M}_2$ and its enclosed surface or volume in the space enclosed by $\mathcal{M}_2$. The same quadruple of integral invariants are observed for the connected components $\mathcal{C}$ and their borders $\partial\mathcal{C}$, which are 1D-manifolds enclosing a 2D-manifold. Furthermore the quadruple is observable for the integral invariants for 1D-manifolds in $\mathbb{R}^2$, which is the special (flat) case of the connected components.

These observations are shown in the following sections to be investigated for future work on integral invariants for manifolds in higher dimensions. Afterwards a conclusion about the mesh processing pipeline and an outlook is given.

Quadruples of Integral Invariants for 3D-Models

There is a total of four integral invariants introduced in Chapter 3, because the volume based integral invariant alone is not sufficient for character extraction within a 2D-manifold $\mathcal{M}_2$ in $\mathbb{R}^3$. These integral invariants are defined in Equations 3.36 to 3.39 on page 61 and they are computed by

$$V_r = |\mathbb{V}_r| \quad , \quad P_r = |\mathcal{P}_r| \quad , \quad S_r = |\mathcal{S}_r| \quad \text{and} \quad L_r = |\mathscr{L}_r| \quad . \tag{6.1}$$

The scale r is not annotated from now on, because the following observation does not depend on a scale as long as $r > 0$. Figure 3.11 on page 60 shows two different examples of the geometric representations of the manifolds leading to this quadruple of integral invariants.

The first integral invariant is the volume of $\mathbb{V}$, which is a subset of $\mathbb{R}^3$. The border $\partial\mathbb{V}$ is the union of two 2D-manifolds $\mathcal{P}$ and $\mathcal{S}$. Their areas are the second and third integral invariant. These two 2D-manifolds share a border $\partial\mathcal{P} = \partial\mathcal{S}$, which is the 1D-manifold $\mathscr{L}$ and its run-length is the fourth integral invariant. Because the border of a border is an empty set no further manifolds are expected:

$$\partial(\partial\mathcal{P}) = \partial(\partial\mathcal{S}) = \partial\mathscr{L} = \emptyset \tag{6.2}$$

In Chapter 4 two intersections $\mathcal{Q}, \mathscr{M}$ were introduced for a 1D-manifold $\mathbf{l}$ in $\mathbb{R}^3$ leading to further integral invariants defined in Equations 4.9 and 4.11 on page 104:

$$\mathcal{Q} \rightsquigarrow Q^{+,-} \quad \text{and} \quad \mathscr{M} \rightsquigarrow \alpha_\kappa \tag{6.3}$$

Then the following can be observed: The definition of $\mathscr{M}$ is analog to $\mathcal{P}$ except for the dimension of the manifold. The same applies for the definition of $\mathcal{Q}$, which is analog to the definition of $\mathcal{P}$. All definitions rely on the manifolds $\mathcal{M}_1 = \mathbf{l}$ and $\mathcal{M}_2$. Both manifolds are borders: $\mathcal{M}_2$ encloses a volume $\mathbb{V}_\mathcal{M}$ respectively an object. $\mathcal{M}_1$ is a border $\partial\mathcal{C}$ of an area of interest $\mathcal{C}$, which is a 2D-manifold. Therefore the enclosed volume is described by the indicator function $\mathbf{1}_{D_\mathcal{M}}$ and the area of interest can be used to define an indicator function $\mathbf{1}_{D_{\mathcal{M}_1}}$. The latter leads to two additional integral invariants:

$$\mathbb{W}_r := B_r \cap \mathbf{1}_{D_{\mathcal{M}_1}} \rightsquigarrow \hat{W} = \frac{W_r}{|B_r^2|} = \frac{|\mathbb{W}_r|}{|B_r^2|} \quad \text{and} \quad \mathcal{T}_r := \mathbb{S}_r \cap \mathbf{1}_{D_{\mathcal{M}_1}} \rightsquigarrow \hat{T} = \frac{T_r}{|\mathbb{S}_r^2|} = \frac{|\mathcal{T}_r|}{|\mathbb{S}_r^2|} \tag{6.4}$$

The result of this definition together with previously defined $\mathcal{Q}, \mathscr{M}$ is another quadruple of elements. These are used to compute the integral invariants for the border $\mathcal{M}_1 = \partial\mathcal{C}$ with $\alpha_\kappa \rightsquigarrow M$ and without indicating r:

$$W = |\mathbb{W}| \quad , \quad Q = |\mathcal{Q}| \quad , \quad T = |\mathcal{T}| \quad \text{and} \quad M = |\mathscr{M}| \quad . \tag{6.5}$$

The quadruple has the same scheme as the one shown in Equation 6.1 for $\mathcal{M}_2$. This is additionally stressed by the annotation of the set's elements using the same fonts for relatives and letters next within the alphabet.

The use of different fonts for the quadruple's elements already indicate an affiliation of each element to other sets, which contain elements being manifolds having the same dimension. This includes $\mathbb{V} \subset \mathbb{R}^3$ as it is a 3D-manifold in $\mathbb{R}^4$. These sets are named with a number referring to their element's dimension:

$$\mathbf{3}_{\mathcal{M}} \coloneqq \left\{ \mathbb{V}(\mathbf{p}) \mid \mathbf{p} \in \mathbb{R}^4 \right\} \tag{6.6}$$

$$\mathbf{2}_{\mathcal{M}} \coloneqq \left\{ \mathcal{P}(\mathbf{p}), \mathcal{S}(\mathbf{p}), \mathbb{W}(\mathbf{p}) \mid \mathbf{p} \in \mathbb{R}^3 \right\} \tag{6.7}$$

$$\mathbf{1}_{\mathcal{M}} \coloneqq \left\{ \mathscr{L}(\mathbf{p}), \mathcal{Q}(\mathbf{p}), \mathcal{T}(\mathbf{p}) \mid \mathbf{p} \in \mathbb{R}^3 \right\} \tag{6.8}$$

$$\mathbf{0}_{\mathcal{M}} \coloneqq \left\{ \mathscr{M}(\mathbf{p}) \mid \mathbf{p} \in \mathbb{R}^3 \right\} \tag{6.9}$$

Then the intersections between the two manifolds, their indicator functions, ball and sphere, can be organized in tables:

$$\mathcal{M}_2 \quad \rightsquigarrow \quad \begin{array}{c|c:c} \cap & B^3 & \mathbb{S}^2 \\ \hline \mathbf{1}_{D_{\mathcal{M}_2}} & \mathbb{V} & \mathcal{S} \\ \hdashline \mathcal{M}_2 & \mathcal{P} & \mathscr{L} \end{array} \quad \in \quad \begin{array}{c:c} \mathbf{3}_{\mathcal{M}} & \mathbf{2}_{\mathcal{M}} \\ \hdashline \mathbf{2}_{\mathcal{M}} & \mathbf{1}_{\mathcal{M}} \end{array} \tag{6.10}$$

$$\mathcal{M}_1 \quad \rightsquigarrow \quad \begin{array}{c|c:c} \cap & B^3 & \mathbb{S}^2 \\ \hline \mathbf{1}_{D_{\mathcal{M}_1}} & \mathbb{W} & \mathcal{T} \\ \hdashline \mathcal{M}_1 & \mathcal{Q} & \mathscr{M} \end{array} \quad \in \quad \begin{array}{c:c} \mathbf{2}_{\mathcal{M}} & \mathbf{1}_{\mathcal{M}} \\ \hdashline \mathbf{1}_{\mathcal{M}} & \mathbf{0}_{\mathcal{M}} \end{array} \tag{6.11}$$

Note that previous chapters use $\mathbf{1}_{D_{\mathcal{M}}}$, B and $\mathbb{S}$ as abbreviation for $\mathbf{1}_{D_{\mathcal{M}_2}}$, B^3 and $\mathbb{S}^2$.

The eight integral invariants from both quadruples are generally normalized. Therefore either the volume and the surface area of the ball B or the area and circumference of a disk B^2 are used. The intervals of the normalized integral invariants are finite or infinite as shown in the tabular representation:

$$\left.\begin{array}{c:c} V & S \\ \hdashline P & L \end{array} \Big/ \begin{array}{|c:c|} \hline B^3 & \mathbb{S}^2 \\ \hdashline B^2 & \mathbb{S}^1 \\ \hline \end{array} \right\} \quad \left.\begin{array}{c:c} W & T \\ \hdashline Q & M \end{array} \Big/ \begin{array}{|c:c|} \hline B^2 & \mathbb{S}^1 \\ \hdashline B^1 & \mathbb{S}^0 \\ \hline \end{array} \right\} \quad \longmapsto \quad \left]\begin{array}{c:c} -1,+1 & -1,+1 \\ \hdashline 0,+\infty & 0,+\infty \end{array}\right[\tag{6.12}$$

Note that in this notation $\hat{M} = |\mathscr{M}|/|\mathbb{S}^0|$ has no clear definition, because $\mathscr{M}$ contains two points and $1/|\mathbb{S}^0| = \infty$. Using an alternate normalization to $]-1,+\infty[$ instead of $]0,+\infty[$ in Equation 6.12 is more conform for all integral invariants. Then increasing values larger than +1 mean surfaces with increasing roughness towards fractals.

However, it is possible to compute an angle $\alpha_\kappa \mapsto]0,\pi]$ using the points in $\mathscr{M}$ and the center of the ball B. As Q is actually the run-length within the ball and $\mathbf{l} = \partial\mathcal{C}$ has an orientation, it can be split into two separate integral invariants namely Q^+ and Q^- depending on the direction provided by $\mathbf{l}$. Similar integral invariants like $Q^{+,-}$ and $M \rightsquigarrow \alpha_\kappa$ are only observed for $\mathcal{M}_1$ embedded in $\mathbb{R}^2$ shown as an example in Figure 3.1 on page 43.

Quadruples of Integral Invariants for High-Dimensional Surfaces

The attentive reader of this thesis will have noticed that processing a 2D-manifold describing a 3D-dimensional object leads to high-dimensional feature spaces shown in Section 3.5 on pages 78ff. As segmenting these spaces is an important step of character extraction it is implicative that a segmented subset of this space has a hypersurface as border. Therefore the properties of integral invariants for curvature estimation and feature detection are extended to higher dimensional surfaces. Such future work is of importance for the wide range of high-dimensional data-sets.

Let $n \in \mathbb{Z}^+$ be the dimension for a high dimensional (feature) space $\mathbb{R}^n$. Assume there is an m-dimensional manifold $\mathcal{M}_m$ with $m \geq 3$, which has a border $\partial\mathcal{M}_m$ and $m < n$. By $\mathbf{1}_{D_{\mathcal{M}_m}} \curvearrowright \mathcal{M}_m$ a quadruple of integral invariants for higher dimensional surfaces $\mathcal{M}_{m-1} = \partial\mathcal{M}_m$ can be assumed. The m-dimensional manifold and its border can be intersected with balls B^n and spheres $\mathbb{S}^{n-1}$. Equation 3.1 on page 42 shows their volume and surface area.

Following the pattern observed for $\mathcal{M}_1, \mathcal{M}_2$ embedded in $\mathbb{R}^3$, leads to quadruples of integral invariants, which are computed using the norm $|\cdot|$ of the intersections:

$$U = |\mathbb{U}| \quad , \quad O = |\mathcal{O}| \quad , \quad R = |\mathcal{R}| \quad \text{and} \quad K = |\mathscr{K}| \quad . \tag{6.13}$$

Note that this pattern is applicable for $\mathcal{M}_1 = \partial\mathbf{1}_{D_{\mathcal{M}_1}}$ embedded in $\mathbb{R}^2$ of the initial example for integral invariants on page 43.

Assigning the intersections to sets $(k)_{\mathcal{M}}$ by their dimension k is assumed to follow the same pattern using the tabular representation:

$$\mathcal{M}_m \in \mathbb{R}^{n>m} \quad \rightsquigarrow \quad \begin{array}{c|c:c} \cap & B^m & \mathbb{S}^{m-1} \\ \hline \mathcal{M}_m & \mathbb{U} & \mathcal{R} \\ \mathcal{M}_{m-1} & \mathcal{O} & \mathscr{K} \end{array} \quad \in \quad \begin{array}{c:c} (m)_{\mathcal{M}} & (m-1)_{\mathcal{M}} \\ (m-1)_{\mathcal{M}} & (m-2)_{\mathcal{M}} \end{array} \tag{6.14}$$

Consequently the normalized integral invariants have to be computed using higher dimensional balls, spheres, disks and circles:

$$\hat{U} = \frac{U}{|B^m|} \quad , \quad \hat{O} = \frac{O}{|B^{m-1}|} \quad , \quad \hat{R} = \frac{R}{|B^{m-1}|} \quad , \quad \hat{K} = \frac{R}{|\mathbb{S}^{m-2}|} \tag{6.15}$$

It is still to be proven, if there exist quadruples of integral invariants as described in Equation 6.16 and if the following assumption about their intervals holds:

$$\begin{array}{c:c} U & R \\ O & K \end{array} \; / \; \left| \begin{array}{c:c} B^m & B^{m-1} \\ B^{m-1} & \mathbb{S}^{m-2} \end{array} \right| \quad \longmapsto \quad \left] \begin{array}{c:c} -1,+1 & -1,+1 \\ 0,+\infty & 0,+\infty \end{array} \right[\tag{6.16}$$

Note that the intervals for $\hat{U}$ and $\hat{R}$ are $]0,1[$ regarding Equation 6.15, while the above equation is normalized as intended in the corresponding Equation 6.12.

Mesh Processing Pipeline

The integral invariants are the principles part of this thesis, and the curvature measures operating in high dimensional feature spaces are implemented in the *GigaMesh* framework for the purpose of character extraction. The development of this framework covers the challenges for real world application of integral invariants.

Therefore a modular design was chosen separating methods from visualization. A high degree of automation is achieved as the processing pipeline requires only one parameter – the size of the largest feature to be expected – as input. This is the radius r used for the largest scale of spheres and balls. Choosing this parameter is assisted by an automatic determination of the upper and lower bound of r for the given discrete meshes as shown in Section 3.2 on pages 58ff.

The mesh processing will terminate in finite time even in case the size of the largest feature is unknown and a mesh's maximal radius $\mathtt{R}_{max}$ is chosen assuming very large features. This may lead to a very long processing time and probably a larger number of scales is wanted respectively a large feature space is gained, but a result is assured. Therefore the estimators for integral invariants have an additional set of parameters, which allows for tuning either the performance or the accuracy. Furthermore an overall computation complexity less than $O(n \log n)$ was observed for the mesh processing pipeline during hundreds of experiments of a variety of real world meshes.

Cuneiform Character Extraction and Visualization

Particularly in – but not limited to – assyriology, this thesis provides highly efficient and accurate means to prepare cuneiform tablets. Using the simplest mesh processing pipeline the outcomes are raster images computed with local 3D-filtering of the surface. Comparable images e.g. as shown in [CD02] can otherwise only be achieved by a complicated and unhealthy chemical procedure [VGL+05].

Applying an extended mesh processing results in vector based layers representing the cuneiform characters in different stages from accuracy in shape/object representation to accuracy regarding contents as shown in Figure 1.10 on page 15:

- Outlines and skeletons are used to determine
- *Bézier splines*, which are reducible to a
- minimal normalized vector representation.

This minimal vector representation normalized by the height of a written line consists of two pairs of vectors. The major vectors represent the position and direction of a wedge and the minor pair holds information about the type of wedge (⊢ vs ◀). Therefore the required formating for *Optical Character Recognition* (OCR) is given, where the minor vector pair is of importance in terms of semantically meaningful relations between neighboring wedges. Furthermore the automated vector drawings share the same file format (SVG) as manual vector drawings, which allow for application of OCR to both kinds of drawings alike.

Future Work

Having the required ingredients for OCR at hand the future work in the domain of assyriology is expected to be the application of further methods used in pattern recognition, artificial intelligence and computer vision. This will ultimately enable machine processes such as machine translation and text mining. Already in the first steps towards this ultimate goal, the workflow from the acquisition of a primary source to its final publication as secondary source will be dramatically improved in terms of performance and increased (re)search capabilities. As cuneiform script is not the only "writing system in 3D" the vast number of inscriptions existing in virtually any language will benefit from using the same mesh processing techniques. This includes paleographic research questions and questions about manufacturing techniques.

Proving the existence and uniformity of the quadruples of integral invariants for borders of higher dimensional manifolds has to be done. A first ansatz is shown on previous pages. This is of great interest to any application operating on high-dimensional space. An examples serve the feature spaces shown in the second part of this thesis, where metrics have to be applied to perform segmentation in high-dimensional spaces. Due to the robustness of the integral invariants against noise, an even more reliable feature detection might be achieved.

The robust mesh processing pipeline will be suitable to many more applications in the domain of pattern recognition, because more and more measuring devices will provide more and more real world 3D-models. The means of processing irregular meshes intrinsically includes the processing of regular meshes. This is supported by the precise algorithms, which are applicable on meshes for any given resolution. Furthermore any volumetric 3D-model e.g. from *Computer Tomography* (CT) can be transformed into a surface based representation using isosurfaces.

Generally speaking: robust curvature measuring methods embedded in an efficient framework for extraction of features is the key to solve questions in many applications as well as it opens up further research questions.

Appendix A

Pseudocode and Datastructure Examples

This appendix shows additional (pseudo)code to be used within the proposed algorithms as well as examples of datastructures used to represent 3D-models and vector drawings.

A.1 Synthetic Cunei

This Section shows the data-structure of the synthetic cuneus from Section 2.3 as *Wavefront OBJ* file and the computation of the circular sectors of the geodesic discs D_r shown in Figure 2.12 on page 36.

```
% Wavefront OBJ
% --- Vertices ------- Indices begin with 1 ----
v  0.0   0.0   0.0
v  3.0   0.0   0.0
v  0.0   3.0   0.0
v  3.0   3.0   0.0 % Vertex A has index 4
v 12.0   0.0   0.0
v 12.0   6.0   0.0 % Vertex B has index 6
v 15.0   0.0   0.0
v 15.0   6.0   0.0
v 15.0  12.0   0.0
v 12.0  12.0   0.0
v  3.0  12.0   0.0
v  3.0   9.0   0.0 % Vertex C has index 12
v  0.0  12.0   0.0
v  0.0   9.0   0.0
v  4.0   6.0  -3.0 % Vertex D has index 15
% --- Cuneus --------- Triangular Faces --------
f  4 15 12
f  4  6 15
f  6 12 15
% --- Flat surface --- Quadtrianglular Faces ---
f  1  2  4  3
f  5  6  4  2
f  5  7  8  6
f  8  9 10  6
f 10 11 12  6
f 11 13 14 12
f 14  3  4 12
```

To compute the circle sectors, we need the vertices $\mathbf{p}_i$ with $i = \{3, 5, 11, 14\}$ from the *OBJ* file:

$$\mathbf{p}_A = \begin{pmatrix} 3 \\ 3 \\ 0 \end{pmatrix}, \mathbf{p}_B = \begin{pmatrix} 12 \\ 6 \\ 0 \end{pmatrix}, \mathbf{p}_C = \begin{pmatrix} 3 \\ 9 \\ 0 \end{pmatrix}, \mathbf{p}_D = \begin{pmatrix} 4 \\ 6 \\ -3 \end{pmatrix} \tag{A.1}$$

Next we compute the length $|\mathbf{e}|$ of the edges of the three triangular faces describing the cuneus

$$\begin{aligned} |\mathbf{e}_{CA}| &= |\mathbf{p}_C - \mathbf{p}_A| = \sqrt{(3-3)^2 + (9-3)^2 + (0-0)^2} = 6 \\ |\mathbf{e}_{AD}| = |\mathbf{e}_{DC}| &= \sqrt{1 + 2 \cdot 3^2} = \sqrt{19} \\ |\mathbf{e}_{AB}| = |\mathbf{e}_{BC}| &= \sqrt{9^2 + 3^2} = \sqrt{90} \\ |\mathbf{e}_{BD}| &= \sqrt{8^2 + 3^2} = \sqrt{73} \end{aligned}$$

to compute the angles α_i, β_i and γ_i within the triangles $\mathbf{t}_0$, $\mathbf{t}_1$ and $\mathbf{t}_2$ with the following Equations. As the previously computed edge length has no orientation we can assume $|\mathbf{e}_{ij}| = |\mathbf{e}_{ji}|$.

$$\begin{aligned} \alpha_0 = \gamma_0 = & \quad arccos\left(\frac{|\mathbf{e}_{CA}|^2 + |\mathbf{e}_{AD}|^2 - |\mathbf{e}_{DC}|^2}{2 \cdot |\mathbf{e}_{CA}| \cdot |\mathbf{e}_{AD}|}\right) = arccos\left(\frac{6^2}{2 \cdot 6 \cdot \sqrt{19}}\right) && = 46,5^\circ \\ \beta_0 = & \quad arccos\left(\frac{|\mathbf{e}_{AD}|^2 + |\mathbf{e}_{DC}|^2 - |\mathbf{e}_{CA}|^2}{2 \cdot |\mathbf{e}_{AD}| \cdot |\mathbf{e}_{DC}|}\right) = arccos\left(\frac{19+19-6^2}{2 \cdot \sqrt{19} \cdot \sqrt{19}}\right) && = 87.0^\circ \\ \alpha_1 = \beta_2 = & \quad arccos\left(\frac{|\mathbf{e}_{DA}|^2 + |\mathbf{e}_{AB}|^2 - |\mathbf{e}_{BD}|^2}{2 \cdot |\mathbf{e}_{DA}| \cdot |\mathbf{e}_{AB}|}\right) = arccos\left(\frac{19+90-73}{2 \cdot \sqrt{19} \cdot \sqrt{90}}\right) && = 64.2^\circ \\ \beta_1 = \alpha_2 = & \quad arccos\left(\frac{|\mathbf{e}_{AB}|^2 + |\mathbf{e}_{BD}|^2 - |\mathbf{e}_{DA}|^2}{2 \cdot |\mathbf{e}_{AB}| \cdot |\mathbf{e}_{BD}|}\right) = arccos\left(\frac{90+73-19}{2 \cdot \sqrt{90} \cdot \sqrt{73}}\right) && = 27.3^\circ \\ \gamma_1 = \gamma_2 = & \quad arccos\left(\frac{|\mathbf{e}_{BD}|^2 + |\mathbf{e}_{DA}|^2 - |\mathbf{e}_{AB}|^2}{2 \cdot |\mathbf{e}_{BD}| \cdot |\mathbf{e}_{DA}|}\right) = arccos\left(\frac{73+19-90}{2 \cdot \sqrt{73} \cdot \sqrt{19}}\right) && = 88.5^\circ \end{aligned}$$

