University of West Bohemia
Faculty of Applied Sciences

Curvature-based shape characteristics of geometrical objects and their use in computer graphics

Ing. Martin Prantl

Doctoral thesis
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Computer Science and Engineering

Supervisor: Prof. Dr. Ing. Ivana Kolingerová
Department: Department of Computer Science and Engineering

Pilsen 2019

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Tvarové charakteristiky založené na křivosti a jejich použití v počítačové grafice

Ing. Martin Prantl

Disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: Prof. Dr. Ing. Ivana Kolingerová
Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2019

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že tuto práci jsem zpracoval samostatně s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni dne Ing. Martin Prantl

Contents

Abstract

Curvature is an important geometric property in computer graphics that provides information about the object surfaces. The exact curvature can only be calculated for a limited set of surface descriptions. Most of the time, we deal with triangles, point sets or some other discrete representation of the surface and for those, curvature can only be estimated.

Curvature and other shape characteristics computed from it can be used for various purposes starting from geometry based problems to rendering. Most of the time, we use shape characteristics for object description and a creation of feature vectors that condense main information about the geometry. The feature vectors are usually further used for object recognition, registration, rendering, etc. Curvature is often used as a starting point. A combination with other characteristics improves the quality of the final description.

In this thesis, we focus on the efficient computation and suitable use of curvature-based shape characteristics. We present a technique for high quality curvature estimation at a price of reduced performance. On the other hand, for interactive use the curvature needs to be estimated fast at a price of quality. A solution for this task is presented as well. Curvature for geometry processing is presented on the models of human heads. We utilize curvature and related characteristics to find key-points on these models. During the data aquisition (eg. from a scanning device), we need to align the scanned, overlapping parts. The problem with symmetrical scans is presented and a possible solution proposed.

Abstrakt

Křivost hraje v počítačové grafice důležitou úlohu. Poskytuje informace o površích objektů. Přesné hodnoty křivosti lze vypočítat pouze pro omezenou množinu dat. Většinu času pracujeme s trojúhelníky, mračny bodů nebo jinou diskrétní reprezentací povrchu. Pro tato data lze křivost pouze odhadnout a přesné hodnoty nejsou k dispozici.

Křivost spolu s dalšími vlastnostmi lze využít pro různé problémy, od geometrických po vizualizační. Většinou používáme tvarové charakteristiky pro popis objektů a tvorbu příznačných vektorů, které obsahují hlavní a nejdůležitější informace o dané geometrii. Příznačné vektory se obvykle dále používají pro rozpoznávání objektů, registraci, vykreslování atd. V mnoha případech se jako výchozí charakteristika používá právě křivost. Její kombinace s dalšími charakteristikami pak zlepšuje kvalitu výsledného popisu geometrie.

V této práci se zaměřujeme především na efektivní výpočet a vhodné využití tvarových charakteristik založených na křivosti. Je prezentován algoritmus pro vysoce kvalitní odhad křivosti za cenu nižší rychlosti algoritmu. Na druhou stranu, často je potřeba mít k dispozici křivost v reálném čase za cenu nižší kvality jejího odhadu. Řešení tohoto problému je v práci také navrženo. Využití křivosti pro zpracování geometrie je prezentováno na modelech lidských hlav. Křivost a související charakteristiky jsou tu využity k nalezení klíčových bodů. Při získávání dat (např. ze skenovacího zařízení) potřebujeme zarovnat naskenované, překrývající se části. Je prezentován problém symetrických skenů a navrženo vhodné řešení.

Acknowledgement

I wish to express my gratitude to all the people who supported me during the realization of this thesis. First and foremost, I would like to thank to my supervisor Prof. Dr. Ing. Ivana Kolingerová for her valued advice and never-ending patience. This thesis would not have been possible without Doc. Ing. Libor Váša, PhD., whose experience in this field of research was very helpful. My thanks also go to my family and my friends whose support was very important during my studies.

Chapter 1

Introduction

There are many ways how to characterize an object. We can describe an object by its weight, color, material, shape etc. The shape of object is probably the most important property and to describe it is an important research goal in computer graphics. One possible way is to use shape characteristics [1] to better understand objects. We can imagine this in the following way: if we take a "black-box" and "feed" it with an object, it will return a shape characteristic, usually in a form of a vector or a single number. It is a simplified representation that tries to carry most of the important information, while being easier to handle, to store and to compare than the shape itself directly. This characterization can be local (created from a neighborhood or a region of a certain size) or global (created from the entire object).

Shape characteristics must meet certain requirements such as: invariance to geometric changes (rotation, translation, sometimes scaling), robustness to noise, robustness to sampling errors etc. The more of these we fulfill the better characteristic we have. To meet these requirements, a solution for a "black-box" have to be proposed.

One of the most important shape characteristics is curvature. This topic is well-known over hundred years. Over the years, there were many different algorithms with a different quality and performance. However, the exact curvature cannot be computed for discrete data. This problem cannot be solved, since we only have a subset of information and in the remaining parts the data can be of any kind. We can only improve the quality to be as closest as possible to the exact values. This can only be tested and compared if we have discrete data together with their original smooth representation. In the end, the quality of the final estimation depends on several factors. The most important are the data itself. More triangles or more points, it all leads to a better estimation of the original model and thus to more exact curvature estimation. Another important factor is the speed of the estimation process. The more exact curvature estimation can be done but the calculation becomes slower. However, to fully describe model or a local region, the curvature itself is not always enough and other characteristics have to be utilized. They can be based on: angles between triangles, distances of points, topology of models, volume, area etc.

[1] An alternative term for the shape characteristics is a shape descriptor. From our point of view, a characteristic of an object provides its description; hence we use the terms characteristic and descriptor interchangeably in this report.

Once the curvature and possible other characteristics of the object are obtained, there are many ways how to use them. They can show important parts, such as: edges, sharp features, regions of interest etc. Other very common use cases are object matching and recognition. Based on a set of descriptors from one model, a similarity with another model can be expressed. The user can build a database of descriptors and try to match an object by descriptors to the ones that are already in the database to find a similarity. Shape characteristics can also be used for visualization purposes. Based on the surface properties, certain effects can be achieved or even computed faster. For example, curvature can be used in lighting to create lighter convex and darker concave areas. Another use can be to show to the user some regions of interest that can be highlighted (by color, different rendering style etc.). There is almost an unlimited number of other use-case scenarios.

In computer graphics, the basic representation of objects are usually triangle meshes or point clouds. However, they both represent only an approximation of the original model and the same dataset can be obtained for different models. Based on that, these representations cause problems, since we are not able to restore the original model but only its approximation. This must be taken into consideration if we work with shape characteristics.

1.1 Problem definition

This thesis aims to the problematic of shape characteristics and their use in computer graphics. Since one of the most important shape characteristic is curvature, the first main objective is therefore curvature estimation that is either fast or very accurate. To meet both criteria together is almost impossible for general cases. Many existing solutions were created to be a trade-off between the performance and the quality. However, only very few of them are easy-to-implement and real-time. To create such a solution is one of our goals. In some cases, the available data are of high quality (almost noise-free, exactly calculated normal vectors etc.). There is no curvature estimation algorithm that is primary targeted to this kind of data. Such an algorithm can benefit from the data quality. In our research, we sometimes deal with this kind of data and therefore an algorithm with the best possible quality is desired.

The second objective is to use the estimated curvature in a combination with other shape characteristics to create a shape description. In this case, the main area of interest are scanned or manually created data of human heads. We aimed our research to this kind of models because of a joint research with our colleagues. The human head models can be used e.g. in computer games for personalization of a user avatar, for a 3D facial composite used by the police etc. The location of anchor points and robust description is crucial since it influences the quality of data modification. In this case, anchor points can be located manually, which is not very convenient, or by an algorithm. The algorithms differ in type of located points and one algorithm cannot be often easily bent to find different points. Another important field is acquiring of human head models. One way is to create them manually in a modeling software, however, this is a time consuming task. An automated solution with the scanning device is preferred. The scanned data need to be registered and we have focused on problems caused by this automatic approach, mainly in the area of data symmetry.

1.2 Summary of contributions

The *first* contribution [PVK16] is a curvature estimation algorithm designed for a better performance. The method is based on an existing and easy to implement solution from [Rus04]. In our contribution, the curvature is estimated in the screen space in real time. This solution can be used as a first, fast estimation that can be later made more precise with non-real time performance. The algorithm can also be used for lighting (shading) purposes as presented in our *second* contribution [PVK17].

The *third* contribution [PV18] presents another curvature estimation algorithm. However, unlike our first contribution, this one is related to the estimation of curvature with respect to the quality of the result. This research is based on our overview of curvature estimation algorithms [Váš+16] where none of them benefits from high-quality input data. Our new curvature estimation algorithm is based on a simple and a well-known idea of surface fitting. However, the fitting process is improved to use a variation of Radial Basis Function interpolation. An advantage of the algorithm is also its possibility to improve the quality of already estimated normal vectors.

The *fourth* contribution [Pra+17] describes a use of curvature and curvature-based shape characteristics for a detection of feature points on the models of human heads. Once the feature points are found, they are used as anchor points for deformations of a human head model. Apart from the deformations, the feature points can be used to obtain feature regions associated with them (eyes, nose, mouth, ears areas). These regions can further be utilized in different applications (modeling, registration, rendering etc.).

The *fifth* contribution [PVK19] is a method for alignment of two partially overlapping scans of symmetrical models with main focus on human heads that were used in our previous research [Pra+17; Sko+17]. The method is based on an existing algorithm [RBB09] that is improved by additional symmetry information. This information is based on local coordinate system in a given vertex and a curvature estimated from its neighborhood.

Our proposed methods are based mainly on triangle meshes. However, some of the proposed solutions (mainly [PV18]) can be directly used or modified to work with point clouds as well. All proposed methods have been implemented and published. Apart from our contributions, the results were also used in other research activities of our collegues.

1.3 Structure of the thesis

The thesis begins with a description of basic terms (Chapter 2) that are used through all sections. Description of some interpolation techniques that we have used for our research is provided in Chapter 3. Basic theory behind curvature is covered in Chapter 4. Description of selected state-of-the-art methods in curvature estimation is presented in Chapter 5. Shape characteristics, mostly curvature-based, are described in Chapter 6. However, other characteristics are discussed as well (Normal vectors, Fourier transformation, Voxelization, Euclidean distances etc.).