With these angles we can determine the circle segments of D_r within the cunei. To compute the complementary circle segments outside the cunei we need to compute the angles α', β' and γ' of the imagined triangle $\mathbf{t}' = \{A, B, C\}$.

$$\begin{aligned} \alpha' = \gamma' = & \quad arccos\left(\frac{|\mathbf{e}_{CA}|^2 + |\mathbf{e}_{AB}|^2 - |\mathbf{e}_{BC}|^2}{2 \cdot |\mathbf{e}_{CA}| \cdot |\mathbf{e}_{AB}|}\right) && = 71,55^\circ \\ \beta' = & \quad arccos\left(\frac{|\mathbf{e}_{AB}|^2 + |\mathbf{e}_{BC}|^2 - |\mathbf{e}_{CA}|^2}{2 \cdot |\mathbf{e}_{AB}| \cdot |\mathbf{e}_{BC}|}\right) && = 36.90^\circ \end{aligned}$$

The complementary circle segments angles α'', β'' and γ'' are computed by subtracting 360° from α', β' and γ'. Figure A.1 shows the computed angles. As the mesh is piecewise flat (linear), we can choose r smaller than the shortest edge length $min(|\mathbf{e}_{ij}|) = |\mathbf{e}_{AD}| = \sqrt{19}$ to compute the area $A(r)$ of the geodesic discs D_r required by the *Bertrand-Diquet-Puiseux* theorem [BDP48, Spi79] as shown in Equation 2.26 on page 36. With e.g. $r = 1$ we can compute the Gaussian curvature κ_G:

$$\mathbf{p}_A, \mathbf{p}_C: \quad A = \pi\frac{\alpha_0+\alpha_1+\alpha''}{360°} = \pi\frac{399.15°}{360°} = 1.11\cdot\pi \quad \kappa_G = 12\frac{\pi - 1.11\cdot\pi}{\pi} = -1.3$$

$$\mathbf{p}_B: \quad A = \pi\frac{\beta_1+\alpha_2+\beta''}{360°} = \pi\frac{377.70°}{360°} = 1.05\cdot\pi \quad \kappa_G = 12\frac{\pi - 1.05\cdot\pi}{\pi} = -0.59$$

$$\mathbf{p}_D: \quad A = \pi\frac{\beta_0+\gamma_1+\gamma_2}{360°} = \pi\frac{264.00°}{360°} = 0.73\cdot\pi \quad \kappa_G = 12\frac{\pi - 0.73\cdot\pi}{\pi} = +3.2$$

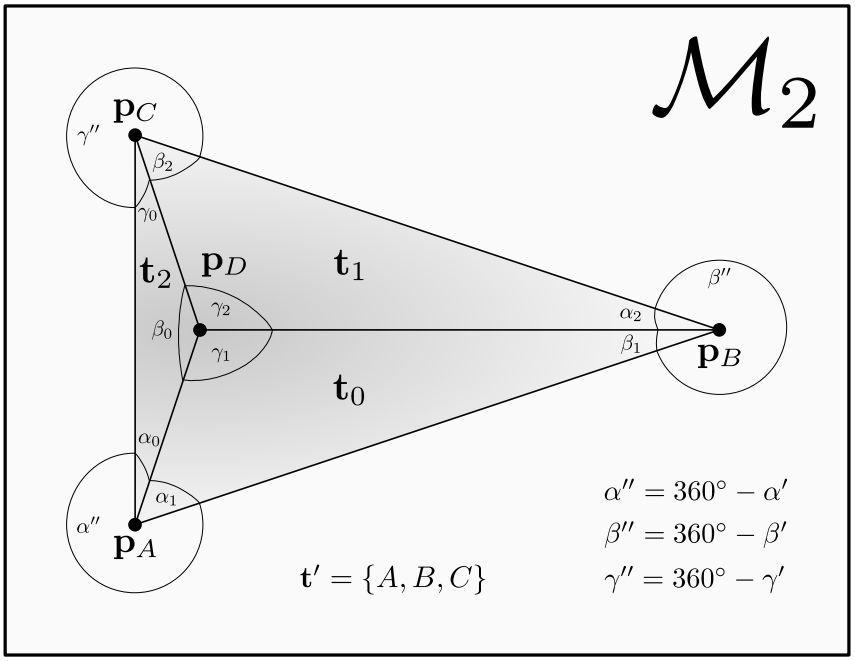
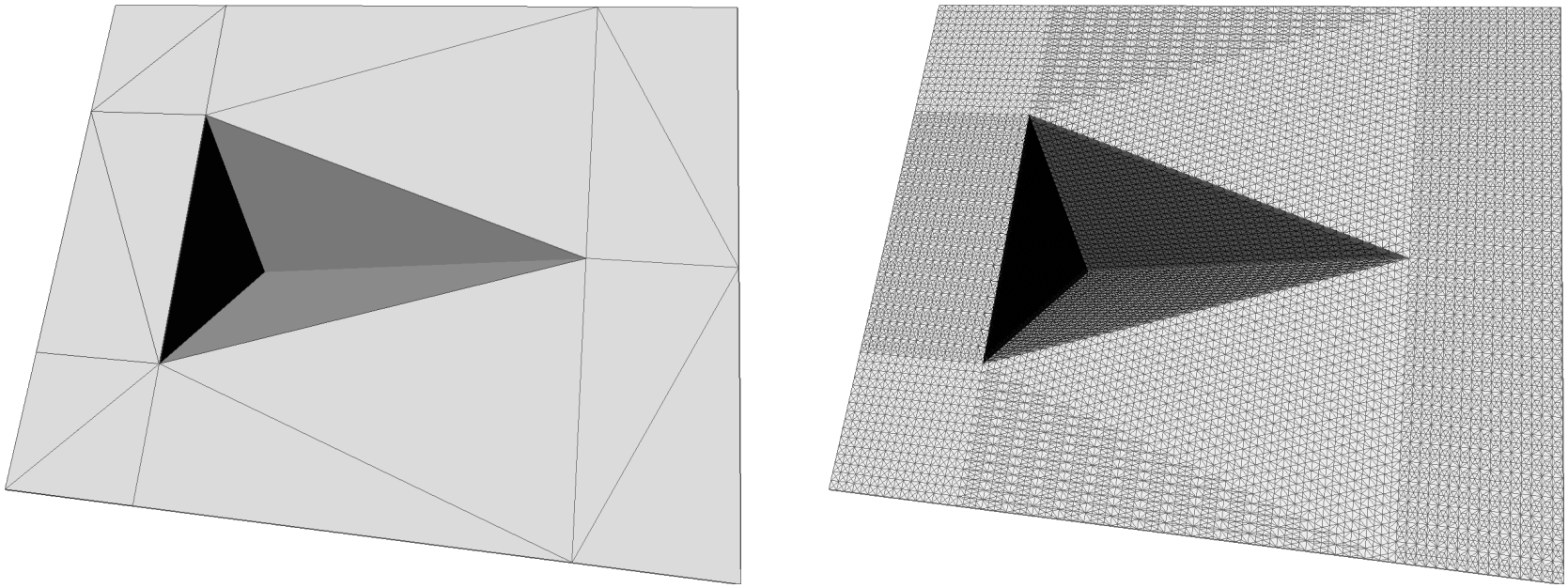


Figure A.1: Ideal cuneus and the angles used to compute the geodesic discs D_r.



(a) **(b)**

Figure A.2: Ideal, synthetic horizontal wedge as (a) minimal mesh as shown as *Wavefront OBJ* file and (b) subdivided using the mid-edge scheme [PR97, HW99].

A.2 Imaginary numbers, $\not\geq$ and < in C++

In Algorithm 3 some of the cells of the arrays become imaginary numbers, when $x^2+y^2 > r^2$ within $\sqrt{r^2-x^2-y^2}$. As programming languages may not allow imaginary numbers, the result of extracting the root of negative numbers leads to an error. In *C++* a global (domain) error is set and the `sqrt` ($\sqrt{\ldots}$) returns *not a number* (`NaN`), which is handled like any other floating-point number e.g. of type `double`. Therefore all the operations available for `double` will accept not a number and comparison operations like "<" and ">" will return `false`, when one of the arguments is not a number . This deterministic behavior can be used to decide if one value is smaller or larger than the other and to determine if one or both values are `NaN`, which allows to skip cells with imaginary numbers. In consequence `not( a >= b )` ($\not\geq$) has other return values than `a < b` (<) as demonstrated by the following source code example. The output of this example is shown in Table A.1.

```
1  #include <cmath>    // pow, sqrt, floor, exp, ...
2  #include <iostream> // cout, endl, ...
3
4  using namespace std;
5
6  int main( void ) {
7    double ab[4] = { -1.0, 0.0, 1.0, sqrt( -1.0 ) };
8    /* a < b */
9    for( int i=0; i<4; i++ ) {
10     for( int j=0; j<4; j++ ) {
11       cout << "" << ab[i] << " < " << ab[j] << " = ";
12       cout << ( ab[i] < ab[j] ) << endl;
13     }
14   }
15   /* not( a >= b ) */
16   for( int i=0; i<4; i++ ) {
17     for( int j=0; j<4; j++ ) {
18       cout << "not( " << ab[i] << " >= " << ab[j] << " ) = ";
19       cout << not( ab[i] >= ab[j] ) << endl;
20     }
21   }
22 }
```

		a			
	<	-1	0	1	NaN
b	-1	false	true	true	false
	0	false	false	true	false
	1	false	false	false	false
	NaN	false	false	false	false

(a) `not( a >= b )`

		a			
	$\not\geq$	-1	0	1	NaN
b	-1	false	true	true	true
	0	false	false	true	true
	1	false	false	false	true
	NaN	true	true	true	true

(b) `a < b`

Table A.1: Comparison of $\not\geq$ and < in *C++*

A.3 Synthetic 3D-Model of the Cuneiform Sign "kaskal"

Wavefront OBJ file defining the cuneiform sign 𒆜 known as *kaskal* used in Section 4.3 on pages 113ff. This file describes the sign i.e. area of interest $\mathcal{C}_{\text{𒆜}}$ without the surrounding mesh i.e. background $\mathcal{C}_{\emptyset}$. This definition describes two horizontal and two vertical wedges, which have to be rotated by 45° for proper representation of the sign. Figure A.3 shows the properly orientated 3D-model. The *Unicode* number of this sign is `0x1219C`.

```
# Wavefront OBJ
# ==== Vertices ======================
# --- Head -- Horiz. wedge, bottom --
v  15  35 -7.5
v   5  15    0
v  30  30    0
v  40  35   -5
v  30  40    0
v   5  55    0
# --- Head -- Horiz. wedge, top -----
v  15  80 -7.5
v   5  60    0
v  30  75    0
v  40  80   -5
v  30  85    0
v   5 100    0
# --- Head -- Vert. wedge, left -----
v  50 115 -7.5
v  30 125    0
v  45 100    0
v  50  90   -5
v  55 100    0
v  70 125    0
# --- Head -- Vert. wedge, right ----
v  95 115 -7.5
v  75 125    0
v  90 100    0
v  95  90   -5
v 100 100    0
v 115 125    0
# --- Head-Intersect, horiz, bot. ---
v  45  30    0
v  50  35 -5.5
v  45  40    0
# --- Head-Intersect, horiz, top ----
v  45  75    0
v  50  80 -5.5
v  45  85    0
# --- Head-Intersect, vert, left ----
v  55  85    0
# --- Head-Intersect, vert, right ---
v  90  85    0
v  95  80 -5.5
v 100  85    0
# --- Tip, vertical, left -----------
v  50  15   -5
v  45  10    0
v  50   5    0
v  55  10    0
v  55  30    0
# --- Line, vertical, left ----------
v  55  40    0
v  55  75    0
# --- Line, horizontal, top ---------
v  90  75    0
# --- Tip, horizontal, top ----------
v 100  75    0
v 120  75    0
v 125  80    0
v 120  85    0
v 115  80   -5
# --- Line, horizontal, bottom and
#           vertical, right ---------
v  95  35 -5.5
v  90  40    0
v  90  30    0
v 100  30    0
v 100  40    0
# --- Tip, vertical, right ----------
v  95  15   -5
v  90  10    0
v  95   5    0
v 100  10    0
# --- Tip, horizontal, bottom -------
v 115 35    -5
v 120 30     0
v 125 35     0
v 120 40     0
# ==== Faces ========================
# --- Head -- Horiz. wedge, bottom --
f  1  2  3  4
f  1  4  5  6
f  1  6  2
# --- Head -- Horiz. wedge, top -----
f  7  8  9 10
f  7 10 11 12
f  7 12  8
# --- Head -- Vert. wedge, left -----
f 13 14 15 16
f 13 16 17 18
f 13 18 14
# --- Head -- Vert. wedge, right ----
```

```
f 19 20 21 22
f 19 22 23 24
f 19 24 20
% --- Head-Intersect, horiz, bot. ---
f 4  3 25 26
f 4 26 27  5
% --- Head-Intersect, horiz, top ----
f 10  9 28 29
f 10 29 30 11
% --- Head-Intersect, vert, left ----
f 16 15 30 29
f 16 29 31 17
% --- Head-Intersect, vert, right ---
f 22 21 32 33
f 22 33 34 23
% --- Tip, vertical, left -----------
f 26 25 36 35
f 35 36 37
f 35 37 38
f 26 35 38 39
% --- Line, vertical, left ----------
f 26 29 28 27
f 26 40 41 29
% --- Line, horizontal, top ---------
f 29 33 32 31
f 29 41 42 33
% --- Tip, horizontal, top ----------
f 33 47 46 34
f 33 43 44 47
f 47 44 45
f 47 45 46
% --- Line, horizontal, bottom and
%           vertical, right ---------
f 26 39 50 48
f 26 48 49 40
f 33 42 49 48
f 33 48 52 43
% --- Tip, vertical, right ----------
f 48 50 54 53
f 48 53 56 51
f 53 54 55
f 53 55 56
% --- Tip, horizontal, bottom -------
f 48 51 58 57
f 48 57 60 52
f 57 58 59
f 57 59 60
%-----------------------------------
```

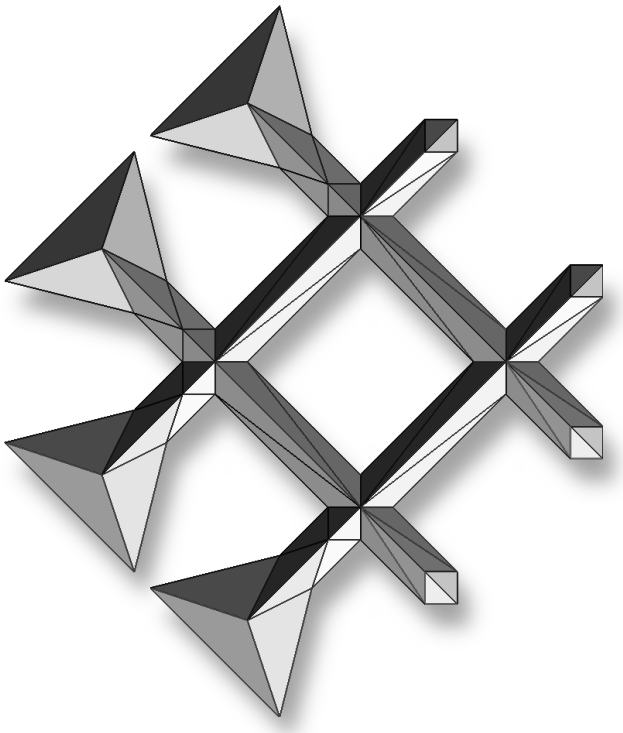


Figure A.3: Synthetic 3D-model of the cuneiform sign *kaskal* having the *Unicode* number `0x1219C`. It consists of 60 vertices and 76 triangular faces.

A.4 Bézier Spline Representation of the Cuneiform Sign "kaskal"

The following source code describes the cuneiform sign *kaskal* using the programming language *tikz*. It is designed for implementing vector graphics and uses *Bézier splines*. Therefore points of the sign's outline are connected by straight lines denoted by "`--`" and by segments of quadratic *Bézier splines* having two control points. These two points are denoted by "`..controls(` x_1, y_1 `)and(` x_2, y_2 `)..`".

This sign becomes the logo ⊛ of the *GigaMesh* software framework by adding a circle $\mathbb{S}^1$ (sphere in $\mathbb{R}^2$). A circle is described with a *Bézier spline* using five points, where the first point equals the last one. The control points are located symmetric on the tangent at the points of the circle.

```
\newcommand{\GIGAMESHLOGO}[1] {
\protect\tikz[baseline=0pt] \fill[scale=0.045*#1]
%--- Circle, outer line ------------------------------------------------
(3.411374,6.812836)..controls(1.524824,6.812836)and(0.000004,5.282836)..
(0.000000,3.406416)..controls(0.000000,1.529996)and(1.524824,0.000000)..
(3.411374,0.000000)..controls(5.297914,0.000000)and(6.822744,1.529996)..
(6.822744,3.406416)..controls(6.822744,5.282836)and(5.297914,6.812836)..
(3.411374,6.812836)
%--- Circle, inner line ------------------------------------------------
(3.411374,6.628306)..controls(5.183014,6.628306)and(6.624434,5.190366)..
(6.624434,3.406416)..controls(6.624434,1.622466)and(5.183014,0.184526)..
(3.411374,0.184526)..controls(1.639724,0.184526)and(0.198304,1.622466)..
(0.198304,3.406416)..controls(0.198304,5.190366)and(1.639724,6.628306)..
(3.411374,6.628306)
%--- Kaskal, outer line ------------------------------------------------
(5.665267,4.222444)..controls(5.665262,4.211122)and(5.661019,4.201218)..
(5.652533,4.192731)--(4.720842,3.337442)--
(5.478505,2.567045)..controls(5.484150,2.558559)and(5.486991,2.550067)..
(5.486991,2.541578)..controls(5.486987,2.518939)and(5.472131,2.492764)..
(5.442422,2.463052)..controls(5.412705,2.434754)and(5.386531,2.420606)..
(5.363898,2.420607)..controls(5.352574,2.420606)and(5.344085,2.424855)..
(5.338430,2.433340)--(4.582890,3.205859)--(3.740334,2.433340)--
(4.587134,1.567439)..controls(4.592795,1.558953)and(4.595621,1.550462)..
(4.595621,1.541972)..controls(4.595616,1.524994)and(4.579345,1.498820)..
(4.546807,1.463447)..controls(4.512846,1.436565)and(4.488794,1.421709)..
(4.474648,1.418879)..controls(4.463326,1.418879)and(4.454130,1.423834)..
(4.447059,1.433735)--(3.600259,2.301758)--
(3.364683,2.085282)..controls(3.339213,2.062645)and(3.298890,2.000390)..
(3.243712,1.898520)..controls(3.099393,1.635354)and(3.027235,1.339647)..
(3.027237,1.011396)..controls(2.969226,0.961877)and(2.935976,0.937116)..
(2.927488,0.937115)..controls(2.907678,0.939951)and(2.877258,0.963292)..
(2.836223,1.007151)--
(2.713136,1.251217)..controls(2.601359,1.471936)and(2.423086,1.641013)..
(2.178315,1.758448)..controls(2.174066,1.761273)and(2.154262,1.773304)..
(2.118891,1.794527)..controls(2.094837,1.815750)and(2.074321,1.846169)..
(2.057344,1.885785)--(2.070078,1.911253)--
(2.140114,1.977045)..controls(2.213686,1.971384)and(2.287259,1.968559)..
(2.360833,1.968559)..controls(2.648050,1.968559)and(2.892822,2.025861)..
```

```
(3.095151,2.140465)..controls(3.117786,2.153199)and(3.164477,2.182203)..
(3.235222,2.227480)--(3.460188,2.437588)--(2.662201,3.258920)--
(2.464826,3.080647)..controls(2.435113,3.053763)and(2.394789,2.991510)..
(2.343855,2.893884)..controls(2.199537,2.617984)and(2.127379,2.322276)..
(2.127380,2.006760)..controls(2.069369,1.957239)and(2.036120,1.932480)..
(2.027631,1.932480)..controls(2.007822,1.932480)and(1.977402,1.955825)..
(1.936373,2.002516)..controls(1.920808,2.037887)and(1.879777,2.120657)..
(1.813279,2.250826)..controls(1.704333,2.464470)and(1.526060,2.632132)..
(1.278458,2.753811)..controls(1.220448,2.782108)and(1.180124,2.824554)..
(1.157487,2.881150)--(1.170221,2.906617)--
(1.240257,2.972409)..controls(1.313830,2.966747)and(1.387402,2.963922)..
(1.460976,2.963922)..controls(1.748194,2.963921)and(1.992966,3.021223)..
(2.195294,3.135829)..controls(2.217930,3.148561)and(2.264621,3.177567)..
(2.335365,3.222843)--(2.526374,3.396872)--
(2.352344,3.575146)..controls(2.322630,3.604856)and(2.257547,3.643057)..
(2.157092,3.689749)..controls(1.865628,3.824161)and(1.567091,3.886415)..
(1.261480,3.876512)..controls(1.209129,3.931691)and(1.182954,3.963525)..
(1.182955,3.972016)..controls(1.182954,4.011631)and(1.286239,4.086619)..
(1.492811,4.196980)..controls(1.673914,4.293190)and(1.828841,4.457314)..
(1.957595,4.689356)..controls(1.967495,4.720480)and(1.983063,4.759389)..
(2.004286,4.806082)..controls(2.060880,4.848525)and(2.092714,4.871870)..
(2.099790,4.876118)..controls(2.108277,4.876116)and(2.138698,4.848525)..
(2.191049,4.793348)--
(2.191049,4.727557)..controls(2.191048,4.378082)and(2.250472,4.100060)..
(2.369322,3.893491)..controls(2.399034,3.842554)and(2.438650,3.781714)..
(2.488171,3.710973)--(2.666445,3.528455)--(3.509001,4.305219)--
(3.243712,4.574751)..controls(3.213998,4.605875)and(3.148913,4.644077)..
(3.048460,4.689355)..controls(2.754165,4.823766)and(2.455629,4.886019)..
(2.152848,4.876118)..controls(2.100496,4.931295)and(2.074321,4.963130)..
(2.074322,4.971622)..controls(2.074321,5.011235)and(2.177607,5.086223)..
(2.384179,5.196586)..controls(2.565280,5.292794)and(2.720208,5.456918)..
(2.848964,5.688961)..controls(2.858863,5.720085)and(2.874429,5.758993)..
(2.895654,5.805687)..controls(2.952247,5.848130)and(2.984082,5.871474)..
(2.991158,5.875723)..controls(2.999644,5.875720)and(3.030064,5.848130)..
(3.082417,5.792953)--
(3.082417,5.727161)..controls(3.082414,5.377686)and(3.141839,5.099665)..
(3.260691,4.893096)..controls(3.290400,4.842159)and(3.330017,4.781319)..
(3.379540,4.710579)--(3.649072,4.432556)--
(4.621088,5.328169)..controls(4.626749,5.333819)and(4.635940,5.336656)..
(4.648678,5.336656)..controls(4.668482,5.336652)and(4.692535,5.321796)..
(4.720836,5.292088)..controls(4.750544,5.262372)and(4.765401,5.237612)..
(4.765405,5.217806)..controls(4.765401,5.206485)and(4.761156,5.196581)..
(4.752671,5.188095)--(3.782778,4.296727)--(4.587131,3.47540)--
(5.520945,4.332806)..controls(5.522357,4.334218)and(5.524476,4.335631)..
(5.527312,4.337054)..controls(5.531550,4.339880)and(5.537918,4.341292)..
(5.546414,4.341303)..controls(5.567631,4.341301)and(5.592392,4.326445)..
(5.620694,4.296734)..controls(5.650401,4.268434)and(5.665258,4.243675)..
(5.665263,4.222454)
%--- Kaskal, inner line (rhomb) ----------------------------------------
(4.447057,3.345946)--(3.642703,4.167279)--(2.800149,3.388393)--
(3.604502,2.567060)--(4.447057,3.345946)
;}
```

Appendix B

3D Models

This appendix contains additional visualizations of real-world 3D-models used within this thesis. Synthetic 3D-models are shown in Appendix A.