Our contributions are presented in the second part of the thesis. This section is based mainly on our publications from [PVK16; PV18; Pra+17].

The screen space version of curvature estimation algorithm for fast estimation is described in Chapter 7. This solution is targeted to an interactive curvature estimation in the screen space on a GPU. The algorithm for high-quality curvature estimation based on Hermite Radial Basis Functions (HRBF) is presented in Chapter 8. Further use of curvatures for finding feature points on human heads is described in Chapter 9. To deal with a problem of incorrect registration of symmetrical parts of human head, symmetry-aware registration algorithm is presented in Chapter 10.

The final Chapter 11 concludes the thesis.

The used notation is as follows: ***bolditalic*** symbols represent vectors (column order), symbol "$\cdot$" denotes the dot product, "$\times$" denotes the cross product, $|\boldsymbol{x}|$ is the vector length and $det(X)$ is the determinant of a matrix X.

The color gradient used for all visualizations goes from the blue for negative values to the red color for positive values. The green color in the middle represents zero. See Figure 1.1.

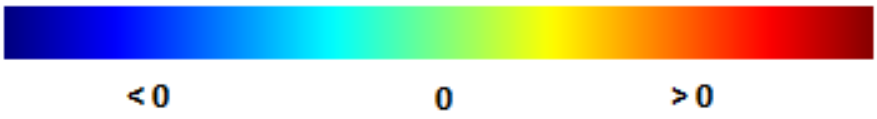


Figure 1.1: The color gradient used in all presented visualizations

Chapter 2

Basic terms

First, we will briefly introduce some of the basic terms needed for this thesis. In this work, we concentrate primarily on geometrical objects represented by triangle meshes. For this representation, we need to define basic terms regarding triangles and triangulated geometry. The basic notation used in this text is as follows, see also Figure 2.1.

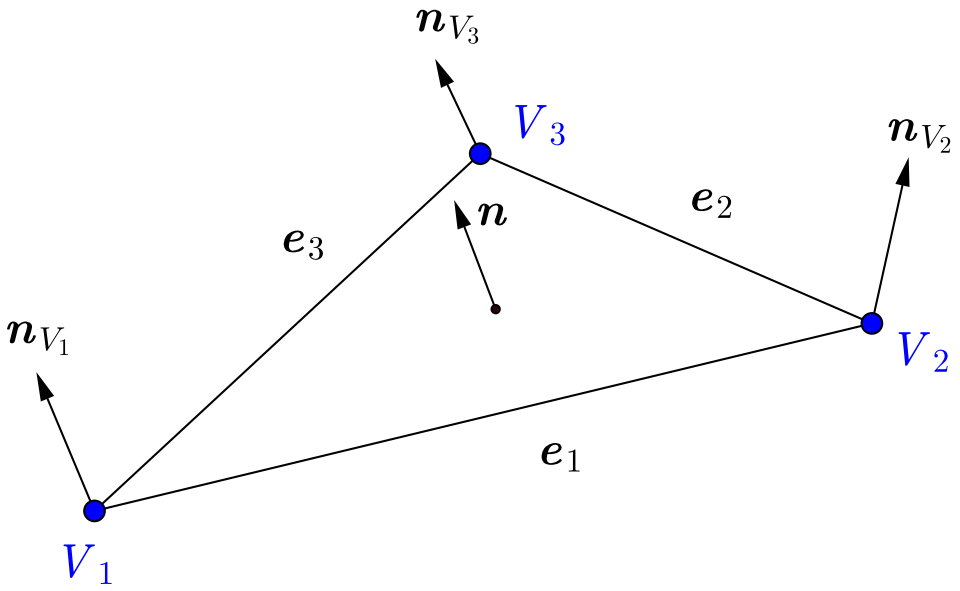


Figure 2.1: Triangle labeling

The geometry is represented as a set of points $P = \{P_i\}_{i=1,2...M}$, where M is a total number of points. For point clouds, the cloud is represented by this set of points.

In case of triangle meshes, each triangle consists of three points denoted as vertices V_i, where $i = 1, 2, 3$. In the used equations, notation $i+1$ is actually $(i\,modulo\,3)+1$ because of circular indexing of triangle vertices. However, we used the shortened notation to keep equations more readable. We expect that the vertices V_i are located on the original surface from which the triangulated version is created.

Vectors $\boldsymbol{e}_i$ are triangle edges computed as $\boldsymbol{e}_i = V_{i+1} - V_i$. Vector $\boldsymbol{n}$ is the triangle

normal vector and vectors $\boldsymbol{n}_{V_i}$ are normal vectors at particular triangle vertices.

We also need to establish the term *neighborhood*. For triangulated geometry, the neighborhood of a vertex is consisting of all vertices that are distant up to some threshold. This threshold distance can be expressed either by the number of edges k (this is referred as k-ring) or as an Euclidean distance.

In case of point clouds, the neighborhood of a point consist of all points that are withing the Euclidean distance from this point. There is no equivalent for k-ring.

2.1 Normal vectors

We often work with a discrete representation of the original model and we are usually not able to retrieve the original (and therefore exact) normal vectors. We can compute only an estimation. The quality of this estimation is crucial for many geometry processing algorithms such as curvature calculation, surface reconstruction, matching and recognition of shapes etc.

For a triangulated geometry, we can talk about two types of normal vectors - normals of triangle faces and normals at triangles vertices.

Normal vector $\boldsymbol{n}$ of a triangle face can be computed exactly by the direct approach using the cross product. The normal $\boldsymbol{n}$ for a triangle is computed as

$$\boldsymbol{n} = \frac{\boldsymbol{e}_i \times \boldsymbol{e}_{i+1}}{|\boldsymbol{e}_i \times \boldsymbol{e}_{i+1}|}, \tag{2.1}$$

where $\boldsymbol{e}_i$ are edges of the triangle, $i \in \{1, 2, 3\}$.

A problem is with normal vectors at the vertices. The simplest solution to obtain a single vertex normal $\boldsymbol{n}_{V_j}$ at a vertex V_j is by averaging normal vectors $\boldsymbol{n}_k$ from neighboring triangles

$$\boldsymbol{n}_{V_j} = \frac{1}{N} \sum_{k=1}^{N} \boldsymbol{n}_k, \tag{2.2}$$

where N is a number of triangles sharing the vertex P_j (a triangle fan with a center at P_j).

This approach is fast, but the resulting quality can be often low, because all adjacent triangles have the same weight regardless their area. This can be sufficient for the purposes of rendering where accuracy problems can be hidden (e.g. due to the movement of the camera). However, for further computations, normal vectors with a higher quality are preferred. Today, Max [Max99] is considered a basic algorithm used in various applications. The algorithm computes the vertex normal $\boldsymbol{n}_{V_j}$ as

$$\boldsymbol{n}_{V_j} = \sum_{k=1}^{N} \frac{\boldsymbol{e}_{k_i} \times \boldsymbol{e}_{k_{i+1}}}{|\boldsymbol{e}_{k_i}|^2 |\boldsymbol{e}_{k_{i+1}}|^2}, \tag{2.3}$$

where $\boldsymbol{e}_{k_i}$ is the i-th edge vector of k-th triangle and N is a number of triangles sharing the vertex P_j. This solution offers a good trade-off between quality and performance. It takes account of triangle areas, while the computation is quite simple.

Other algorithms have been presented by various authors. Main difference between the algorithms is usually in using different weighting scheme of triangle normals during summation. The weight can be the area of the triangle, the angle between triangles etc. The comparison of various approaches has been done by Jin et al. [Jin+05].

A similar problem as for a triangle mesh is with normal vectors for point clouds. However, the estimation is more challenging task, since there is no connectivity information and we are not aware of the surface itself. Using only the neighborhood of a point is not enough, since the points from the opposite sides of the surface can be presented in the neighborhood. A possible solution is to first create the triangulation of the point cloud and estimate the normals from the triangle mesh. However, this solution is not ideal, since it increases computational cost and shifts the problem with the neighborhood to the triangulation part of the algorithm.

A basic solution is to use a local surface fitting and estimate the normal vectors as derivatives of the surface in desired points. Overview and comparison of some other, more robust, techniques can be found in [Kla+09].

2.2 Tangent space

A tangent space (sometimes also called local space) is a coordinate system defined by a triangle normal vector $\boldsymbol{n}$, tangent $\boldsymbol{T}_u$ and bitangent $\boldsymbol{T}_v$. The origin can be any point in the space, but one of the triangle vertices is usually used. The main idea is to express every triangle in its own coordinate space based on the tangent plane of the triangle. It reduces the dimensionality from 3D to 2D, because the tangent space triangle is located in a tangent plane. The transformed triangle is labeled with vertices V_{L1}, V_{L2}, V_{L3} and normals $\boldsymbol{n}_{L1}, \boldsymbol{n}_{L2}, \boldsymbol{n}_{L3}$ (see Figure 2.2).

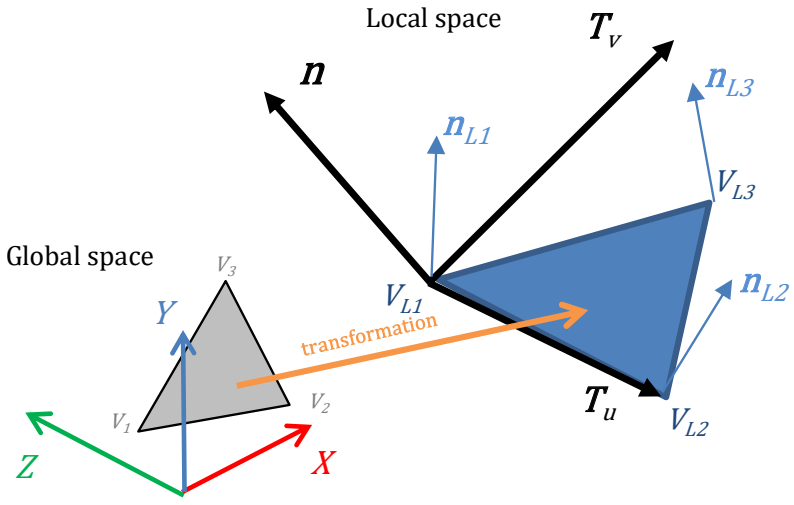


Figure 2.2: Transformation of triangle from Global (world) space to Tangent (local) space.