The majority of 3D-models for testing the *GigaMesh* software framework described in Chapter 4 are objects with cuneiform scripts. These objects are typically tablets and were acquired in *Graz, Austria*; *Heidelberg*; and *Berlin, Germany* and are shown in the next three sections. At the end of this section the key data of the 3D-models is shown in Table B.1 on page 194.

B.1 Sammlung der Assyriologie Heidelberg

The *Sammlung des Seminars für Sprachen und Kulturen des Vorderen Orients, Assyriologie, Universität Heidelberg* is a collection of objects within the *Heidelberg University*. It consists of original objects and cast copies. Additional cast copies and objects of past and present projects of the *Heidelberg*'s assyriology group were acquired to demonstrate the large variability of object's shapes and the cuneiform script.

B.1.1 Curvature Estimation Methods — Inv. No. 88/677

This tablet is shown in Figure 2.3 for comparing 3D-scanners; one of it's wedges is shown to visualize the 1-ring neighborhood in Figure 2.6; ambient occlusion is shown in Figure 2.10; a smoothed version is compared to the original 3D-model in Figure 2.11; and one of it's wedges is shown using curvature estimation methods in Figure 2.15 – the same methods are shown in Figure B.1.

This tablet has an almost black surface. A part on the backside is discarded by an inventory sticker as shown in Figure B.2. The figures on the following pages show the *fat-cross* visualization of this tablet using virtual illumination and volume based integral invariants as suggested in Section 3.5.2.

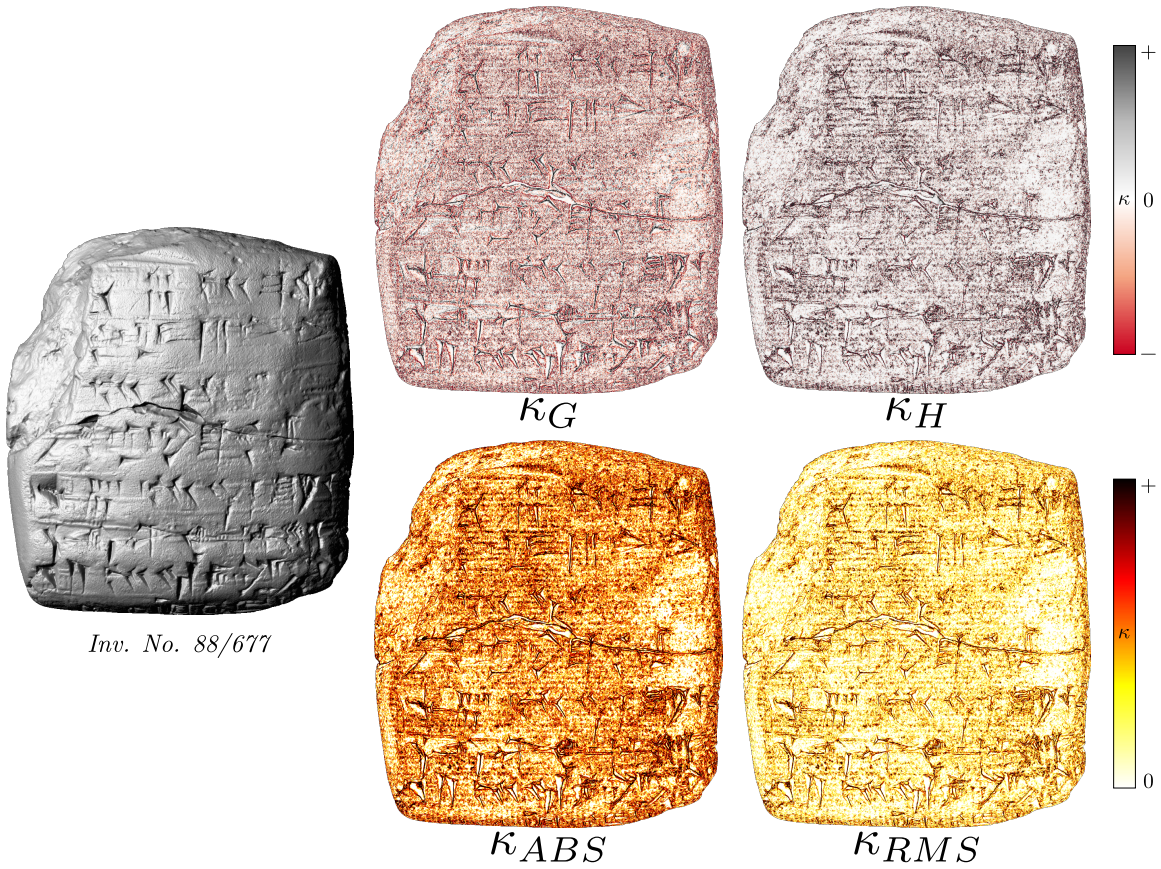


Figure B.1: Visualizations of the mean curvature κ_H, the Gaussian curvature κ_G, the root mean square curvature κ_{RMS} and the the absolute curvature κ_{ABS}. The color ramps indicate the interval of the curvature. Note that κ_H and κ_G can change sign, while κ_{RMS} and κ_{ABS} are only positive.



Figure B.2: Inv. No. 88/677 with monochrome texture map – 8 views, scale 1 : 0.75

Max. dim.	$4{,}4 \times 4{,}9 \times 1{,}8\,cm$	Surface, acq.	$5\,654\,mm^2$
Vertices	5 694 192	Res., avg	$1\,007\,mm^{-2}$
Faces	11 389 024		$806\,DPI$
Material	original	Volume, est.	$24\,cm^3$



Figure B.3: Inv. No. 88/677 with MSII texture map – 6 views, scale 1 : 0.75

Max. dim.	$4{,}4 \times 4{,}9 \times 1{,}8\,cm$	Surface, acq.	$5\,654\,mm^2$
Vertices	5 694 192	Res., avg	$1\,007\,mm^{-2}$
Faces	11 389 024		$806\,DPI$
Material	original	Volume, est.	$24\,cm^3$

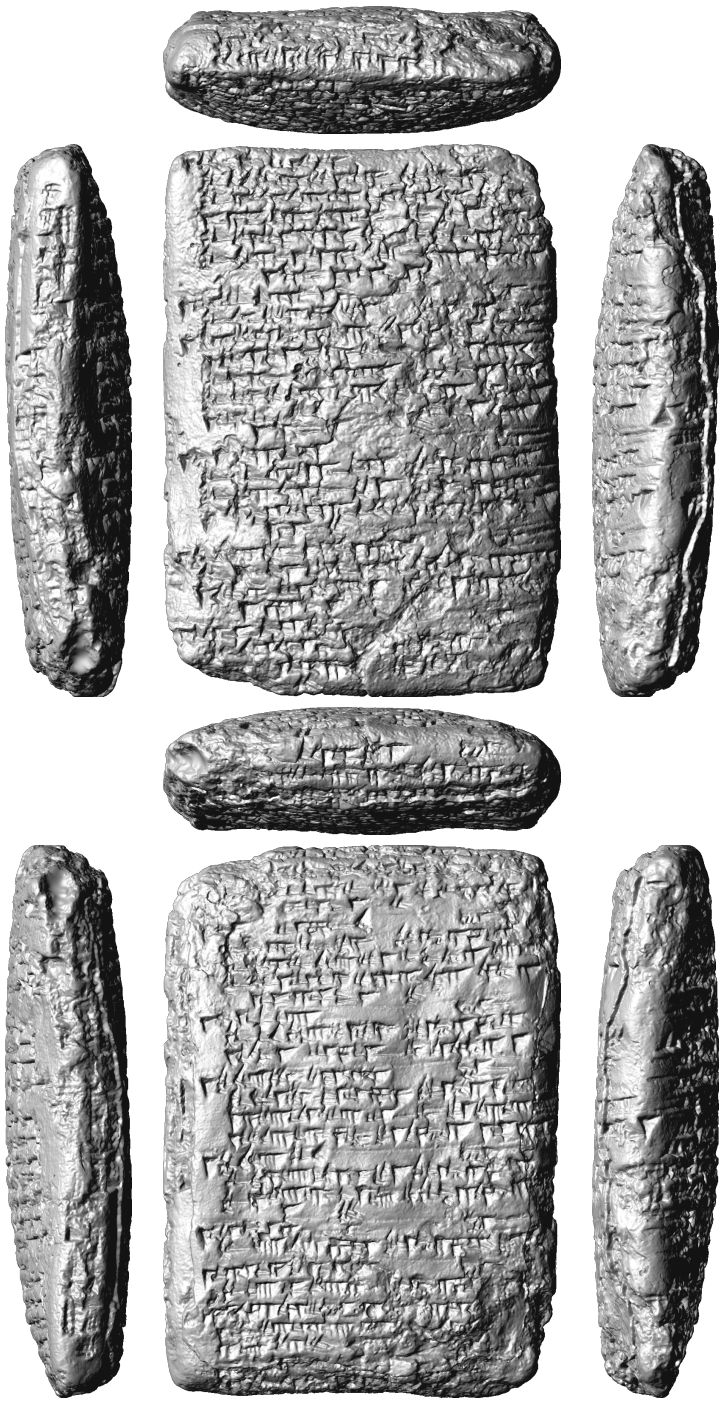


Figure B.4: TCH.92,G.127 with monochrome texture map – 8 views, scale 1 : 1.5

Max. dim.	$8 \times 11{,}1 \times 2{,}5\,cm$	Surface, acq.	$23\,442\,mm^2$
Vertices	4 466 245	Res., avg	$191\,mm^{-2}$
Faces	8 932 486		$351\,DPI$
Material	cast copy, gypsum	Volume, est.	$141\,cm^3$



Figure B.5: TCH.92,G.127 with MSII texture map – 6 views, scale 1 : 1.5

Max. dim.	$8 \times 11{,}1 \times 2{,}5\,cm$	Surface, acq.	$23\,442\,mm^2$
Vertices	4 466 245	Res., avg	$191\,mm^{-2}$
Faces	8 932 486		$351\,DPI$
Material	cast copy, gypsum	Volume, est.	$141\,cm^3$



Figure B.6: HOS G8 with monochrome texture map – 8 views, scale 1 : 1

Max. dim.	$6{,}2 \times 4{,}6 \times 2{,}9\,cm$	Surface, acq.	$7\,487\,mm^2$
Vertices	344 694	Res., avg	$46\,mm^{-2}$
Faces	689 384		$172\,DPI$
Material	cast copy, ceramic	Volume, est.	$43\,cm^3$



Figure B.7: HOS G8 with MSII texture map – 6 views, scale 1 : 1

Max. dim.	$6{,}2 \times 4{,}6 \times 2{,}9\,cm$	Surface, acq.	$7\,487\,mm^2$
Vertices	344 694	Res., avg	$46\,mm^{-2}$
Faces	689 384		$172\,DPI$
Material	cast copy, ceramic	Volume, est.	$43\,cm^3$

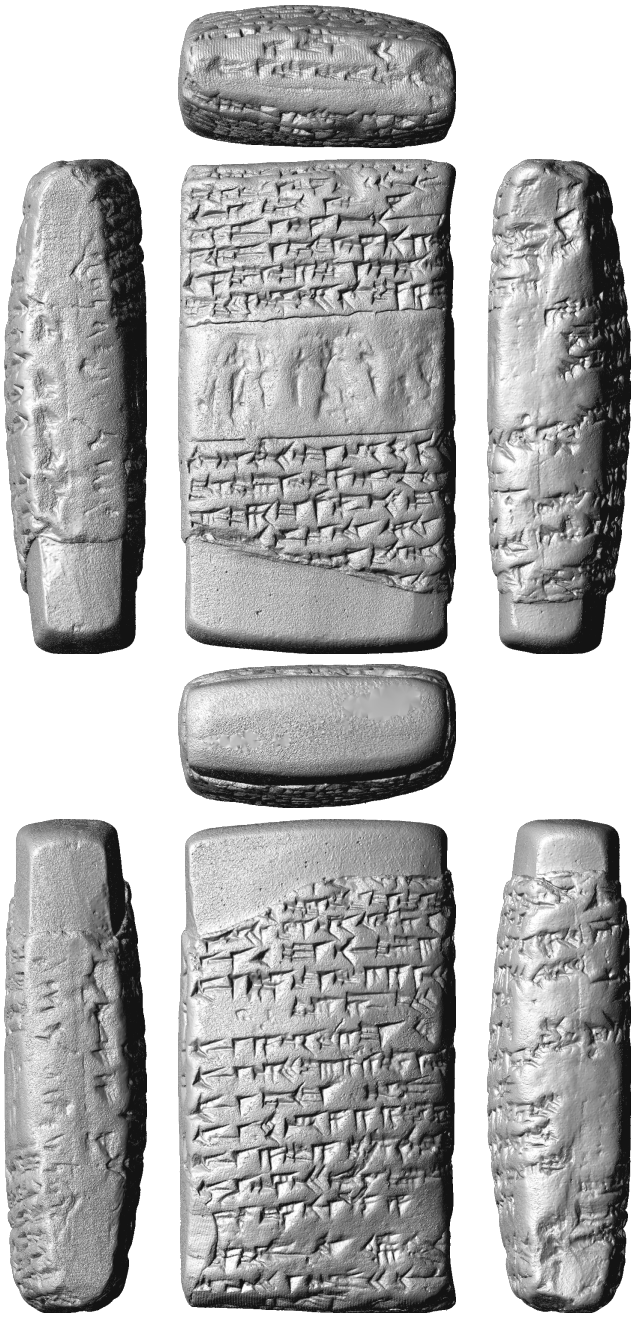


Figure B.8: HOS G10 with monochrome texture map – 8 views, scale 1 : 1

Max. dim.	$3{,}7 \times 6{,}5 \times 1{,}9\,cm$	Surface, acq.	$7\,327\,mm^2$
Vertices	6 596 964	Res., avg	$900\,mm^{-2}$
Faces	13 194 060		$762\,DPI$
Material	cast copy, ceramic	Volume, est.	$33\,cm^3$

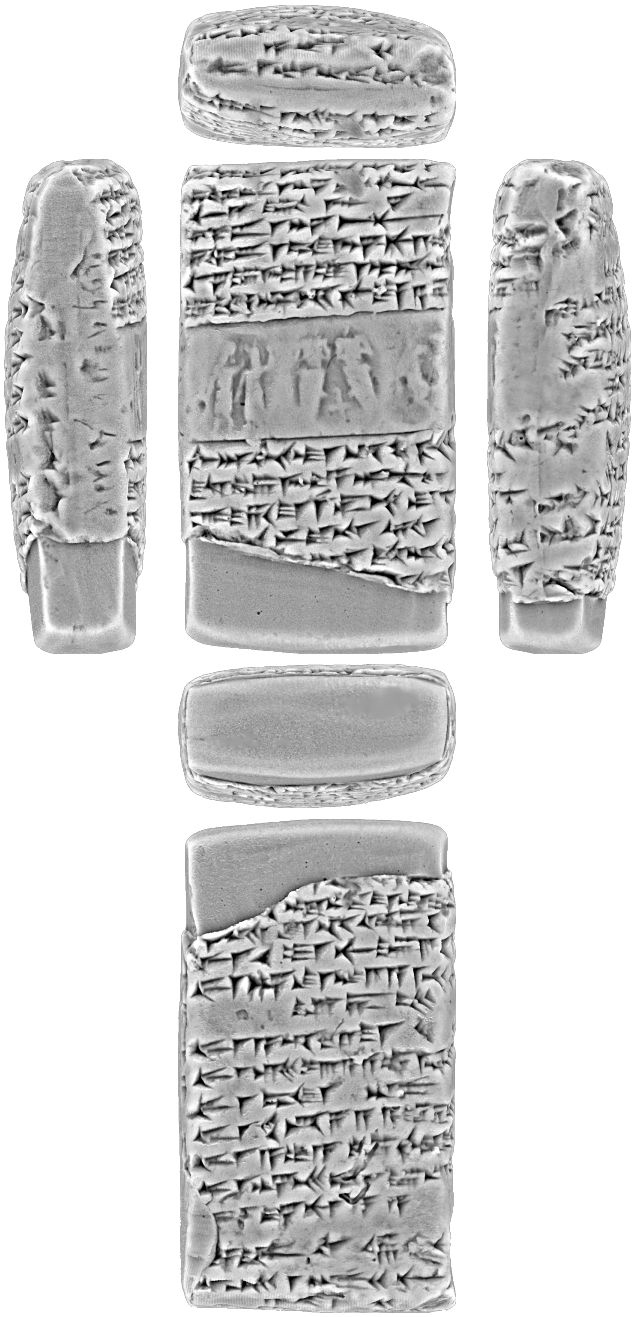


Figure B.9: HOS G10 with MSII texture map – 6 views, scale 1 : 1

Max. dim.	$3{,}7 \times 6{,}5 \times 1{,}9\,cm$	Surface, acq.	$7\,327\,mm^2$
Vertices	6 596 964	Res., avg	$900\,mm^{-2}$
Faces	13 194 060		$762\,DPI$
Material	cast copy, ceramic	Volume, est.	$33\,cm^3$

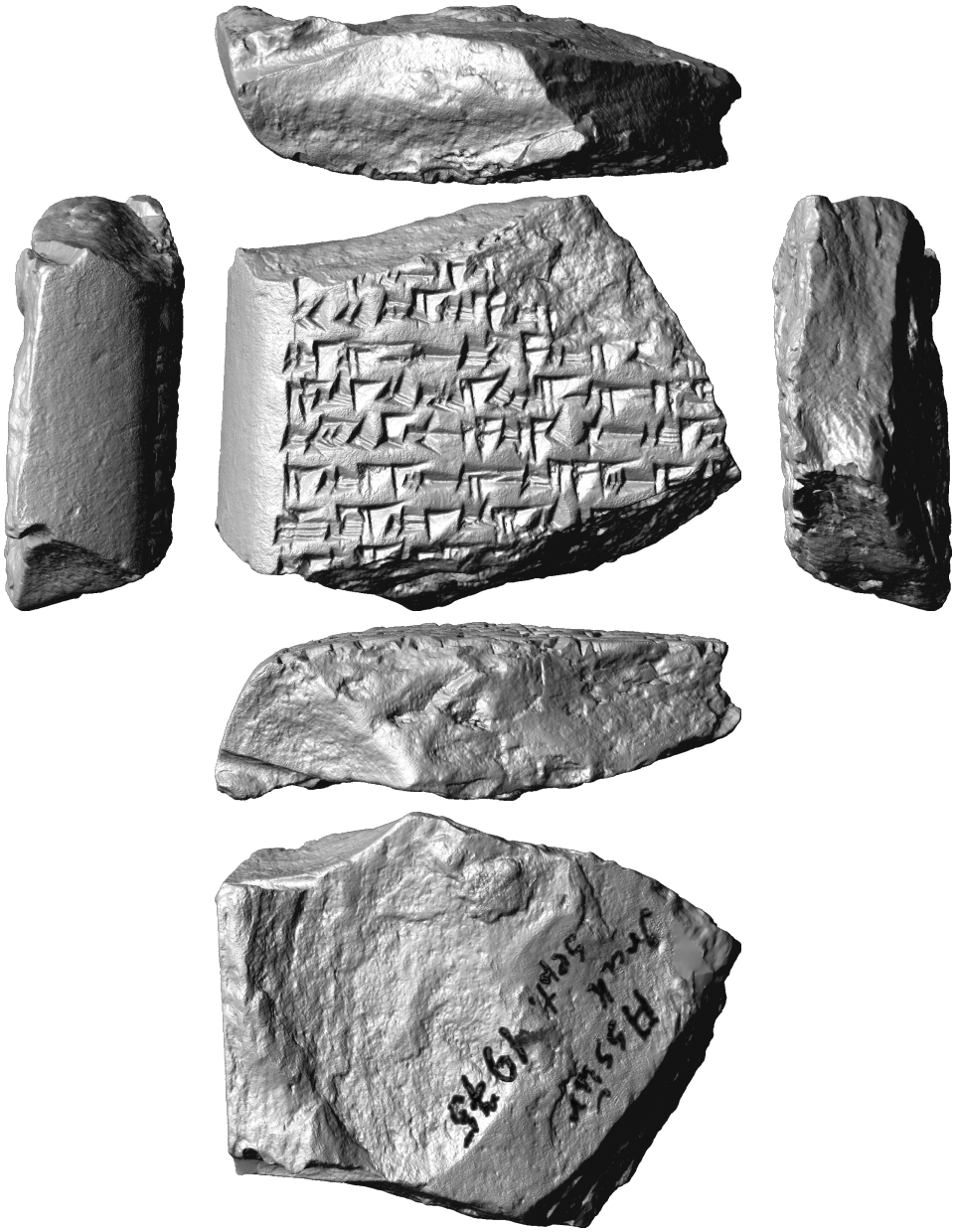


Figure B.10: *Königsinschrift* found 1975 in Assur, Iraq with monochrome texture map – 6 views, scale $1 : 0.75$