To obtain the tangent space, we need a normal vector $\boldsymbol{n} = (n_x, n_y, n_z)$, tangent $\boldsymbol{T}_u = (T_{ux}, T_{uy}, T_{uz})$ and bitangent $\boldsymbol{T}_v = (T_{vx}, T_{vy}, T_{vz})$ at each vertex of the triangle. These vectors are obtained using Equations (2.4):

$$\begin{aligned} \boldsymbol{T}_u &= \tfrac{V_2 - V_1}{|V_2 - V_1|}, \\ \boldsymbol{n} &= \tfrac{\boldsymbol{T}_u \times (V_3 - V_1)}{|\boldsymbol{T}_u \times (V_3 - V_1)|}, \\ \boldsymbol{T}_v &= \tfrac{\boldsymbol{T}_u \times \boldsymbol{n}}{|\boldsymbol{T}_u \times \boldsymbol{n}|}. \end{aligned} \tag{2.4}$$

Tangent and bitangent are both orthogonal to the normal vector. These three vectors create a transformation matrix τ (see Equation (2.5)) to map every point from a global space to a local space. Note that this matrix is different for every triangle.

$$\tau = \begin{bmatrix} T_{ux} & T_{uy} & T_{uz} \\ T_{vx} & T_{vy} & T_{vz} \\ n_x & n_y & n_z \end{bmatrix} \tag{2.5}$$

The original triangle is expressed in the tangent space, resulting in vertices V_{L1}, V_{L2}, V_{L3} and normals $\boldsymbol{n}_{L1}, \boldsymbol{n}_{L2}, \boldsymbol{n}_{L3}$. For example, a conversion of V_2 from the global to the local system is calculated as:

$$V_{L2} = \tau(V_2 - V_1) \ . \tag{2.6}$$

Normal vectors should be converted using the inverse transposed matrix of τ. Due to the orthonormality of the system we do not need to compute this, since inversion of the orthonormal matrix is equal to the matrix transposition.

Tangent space is often used for lighting effects. For example, normal mapping and its variants are calculated using this space (see [AMHH08]). A lot of curvature algorithms ([Rus04; Gri+12; BW07] etc.) exploit directly or indirectly this space as well.

2.3 Registration

Registration is a process of alignment of two or more overlapping parts of geometry. The entire process of registration consists of several steps and the basic pipeline is visualized in Figure 2.3.

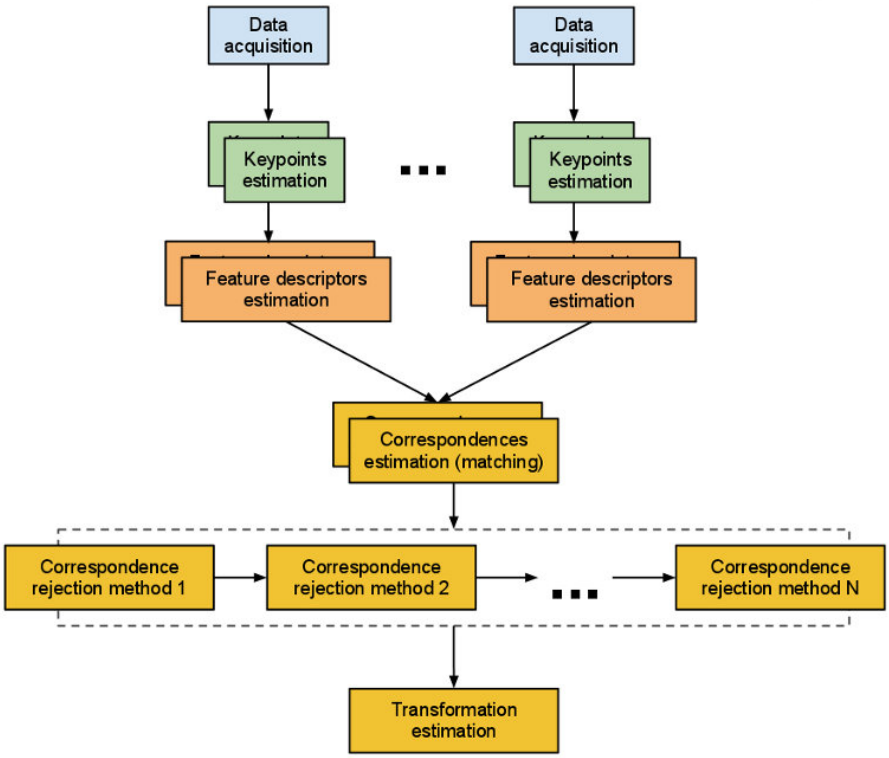


Figure 2.3: Registration pipeline from [RC11]

Based on Figure 2.3, the description of the main steps is as follows. The source of data (data acquisition) is usually a scanning device outputting a point cloud or an already preprocessed triangle mesh.

Once the data are obtained, keypoints and feature descriptors are calculated from them. The keypoints are usually unambigous points with some specific property, such as a high curvature. The feature descriptors are created in keypoins and are used to find correspondences among the input data. The corresponding feature vectors are usually close to each other in the Euclidean space. However, a lot of false correspondences is found which must be eliminated using different rejection methods. Based on the used algorithms, one or several of them are used.

From the resulting correspondences, a transformation is estimated. This transformation is sometimes further improved with ICP (Iterative Closest Point) algorithm [BM92]. This is a well-known method to solve the registration problem numerically. However, if this method is used without almost registered geometry, it often suffers from the local minima leading to an incorrect registration. On the other hand, if the method is used for almost registered geometry, it may further improve the already found transformation that is used to align input data.

Looking at Figure 2.3, the Chapters 9 and 10 are focused on the parts “Keypoint estimation” and “Feature descriptors estimation”. For the final alignment, we have used the existing solutions presented in state-of-the-art publications.

2.4 Other terms

- *Screen space* - during rendering, all triangles are converted from a 3D global (world) space to the 2D screen space by using world - view (camera) - projection matrix. Screen space is therefore the coordinate space of the resulting 2D projection. We can imagine this as a resulting rendered 2D image seen on our monitor. The screen space is often used for post-processing effects, see [Mel+13; Mit07; BSD08].
- *Local reference frame* (LRF) - it is a local, usually orthogonal, coordinate system. It is often used for creating shape descriptors [Guo+13; TSS10; Sha+13].

Chapter 3

Interpolations

We often require some kind of interpolation to create smooth surfaces from our input data, which can be point clouds or triangle meshes. In some cases, we can make use of normal vectors to improve interpolation results. If the normal vectors are not present in the input data, in some cases they can be estimated from the data.

The advantage of interpolation is the possibility to express value at an arbitrary point, which was not a part of the input data, directly. We can directly calculate normal vectors, curvatures, intersections etc. However, based on used functions, the complexity of solution can vary. In some cases (e.g. noisy data, incorrect normal vectors), an approximation may be more suitable than an interpolation. However, the approximation is more complicated to set up correctly. In most of our research we use mainly interpolation techniques.

There are many techniques for this task and we describe only those which are further used in our research,

3.1 Bézier surfaces

Bézier surfaces (or patches) are a type of mathematical splines. They are given as the Cartesian product of the blending functions of two orthogonal Bézier curves. A general Bézier patch is defined as:

$$F(u,v) = \sum_{i=0}^{N}\sum_{j=0}^{N} P_{ij} B_i^N(u) B_j^N(v);\ u,v \in \langle 0,1\rangle\,, \tag{3.1}$$

$$B_i^N(u) = \frac{N!}{i!(N-i)!} u^i (1-u)^{N-i}, \tag{3.2}$$

where N is the patch degree and P_{ij} are patch control points. The function $B_i^N(u)$ is called the Bernstein polynomial.

Bézier patches can also been computed for triangles. In this case, we often use the cubic form of the patch ($N = 3$) and every triangle in the mesh can be replaced with a Bézier patch defined as:

$$F(u,v,w) = \sum_{i+j+k=3} P_{ijk} B_{ijk}^3(u,v,w), \tag{3.3}$$

$$B^3_{ijk}(u,v,w) = \frac{3!}{i!j!k!}u^i v^j w^k,\ 0 \leq u, v,\ w \leq 1, \tag{3.4}$$

$$u + v + w = 1;\ i + j + k = 3, \tag{3.5}$$

where the patch degree is 3 and P_{ijk} are control points. Every triangle is defined by three corner points (P_{003}, P_{030}, P_{300}) and normal vectors in them. Triangle points and normals are used to obtain remaining control points P_{ijk} (see Figure 3.1) for the patch.

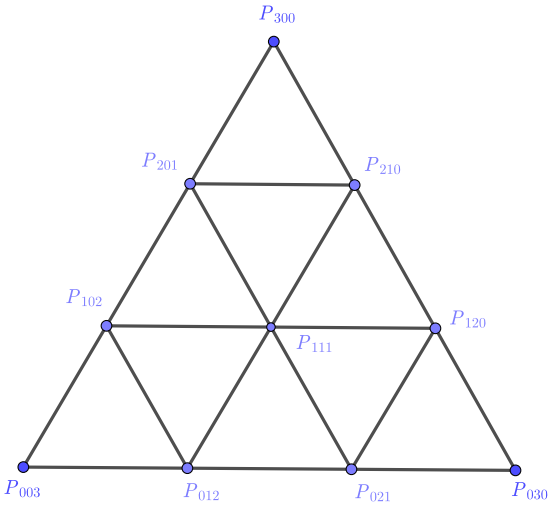


Figure 3.1: Control points of Bézier patch for a single triangle (top view)

The control points P_{ijk} need to be selected. Vlachos et al. in [Vla+01] computed missing control points on edges using a projection. Each edge of the triangle is divided into three equally long parts which leads to control points ($P_{012}, P_{021}, P_{120}, P_{102}$...). They are further projected on the plane created by the nearest vertex and its normal vector. Figure 3.2 shows this for one edge of the triangle. Each of the two endpoints (P_{003}, P_{030}) has its own normal ($\boldsymbol{n}_{003}, \boldsymbol{n}_{030}$) (green). The combination of a point and a normal determines a plane. We take the two control points (P_{012}, P_{021}) and project them to the plane of the nearest vertex (see green arrows at Figure 3.2). The center control point P_{111} is calculated from all other control points as defined in [Vla+01].

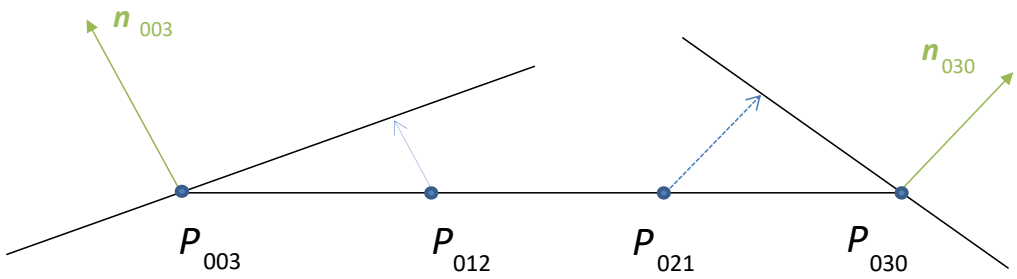


Figure 3.2: Calculation of control points on the edge of the triangle (side view)

3.1.1 Blended Bézier surfaces with G^1 continuity

The problem with the classic Bézier patch is its continuity. There is no G^1 continuity between neighboring surfaces, which means that surfaces are not sharing a common tangent direction at the join points between them. Therefore, on the edges there could be a sharp turn leading to a steep change in the derivation. Fünfzig et al. in [Fün+08] proposed a solution called *PNG1* to overcome the problem. See Figure 3.3 for difference between *PNG1* and a classic Bézier patch. *PNG1* patch is created by blending standard Bézier patches from neighboring triangles, see Figure 3.4. The final patch (gray color) is created from a higher number of normal vectors then in the case of a classic Bézier surface.

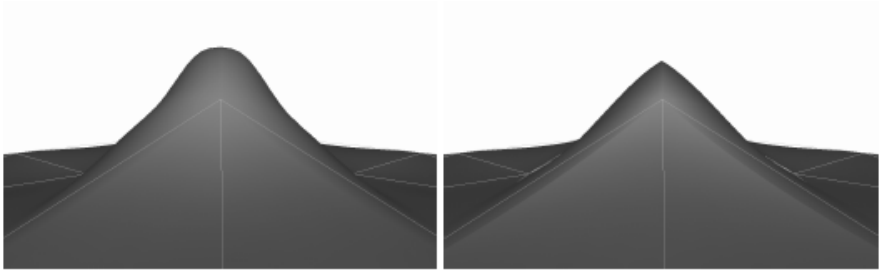


Figure 3.3: Comparison of the *PNG1* [Fün+08] (left) and Bézier patch (right). Sharp change of the surface can be observed for a classic Bézier patch.

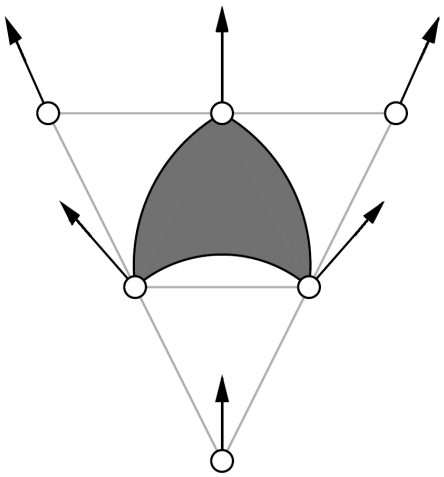


Figure 3.4: *PNG1* patch (gray) and neighboring triangles used for blending in [Fün+08].

The newly created patch is not a Bézier one. It was only created as a blend from several Bézier patches and the result does not meet conditions for a general Bézier patch and its definition. However, the description of this blended *PNG1* patch is analytical.

3.2 Radial basis functions

Radial basis functions (RBF) interpolation can be computed for any D-dimensional scattered point cloud of M points P_i. For the surface reconstruction from scattered data, it produces an implicit function representing the surface. In comparison with other surface fitting algorithms, RBF is more robust and more universal solution.

We want the resulting interpolant to be equal to zero everywhere the surface lies. For off-surface points, there are two choices of their location with respect to the model. For inside points we usually consider function values of interpolant to be negative and for outside points positive.

The quality of the result can be considerably affected by the selection of the appropriate basis function. Some basis functions have an additional shape parameter ε (in some literature also known as the scale parameter) that affects the shape of the function. Commonly used types of radial basis functions are summarized in Table 3.1.

Gaussian	$exp(-(\varepsilon r)^2)$
Multiquadric	$\sqrt{1+(\varepsilon r)^2}$
Inverse quadratic	$\frac{1}{1+(\varepsilon r)^2}$
Inverse multiquadric	$\frac{1}{\sqrt{1+(\varepsilon r)^2}}$
Polyharmonic spline	r^k, where $k = 1, 3, 5...$
Polyharmonic spline	$r^k ln(r)$, where $k = 2, 4, 6...$
Wendland functions	$(1-r)^3(3r+1)$ (for more, see [Wen95])

Table 3.1: Commonly used types of radial basis functions

Using basis functions with the additional shape parameter ε is problematic. Small values of ε can lead to a better interpolation, but also decrease numerical stability. Finding the optimal parameter is a non-trivial task. Polyharmonic splines or Wendland functions are free of an additional parameter and are usually preferred. From polyharmonic splines, a lower order has a higher numerical robustness.

To reconstruct a surface and create its functional representation, there are two main approaches - a standard RBF and a Hermite variant.

3.2.1 Standard RBF interpolation

The standard RBF interpolation for D-dimensional data can be used in the format

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} \alpha_i \phi(r_i), \tag{3.6}$$

where M is the number of input points P_i, $\boldsymbol{x} \in \mathbb{R}^D$, $r_i = ||\boldsymbol{x} - P_i||$, $\phi(r_i)$ is a basis function (see Table 3.1) and $\alpha_i \in \mathbb{R}$ are the weights.

To solve the interpolation, we need to find weights α_i. As an example for $D = 2$, this can be achieved by solving the system in a form:

$$\begin{bmatrix} \phi_{1,1} & \cdots & \phi_{1,M} \\ \vdots & \ddots & \vdots \\ \phi_{M,1} & \cdots & \phi_{M,M} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_M \end{bmatrix} = \begin{bmatrix} f(P_1) \\ \vdots \\ f(P_M) \end{bmatrix}, \tag{3.7}$$

where $\phi_{i,j}$ are the values of basis function obtained from all combinations of input points P_i (eg.: $P_0 - P_0$, $P_0 - P_1$... $P_M - P_M$). On the diagonal of the matrix ϕ, the values of basis function are computed from distances between the same points, which leads to $r_i = 0$.

To find the weights for our representation of the surface, we have a system in the form

$$f(P_i) = 0, \forall i \tag{3.8}$$

for all on-surface points. However, this leads to a trivial solution $\alpha_i = 0, \forall i$. To obtain a non-trivial solution, normal vectors $\boldsymbol{n}_i$ scaled by a constant c at the points P_i are used to offset the surface in the positive ($f(P_i + c\boldsymbol{n}) = 1$) and/or negative ($f(P_i - c\boldsymbol{n}) = -1$) direction. However, not all points have to be offseted. A few sample points (P_0... P_8) are presented in Figure 3.5. Original surface points are green, while the new offset points are blue.

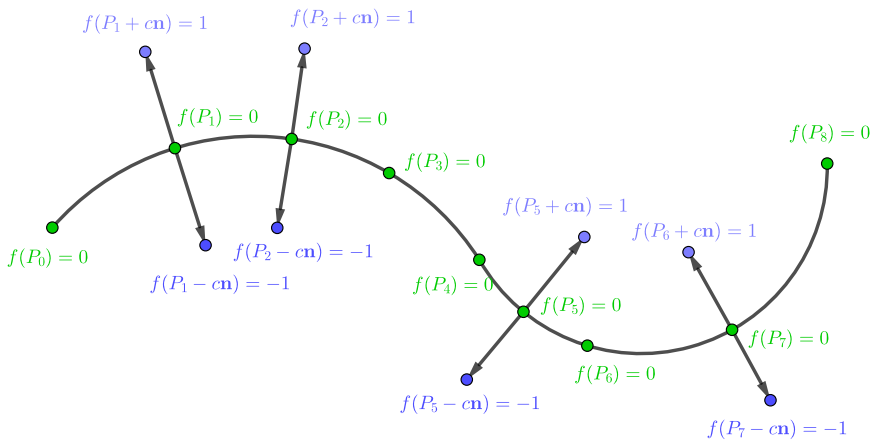


Figure 3.5: Adding off-surface points to obtain a non-trivial solution for RBF surface interpolation

With the newly added off-surface points, a non-trivial solution can be found for Equation 3.6. This approach was described in [Car+01]. However, using this solution can be problematic because of the difficulty of choosing the c parameter, since it has to be set in a way that avoids self-intersections.

3.2.2 Hermite RBF interpolation

A more stable approach without the need for off-surface points uses a Hermite version of the RBF called Hermite RBF interpolation (HRBF) (see [MGV09] and [MGV11]). This solution is applicable only if the input data are Hermite, i.e. equipped with a normal vector at each point. The improved interpolation scheme for D-dimensional data is defined as:

$$f(\boldsymbol{x}) = \sum_{i=1}^{M} \alpha_i \phi(r_i) + \boldsymbol{\beta}_i \cdot \boldsymbol{\nabla}[\phi(r_i)]. \tag{3.9}$$

where M is the number of points, $\boldsymbol{x} \in \mathbb{R}^D$, $r_i = ||\boldsymbol{x} - P_i||$, $\phi(r_i)$ is a basis function (see Table 3.1), $\alpha_i \in \mathbb{R}$ are weights for points and $\boldsymbol{\beta_i} \in \mathbb{R}^D$ are vectors of the weights for normal vectors. Weights need to be solved using a solution similar to the one used for a classic RBF, but in this case, no trivial solution is found.