Max. dim.	$5{,}2 \times 4{,}1 \times 1{,}8\,cm$	Surface, acq.	$5\,026\,mm^2$
Vertices	3 477 399	Res., avg	$692\,mm^{-2}$
Faces	6 954 802		$668\,DPI$
Material	original, clay	Volume, est.	$18\,cm^3$



Figure B.11: *Königsinschrift* found 1975 in Assur, Iraq with MSII texture map – 6 views, scale $1 : 0.75$

Max. dim.	$5{,}2 \times 4{,}1 \times 1{,}8\,cm$	Surface, acq.	$5\,026\,mm^2$
Vertices	$3\,477\,399$	Res., avg	$692\,mm^{-2}$
Faces	$6\,954\,802$		$668\,DPI$
Material	original, clay	Volume, est.	$18\,cm^3$



Figure B.12: Inv. No. BM 90922, King, Boundary Stones Nr. XXVIII with monochrome texture map – 6 views, scale 1 : 2

Max. dim.	$11{,}9 \times 17{,}7 \times 4{,}3\,cm$	Surface, acq.	$57\,567\,mm^2$
Vertices	12 243 633	Res., avg	$213\,mm^{-2}$
Faces	24 489 764		$370\,DPI$
Material	cast copy, ceramic	Volume, est.	$623\,cm^3$

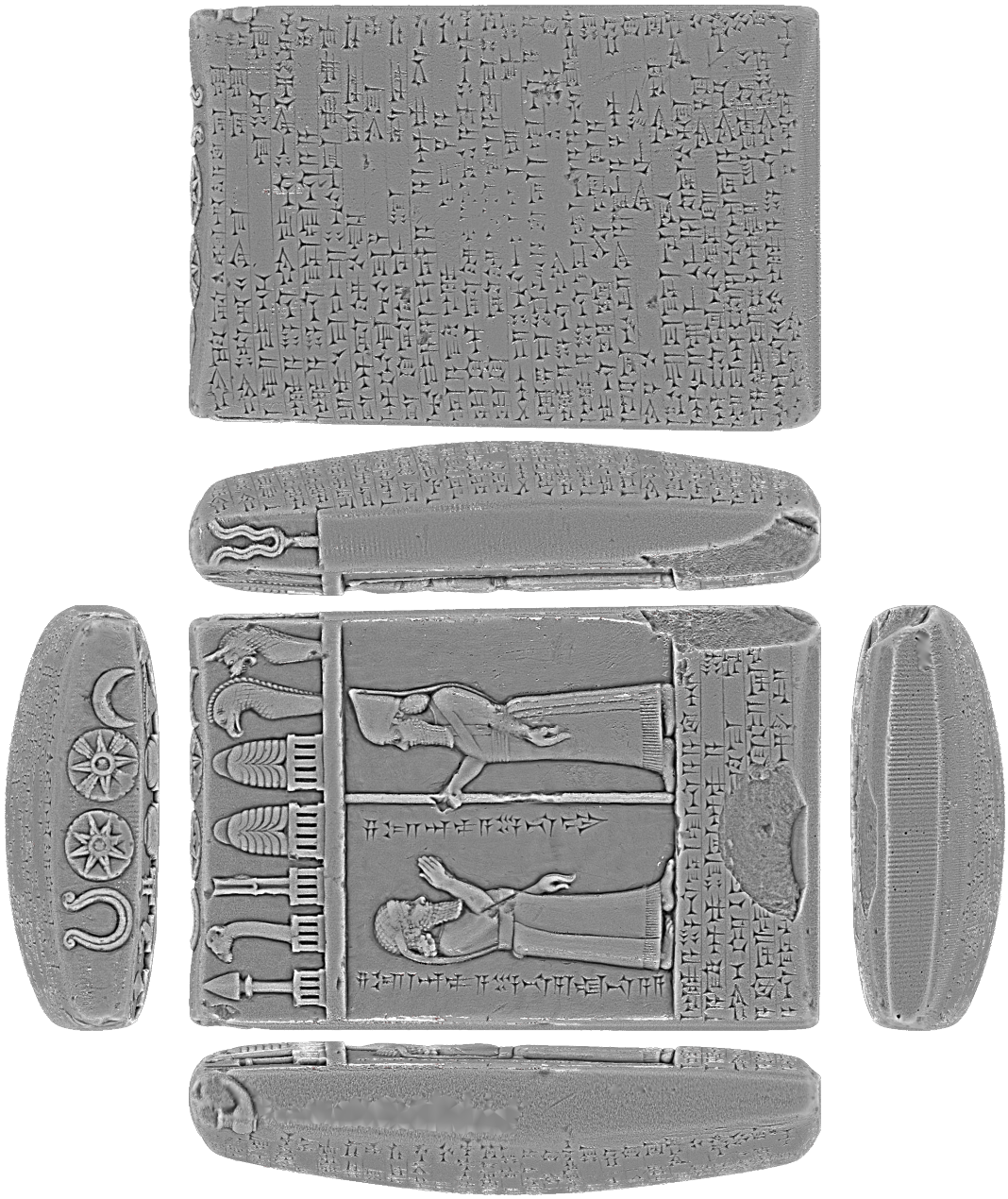


Figure B.13: Inv. No. BM 90922, King, Boundary Stones Nr. XXVIII with MSII texture map – 6 views, scale 1 : 2

Max. dim.	$11{,}9 \times 17{,}7 \times 4{,}3\,cm$	Surface, acq.	$57\,567\,mm^2$
Vertices	12 243 633	Res., avg	$213\,mm^{-2}$
Faces	24 489 764		$370\,DPI$
Material	cast copy, ceramic	Volume, est.	$623\,cm^3$

B.1.2 Unwrapping a Cone — Tonnagel, Text Ur-Baba 7

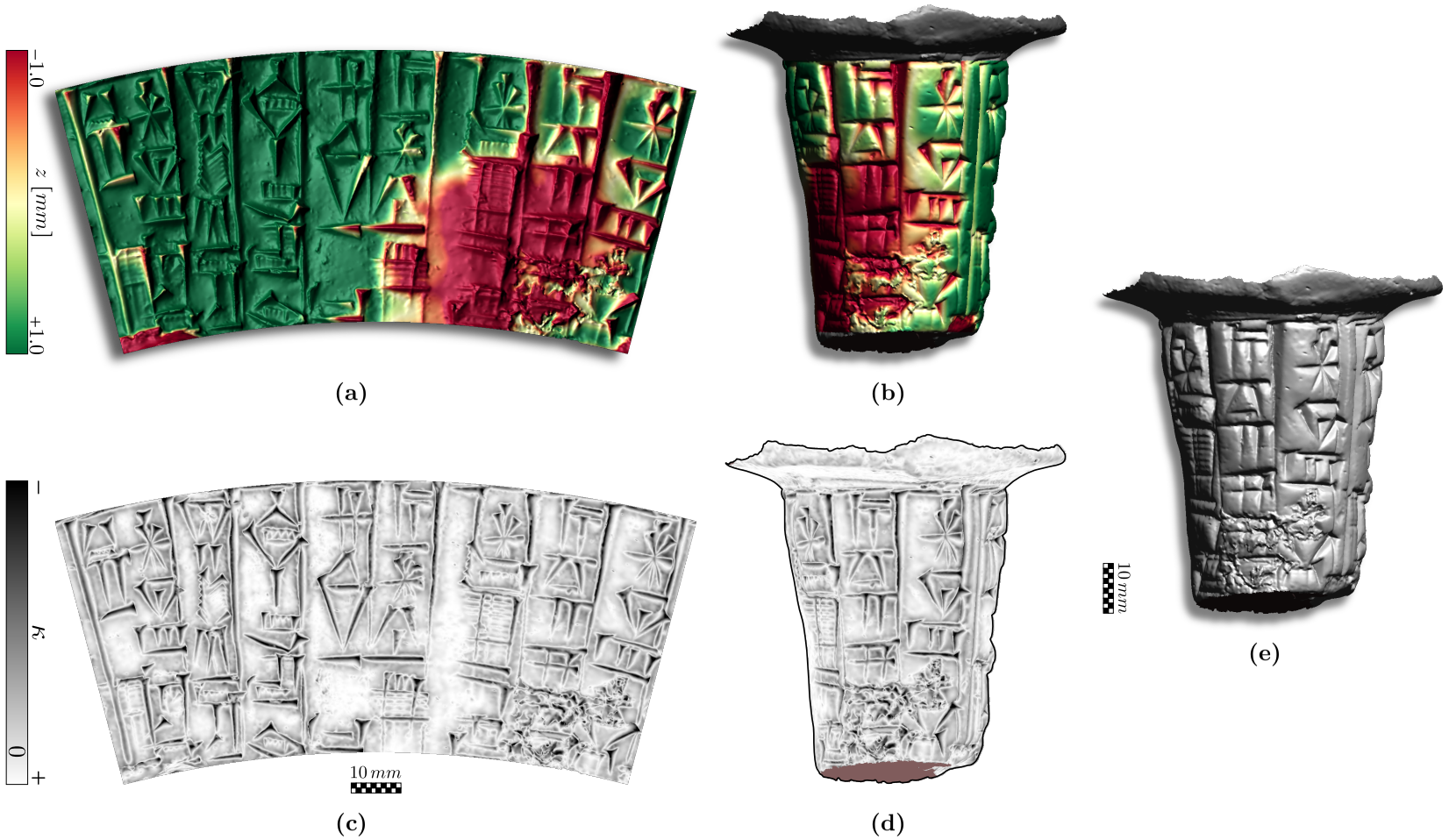


Figure B.14: 3D model of (e) a building inscription with cuneiform characters on a *Tonnagel* (clay cone), ca. 2200–2000 B.C., *Sammlung des Seminars für Sprachen und Kulturen des Vorderen Orients, Assyriologie, Universität Heidelberg*. For information about the inscription and its known 451 ancient duplicates see [Ste91], *Text Ur-Baba 7*. (a,b) Distance to an ideal cone. (c,d) Visualization of the volume based ulti-Scale Integral Invariant (MSII). (a,c) Unwrapped 3D-model using an ideal cone.

B.1.3 Unwrapping a Spherical Clay Envelope — Tonhülle einer Tafel

Clay tablets often have a spherical shape and in case of letters they were wrapped into an envelope made of clay. The letters typically leave negative imprints of cuneiform characters on the inside of an envelope. By chance only the envelope is found by archaeologists, because the envelope was thrown away, while the letter was kept and lost in another place. Due to the inversion of the wedges and the mirrored characters, it is a virtually impossible task to read the inside of an envelope by traditional means. Therefore Figure B.15 shows the inversion and unwrapping of a 3D-model[1] of a spherically shaped envelope. Finally a MSII texture map of the 3D-model is shown in Figure B.15.

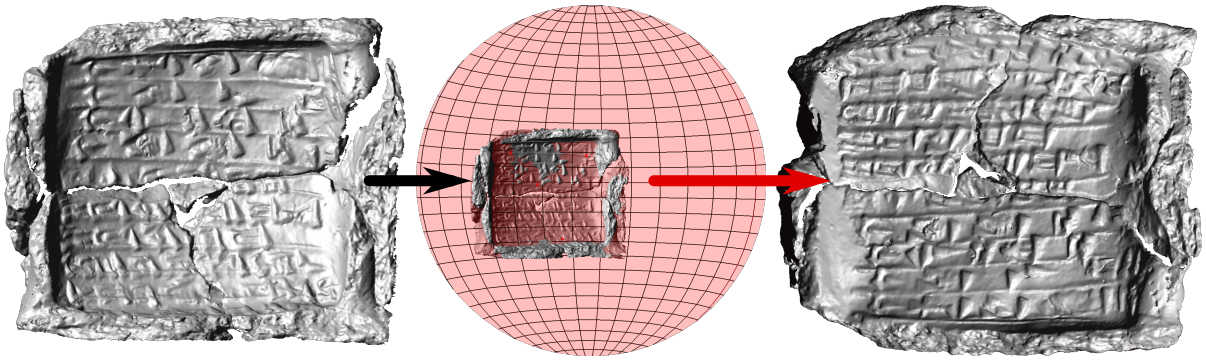


Figure B.15: The left shows an envelope of a cuneiform letter with inverted characters of the missing tablet. On the right hand side an ideal sphere is shown, which is fitted onto the envelope's surface. This sphere is used to flatten the surface. The flattened and inverted surface is shown in the bottom left with virtual illumination.



Figure B.16: Flattened and inverted clay envelope visualized using the curvature based MSII texture map. The envelope/hull is shown scaled $1:1$ having the same appearance as a tablet.

[1]Thanks to the timely acquisition by Sonja Speck, it was added right before submitting this thesis.

B.2 Uruk-Warka Sammlung, Heidelberg

Thanks to the custos Prof. Markus Hilgert a selection of tablets found in the ancient town of Uruk – today known as Warka, Iraq – were acquired in 2009. This collection is part of the Deutsches Archäologisches Institut (DAI).

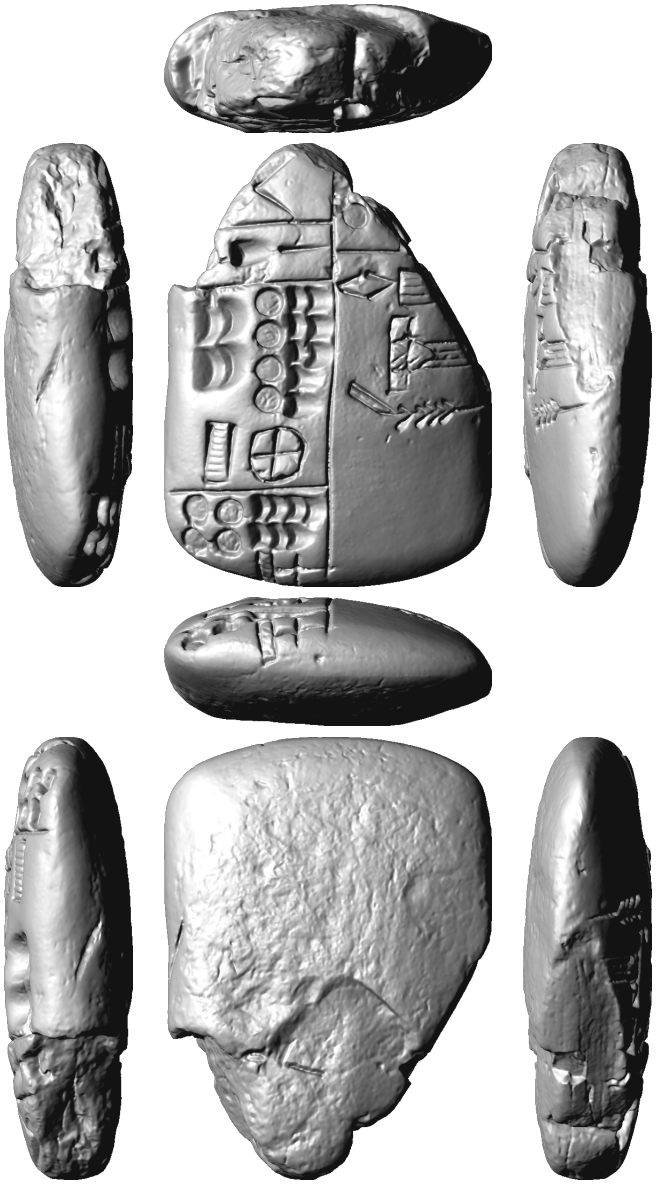


Figure B.17: W 20246,2 with monochrome texture map – 6 views, scale 1 : 1

Max. dim.	$4{,}3 \times 5{,}9 \times 1{,}7\,cm$	Surface, acq.	$5\,557\,mm^2$
Vertices	757 624	Res., avg	$136\,mm^{-2}$
Faces	1 515 244		$297\,DPI$
Material	original, clay	Volume, est.	$22\,cm^3$



Figure B.18: W 20246,2 with MSII texture map – 6 views, scale 1 : 1

Max. dim.	$4{,}3 \times 5{,}9 \times 1{,}7\,cm$	Surface, acq.	$5\,557\,mm^2$
Vertices	757 624	Res., avg	$136\,mm^{-2}$
Faces	1 515 244		$297\,DPI$
Material	original, clay	Volume, est.	$22\,cm^3$

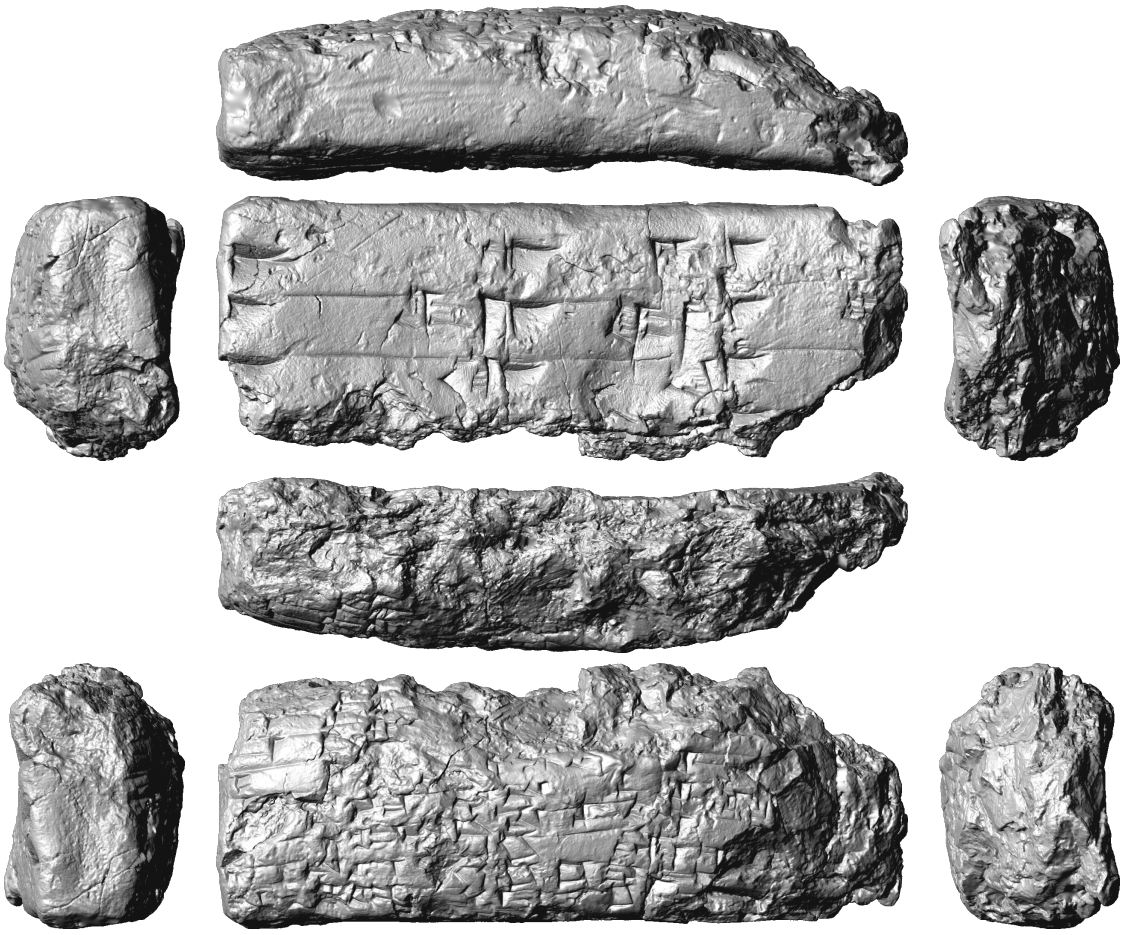


Figure B.19: W 20248,10 with monochrome texture map – 8 views, scale 1 : 1.5

Max. dim.	$13{,}7 \times 5{,}2 \times 3{,}6\,cm$	Surface, acq.	$20\,592\,mm^2$
Vertices	3 878 448	Res., avg	$188\,mm^{-2}$
Faces	7 756 892		$349\,DPI$
Material	original, clay	Volume, est.	$135\,cm^3$