Let the gradient $\boldsymbol{\nabla}$ be defined as

$$\boldsymbol{\nabla} = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_3}, ..., \frac{\partial}{\partial x_N} \right). \tag{3.10}$$

Every on-surface point $\boldsymbol{x}$ meets these two criteria:

- $f(\boldsymbol{x}) = 0$ (this is the same as in the standard RBF interpolation)
- $\boldsymbol{\nabla} f(\boldsymbol{x}) = \boldsymbol{n}$ (an additional condition for a normal vector)

The system to be solved for finding weights is therefore in the form of a block matrix:

$$\begin{bmatrix} f(P_i) \\ \boldsymbol{\nabla} f(P_i) \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{n}_i \end{bmatrix}, \forall i \tag{3.11}$$

where the control points P_i, located on the surface, and their respective normals $\boldsymbol{n}_i$, are used to create the interpolation. Each block has the dimension $(d+1) \times (d+1)$, where 1 corresponds to scalar part and d corresponds to a gradient part. A full and clear derivation of final equations can be found in [Vai13].

Chapter 4

Curvature

Curvature itself plays an important role in computer graphics. We can use curvature as our "black-box" in shape characteristic, since it is a local characteristic and describes how bent a curve is at a particular point on the curve. In other words, it tells us how much the curve deviates from a straight line at this point.

4.1 2D space

If the curve is defined parametrically in Cartesian coordinates as $x = x(t)$ and $y = y(t)$, the curvature κ at the point P (with a normal vector n and a tangent vector $\boldsymbol{T}$) is computed as:

$$\kappa = \frac{d\omega}{dS}, \tag{4.1}$$

where $d\omega$ is the rate of change of the tangential angle with respect to the arc length dS.

The infinitesimal neighborhood of P can be replaced by the osculating circle. It is a circle that approximates the curve in a neighborhood of a point P. It is defined as the circle with a radius r passing through P and a pair of additional points on the curve infinitesimally close to P. The center of the circle is located on a half-line passing through P in the direction of the normal vector at the point P. The osculating circle has also a tangent vector $\boldsymbol{T}$ equal to the tangent vector at the point P. To better understand Equation (4.1) and the osculating circle, see Figure 4.1 and [Rob01].

From Equation 4.1 we can obtain ([Kre91]) the equation

$$\kappa = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{\frac{3}{2}}}. \tag{4.2}$$

If the curve is defined explicitly in a form $y = f(x)$, Equation (4.2) can be rewritten as

$$\kappa = \frac{\frac{d^2y}{dx^2}}{\left(1 + \left(\frac{dy}{dx}\right)^2\right)^{\frac{3}{2}}}. \tag{4.3}$$

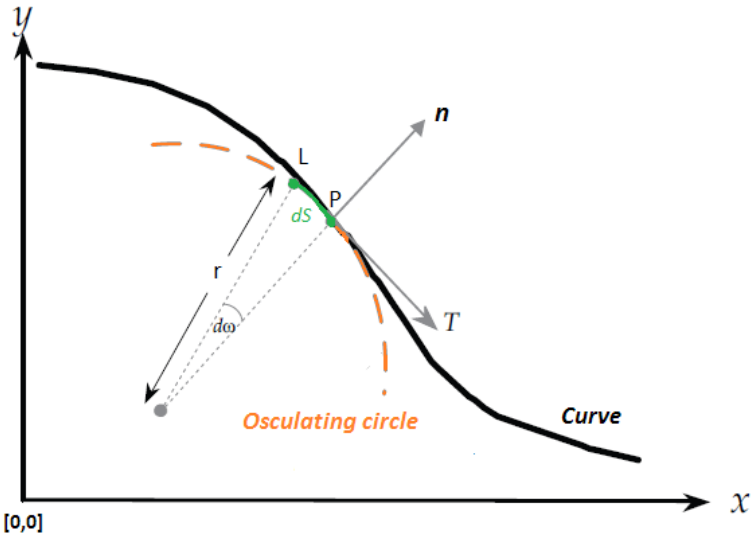
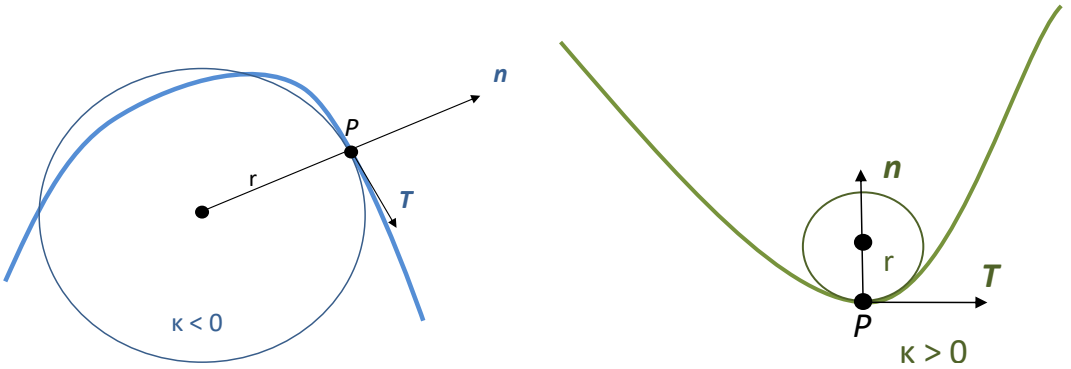


Figure 4.1: Definition of curvature using tangential angle $d\omega$ and arc length dS. [Rob01]

For more detailed explanations and derivations of Equations (4.1) - (4.3), see [Kre91].

From a geometrical point of view, curvature in a point P is defined by an osculating circle. The sign of curvature is defined by the curve parametrization. See two osculating circles, curvature sign and a curve in Figure 4.2.



(a) Negative curvature (b) Positive curvature

Figure 4.2: Plane curve with two osculating circle at points P and curvature sign given by curve parametrization

The osculating circle radius r, which equals to the distance of the circle center from the point P, is called the radius of curvature. The curvature κ of a curve at point P is derived from Equation (4.1) using polar coordinates (see [Kre91]). The final equation is defined as

$$\kappa = \frac{1}{r}. \tag{4.4}$$

The smaller the radius of curvature r is, the more bent the curve is and, therefore, the larger curvature we have. The limiting case is a straight line. The osculating circle

that describes it would have an infinite radius and the curvature $\kappa = \frac{1}{\infty}$ converges to zero.

The sign of the curvature depends on the orientation of the normal vector in the point P (see Figure 4.2).

4.2 3D space

So far we have been talking about curvature in a 2D space. In 3D space, however, we have to distinguish between curves and surfaces. The curvature for 3D space curves can be calculated similarly to 2D curves. However, since the 3D curves are not used in our research, we are not going further in their description.

Solution for surfaces is more complicated. The curvature itself is still related to the curves and is therefore not calculated for the surface directly. The curvature is calculated for a curve in a particular plane defined by a slice of the surface. There are different types of curvature for surfaces - normal (k_n), mean (K_H), Gaussian (K_G), dip (k_d), strike (k_s) etc. We will cover the first three named curvatures in more detail, the other can be seen, e.g., in [Rob01].

4.2.1 Normal curvature

If we cut the surface at a point P with a normal plane (plane containing the normal vector $\boldsymbol{n}$), we have a 2D slice (see Figure 4.3). There is a 2D curve within this slice and we are looking for its curvature at a point P.

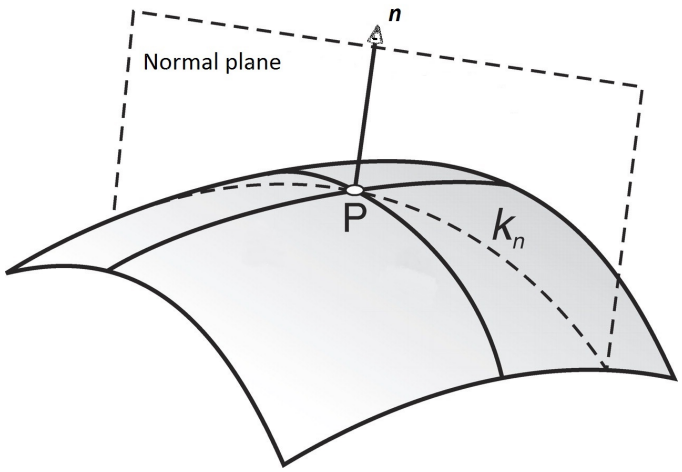


Figure 4.3: Normal curvature k_n within a normal place

The problem is that there is an infinite number of normal planes at a point P, because they can have various rotations around the normal vector $\boldsymbol{n}$. From all of these possible rotations, two of them lead to the maximal (K_{max}, K_1) and minimal (K_{min},

K_2) curvature. These two curvatures are known as principal curvatures. The normal curvature is connected with principal curvatures by the following formula

$$k_n = K_1 cos^2\alpha + K_2 sin^2\alpha, \tag{4.5}$$

where α is the angle between the plane of K_1 and the plane for k_n (see Figure 4.4).

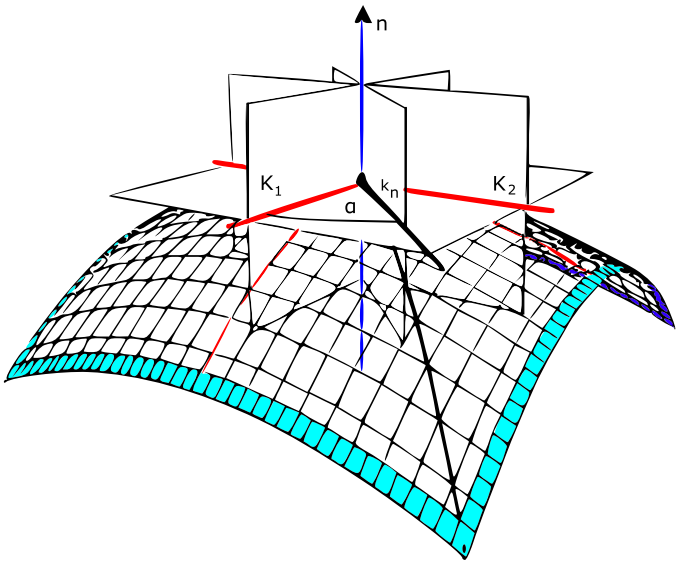


Figure 4.4: Angle between planes K_1 and k_n. [Unk17]

The principal curvatures are the most important for research. They can be used to identify the type of the surface. Based on the sign, we can subdivide surfaces into different categories - see Table 4.1.

	$K_1 < 0$	$K_1 = 0$	$K_1 > 0$
$K_2 < 0$	concave ellipsoid	concave cylinder	hyperboloid surface
$K_2 = 0$	concave cylinder	plane	convex cylinder
$K_2 > 0$	hyperboloid surface	convex cylinder	convex ellipsoid

Table 4.1: Surface Shape Classes as defined in [Fis89]

For every plane orientation, we also have its tangent vector. For principal curvatures, these two tangent vectors are called principal directions ($\boldsymbol{K}_{1,2}$). They are always orthogonal to each other. However, their direction is ambiguous, since they can differ in sign and still represent the correct principal direction.

4.2.2 Mean and Gaussian curvature

We can derive other curvatures with a different meaning from principal curvatures. The best known ones are the mean (K_H) and Gaussian (K_G) curvature.

Mean curvature is defined as an average of any two orthogonal normal curvatures as

$$K_H = \frac{K_1 + K_2}{2}. \tag{4.6}$$

Gaussian curvature is defined as

$$K_G = K_1 K_2. \tag{4.7}$$

The mean and Gaussian curvature can be used to obtain principal curvatures from equation

$$K_{1,2} = K_H \pm \sqrt{K_H^2 - K_G}. \tag{4.8}$$

4.3 Curvature computation

The curvature computation can be divided into two main approaches - direct and discretized computations. The direct computations give us exact results but can be used only if we have function description of the surface. Discretized computations, on the other hand, are useful if we have only discrete geometry. Both solutions are described in the following subsections.

4.3.1 Implicit surface

The direct computation of curvature for implicit surfaces is a straightforward solution. We can use the approach from [Gol05] directly. To compute the principal curvatures ($K_{1,2}$) at a certain point P, its gradient $\boldsymbol{\nabla F}$, Hessian matrix H and adjoint of Hessian matrix $H^*$are used. Principal curvatures are not computed directly but from Gaussian (K_G) and mean (K_H) curvature. The required steps are as follows:

$$H^* = \begin{bmatrix} H_{11}H_{33} - H_{23}H_{32} & H_{23}H_{31} - H_{21}H_{33} & H_{21}H_{32} - H_{22}H_{31} \\ H_{13}H_{32} - H_{12}H_{33} & H_{11}H_{33} - H_{13}H_{31} & H_{12}H_{32} - H_{11}H_{32} \\ H_{12}H_{23} - H_{13}H_{22} & H_{21}H_{13} - H_{11}H_{23} & H_{11}H_{22} - H_{12}H_{21} \end{bmatrix}, \tag{4.9}$$

$$K_G = \frac{\boldsymbol{\nabla F}\, H^*\, \boldsymbol{\nabla F}^T}{|\boldsymbol{\nabla F}|^4}, \tag{4.10}$$

$$K_H = \frac{\boldsymbol{\nabla F}\, H\, \boldsymbol{\nabla F}^T - |\boldsymbol{\nabla F}|^2 trace(H)}{2|\boldsymbol{\nabla F}|^3}, \tag{4.11}$$

$$K_{1,2} = K_H \pm \sqrt{K_H^2 - K_G}, \tag{4.12}$$

where $trace(H)$ is a sum of elements on the main diagonal of the square matrix H. To see full derivations of the above equations, see [Gol05].

4.3.2 Explicit surface

For explicit surfaces, we have to use a different approach based on fundamental forms. The first fundamental form (I) is constructed from the first order derivatives at a surface point, which give us two tangent vectors $(\boldsymbol{T}_u, \boldsymbol{T}_v)$, see Figure 4.5.

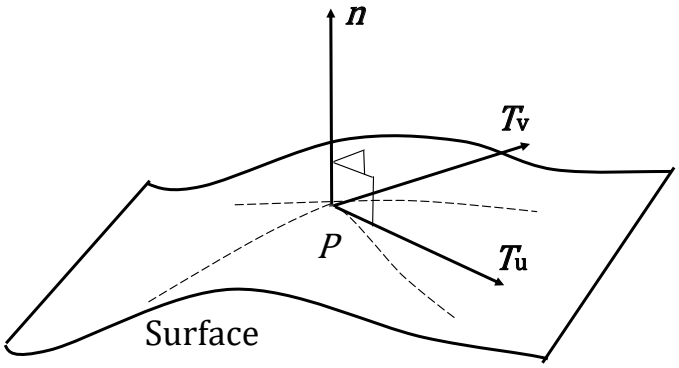


Figure 4.5: Tangent vectors of surface at point P

Vectors $\boldsymbol{T}_u, \boldsymbol{T}_v$ are in general not orthogonal. They are, however, orthogonal to the normal vector $\boldsymbol{n}$ to the surface at the given point. Elements of the matrix I are computed as

$$\mathrm{I}=\begin{bmatrix} E & F \\ F & G \end{bmatrix}, \qquad E = \boldsymbol{T}_u \cdot \boldsymbol{T}_u,\ F = \boldsymbol{T}_u \cdot \boldsymbol{T}_v,\ G = \boldsymbol{T}_v \cdot \boldsymbol{T}_v. \tag{4.13}$$

The second fundamental form (II) is calculated from the second partial derivatives $(\boldsymbol{T}_{uu}, \boldsymbol{T}_{vv}, \boldsymbol{T}_{uv})$ and the normal vector $(\boldsymbol{n})$. The elements of the matrix II are computed as

$$\mathrm{II} = \begin{bmatrix} L & M \\ M & N \end{bmatrix}, \qquad \boldsymbol{n} = \frac{\boldsymbol{T}_u \times \boldsymbol{T}_v}{|\boldsymbol{T}_u \times \boldsymbol{T}_v|}, \qquad L = \boldsymbol{T}_{uu} \cdot \boldsymbol{n},\ M = \boldsymbol{T}_{uv} \cdot \boldsymbol{n},\ N = \boldsymbol{T}_{vv} \cdot \boldsymbol{n}. \tag{4.14}$$

Combining the fundamental forms gives the shape operator W (also known as the Weingarten operator). For every point of the surface it tells us the change of the normalized normal vector in the direction of the tangent vector at this point. W is a 2×2 symmetric matrix that can be obtained from the first (I) and second (II) fundamental forms:

$$W = \mathrm{I}^{-1}\mathrm{II}. \tag{4.15}$$

The matrix W has two real eigenvalues that correspond to the first (K_1) and second (K_2) principal curvatures. The eigenvectors of the matrix W correspond to the directions of the principal curvature within the tangent plane.

4.3.3 Monge patch

For a regular height field, curvature can be calculated directly using the Monge Patch [Gra97]. This is a patch in a form

$$x(u, v) = (u, v, h(u, v)), \tag{4.16}$$

which is basically a 2.5D heightfield with a height defined as $h(u, v)$. For this, we need to obtain derivatives of the function $h(u, v)$. Final curvatures are calculated as follows:

$$K_H = \frac{h_{uu}h_{vv} - h_{uv}^2}{(1 + h_u^2 + h_v^2)^2}, \tag{4.17}$$

$$K_G = \frac{(1 + h_v^2)h_{uu} - 2h_u h_v h_{uv} + (1 + h_u^2)h_{vv}}{2(1 + h_u^2 + h_v^2)^{\frac{3}{2}}}, \tag{4.18}$$

where h_u, h_v, h_{uu}, h_{vv} and h_{uv} are derivatives of the function $h(u, v)$. Based on Equation 4.8, we can obtain principal curvatures.

Chapter 5

Existing methods for curvature computation

We often deal with a discretized representation of the geometry that can have various forms - triangles, volumetric data sets, height fields, point clouds etc. All of them cause a problem with curvature in vertices, since we are not able to compute exact values but only an estimation. Usually, with a more detailed discretization, this estimation offers better results, but the calculation itself is usually tied with a loss of performance.

Curvature estimation can be divided into two main categories of approaches - discrete and surface fitting. The discrete methods calculate curvature directly from the data, while the surface fitting construct a local approximation of the surface and then calculates the curvature of this approximation directly. Usually, discrete methods are faster but less accurate. There is also a "third" category that includes algorithms combining discrete and surface fitting approaches.

In this chapter, we introduce some of the existing methods for curvature estimation. A more detailed description is provided for the solutions relevant to our current and future research. We have directly tested these methods, either by using implementation from original research articles or by re-implementation from description of the algorithm.

5.1 Discrete methods

We start with a description of several discrete-based algorithms. These algorithms are more often used in a combination with triangulated geometry, since they utilize the known connectivity and the neighborhoods.

5.1.1 Monge patch

For a simple representation of 2.5D heightfield, we can use directly Monge patch (see subsection 4.3.3). Required derivatives can be estimated with a discrete finite difference.

5.1.2 Tangent space methods

The discrete method proposed by Rusinkiewicz [Rus04] uses the second fundamental form matrix II. Every triangle in the mesh is converted to the tangent space and has a unique II matrix. The elements of this matrix are unknown and need to be computed. For a single triangle $V_1V_2V_3$ with a normal vector $\boldsymbol{n}$ and edges $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ this leads to a system of equations:

$$\begin{aligned}
\mathrm{II} \begin{bmatrix} \boldsymbol{e}_1 \cdot \boldsymbol{T}_u \\ \boldsymbol{e}_1 \cdot \boldsymbol{T}_v \end{bmatrix} &= \begin{bmatrix} (\boldsymbol{n}_{V_3} - \boldsymbol{n}_{V_2}) \cdot \boldsymbol{T}_u \\ (\boldsymbol{n}_{V_3} - \boldsymbol{n}_{V_2}) \cdot \boldsymbol{T}_v \end{bmatrix}, \\
\mathrm{II} \begin{bmatrix} \boldsymbol{e}_2 \cdot \boldsymbol{T}_u \\ \boldsymbol{e}_2 \cdot \boldsymbol{T}_v \end{bmatrix} &= \begin{bmatrix} (\boldsymbol{n}_{V_1} - \boldsymbol{n}_{V_3}) \cdot \boldsymbol{T}_u \\ (\boldsymbol{n}_{V_1} - \boldsymbol{n}_{V_3}) \cdot \boldsymbol{T}_v \end{bmatrix}, \\
\mathrm{II} \begin{bmatrix} \boldsymbol{e}_3 \cdot \boldsymbol{T}_u \\ \boldsymbol{e}_3 \cdot \boldsymbol{T}_v \end{bmatrix} &= \begin{bmatrix} (\boldsymbol{n}_{V_2} - \boldsymbol{n}_{V_1}) \cdot \boldsymbol{T}_u \\ (\boldsymbol{n}_{V_2} - \boldsymbol{n}_{V_1}) \cdot \boldsymbol{T}_v \end{bmatrix}.
\end{aligned} \tag{5.1}$$

From the above presented Equations (5.1), a solution of 2×2 matrix II is found using the least squares method. Principal curvatures are calculated as eigenvalues of II. However, this gives us the curvature of the triangle face, while we are looking for curvature in vertices. To estimate curvatures in vertices, curvatures from all neighboring faces are used. Since every triangle was expressed in the tangent space, the fundamental forms must be unified by expressing the fundamental form in the tangent space of the vertices.