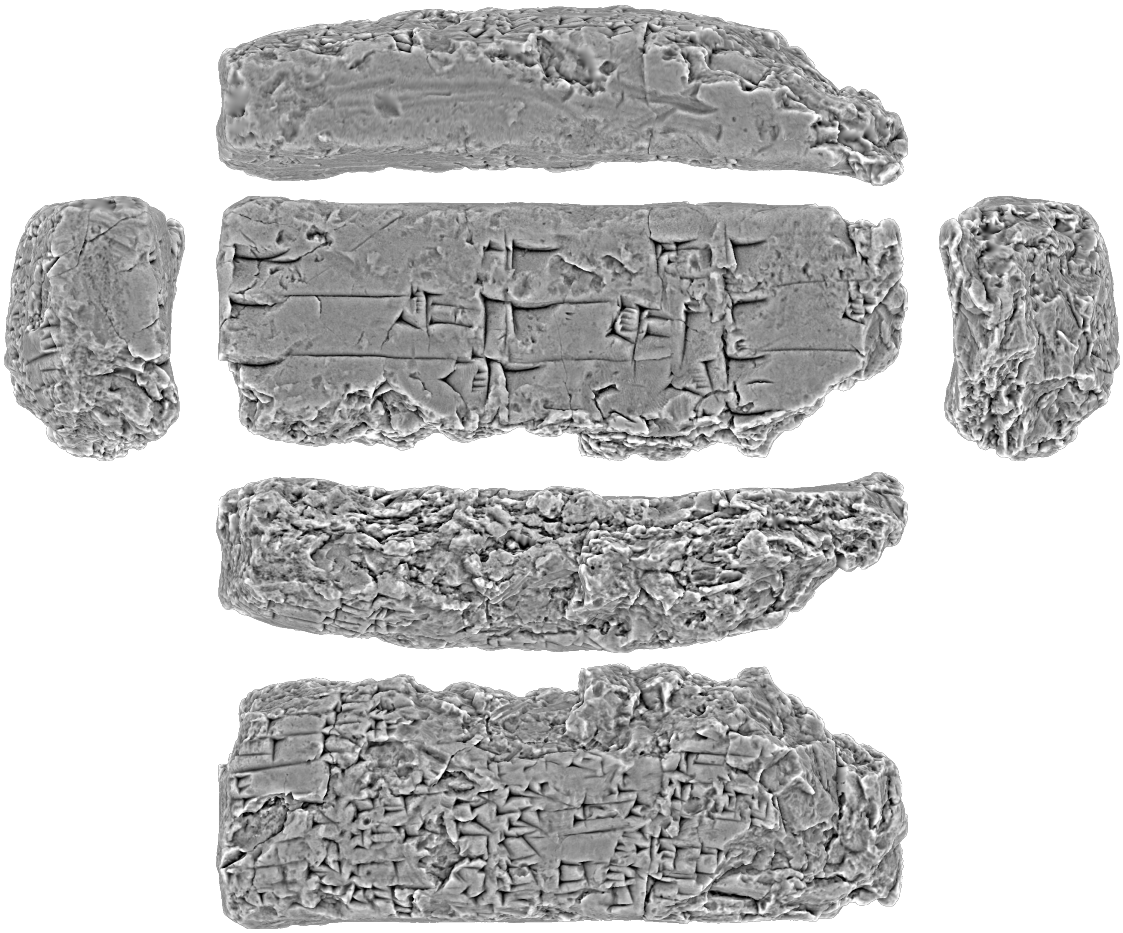


Figure B.20: W 20248,10 with MSII texture map – 6 views, scale 1 : 1.5

Max. dim.	$13{,}7 \times 5{,}2 \times 3{,}6\,cm$	Surface, acq.	$20\,592\,mm^2$
Vertices	3 878 448	Res., avg	$188\,mm^{-2}$
Faces	7 756 892		$349\,DPI$
Material	original, clay	Volume, est.	$135\,cm^3$

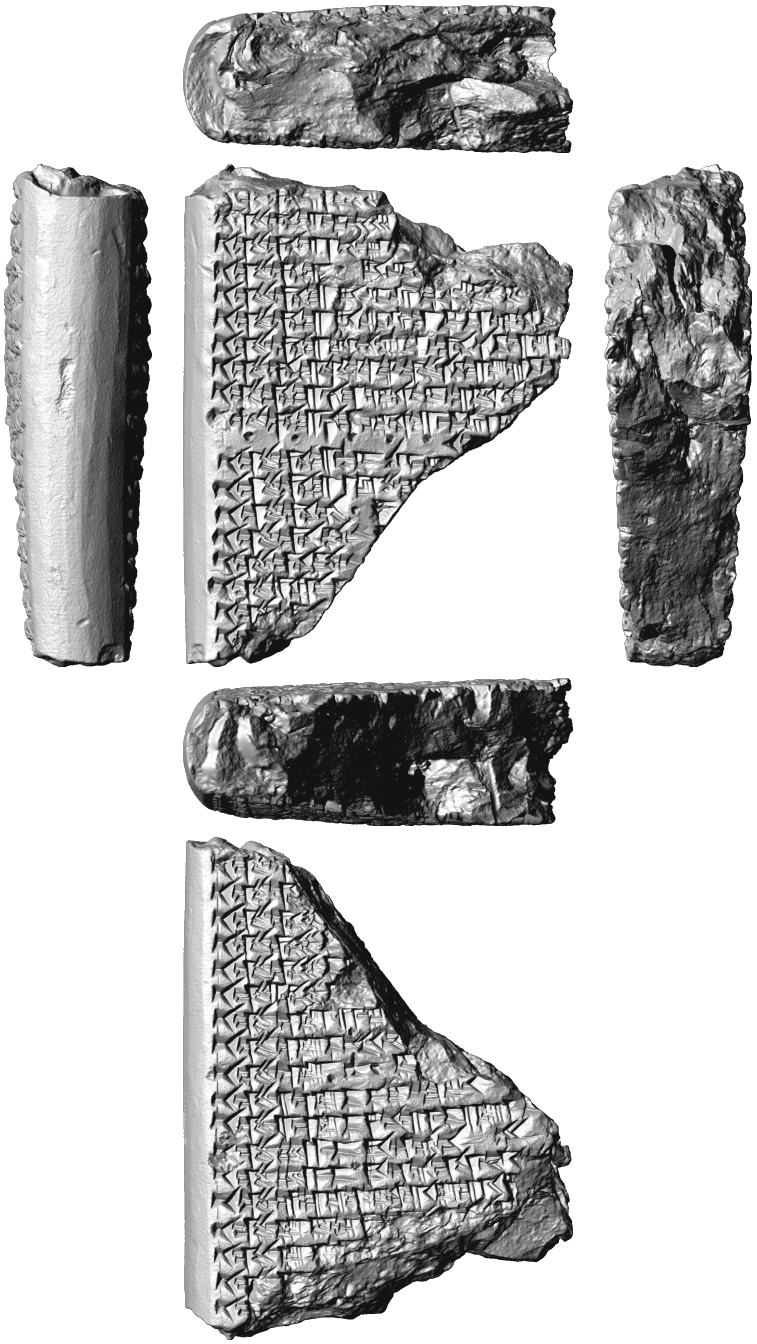


Figure B.21: W 20430,101 with monochrome texture map – 6 views, scale 1 : 1.25

Max. dim.	$6{,}4 \times 8{,}3 \times 2{,}4\,cm$	Surface, acq.	$13\,195\,mm^2$
Vertices	2 415 892	Res., avg	$183\,mm^{-2}$
Faces	4 831 780		$344\,DPI$
Material	original, clay	Volume, est.	$59\,cm^3$

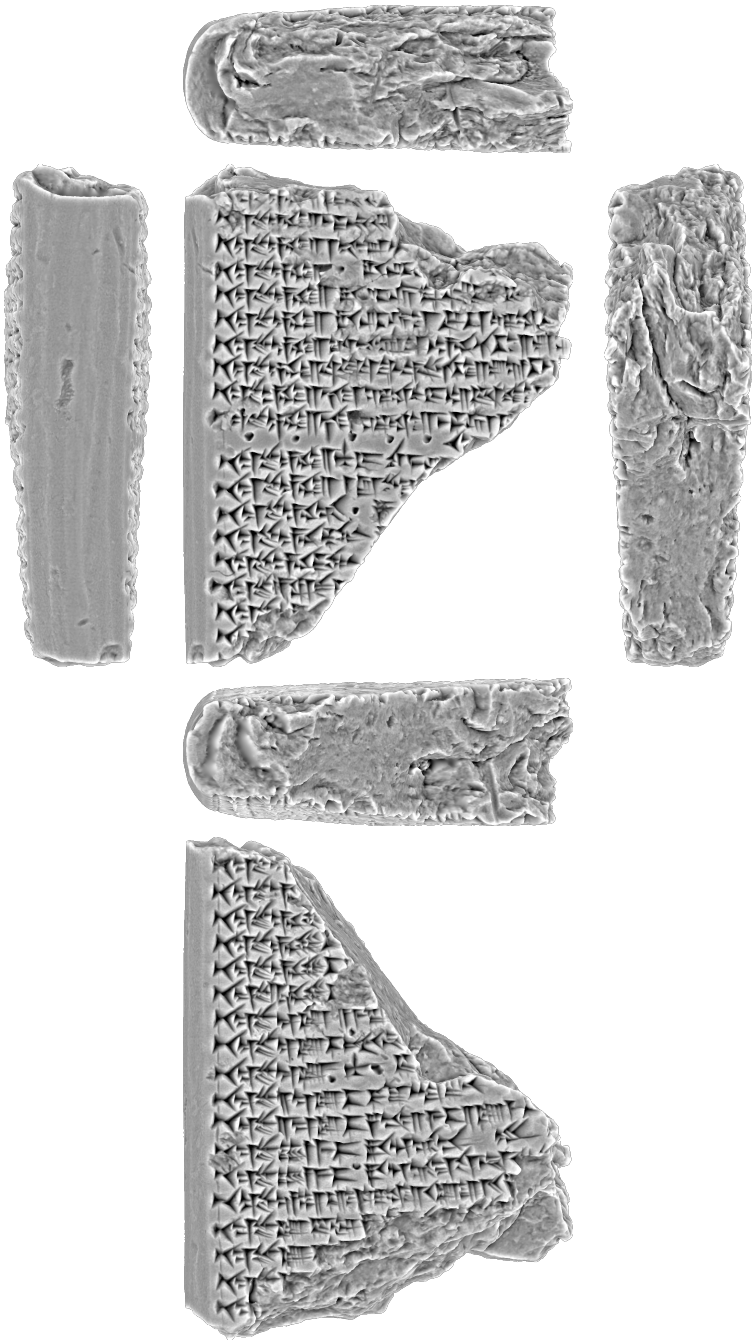


Figure B.22: W 20430,101 with MSII texture map – 6 views, scale 1 : 1.25

Max. dim.	$6{,}4 \times 8{,}3 \times 2{,}4\,cm$	Surface, acq.	$13\,195\,mm^2$
Vertices	2 415 892	Res., avg	$183\,mm^{-2}$
Faces	4 831 780		$344\,DPI$
Material	original, clay	Volume, est.	$59\,cm^3$

B.3 Vorderasiatisches Museum, Berlin



Figure B.23: VAT 9728 without texture map – 6 views, scale 1 : 2

Max. dim.	$165{,}9 \times 105{,}2 \times 26{,}5\,cm$	Surface, acq.	$43\,291\,mm^2$
Vertices	9 897 424	Res., avg	$229\,mm^{-2}$
Faces	19 692 569		$384\,DPI$
Material	original, clay	Volume, est.	$294 \pm 18\,cm^3$

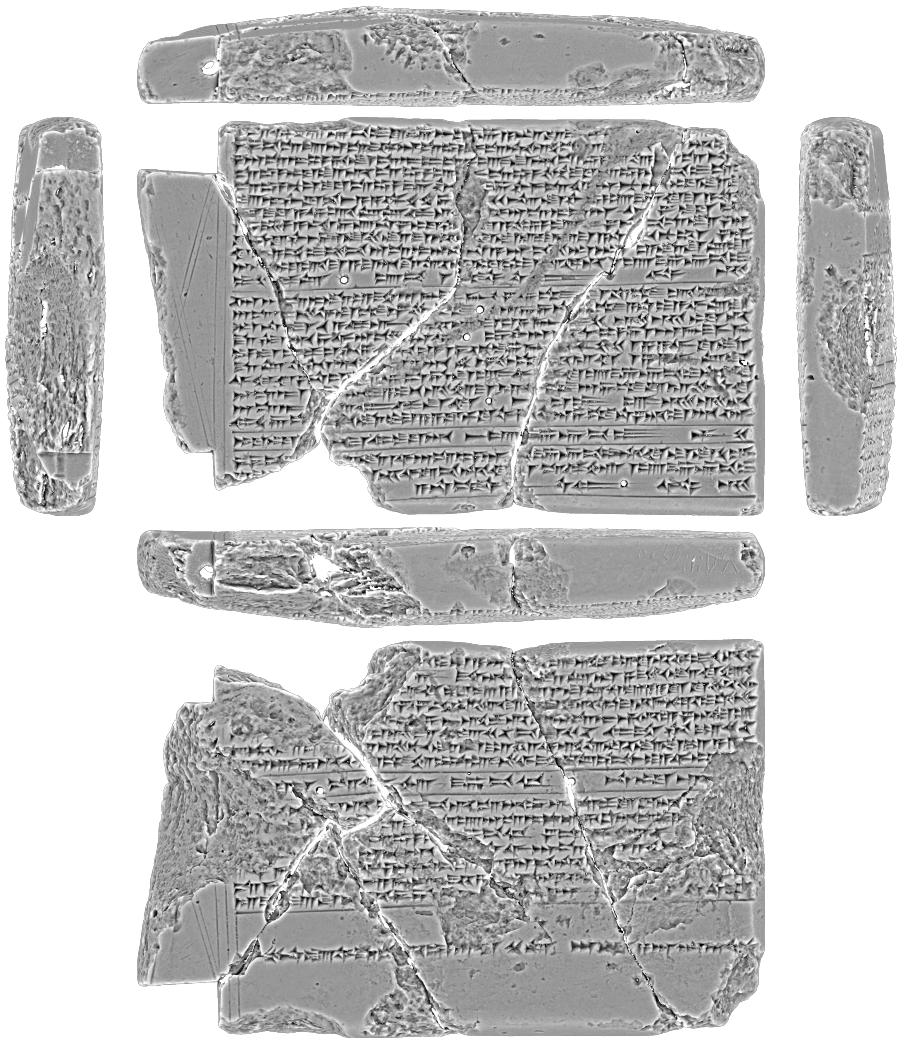


Figure B.24: VAT 9728 with MSII texture map – 6 views, scale 1 : 2

Max. dim.	$165{,}9 \times 105{,}2 \times 26{,}5\,cm$	Surface, acq.	$43\,291\,mm^2$
Vertices	9 897 424	Res., avg	$229\,mm^{-2}$
Faces	19 692 569		$384\,DPI$
Material	original, clay	Volume, est.	$294 \pm 18\,cm^3$

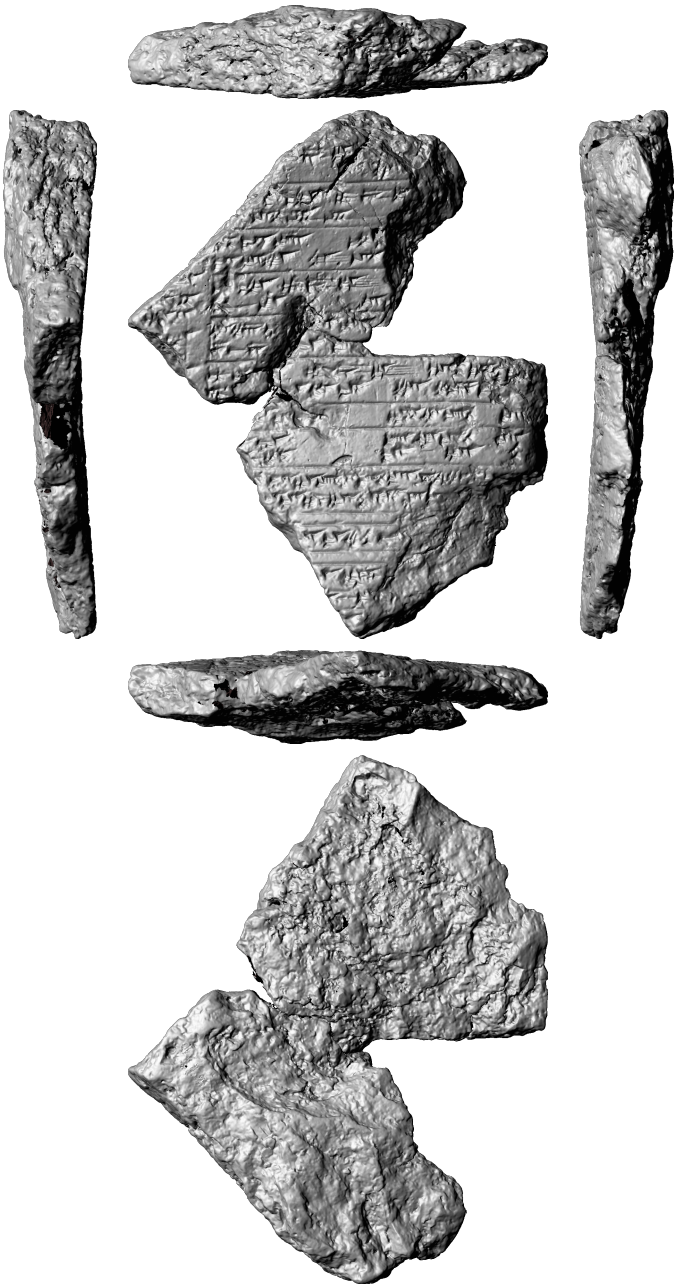


Figure B.25: VAT 9970 und 9927 without texture map – 6 views, scale 1 : 1.5

Max. dim.	$8{,}3 \times 10{,}5 \times 1{,}9\,cm$	Surface, acq.	$12\,281\,mm^2$
Vertices	2 295 734	Res., avg	$187\,mm^{-2}$
Faces	4 569 655		$347\,DPI$
Material	original, clay	Volume, est.	$38 \pm 3\,cm^3$

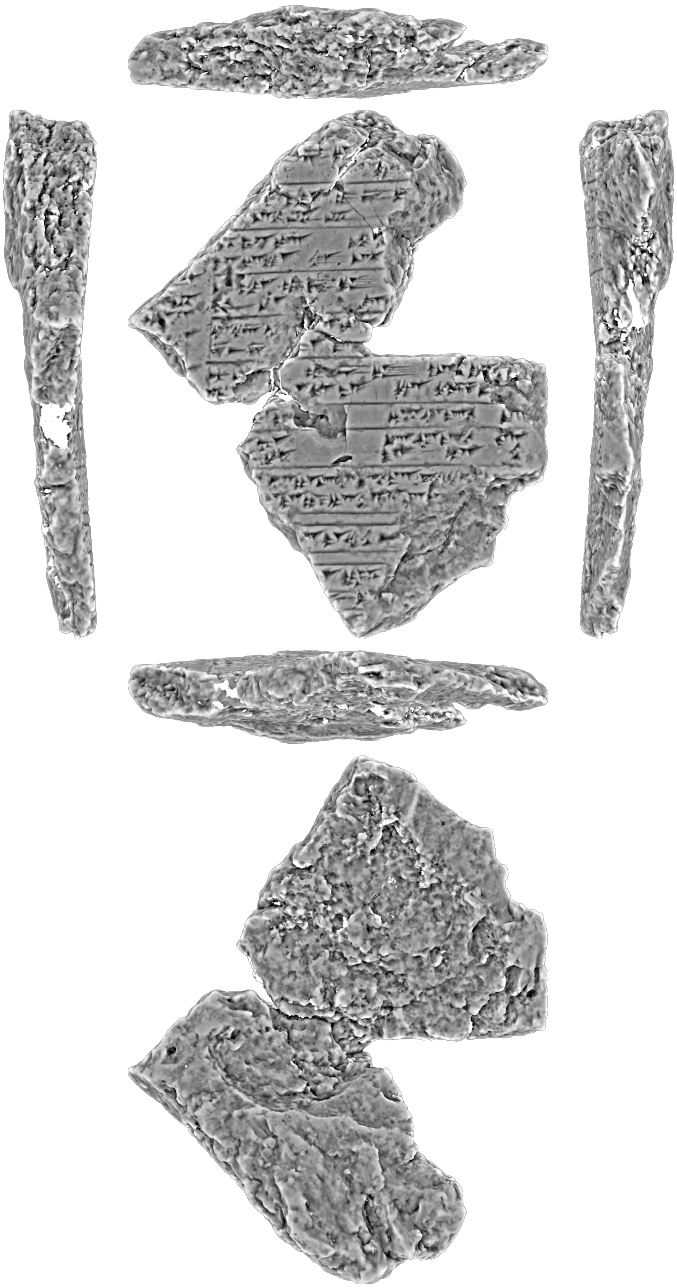


Figure B.26: VAT 9970 und 9927 with MSII texture map – 6 views, scale 1 : 1.5

Max. dim.	$8{,}3 \times 10{,}5 \times 1{,}9\,cm$	Surface, acq.	$12\,281\,mm^2$
Vertices	2 295 734	Res., avg	$187\,mm^{-2}$
Faces	4 569 655		$347\,DPI$
Material	original, clay	Volume, est.	$38 \pm 3\,cm^3$



Figure B.27: VAT 10908 without texture map – 6 views, scale 1 : 1.25

Max. dim.	$9 \times 7 \times 2{,}5\,cm$	Surface, acq.	$13\,614\,mm^2$
Vertices	2 569 362	Res., avg	$189\,mm^{-2}$
Faces	5 110 602		$349\,DPI$
Material	original, clay	Volume, est.	$72 \pm 8\,cm^3$