The final curvature is weighted in a way similar to Meyer et al. [Mey+03]. They use Voronoi areas to give a greater influence to curvatures from larger areas. This step is also sometimes used for normal vector calculation in triangle vertices, where a weighting of normals from adjacent triangle faces is used.

Rusinkiewicz's approach [Rus04] is quite popular because of its simplicity and quite accurate results. Many other authors use the same basic idea. Theisel et al. [The+04] calculate curvature for every triangle based on triangle vertices positions and unnormalized normals. These normal vectors are created from the cross product of triangle edges and because they are not normalized, they describe the area of the triangle. By a linear interpolation, a single point and a normal is calculated for each triangle. All these values are used for curvature estimation. The estimated curvature weight depends on the length and quality of the normals. Batagelo et al. [BW07] use the basic idea from Rusinkiewicz [Rus04] and transfer the solution to the GPU. They also improve numerical robustness.

5.1.3 Statistics-based methods

A statistics-based approach was presented by Kalogerakis et al. [Kal+07]. The algorithm is based on curvature tensor fitting using the finite normal differences in the way similar to Rusinkiewicz [Rus04]. For this, normal vectors are needed. However, the presented solution does not require normal at its input. If normal vectors are not present, they calculate their own using Max algorithm [Max99]. Input normals are not used directly. They are further re-estimated in several iterations. Kalogerakis et al. use this re-estimation approach to increase the robustness of the algorithm based on the assumption that input

normals are usually incorrect. Of course, if we have exact normals at the input, this would not be an improvement.

The algorithm computes several curvature estimations with a different neighborhood sizes. Based on statistic approach and iterative curvature re-estimation, several versions of curvatures are estimated. The final one is selected based on weights that are constructed with respect to boundaries and feature lines of geometry. This leads to a correct estimation on neighborhoods with feature boundaries. After initial weighting, curvature is incorrectly spread across the entire neighborhood due to the weights. After several steps, the final weights are obtained and curvature is cut off at the boundary edge, as can be seen in Figure 5.1.

Weighting is also used to suppress effects of the noise and to find the most smooth result. As a result, this method performs reasonably well for noisy data. This, on the other hand, can be a problem as well, since the algorithm can also smooth out the fine details that are not noise and we want to keep them in data.

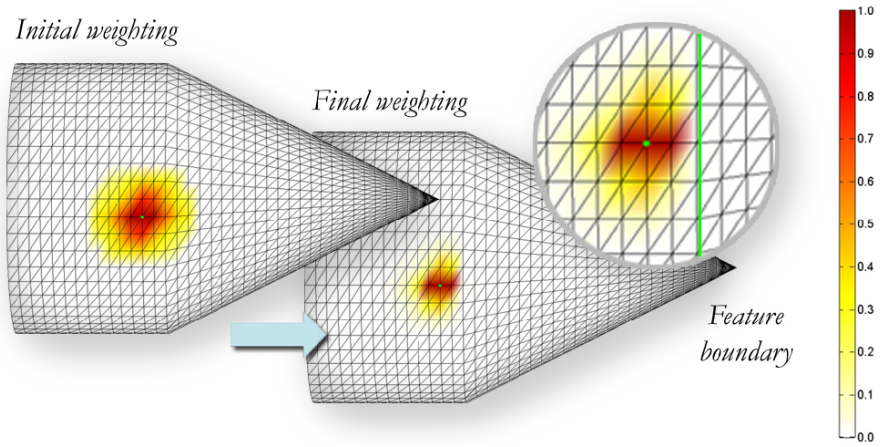


Figure 5.1: Re-estimation of curvature after initial estimation. Re-estimated curvature is correctly cut on the feature boundary (see detailed subimage). [Kal+07]

5.2 Surface fitting methods

The second large group of methods for curvature estimation are surface fitting methods. These algorithms try to find a surface that is fitted to the neighborhood of a point of interest. The final curvature is directly computed from the functional representation of the fitted surface. Due to the nature of these methods, they are often used not only for triangle meshes but also for point clouds and volumetric models.

5.2.1 Polynomial fitting

A common class of methods, used in many geometric modeling applications, is based on polynomial fitting. Fitting polynomials to sample points of a smooth surface yields an

approximation of the curvature at a point of the smooth surface. For faster computations, lower order polynomials are used, usually quadratic or cubic surface approximations.

Taubin [Tau95] presented this as one of the first in 1995. A well-known method is from Goldfeather and Interrante [GI04]. They use a cubic surface approximation. However, the third-order fit of the surface greatly increases both time and space required for computation. Their fitting scheme is not done only by interpolating through points but normal vectors from 1-ring neighborhood of vertex are used as well. The problem is with the neighborhood that has many vertices or an oscillating shape. In this case, the approximation is not accurate and the resulting error can be quite high.

A Bézier patch representation (see Section 3.1) is quite popular due to its simplicity. Several authors used them for curvature estimation, see the following subsections.

Biquadratic patch

Bézier patch of degree two ($N = 2$) is called biquadratic. This solution was used by Razdan et al. in [RB05]. To construct the patch, they use points from the k-ring neighborhood of a vertex (authors recommend to use $k = 2$). The standard least squares method is used to obtain the set of control points from the neighborhood. Sometimes, smoothing can be added if the original mesh contains noise. This is done by adding weights to the control points. The final curvature is computed at a single vertex directly from the patch itself.

The computational cost is very low, but if the selected neighborhood occupies a small area, the results can be incorrect.

Bézier patches as triangle replacements

Based on the subdivision scheme by Vlachos et al. ([Vla+01]), a curvature estimation solution was proposed by Zhihong et al. in [Zhi+11]. Derivatives needed for curvature estimation are directly calculated from Bézier patch using Equation (3.3). The final curvature is computed as an average value from center vertices from adjacent triangles. The average can be simply an arithmetic mean or a weighted average using Voronoi area as described in [Mey+03].

PNG1 surfaces

As mentioned in Subsection 3.1.1, the problem with a classic Bézier patch is its continuity. The curvature is not directly defined on edges. In practice, curvature is calculated as an average from triangles that share the edge. However, there could be a sharp turn leading to a steep change in the curvature. Solution from Fünfzig et al. [Fün+08] can be altered and used for curvature estimation. Fünfzig et al. in [Bos+12] proposed that the curvature can be directly calculated using an analytic solution based on PNG1 surface description. A drawback of this method is that the second derivatives are much more complex than for a simple Bézier patch and therefore takes longer time to compute.

5.2.2 Point clouds

Algorithms for point clouds can be used also for triangulated geometry. We just simply omit triangles and use only their vertices as a point cloud representation. A curvature estimation algorithm for point clouds was presented by Yang et al. in [YQ07]. This method approximates the surface by the least squares technique. Normal vectors are required for input points. If they are not present in the input datasets, they can be calculated. This can be done using a statistical analysis of the neighboring samples leading to a covariance matrix. Normal vectors are obtained from eigenvectors of this matrix.

The final curvature is calculated directly from approximated functions that describe the point set in a local neighborhood of the selected point. In this case, mean and Gaussian curvature are calculated and from them, principal curvatures can be derived.

5.3 Other methods

The last section is used for algorithms that cannot be directly assigned to previously mentioned groups. They use combinations of approaches and often take knowledge of both worlds - discrete and surface fitting.

5.3.1 Tensor-based method

Curvature can be computed from the eigenvalues of a tensor average over a small area of the polygonal mesh as done by Cohen-Steiner and Morvan in [CSM03]. This algorithm was also used for remeshing in [All+03]. A triangle mesh is a piecewise-linear surface and curvature tensors cannot be expressed directly, they are estimated at vertices of each triangle. To obtain a continuous tensor field, tensors from triangle vertices are linearly interpolated over triangles. However, to define tensors directly at the vertices is not very natural. A better way is to define tensors on the edges of triangle. Each edge $\boldsymbol{e}$ contains an infinite number of tensors, leading to a computation of an integral. To simplify this, a discretization is used and the integral is expressed via summing over an arbitrary region R surrounding given vertex. The regions are usually discs with a radius that can be selected by the user. A simple equation is used to create a 3×3 matrix:

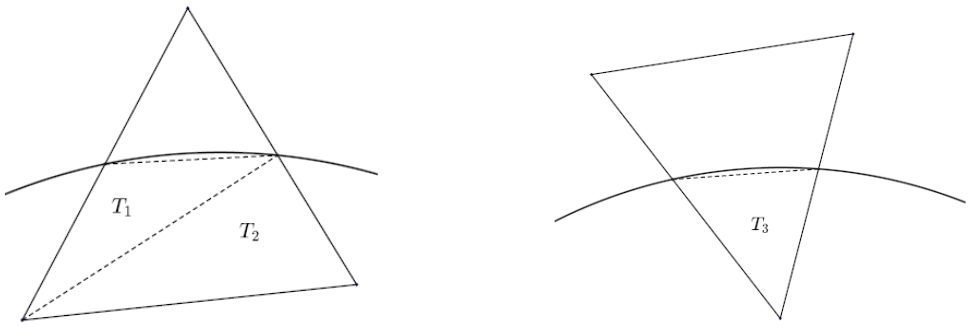
$$S = \frac{1}{|R|} \sum_{\forall \boldsymbol{e}} \beta(\boldsymbol{e}) \, |e \cap R| \, \bar{\boldsymbol{e}} \, \bar{\boldsymbol{e}}^T, \tag{5.2}$$

where $|R|$ is a surface area of the region R, $\beta(\boldsymbol{e})$ is an angle between the normals of two oriented triangles incident to the edge $\boldsymbol{e}$ and $\bar{\boldsymbol{e}}$ is the unit vector in the direction of $\boldsymbol{e}$. From the matrix S, three eigenvectors and their corresponding eigenvalues are calculated. The eigenvectors associated with the minimal-magnitude eigenvalue is a normal vector, the remaining two eigenvalues and eigenvectors represent principal curvatures $K_{1,2}$ and their swapped principal directions $\boldsymbol{K}_{1,2}$.