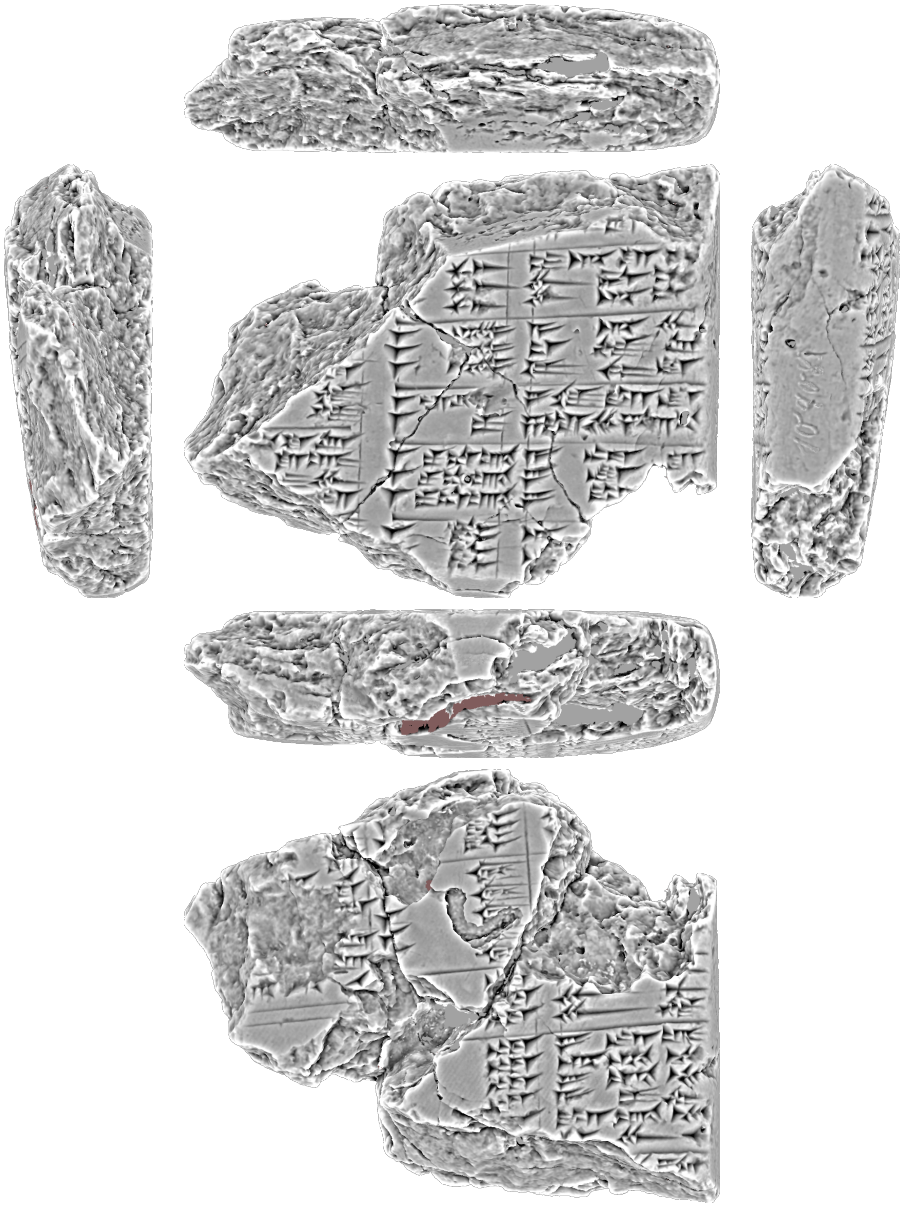


Figure B.28: VAT 10908 with MSII texture map – 6 views, scale 1 : 1.25

Max. dim.	$9 \times 7 \times 2{,}5\,cm$	Surface, acq.	$13\,614\,mm^2$
Vertices	2 569 362	Res., avg	$189\,mm^{-2}$
Faces	5 110 602		$349\,DPI$
Material	original, clay	Volume, est.	$72 \pm 8\,cm^3$

B.4 Landesmuseum Joanneum, Graz

This object is a rare clay brick with a (flat) cuneiform seal impression [Gal06] and was acquired in July 2009 in Graz, Austria thanks to Stephan Karl.

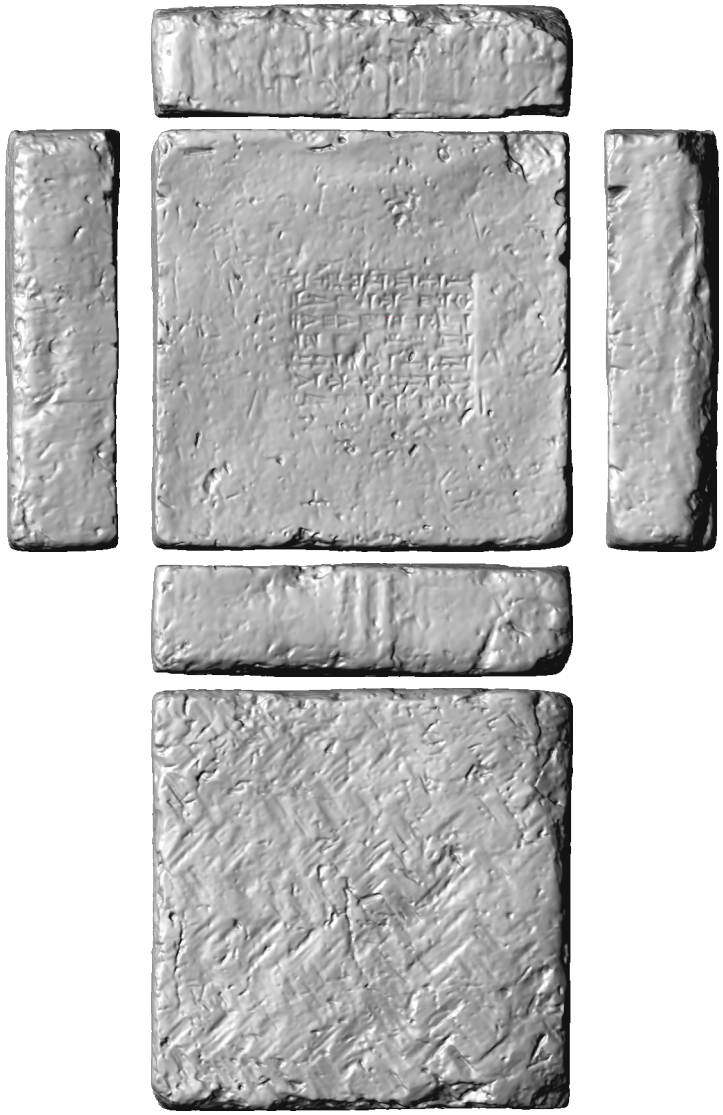


Figure B.29: UMJ 4279 with monochrome texture map – 6 views, scale 1 : 6

Max. dim.	$33{,}4 \times 33{,}5 \times 9{,}1\,cm$	Surface, acq.	$315\,658\,mm^2$
Vertices	365 951	Res., avg	$1\,mm^{-2}$
Faces	731 898		$27\,DPI$
Material	original, clay	Volume, est.	$8\,854\,cm^3$

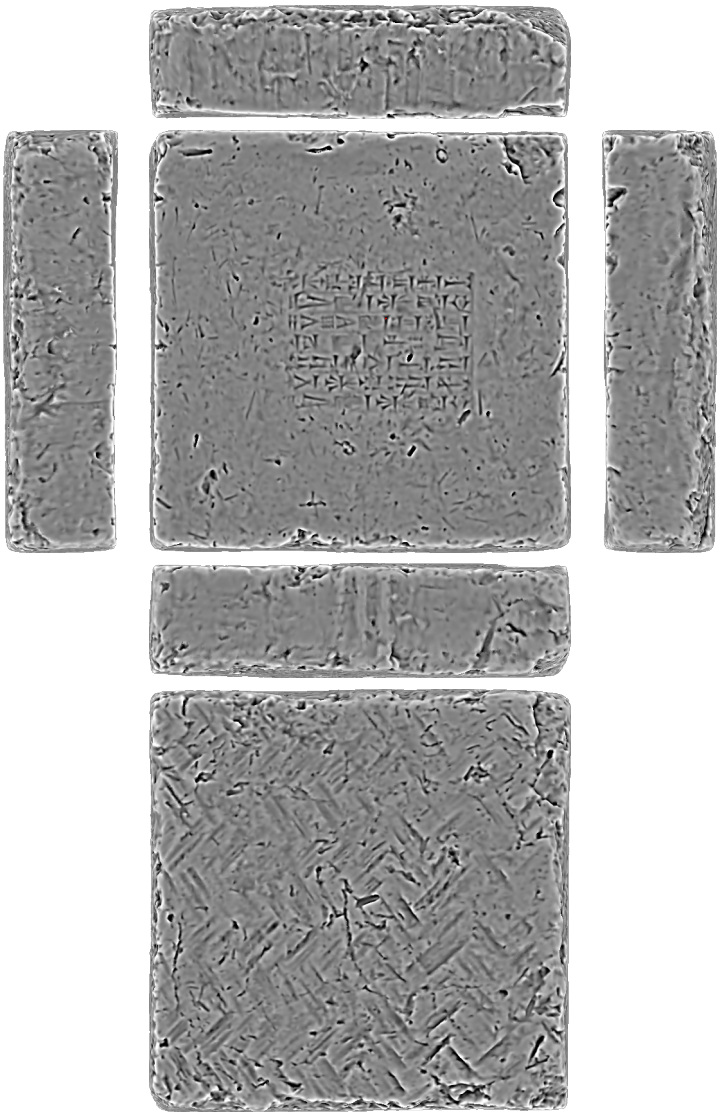


Figure B.30: UMJ 4279 with MSII texture map – 6 views, scale 1 : 6

Max. dim.	$33{,}4 \times 33{,}5 \times 9{,}1\,cm$	Surface, acq.	$315\,658\,mm^2$
Vertices	365 951	Res., avg	$1\,mm^{-2}$
Faces	731 898		$27\,DPI$
Material	original, clay	Volume, est.	$8\,854\,cm^3$

B.5 Key Data of the 3D-Models

The following table shows the key data of the acquired objects and the 3D-models presented in this appendix.

Inventory No.	Max. dimension $[cm]$	Vertices $\lvert\mathcal{L}_p\rvert$	Faces $\lvert\mathcal{L}_t\rvert$	Surface $[mm^2]$	Res., avg. $[mm^{-2}]$	Vol., est. $[cm^3]$	Material
88/677	4,4 × 4,9 × 1,8	5 694 192	11 389 024	5 654	1 007	24	original
Königsinschrift	5,2 × 4,1 × 1,8	3 477 399	6 954 802	5 026	692	18	original, clay
TCH.92,G.127	8 × 11,1 × 2,5	4 466 245	8 932 486	23 442	191	141	cast copy, gypsum
HOS G8	6,2 × 4,6 × 2,9	344 694	689 384	7 487	46	43	cast copy, ceramic
HOS G10	3,7 × 6,5 × 1,9	6 596 964	13 194 060	7 327	900	33	cast copy, ceramic
BM 90922	11,9 × 17,7 × 4,3	12 243 633	24 489 764	57 567	213	623	cast copy, ceramic
Tonnagel	6,8 × 7 × 6,8	613 640	1 223 004	10 676	57	n.a.	original, clay
Tonhülle e. Tafel	7,1 × 5,4 × 3,8	226 189	436 335	5 226	43	n.a.	original, clay
W 20246,2	4,3 × 5,9 × 1,7	757 624	1 515 244	5 557	136	22	original, clay
W 20248,10	13,7 × 5,2 × 3,6	3 878 448	7 756 892	20 592	188	135	original, clay
W 20430,101	6,4 × 8,3 × 2,4	2 415 892	4 831 780	13 195	183	59	original, clay
VAT 9728	166 × 105 × 26	9 897 424	19 692 569	43 291	229	294 ± 18	original, clay
VAT 9970, 9927	8,3 × 10,5 × 1,9	2 295 734	4 569 655	12 281	187	38 ± 3	original, clay
VAT 10908	9 × 7 × 2,5	2 569 362	5 110 602	13 614	189	72 ± 8	original, clay
UMJ Inv. 4279	33,4 × 33,5 × 9,1	365 951	731 898	315 658	1	8 854	original, clay

Table B.1: Key data of the 3D-models.

Explanation of the table's columns:

- *Max. dimension* shows the length of the edges (width, height, thickness) of the so-called bounding box of the 3D-model of an oriented tablet.
- *Vertices* $\lvert\mathcal{L}_p\rvert$ shows the number of measuring points $\mathbf{p}$ of the final 3D-model.
- *Faces* $\lvert\mathcal{L}_t\rvert$ shows the number of triangles $\mathbf{t}$ connecting the measuring points.
- *Surface* shows the acquired area of a tablet. It is computed as sum of the area of all triangles $\mathbf{t}$.
- *Res., avg.* – *average resolution*: number of measuring points divided by the *surface*.
- *Vol., est.* – *estimated volume*: computed using the 3D-model and only given for tablets, where $>\approx 95\,\%$ was acquired.
- *Material* is noted as it influences the quality of the text image. Cast copies in general do have a lower quality of the text image than originals – espically casts using ceramic.

List of Acronyms

ADWM	*Akademie der Wissenschaften und Literatur, Mainz* Academy of Sciences and Literature ... 14
ALS	Airborne Laser Scanner ... 19
Ass.	Assur Fundnummer – Inventory number of an object of the ancient city *Assur* ... 129
BSP	Binary Space Partitioning ... 58
CAD	Computer Aided Design ... 30
CDFP	Cuneiform Digital Forensic Project ... 11
CDLI	Cuneiform Digital Library Initiative ... 14
CDP	Cuneiform Digital Palaeography Project ... 11
CG	Computer Graphics ... 14
CH	Cultural Heritage ... 20
CT	Computer Tomography also known as Computed Tomography ... 11
CV	Computer Vision ... 25
DAI	*Deutsches Archäologisches Institut* ... 180
DFG	*Deutsche Forschungsgemeinschaft* – German Research Foundation ... 12
DPI	Dots Per Inch ... 20
DTM	Digital Terrain Model ... 87
FOV	Field of View ... 16
FFT	Fast Fourier Transform ... 81
GB	GigaByte (1024^3 byte) ... 117
GIMP	GNU Image Manipulation Program
GNU	GNU's Not Unix!
GUI	Graphical User Interface ... 7
GPL	GNU General Public License ... 120
GTK+	GIMP Toolkit
HAW	*Heidelberger Akademie der Wissenschaften* Heidelberg Academy of Sciences and Humanities ... 1
HDRI	High Dynamic Range Imaging ... 17
HGS MathComp	Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences ... 1
HNF	Hesse normal form ... 66
HOS	Heidelberger Orientalisches Seminar – Prefix for cuneiform tablets located at the Assyriology, Heidelberg University[2]
HPM	Hethitologie Portal Mainz – Web portal of the Assyriology groups of the ADWM and the Würzburg University ... 14
HSV	hue, saturation and value ... 123
ICE	Initiative for Cuneiform Encoding ... 13
ICP	Iterative Closest Point ... 16
IDE	Integrated development environment ... 124
IPP	IWR Pioneering Projects ... 1
IWR	Interdisciplinary Center for Scientific Computing ... 1
JHUAPL	Johns Hopkins University Applied Physics Laboratory ... 11

[2]Refers to the former name of the institute.

KAL	Keilschrifttexte aus Assur literarischen Inhalts ... 8
LA	Linear Algebra ... 25
LCDS	Leuven Camera Dome System ... 12
LIDAR	Light Detection And Ranging ... 19
LOD	Level of Detail ... 35
MPIWG	*Max Planck Institut für Wissenschaftsgeschichte* Max Planck Institute for History of Sciences ... 14
`NaN`	not a number Bit pattern for floating point datatypes in *C++* ... 159
OCR	Optical Character Recognition ... 7
OpenGL	Open Graphics Language ... 29
OS	Operating System ... 13
PCA	Principal Component Analysis ... 37
PLY	Stanford Polygon ... 115
PTM	Polynomial Texture Mapping ... 12
RTI	Reflection Transformation Imaging ... 12
SVD	Single Value Decomposition ... 39
SVG	Scalable Vector Graphics ... 126
SVM	Support Vector Machine ... 8
TCH	Cuneiform tablet signatur of the excavation at *Tell Chuera*, Syria
TIN	Triangulated Irregular Network ... 25
TLS	Terrestrial Laser Scanner ... 19
TNB frame	Frenet-Serret frame ... 47
TOF	Time-Of-Flight ... 19
VAM	Vorderasiatische Abteilung der Berliner Museen – Collection of the Ancient near east of the museums in Berlin, Germany ... 117
VAT	Tafelsignatur der Vorderasiatischen Abteilung der Berliner Museen Tablet index of the museums in Berlin, Germany ... 1
VBO	Vertex Buffer Object ... 122
VRML	Virtual Reality Modeling Language ... 11
W3C	World Wide Web Consortium ... 127
W	Prefix for objects from the Uruk-Warka Collection
WT	Wavelet Transformation ... 81
UCLA	University of California at Los Angeles ... 14
XML	eXtensible Markup Language ... 127

Glossary

Notation	Description	Symbol	Page
Area	generally denoted using	A	
	alternate notation for an area, e.g. $A_{\mathcal{M}} \equiv \lvert\mathcal{M}_2\rvert$	$\lvert\cdot\rvert$	27
	of a 3D-model's surface $\mathcal{M}_2$	$A_{\mathcal{M}}$	27
	of a triangle $\mathbf{t}$	$A(\mathbf{t})$	27
	area of a triangle intersected by a ball $\lvert\mathbf{t} \cap B_r\rvert$	$A_B(\mathbf{t})$	62
	area of a circle segment	$A_C(r, \alpha)$	66
Array	generally denoted using	a	47
	voxel cells, addressed using three indices	$a[i, j, k]$	47
	cell of an array of voxel holding a ball $B_{r_\boxplus}$	$a_\circ$	48
	cells describing voxels: $\mathcal{P}_i \rightsquigarrow a_{\mathcal{P}_i} \subset \mathbf{1}_{D_{\mathcal{M}}}$	$a_{\mathcal{P}_i}$	48
	size of an array of voxels	$n_\boxplus^3$	47
	cells holding a transformed sphere, q.v. function	$f^\oplus[j, k]$	50
Assign	an element $\cdot$ to a (q.v.) list $\mathcal{L}$	$\mathcal{L} \hookleftarrow \cdot$	94
Ball	q.v. sphere $\mathbb{S}$	B	27
	about a point $\mathbf{p}$ having a radius r in $\mathbb{R}^2$	$B_r^2(\mathbf{p})$	43
	about a point $\mathbf{p}$ having a radius r in $\mathbb{R}^3$	$B_r(\mathbf{p})$	42
	about a point $\mathbf{p}$ having a radius r in $\mathbb{R}^n$	$B_r^n(\mathbf{p})$	42
	volume approximated using voxels, q.v. array	$\lvert\widetilde{B}_\boxplus\rvert$	48
Border	or boundary	∂	
	of a region Ω	$\partial\Omega$	27
	of the surface of a 3D-model	$\partial\mathcal{M}_2$	28, 53
	of a n-ball B^n, $\partial B^n \equiv \mathbb{S}^{n-1}$, q.v. sphere and ball	∂B^n	42
Connected	component, generally denoted using	$\mathcal{C}$	94
	components have an unique id $\in \mathbb{Z}^+$ called label id	$l_{\mathcal{C}}$	94
	components $\mathcal{C}_{l_{\mathcal{C}}}$ with an id $l_{\mathcal{C}}$ are denoted by	$\mathcal{C}_l$	94
	component of no area of interest i.e. background	$\mathcal{C}_\varnothing$	94
	component's area	$\lvert\mathcal{C}\rvert$	97
	component's boundary length	$\lvert\partial\mathcal{C}\rvert$	97
Correlation	generally denoted using	R	
	of two functions/signals f and g	R_{fg}	81
	of a function/signal f with itself	R_{ff}	81
	of a feature vector $\mathfrak{F}_i$ with itself	R_i	82
	of a feature vector $\mathfrak{F}_i$ with a reference $\mathfrak{F}_k$	$R_{i,k}$	84
	combined by addition of R_i and $R_{i,k}$	$R_{i,i,k}$	85
Cunei	Latin for wedge, q.v. Winkelhaken	$\shortmid$, $\rightarrowtail$	7, 136

Notation	Description	Symbol	Page
Curvature	generally denoted using	κ	36
	mean curvature	κ_H	36
	Gaussian curvature	κ_G	36
	principle curvatures for a point $\mathbf{p}_i$ of $\mathcal{M}_2$	κ_1, κ_2	37
	root mean square curvature	κ_{RMS}	39
	absolute curvature	κ_{ABS}	39
	estimated curvature	$\widetilde{\kappa}$	101
	signed curvature	$\overset{\pm}{\kappa}$	101
	curvature smoothed using a Gaussian kernel	$\bar{\kappa}$	102
Domain	generally denoted using	D	42
	in $\mathbb{R}^2$ defined by a 1D-manifold	$D_{\mathcal{M}_1}$	42
	in $\mathbb{R}^3$ defined by a 3D-model's surface	$D_{\mathcal{M}}$	45
Empty set	or list: $\mathcal{L} = \{\} \equiv \emptyset \rightarrow \lvert\mathcal{L}\rvert = 0$	$\emptyset$	28
Euclidean	distance $\Delta_{..}$, q.v. vectors & Manhattan distance	$\lvert\cdot - \cdot\rvert_2$	81, 90
Euler	characteristic of a manifold	$\chi(\mathcal{M})$	49
Function	in general, denoted using	f	
	for a transformed sphere, q.v. sphere and array	$f^{\oplus}$,$f^{\ominus}$	50
	multi-purpose function value of a vertex $\mathbf{p}_i$	$f(\mathbf{p}_i)$	94
Gaussian	or normal distribution: mean μ, variance σ^2	$\mathcal{N}(\mu, \sigma^2)$	102
Geodesic	distance between a point $\mathbf{p}_i$ and $\mathbf{p}_j$	$\mathfrak{g}_i(\mathbf{p}_j)$	109
Hausdorff	distance of two manifolds	$d_H(\cdot, \cdot)$	35, 129
Indicator	function	$\mathbf{1}$	43
	for a domain $D_{\mathcal{M}_1}$	$\mathbf{1}_{D_{\mathcal{M}_1}}$	43
	for a domain $D_{\mathcal{M}}$ defined by a mesh $\mathcal{L}_{\mathcal{M}}$	$\mathbf{1}_{D_{\mathcal{M}}}$	45
Integral	invariants – normalized are tagged with $\hat{}$		
	volume integral invariant $\lvert B_r(\mathbf{p}) \cap \mathbf{1}_{D_{\mathcal{M}}}\rvert$	$V_r(\mathbf{p})$	45
	normalized volume integral invariant for a scale r	$\hat{V}_r$	45
	$\hat{V}_r$ estimated using voxels, q.v. array	$\hat{V}_{\boxplus}$	48
	surface based using spheres $\mathbb{S}$, q.v. manifolds	S_r,$\hat{S}$	61
	surface based using balls B, q.v. manifolds	P_r,$\hat{P}$	62
	line based — border of $\mathcal{P}$ and $\mathcal{S}$, q.v. manifolds	L_r,$\hat{L}$	62
	fast estimator for L_r	$\widetilde{L}_r$	63
	run-length based integral invariant	$Q_r^{+,-}$, $\hat{Q}^{+,-}$	104
	angle based integral invariant	M_r, α_κ	104
Landau	notation for terms of higher order	$O(\cdot)$	47
	notation for computational complexity		39
List	contains sorted primitives, indices, etc.	$\mathcal{L}$	
	has a cardinality of elements, q.v. empty set	$\lvert\mathcal{L}\rvert$	26
	of vertices $\mathbf{p}_i$	$\mathcal{L}_v$	25
	of faces $\mathbf{t}_i$	$\mathcal{L}_f$	26
	of lists describing a 3D-model: $\mathcal{L}_{\mathcal{M}} = \{\mathcal{L}_v, \mathcal{L}_f\}$	$\mathcal{L}_{\mathcal{M}}$	26
	of edges $\mathbf{e}_k$	$\mathcal{L}_e$	28
	of solo vertices $\mathbf{p}_s$	$\mathcal{L}_{v_s}$	28
	of edges along the boundary of a mesh $\partial\mathcal{M}_2$	$\mathcal{L}_{\partial e}$	49

Notation	Description	Symbol	Page
	organized as an explicit stack of vertices	$\mathcal{L}_{\boxplus}$	92
	of connected components $\mathcal{C}$	$\mathcal{L}_C$	95
	of polygonal lines $\mathbf{l}$, e.g. $\partial\mathcal{C}$	$\mathcal{L}_l$	100
	of points of interest within a polyline, e.g. $\mathbf{l} \equiv \partial\mathcal{C}$	$\mathcal{L}_{\textcircled{1}}$	108
	of points of interest within an area, e.g. $\mathcal{C}$	$\mathcal{L}_{\circledast}$	109
Manhattan	distance, q.v. Euclidean distance	$\lvert\cdot - \cdot\rvert_1$	81
Manifold	nD-subset of an Euclidean space in $\mathbb{R}^{>n}$	$\mathcal{M}_n$	
	1D-manifold, e.g. a curve or a polygonal line	$\mathcal{M}_1$	43
	2D-manifold, abbreviated $\mathcal{M}$ for $\mathcal{M}_2$ in $\mathbb{R}^3$	$\mathcal{M}_2$	26
Manifolds	leading ($\rightsquigarrow$) to integral invariants		
	$\mathcal{S}_r \coloneqq \mathbb{S}_r \cap \mathbf{1}_{D_\mathcal{M}} \rightsquigarrow S_r$	$\mathcal{S}_r$	61
	$\mathcal{P}_r \coloneqq B_r \cap \mathcal{M}_2 \rightsquigarrow P_r$	$\mathcal{P}_r$	61
	$\mathscr{L}_r \coloneqq \mathbb{S}_r \cap \mathcal{M}_2 \rightsquigarrow L_r$	$\mathscr{L}_r$	61
Matrix	arbitrary transforming homogeneous coordinates	$\mathbf{A}$	
	surface patch $\mathcal{P}_i$ transform: $\mathbf{p}_i \leftarrow \mathbf{o}$ and $\hat{\mathbf{n}}_i \leftarrow \mathfrak{z}$	$\mathbf{A}_{0z_i}$	51
	rotation about the z-axis about an angle ϕ	$\mathbf{R}_z(\phi)$	51
	translation using $-\mathbf{p}_i \Longrightarrow \mathbf{p}_i\mathbf{T}_{\mathbf{p}_i} = \mathbf{o}$	$\mathbf{T}_{\mathbf{p}_i}$	51
	related to rotation about an arbitrary axis $(\mathbf{p}, \mathfrak{r})$	$\mathbf{T}_z$. $\mathbf{T}_{xz}$	51
Patch	of a surface about $\mathbf{p}_i$ within the radius r	$\mathcal{P}_i$,$\mathcal{P}_{i,r}$	47
Plane	typically described using the Hesse normal form	$\boldsymbol{\tau}$	66
Poisson	surface reconstruction of a (q.v.) manifold	$\delta\mathcal{M}_2$	35
Primitive	elements of triangulated surfaces	$\in \mathcal{L}_\mathcal{M}$	
	measuring point also known as vertex, q.v. vectors	$\mathbf{p}$	25
	triangle connecting vertices also known as face	$\mathbf{t}$	26
	edge, e.g. of a triangle or polyline	$\mathbf{e}$	26
	faces of a 1-ring: faces adjacent to a vertex $\mathbf{p}_i$	$\mathring{\mathbf{t}}_i$	29
	vertices of a 1-ring: $\mathring{\mathbf{p}}_i = \mathbf{p}(\mathring{\mathbf{t}}_i)$	$\mathring{\mathbf{p}}_i$	29
	triangles $\mathbf{t} \cap B_r(\mathbf{p}_i)$ resulting in circle segments	$\mathbf{t}_{\subset\partial i}$	64
	orthogonal projection of a point onto a plane $\boldsymbol{\tau}$	$\mathbf{p}^\tau$	74
	polyline described by a chain of pairs $(\mathbf{p}, \mathfrak{n})$	$\mathbf{l}$	100
Prism	defined by a base e.g. a square and a direction $\mathfrak{n}$	$\square \times \mathfrak{n}$	72
Prism set	of elements of a patch $\mathcal{P}_i$ and its direction $\mathfrak{n}_i$	$\mathscr{L} \times \mathfrak{n}_i$	73
Radius	generally denoted using	r	36
	discrete radius related to $n_\boxplus$ – q.v. array	$r_\boxplus$	48
	smallest radius of a ball $B(\mathbf{p}_i)$ enclosing $\mathcal{M}_2$	$\mathtt{r}_{max_i}$	58
	minimum of $\mathtt{r}_{max_i}$ of all $\mathbf{p}_i \in \mathcal{L}_v$	$\mathtt{R}_{max}$	58
	radius of a ball $B(\mathbf{p}_i)$ defined by the 1-ring $\mathring{\mathbf{t}}_i$	$\mathtt{r}_{min_i}$	58
	minimum of $\mathtt{r}_{min_i}$ of all $\mathbf{p}_i \in \mathcal{L}_v$	$\mathtt{R}_{min}$	59
Region	enclosing a surface	Ω	27
Sphere	q.v. ball B	$\mathbb{S}$	
	boundary $\partial B_r^n(\mathbf{p})$ of a ball	$\mathbb{S}_r^{n-1}$	42
	parameter form of a sphere, q.v. function & array	$f_r(x, y)$	50
	parameter form, upper ½ of a sphere	$f_r^+(x, y)$	50
	parameter form, lower ½ of a sphere	$f_r^-(x, y)$	50
	small circle $\mathbb{S}_r \cap \boldsymbol{\tau}$	$\mathbb{S}^1_{r_s}(\mathbf{p}_\tau)$	66

Notation	Description	Symbol	Page
Threshold	for function values $f(\mathbf{p}_i)$, q.v. function	$\mathfrak{t}_{\mathbf{p}}$	94
Vector	operations		
	length or magnitude or norm of a vector	$\lvert\cdot\rvert$	26
	inner product of two vectors	$\langle\cdot,\cdot\rangle$	51
Vectors	for directions		
	tangent vector	$\mathfrak{t}$	47
	normal vector	$\mathfrak{n}$	27, 47
	bi-tangent vector	$\mathfrak{b}$	47
	orientation of the z-axis: $(0,0,1)^T$	$\mathfrak{z}$	51
	orientation of a rotation axis	$\mathfrak{r}$	51
Vectors	spanning feature spaces, having elements $\mathfrak{f}_j$	$\mathfrak{F}$	81
	having elements $\mathfrak{v}_j \leftarrow \hat{V}_r$ for a scale $r \mapsto j$	$\mathfrak{V}$	79
	having elements $\mathfrak{p}_j \leftarrow \hat{P}_r$ for a scale $r \mapsto j$	$\mathfrak{P}$	79
	having elements $\mathfrak{l}_j \leftarrow \hat{L}_r$ for a scale $r \mapsto j$	$\mathfrak{L}$	79
Vectors	for positions		
	position vector, q.v. primitive	$\mathbf{p}$	25
	balance point of a primitive, e.g. a triangle	$\mathbf{s}$	27, 72
	origin, e.g. of $\mathbb{R}^3$: $(0,0,0)^T$	$\mathbf{o}$	26, 51
Volume	enclosed by any arbitrary region	V	
	enclosed by a unit cube	$V_{\square}$	27
	enclosed by a 3D-model	$V_{\mathcal{M}}$	27
	alternate notation, q.v. area and vector	$\lvert\mathbb{V}_{\mathcal{M}}\rvert$	27
	of a ball B	$\lvert B\rvert$	27
	leading to the volume integral invariant $V_r = \lvert\mathbb{V}_r\rvert$	$\mathbb{V}_r$	61
Winkelhaken	German for a variant of a wedge, q.v. cunei	◀, ◀	7, 136

Bibliography

[AA05] N. Applbaum and Y. Applbaum. *The Use of Medical Computed Tomography (CT) Imaging in the Study of Ceramic and Clay Archaeological Artifacts from the Ancient Near East.* Humanities, Social Sciences and Law. Springer Netherlands, 2005. cited p. 11

[AHB87] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–800, 1987. cited p. 16

[AL02] S. E. Anderson and M. Levoy. Unwrapping and Visualizing Cuneiform Tablets. *IEEE Computer Graphics and Applications*, 22(6):82–88, 2002. cited p. 11, 34

[BVD+94] G. von Bally, D. Vukicevic, N. Demoli, H. Bjelkhagen, G. Wernicke, U. Dahms, H. Gruber, and W. Sommerfeld. Holography and Holographic Pattern Recognition for Preservation and Evaluation of Cultural-Historic Sources. *Biomedical and Life Sciences*, 81(12):563–565, 1994. cited p. 10

[BB82] D. H. Ballard and C. M. Brown. *Computer Vision.* Prentice Hall, 1982. cited p. 25, 28

[BBS10] C. Bathow, B. Breuckmann, and R. Scopigno. Verification and Acceptance Tests for High Definition 3D Surface Scanners. In A. Artusi, M. Joly, G. Lucet, D. Pitzalis, and A. Ribes, editors, *Proc. VAST Int. Symposium on Virtual Reality, Archaeology and Cultural Heritage*, Palais du Louvre, Paris, France, 2010. Eurographics Association. cited p. 20

[BDP48] J. Bertrand, C. Diquet, and V. Puiseux. Démonstration d'un théorème de Gauss. *Journal de Mathématiques*, 13:80–90, 1848. cited p. 36, 157

[BE92] M. Bern and D. Eppstein. *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes on Computing*, chapter Mesh Generation and Optimal Triangulation, pages 47–123. World Scientific, 2 edition, 1992. cited p. 26

[Ber05] G. Bertrand. On Topological Watersheds. *Journal of Mathematical Imaging and Vision*, 22(2-3):217–230, 2005. cited p. 40

[BHH03] C. A. Brewer, G. W. Hatchard, and M. A. Harrower. ColorBrewer in Print: A Catalog of Color Schemes for Maps. *Cartography and Geographic Information Science*, 30:5–32, 2003. cited p. 123

[Bis05] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, USA, reprint edition, 2005. cited p. 10

[BKMK10] S. Bechtold, S. Krömker, H. Mara, and B. Kratzmüller. Rollouts of Fine Ware Pottery using High Resolution 3D Meshes. In A. Artusi, M. Joly, G. Lucet, D. Pitzalis, and A. Ribes, editors, *Proc. VAST Int. Symposium on Virtual Reality, Archaeology and Cultural Heritage*, pages 79–86, Palais du Louvre, Paris, France, 2010. Eurographics Association. cited p. 124

[BL79] S. Beucher and C. Lantuéjoul. Use of Watersheds in Contour Detection. In *Proc. of International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation*, Rennes, France, 1979. cited p. 40

[BM92] P. J. Besl and N. D. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. cited p. 16

[BMV09] B. Breuckmann, H. Mara, and Z. Végvári. Combined 3D and Multispectral Fresco Documentation of the Villa Oplontis, Pompei. In A. Ranchordas and H. Araújo, editors, *Proc. of the 4th International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 1, pages 615–620, Lisboa, Portugal, 2009. Springer. Subtitle: High-Resolution and High-Performance Digitization of Cultural Heritage. cited p. 20

[Bod01] J. Bodel, editor. *Epigraphic Evidence: Ancient History from Inscriptions*. Routledge, Taylor & Francis Group, 2001. cited p. 2

[Boo72] C. de Boor. On Calculating with B-Splines. *Journal of Approximation Theory*, 6:50–62, 1972. cited p. 34

[Bor10] R. Borger. *Mesopotamisches Zeichenlexikon*, volume 305 of *Alter Orient und Altes Testament – Veröffentlichungen zur Kultur und Geschichte des Alten Orients und des Alten Testaments (AOAT)*. Ugarit-Verlag, 2 edition, 2010. cited p. 13

[BR07] B. J. Brown and S. Rusinkiewicz. Global Non-Rigid Alignment of 3-D Scans. *ACM Transactions on Graphics (TOG)*, 26(3):CDROM, 2007. cited p. 25

[Bra65] R. Bracewell. *The Fourier Transform and Its Applications*, chapter The Autocorrelation Function, pages 40–45. 1965. cited p. 81

[Bre65] J. E. Bresenham. Algorithm for Computer Control of a Digital Plotter. *IBM Systems Journal*, 4(1):25–30, 1965. cited p. 52

[Bre94] C. A. Brewer. *Visualization in Modern Cartography*, chapter 7 – Color use guidelines for mapping and visualization, pages 123–147. Modern cartography. Pergamon, Oxford, United Kingdom, 1 edition, 1994. cited p. 123

[Bre01] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001. cited p. 8

[BS08] J. Blanchette and M. Summerfield. *C++ GUI programming with Qt 4*. Open Source Software Development. Prentice Hall, 2 edition, 2008. cited p. 124

[Bun05] M. Bunnell. *GPU Gems 2*, chapter Dynamic Ambient Occlusion and Indirect Lighting, pages 223–233. Addison Wesley, 2005. cited p. 33

[But97] D. R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. cited p. 116

[CB97] M. Couprie and G. Bertrand. Topological Gray-Scale Watershed Transform. In *Proc. of SPIE Vision Geometry V*, volume 3168, pages 136–146, 1997. cited p. 40

[CD02] D. Charpin and J.-M. Durand, editors. *Recueil d'études à la mémoire d'André Parrot, Florilegium marianum 6*, volume 7 of *Mémoires de N.A.B.U.* Paris, France, 2002. cited p. 10, 153

[CDS+04] J. Cohen, D. Duncan, D. Snyder, J. Cooper, S. Kumar, D. Hahn, Y. Chen, B. Purnomo, and J. Graettinger. iClay: Digitizing Cuneiform. In Y. Chrysanthou, K. Cain, N. Silberman, and F. Niccolucci, editors, *Proc. of the 5th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, 2004. cited p. 11

[CGOS11] F. Chazal, L. Guibas, S. Oudot, and P. Skraba. Persistence-based clustering in Riemannian manifolds. In *Proc. 27th Annual ACM Symposium on Computational Geometry*, pages 97–106, France, 2011. cited p. 81

[CKLS07] M. Cheriet, N. Kharma, C.-L. Liu, and C. Y. Suen. *Character Recognition Systems – A Guide for Students and Practiitioners*. John Wiley & Sons, Inc, 2007. cited p. 7

[CRS01] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, 17(2):167–174, 2001. cited p. 35

[CT65] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965. cited p. 81

[CV95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. cited p. 8, 148

[DA09] B. Devereux and G. Amable. *Laser Scanning for the Environmental Sciences*, chapter Airborne LiDAR: Instrumentation, Data Acquisition and Handling, pages 49–66. Wiley, Oxford, United Kingdom, 2009. cited p. 19

[DB96] P. T. Daniels and W. Bright. *The World's Writing Systems*. Oxford University Press, 1996. cited p. 2, 7

[DBS+10] R. Donner, E. Birngruber, H. Steiner, H. Bischof, and G. Langs. Localization of 3D Anatomical Structures Using Random Forests and Discrete Optimization. In *Proc. of Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging - International MICCAI Workshop, MCV*, pages 86–95, Beijing, China, 2010. cited p. 8

[DCCS06] M. Dellepiane, M. Corsini, M. Callieri, and R. Scopigno. High Quality PTM Acquisition: Reflection Transformation Imaging for Large Objects. In M. Ioannides, D. Arnold, F. Niccolucci, and K. Mania, editors, *Proc. of the 7th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, 2006. cited p. 12

[Dig10] J. Digne. *Inverse Geometry: From the raw point cloud to the 3D surface – Theory and Algorithms*. PhD thesis, Ecole normale supérieure de Cachan (ENS Cachan), Cachan, France, 2010. cited p. 47

[DKK+02] N. Demoli, J. Kamps, S. Krüger, H. Gruber, and G. Wernicke. Recognition of Cuneiform Inscription Signs by use of a Hybrid-Optoelectronic Correlator Device. *Applied Optics*, 41(23):4762–4774, 2002. cited p. 10

[DM11] J. Digne and J.-M. Morel. A numerical analysis of differential operators on raw point clouds. *SIAM Journal on Numerical Analysis (SINUM)*, 2011. submitted. cited p. 47

[EH08] H. Edelsbrunner and J. Harer. Persistent Homology —- a Survey. In *Surveys on discrete and computational geometry*, volume 453 of *Contemporary Mathematics*, pages 257–282. American Mathematical Society, 2008. cited p. 81

[Eis02] J. D. Eisenberg. *SVG Essentials*. O'Reilly, 1 edition, 2002. cited p. 127

[ES95] M. Etherington-Smith. *The Persistence of Memory: A Biography of Dali*. Da Capo Press, 1995. cited p. 50

[Feu81] E. Feucht. Ein Relief Scheschonqs I. beim Erschlagen der Feinde aus El-Hibe. *Studien zur altägyptischen Kultur (SAK)*, 8:105ff., 1981. cited p. 117

[Feu00] K. Feuerherm. The Computer Representation of Cuneiform: Towards the Development of a Character Code. Univ. of Birmingham, 2000. Listed as publication of the CLDI related to the 2nd Initiative for Cuneiform Encoding (ICE) Workshop. cited p. 13

[FKN80] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On Visible Surface Generation by A Priori Tree Structures. In *ACM Computer Graphics*, volume 14, pages 124—133, New York, NY, USA, 1980. cited p. 58, 90

[For85] A. R. Forrest. *Fundamental Algorithms for Computer Graphics*, chapter Antialiasing In Practice, pages 113—134. NATO ASI Series F.17. Springer, Berlin, Germany, 1985. cited p. 52

[Gal06] H. D. Galter. Die altorientalischen Altertümer im Landesmuseum Joanneum

– ihre Herkunft und ihre Bedeutung. *Schild von Steier – Archäologische und numismatische Beiträge aus dem Landesmuseum Joanneum*, 19:15–18, 2006. cited p. 192

[GMH07] C. F. Gauss, J. C. Morehead, and A. M. Hiltebeitel. *General Investigations of Curved Surfaces*. Watchmaker Publishing, 2007. cited p. 39

[Gra30] W. C. Graustein. *Homogeneous Cartesian Coordinates. Linear Dependence of Points and Lines*, chapter Introduction to Higher Geometry, pages 29–49. New York: Macmillan, 1 edition, 1930. cited p. 26

[GTHD03] A. Gardner, C. Tchou, T. Hawkins, and P. Debevec. Linear Light Source Reflectometry. In *Proc. of Int. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 749–758, San Diego, CA, USA, 2003. ACM. cited p. 38

[GWLU05] B. Groneberg, F. Weiershäuser, T. Linnemann, and D. Ullrich. *Jahrbuch der Max-Planck-Gesellschaft*, chapter Digitale Keilschriftbibliothek Lexikalischer Listen aus Assur. Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Göttingen, Germany, 2005. cited p. 14

[Ham41] R. Hampe. *Der Wagenlenker von Delphi*. F. Bruckmann, Munich, Germany, 1941. cited p. 20

[HBD07] D. V. Hahn, K. C. Baldwin, and D. D. Duncan. Non-laser-based scanner for three-dimensional digitization of historical artifacts. *Applied Optics Journal*, 46:2838—-2850, 2007. cited p. 11

[Hec66] K. Hecker. *Die Keilschrifttexte der Universitätsbibliothek Giessen unter Benutzung nachgelassener Vorarbeiten von Julius Lewy*, volume 9 of *Berichte und Arbeiten aus der Universitätsbibliothek Giessen*. Justus Liebig-Universität, Giessen, Germany, 1966. cited p. 12

[HFG+06] Q.-X. Huang, S. Flöry, N. Gelfand, M. Hofer, and H. Pottmann. Reassembling Fractured Objects by Geometric Matching. In *Proc. of Int. Conference on Computer Graphics and Interactive Technique (SIGGRAPH)*, pages 569–578, Boston, MA, USA, 2006. ACM. cited p. 41

[Hil08] M. Hilgert. Cuneiform Texts in the Collection of St. Martin Archabbey Beuron. *Cuneiform Digital Library Journal (CDLJ)*, 2:online, 2008. cited p. 14

[HK09] C. Hoppe and S. Krömker. Adaptive Meshing and Detail-Reduction of 3D-Point Clouds from Laser Scans. In F. Remondino, S. El-Hakim, and L. Gonzo, editors, *Proc. of the 3rd ISPRS International Workshop 3D-ARCH: "3D Virtual Reconstruction and Visualization of Complex Architectures"*, Trento, Italy, 2009. ISPRS Commission V - WG 4. cited p. 26

[Ho95] T. Ho. Random Decision Forest. In *Proc. of the 3rd International Conference on Document Analysis and Recognition (ICDAR)*, pages 278—-282, Montreal, Canada, 1995. IEEE Computer Society. cited p. 8

[HT03] D. Hulin and M. Troyano. Mean Curvature and Asymptotic Volume of Small Balls. *The American Mathematical Monthly*, 110(10):947–950, 2003. cited p. 47, 62

[HW99] A. Habib and J. Warren. Edge and Vertex Insertion for a Class of C^1 Subdivision Surfaces. *Computer Aided Geometric Design*, 16(4):223–247, 1999. cited p. 38, 158

[HW11] H. Hameeuw and G. Willems. New Visualization Techniques for Cuneiform Texts and Sealings. *Akkadica*, 132(2):in print, 2011. cited p. 12

[Jäh05] B. Jähne. *Digital Image Processing*. Springer, 6 edition, 2005. cited p. 52, 94

[Jak09] S. Jakob. *Die mittelassyrischen Texte aus Tell Chuera in Nordost-Syrien*. Harrassowitz, O., Wiesbaden, Germany, 2009. Tafel 26. cited p. 3, 4

[Jol02] I. Jolliffe. *Principal Components Analysis*. Springer Series in Statistics. Springer, 2 edition, 2002. cited p. 37

[JSC04] W. Jung, H. Shin, and B. Choi. *Computer-Aided Design and Applications*, chapter Self-Intersection Removal in Triangular Mesh Offsetting, pages 477–484. 2004. cited p. 32

[Jun10] D. Jungblut. *Rekonstruktion von Oberflächenmorphologien und Merksmalketten aus dreidimensionalen Daten unter Verwendung hochparalleler Rechnerarchitekturen*. PhD thesis, Goethe-Universität Frankfurt am Main, Goethe-Zentrum für Wissenschaftliches Rechnen (G-CSC), Arbeitsgruppe Simulation und Modellierung, 2010. cited p. 11

[Kap10] A. Kapornaki. The Military's Role in Seizing War Booty and Protecting Cultural Treasures. *Forum Archaeologiae*, 55(VI):online, 2010. cited p. 3

[KBH06] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson Surface Reconstruction. In *Eurographics Symposium on Geometry Processing*, 2006. cited p. 35

[Kil04] M. Kilian. Differentialgeometrische Konzepte fur Dreiecksnetze. Master's thesis, Universität Karlsruhe, Institut für Betriebs- und Dialogsysteme und Mathematisches Institut II, 2004. cited p. 37

[KMT08] R. Kalasek, H. Mara, and H. Taeuber. Reading Weathered Ancient Laws in 3rd Dimension. In *13. Int. Tagung: Kulturelles Erbe und Neue Technologien*, Vienna, Austria, 2008. cited p. 5

[Köt08] U. Köthe. *Reliable Low-Level Image Analysis*, chapter Geometric Sampling Theorems, pages 165–190. book version in preparation, Department Informatik, University of Hamburg, Germany, 2008. Habilitation. cited p. 19, 137

[Kra07] A. Krause. *Foundations of GTK+ Development*. APress, New York City, United States, 1 edition, 2007. cited p. 124

[Lab88] R. Labat. *Manuel D'epigraphie Akkadienne*. Librairie Orientaliste Paul Geuthner, Paris, France, 1988. cited p. 13

[Lan03] M. Landfester. Der Beitrag der Gießener Papyrussammlungen als Quellen für die Geschichte Ägyptens und ihre Digitalisierung. *Spiegel der Forschung (Giessen)*, 20:136–145, 2003. H.1/2. cited p. 12

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, volume 21, New York, NY, USA, 1987. cited p. 92

[Led04] L. Ledderose. Carving Sutras into Stone before the Catastrophe: The Inscription of 1118 at Cloud Dwelling Monastery near Beijing. In *Proc. of The British Academy*, volume 125 of *Elsley Zeitlyn Lecture on Chinese Archaeology and Culture*, pages 381–454. The British Academy, 2004. cited p. 4

[Lie03] P. Liepa. Filling Holes in Meshes. In *Proc. of Eurographics Symposium on Geometry Processing (SGP)*, pages 200–207, Aire-la-Ville, Switzerland, 2003. Eurographics Association. cited p. 121

[LZH+07] Y.-K. Lai, Q.-Y. Zhou, S.-M. Hu, J. Wallner, and H. Pottmann. Robust Feature Classification and Editing. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 13(1):34–45, 2007. cited p. 41

[Mar06] H. Mara. Documentation of Rotationally Symmetrical Archaeological Finds by 3D Shape Estimation. Master's thesis, Pattern Recognition and Image Processing (PRIP) Group, Vienna University of Technology, 2006. Supervisor: Robert Sablatnig – see PRIP technical report nr. 83. cited p. 20, 25, 39, 109

[Mar09] H. Mara. *New Technologies for Archaeology - Multidisciplinary Investigations in Palpa and Nasca, Peru*, chapter Pottery plotted by Laser - 3D-Acquisition

for Documentation and Analysis of Symmetry of Ancient Ceramics, Part V – Geomatics, pages 379–408. Number XVII in Natural Science in Archaeology. Springer, 1 edition, 2009. cited p. 20

[Mau05] S. M. Maul. *Das Gilgamesch-Epos*. C.H.Beck, Munich, Germany, 3 edition, 2005. cited p. 2, 115

[MB05] T. McReynolds and D. Blythe. *Advanced Graphics Programming using OpenGL*. Elsevier, 2005. cited p. 30, 33, 122

[MCH+06] S. Manay, D. Cremers, B.-W. Hong, A. J. Y. Jr., and S. Soatto. Integral Invariants for Shape Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1602–1616, 2006. cited p. 90

[MDSB03] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. *Visualization and Mathematics III*, chapter Discrete Differential-Geometry Operators for Triangulated 2-Manifolds, pages 35–58. Mathematics and Visualization. Springer, Heidelberg, 2003. cited p. 37, 38, 39, 40

[MGW01] T. Malzbender, D. Gelb, and H. Wolters. Polynomial Texture Maps. In *Proc. of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH).*, Los Angeles, CA, USA, 2001. cited p. 12

[MHK09] H. Mara, J. Hering, and S. Krömker. GPU based Optical Character Transcription for Ancient Inscription Recognition. In *Proc. of 15th International Conference on Virtual Systems and Multimedia (VSMM) – "Vision or Reality? Computer Technology and Science in Art, Cultural Heritage, Entertainment and Education"*, pages 154–159, Vienna, Austria, 2009. cited p. 4

[MHYS04] S. Manay, B. Hong, A. J. Yezzi, and S. Soatto. Integral Invariant Signatures. In *Proc. Eighth European Conf. Computer Vision (ECCV)*, pages 87–99. Springer, 2004. cited p. 41

[MK03] H. Mara and M. Kampel. Automated Extraction of Profiles from 3D Models of Archaeological Fragments. In *Proc. of the XIX CIPA Int. Symposium: New Perspectives to Save Cultural Heritage,*, pages 87–93, Antalya, Turkey, 2003. cited p. 42, 44

[MKS02] H. Mara, M. Kampel, and R. Sablatnig. Preprocessing of 3D-Data for Classification of Archaeological Fragments in an Automated System. In *Proc. of the 26th Workshop "Vision with Non-Traditional Sensors" of the Austrian Association for Pattern Recognition (ÖAGM/AAPR)*, pages 257–264, Graz, Austria, 2002. cited p. 20

[MKT08] H. Mara, P. Kammerer, and E. Trinkl. Documentation of Polychrome Ceramics using 3D-Acquisition and Multi-Spectral Readings. In *36. Computer Applications in Archaeology – "On the Road to Reconstructing the Past"*, page to appear, Budapest, Hungary, 2008. cited p. 23

[MMS08] H. Mara, A. Monitzer, and J. Stöttinger. Introducing 3D Vision and Computer Graphics to Archaeological Workflow – an Applicable Framework. In *3rd International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 305–312, Funchal, Madeira, Portugal, 2008. Springer. cited p. 20

[Moo65] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, 1965. cited p. 116

[MP13] H. Mara and J. Portl. *Neue interdisziplinäre Dokumentations- und Visualisierungsmethoden*, chapter Acquisition and Documentation of Vessels using High-Resolution 3D-Scanners. Corpus Vasorum Antiquorum Österreich, Beiheft 1. Verlag der Österreichischen Akademie der Wissenschaften (VÖAW), Vienna, Austria, 2013. Trinkl, E. (ed.). cited p. 17

[MSS84] C. Montani, R. Scateni, and R. Scopigno. Discretized Marching Cubes. In *Proc. of Visualization '94*, pages 281–287. IEEE Computer Society Press, 1984. cited p. 92

[MTKZ07] H. Mara, E. Trinkl, P. Kammerer, and E. Zolda. 3D-Acquisition and Multi-Spectral Readings for Documentation of Polychrome Ceramics of the Antiquities Collection of the *Kunsthistorisches Museum* Vienna. In J. Trant and D. Bearman, editors, *Proc. of the International Cultural Heritage Informatics Meeting (ICHIM07)*, pages CD–ROM, Toronto, Ontario, Canada, 2007. Toronto: Archives & Museum Informatics. cited p. 20

[NDE91] H. J. Nissen, P. Damerow, and R. K. Englund. *Frühe Schrift und Techniken der Wirtschaftsverwaltung im alten Vorderen Orient*, page 200. Franzbecker, 2 edition, 1991. Fig. 19. cited p. 7

[Neu07] H. Neumann. *Das geistige Erfassen der Welt im Alten Orient. Beiträge zu Sprache, Religion, Kultur und Gesellschaft*, chapter "Gib mir mein Geld zurück!" – Zur rechts- und wirtschaftsgeschichtlichen Bedeutung keilschriftlicher Privatarchive des 3. Jahrtausends v. Chr., pages 281–299. Harrassowitz Verlag, Wiesbaden, Germany, 1 edition, 2007. cited p. 2

[NK02] M. Novotni and R. Klein. Computing Geodesic Distances on Triangular Meshes. In *Proc. of 10th Int. Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 341–347, 2002. cited p. 92, 109

[OBB00] Y. Ohtake, A. G. Belyaev, and I. A. Bogaevski. Polyhedral Surface Smoothing with Simultaneous Mesh Regularization. In *Proc.Geometric Modeling and Processing*, 2000. cited p. 35

[OTV93] C. Orton, P. Tyers, and A. Vince. *Pottery in Archaeology*. Cambridge University Press, 1993. cited p. 42

[Owe75] D. Owen. *The John Frederick Lewis Collection Texts from the Third Millennium in the Free Library of Philadelphia*, volume 3 of *Materiali per il Vocabolario Neosumerico*. Unione Accademica Nazionale - Multigrafica Editrice, Rome, Italy, 1975. cited p. 10

[PCK04] B. Purnomo, J. D. Cohen, and S. Kumar. Seamless Texture Atlases. In R. Scopigno and D. Zorin, editors, *Proc. of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, volume 71 of *ACM International Conference Proceeding Series*, pages 65–74, Nice, France, 2004. Eurographics Association, Association for Computing Machinery. cited p. 11

[PCMA08] D. Pitzalis, P. Cignoni, M. Menu, and G. Aitken. 3D Enhanced Model from Multiple Data Sources for the Analysis of the Cylinder Seal of Ibni-Sharrum. In M. Ashley, S. Hermon, A. Proenca, and K. Rodriguez-Echavarria, editors, *Proc. 9th International Symposium on VAST International Symposium on Virtual Reality, Archaeology and Cultural Heritage*, pages 79–84. The Eurographics Association, 2008. cited p. 11

[Ped97] O. Pedersén. *Katalog der beschrifteten Objekte aus Assur*, page 209. Number 23 in Abhandlungen der Deutschen Orient-Gesellschaft (ADOG). Saarbrücken, Germany, 1997. cited p. 129

[PH12] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier, 4 edition, 2012. cited p. 116

[Píš99] K. Píška. Fonts for Neo-Assysian Cuneiform. In G. Partosch and G. Wilhelms, editors, *Proceedings of the EuroTEX Conference (Paperless TEX)*, pages 142–154, Heidelberg, Germany, 1999. cited p. 13

[Píš04] K. Píška. Creating Type 1 fonts from Metafont sources: comparison of tools, techniques and results. In *TeX, XML, and Digital Typography*, volume 3130 of *Lecture Notes Computer Science*, pages 240—-256, Berlin, Germany, 2004. Springer. cited p. 130

[Por10] J. Portl. Subdivision Curve and Surface Fitting for Mesh Compression. Master's thesis, IWR – Interdisciplinary Center for Scientific Computing, Heidelberg University, 2010. cited p. 34

[PR97] J. Peters and U. Reif. The Simplest Subdivision Scheme for Smoothing Polyhedra. *ACM Transactions on Graphics (TOG)*, 16(4):420–431, 1997. cited p. 38, 158

[PR08] J. Peters and U. Reif. *Subdivision Surfaces*, volume 3 of *Geometry and Computing*. Springer, 2008. cited p. 22

[Pre10] A. Pressley. *Elementary Differential Geometry*. Springer Undergraduate Mathematics Series (SUMS). Springer, 2 edition, 2010. cited p. 36

[PSK+02] D. Page, Y. Sun, A. Koschan, J. Paik, and M. Abidi. Normal Vector Voting: Crease Detection and Curvature Estimation on Large, Noisy Meshes. *Graphical Models*, 64(3–4):199–229, 2002. Special issue: Processing on large polygonal meshes. cited p. 37

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in Fortran 77*, volume 1, chapter Evaluation of Functions - Quadratic and Cubic Equations, pages 178–180. Cambridge University Press, 1992. cited p. 65

[PWHY09] H. Pottmann, J. Wallner, Q.-X. Huang, and Y.-L. Yang. Integral Invariants for Robust Geometry Processing. *Computer Aided Geometric Design*, 26(1):37–60, 2009. cited p. 41, 47, 62, 81

[QTK+09] U. Quatember, B. Thuswaldner, R. Kalasek, C. Bathow, and B. Breuckmann. The Virtual and Physical Reconstruction of the Octagon and Hadrian's Temple in Ephesus. In H. G. Bock, W. Jäger, H. Mara, and M. Winckler, editors, *Proc. 2nd Workshop on Scientific Computing for Cultural Heritage (SCCH)*, page in print, Heidelberg, Germany, 2009. Springer. cited p. 20, 23

[Ray03] E. T. Ray. *Learning XML*. O'Reilly, 2 edition, 2003. cited p. 127

[RHP08] M. Rutzinger, B. Höfle, and N. Pfeifer. *Object-Based Image Analysis*, chapter Object detection in airborne laser scanning data - an integrative approach on object-based image and point cloud analysis, pages 645–662. Springer, 2008. cited p. 19

[Ric53] N. Ricker. Wavelet Contraction, Wavelet Expansion, and the Control of Seismic Resolution. *Geophysics*, 18(4):769–, 1953. cited p. 81

[RML12] B. Rieck, H. Mara, and H. Leitte. Multivariate Data Analysis Using Persistence-based Clustering and Topological Signatures. In *Proc. of VisWeek*, page accepted, Seattle, Washington, USA, 2012. cited p. 81

[Ros97] A. Roshop. *Digitale Mustererkennung an holographischen Bildern von Keilschrifttafeln*. Physik. Herbert Utz Verlag, 1997. Phd Thesis. cited p. 10, 11

[RP66] A. Rosenfeld and J. L. Pfaltz. Sequential Operations in Digital Picture Processing. *Journal of the Association for Computing Machinery (JACM)*, 13(4):471–494, 1966. cited p. 115

[Ruc96] W. Rucklidge. *Efficient Visual Recognition using the Hausdorff Distance*. Springer, 1996. cited p. 35

[RWPD05] E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*. The Morgan Kauf-

mann Series in Computer Graphics. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. cited p. 17

[SBK10] B. Snyder, R. Barzilay, and K. Knight. A Statistical Model for Lost Language Decipherment. In *Proc. of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1048–1057, Uppsala, Sweden, 2010. cited p. 13

[Sch11] B. Schäling. *The Boost C++ Libraries*. XML Press, 2011. cited p. 124

[SG66] G. W. Stroke and D. Gabor. *An Introduction to Coherent Optics and Holography*. Academic Press, 1966. cited p. 10

[SGFT12] C. Siart, M. Ghilardi, M. Forbriger, and K. Theodorakopoulou. Terrestrial Laser Scanning and Electrical Resistivity Tomography as Combined Tools for the Geoarchaeological Study of the Kritsa-Latô Dolines (Mirambello, Crete, Greece). *Géomorphologie: relief, processus, environnement*, (1):59–74, 2012. cited p. 19

[Sha48] C. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948. cited p. 19, 37

[Sha84] C. Shannon. Communication in the Presence of Noise. *Proceedings of the IEEE*, 72(9):1192–1201, 1984. cited p. 19

[SM92] R. Sablatnig and C. Menard. Stereo and Structured Light as Acquisition Methods in the Field of Archaeology. In S. Fuchs and R. Hoffmann, editors, *Mustererkennung '92, 14. DAGM-Symposium Dresden*, pages 398–404. Springer, 1992. cited p. 11

[Sod94] W. von Soden. *The ancient Orient: an introduction to the study of the ancient Near East*. Wm. B. Eerdmans Publishing Co., 1994. cited p. 2, 7

[Spe81] G. Sperl. *Erkennen von Keilschriftzeichen mit Hilfe Elektronischer Rechenanlagen*. PhD thesis, Leopold-Franzens-Universität Innsbruck, Innsbruck, Austria, 1981. cited p. 10

[Spi79] M. Spivak. *A Comprehensive Introduction to Differential Geometry*, volume 2 of *Mathematics*. Publish or Perish, inc., 1979. cited p. 36, 157

[Ste91] H. Steible. *Die neusumerischen Bau- und Weihinschriften I*, volume 9 of *Freiburger altorientalische Studien*. Steiner, Stuttgart, Germany, 1991. cited p. 178

[Str00] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, Amsterdam, Netherlands, 3 edition, 2000. cited p. 117

[SV99] J. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, 1999. cited p. 8, 148

[TF95] E. Trucco and R. B. Fisher. Experiments in Curvature-Based Segmentation of Range Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):177–182, 1995. cited p. 39

[Thi87] W. C. Thibault. *Application of Binary Space Partitioning Trees to Geometric Modeling and Ray-Tracing*. PhD thesis, Georgia Institute of Technology Atlanta, GA, USA, 1987. cited p. 90

[Uni06] Unicode Consortium. *The Unicode Standard, Version 5.0*. Addison-Wesley, Boston, MA, USA, 5 edition, 2006. cited p. 13

[VGL+05] C. Vandecasteele, L. Van Gool, K. Van Lerberghe, J. Van Rompay, and P. Wambacq. *Images and Artifacts of the Ancient World*, chapter Digitizing the Cuneiform Tablets from Beydar, pages 85–96. Royal Society/Britsh Academy, London, UK, 2005. cited p. 10, 153

[VH96] T. Várady and T. Hermann. Best Fit Surface Curvature at Vertices of Topologically Irregular Curve Networks. In *Proc. of the 6th IMA Conference on the*

Mathematics of Surfaces, 1996. cited p. 39

[WDF+02] S. I. Woolley, T. Davis, N. Flowers, J. Pinilla-Dutoi, A. Livingstone, and T. Arvanitis. Communicating Cuneiform: The Evolution of a Multimedia Cuneiform Database. *The Journal of Visible Language*, 36(3):308–324, 2002. cited p. 11, 16, 20

[Wei09] F. Weiershäuser. Die Online-Datenbank der lexikalischen Texte aus Assur. *Mitteilungen der Deutschen Orient-Gesellschaft zu Berlin (MDOG)*, pages 117–141, 2009. cited p. 14

[Wel78] W. Welling. *Photography in America: The Formative Years, 1839-1900 - A Documentary History*. Crowell, 1978. cited p. 10

[WGD+98] G. K. Wernicke, H. Gruber, N. Demoli, M. Senoner, G. von Bally, F. Dreesen, and A. Roshop. Applications of Color Holography in the Investigation of Handwritten Cultural Historic Sources. In *Proc. of SPIE:: 6th Int. Symposium on Display Holography*, volume 3358, Lake Forest, IL, USA, 1998. SPIE – International Society for Optical Engineering. cited p. 10

[Whi80] T. Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–349, 1980. cited p. 33

[WMW06] S. Winkelbach, S. Molkenstruck, and F. M. Wahl. Low-Cost Laser Range Scanner and Fast Surface Registration Approach. In *Proc. of DAGM 2006*, volume 4174 of *Lecture Notes in Computer Science*, pages 718–728. Springer, 2006. cited p. 19

[WS03] L. Watkins, Jr. and D. A. Snyder. The Digital Hammurabi Project. In *Proc. of Museums and the Web*, pages CD–ROM, Online, Charlotte, NC, USA, 2003. Toronto: Archives & Museum Informatics. cited p. 11

[WVM+05] G. Willems, F. Verbiest, W. Moreau, H. Hameeuw, K. Van Lerberghe, and L. Van Gool. Easy and Cost-Effective Cuneiform Digitizing. In M. Mudge, N. Ryan, and R. Scopigno, editors, *Proc. of 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, pages 73–80, Pisa, Italy, 2005. cited p. 12

[YLHP06] Y.-L. Yang, Y.-K. Lai, S.-M. Hu, and H. Pottmann. Robust Principal Curvatures on Multiple Scales. In *SGP 2006: 4th Eurographics Symposium on Geometry processing*, pages 223–226. Eurographics Association, 2006. cited p. 47

[Zus93] K. Zuse. *The Computer, My Life*. Springer, 1993. cited p. 10