The quality of the resulting curvature depends on the size of the selected area. If the size of the area is too large, fine details in curvature are lost. If the neighborhood is too small, it can, on the other hand, lead to a noisy result with incorrect curvatures.

5.3.2 Generalized shape operator

An approach based on a generalized version of the shape operator (see Equation (4.15)) has been proposed by Hildebrandt and Pohltier in [HP11]. The algorithm uses triangles within neighborhood of a vertex. The neighborhood is not a k-ring, but a disc with a certain radius is used (Euclidean neighborhood). If the triangle of the mesh is fully in the neighborhood, the whole face is used. If only a part is in the neighborhood, the triangle is split and only relevant parts are used. The splitting scheme can create two triangles T_1 and T_2 (Figure 5.2a) or only one triangle T_3 (Figure 5.2b).



(a) Two triangles (b) One triangle

Figure 5.2: Two possible splitting schemes of triangle with neighborhood intersection

Once the neighborhood of the vertex is created, a surface integral is computed. Since triangles are used, the integral is discretized and the areas of faces are summed together resulting in a generalized version of the shape operator (expressed using a 3×3 matrix while the classic shape operator is described by a 2×2 matrix). The three eigenvalues of generalized shape operator matrix represent two principal curvatures and negative mean curvature. The mathematics behind computation is quite complex and out of the scope of this chapter. The reader can find more in-depth details with proofs in the original research publication [HP11].

5.3.3 Integral invariants

An integral-invariant-based method was presented by Pottmann et al. in [Pot+07] and [Pot+09]. The basic idea of this algorithm is to estimate curvature on a voxelized version of the triangle mesh. Therefore, this algorithm can also be used for volumetric data (see an example with mean curvature in Figure 5.3). The modification of this algorithm by Levallois et al. is a part of DGtal library [Lev15].

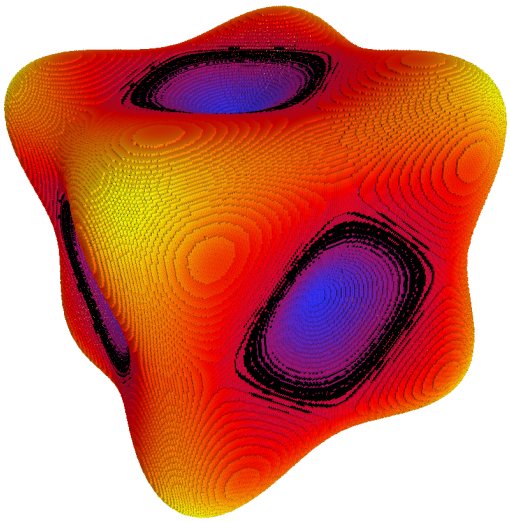


Figure 5.3: Mean curvature calculated on volume data. [Lev15]

A sphere is circumscribed around each point and an integral of the delimited area is computed. This is done using a voxelization (rasterization), where the integral is discretized. A 2D example can be seen in Figure 5.4. The rasterized part of the sphere $B_r(P)$ with a radius r and a center P represents the target volume $V_r(P)$ that is used to estimate curvature. The finer the rasterization is, the more detailed curvature we can get.

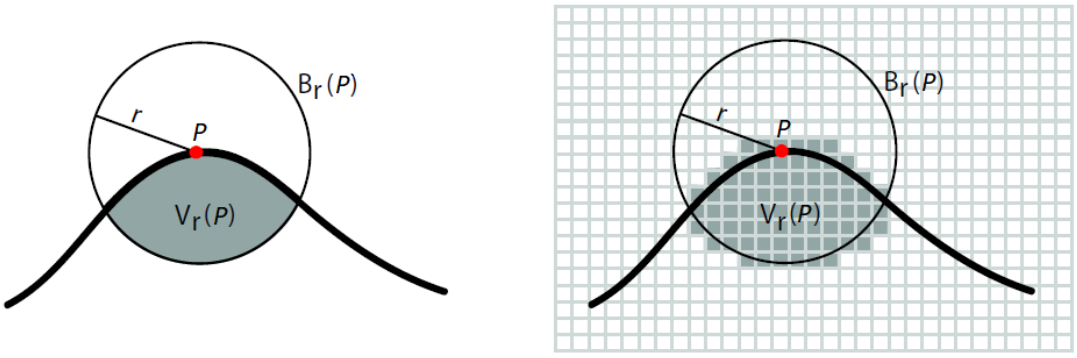


Figure 5.4: Example of rasterization between sphere and curve intersection in 2D. [Gel+05]

This process is, however, quite slow and memory-heavy. To speed up the process and save memory, an octree is constructed. It has the highest precision near the faces of the triangles and inside the model, a lower precision is used.

5.4 Dynamic curvature estimation

The so far presented solutions were primarily designed for the static geometry. They can be also used for the dynamic geometry (deformations, animations etc.), however, it will lead to a curvature re-estimation for the entire model after every change (e.g. a frame of an animation). If we have a high-detailed model representation, it can cause a non-interactive or slow response of a modeling software if the geometry is changed. Of course, in some cases the recomputations can be limited to a certain local part of a model and then the performance of classic solutions may be sufficient. However, we cannot rely on this and it is better to have an algorithm designed directly for the dynamic geometry.

The simplest solution to this problem is to run the computation of curvature estimation in parallel. This approach has been presented by Griffin et al. [Gri+12]. They have created a parallel version of the Rusinkiewicz algorithm [Rus04]. The paralellization is done directly on the GPU. The algorithm uses vertex neighborhood and this information must be available for each vertex. In every frame, normal vectors and Voronoi areas are recomputed from 1-ring neighborhood of the vertex. Results are stored for every vertex in a single texture. The curvature is estimated the same way as Rusinkiewicz's solution described in Section 5.1.2. In the final step of the original Rusinkiewicz algorithm, contributions from the vertex neighborhood are weighted and summed for the vertex. In this step, there is a need for a synchronization of threads, since the summation is over neighboring vertices and each vertex is computed in its own thread. Synchronization slows down computations, however, the main speed-up of the algorithm is in the curvature solving for a single vertex via the least square method and computing transformations from the object to the tangent space and vice versa.

Another approach designed directly for the dynamic geometry has been presented by Kalogerakis et. al. [Kal+09]. They used this algorithm for line drawing based on curvature.

The curvature estimation is based on a mapping function between a shape representation and the curvature with other attributes (value and directions). The shape is represented by a state vector, whose values can be joint angles, blending weights etc. These parts are dependent on the model we are describing and what information is available to us. The shape vector is expressed in a reduced dimension, this vector is 2D for a 3D model.

The solution for curvature estimation consists of two steps - preprocessing and main rendering. In the preprocessing step, the mapping function is obtained using the learning process. Curvature values for a given mesh are estimated by one of the existing algorithms (the authors, like many others, use the Rusinkiewicz's solution). The curvature estimations are mapped to the state vector. Training is done using regression (back-propagation training). In the rendering step, each frame or geometry update has its own unique state vector. This vector is used together with the mapping function and the curvature is reconstructed without the need of recomputing an entire model directly. The difference between the curvature computed directly and from the mapping function can be seen in Figure 5.5. The visual quality of both results is quite similar, but the solution by Kalogerakis runs 1.7 ms, while Rusinkiewicz took 91 ms.

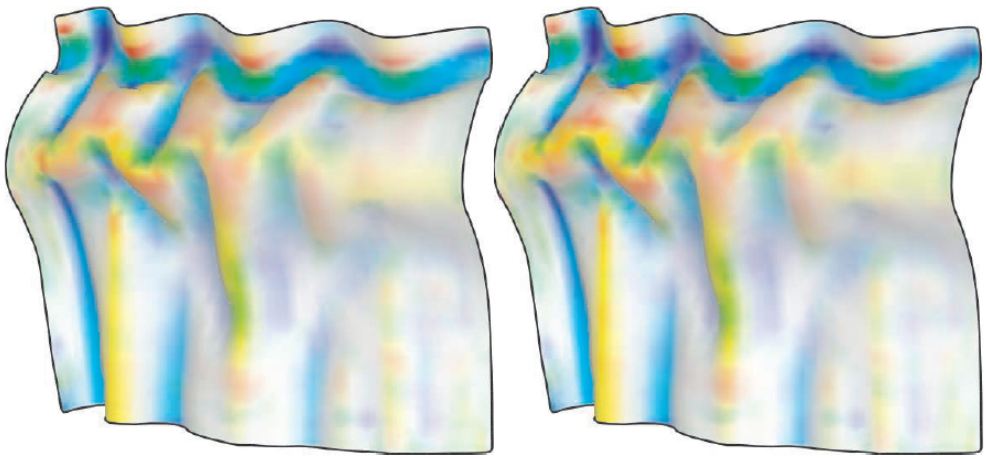


Figure 5.5: Comparison of principal curvatures produced by the method of Rusinkiewicz [Rus04] and the dynamic Kalogerakis [Kal+09]

5.4.1 Screen space

For dynamic computations the screen space can also be used, mainly for visualization purposes. We are not able to assign the screen space curvature estimations back to the geometry vertices. The curvature computations in the screen space are not very common. The only algorithm dealing with this problem known to us is by Mellado et al. presented in [Mel+13]. They propose the screen space curvature calculation by a sphere fitting. A point cloud is created from the screen space pixels and for each pixel, the best fitting sphere is searched.

For every pixel p on the screen, its neighboring pixels within a limited radius are collected. Pixels, whose depth differences against the p are greater than a threshold value, are rejected. They can be from the background or from another model. The final set of points (a local point cloud) is converted from the screen space to the view space (or world space). These converted values are fitted by the sphere. Final curvature is calculated from the sphere radius. With this approach, however, only the mean curvature is calculated. The Gaussian curvature cannot be computed this way and, therefore, principal curvatures cannot be calculated either.

The result of the method can be seen in Figure 5.6. Another limitation of the screen space version is a loss of details if the objects are further away from the camera. This effect can be seen in the upper right part of Figure 5.6, where the fine details of the hair are lost.

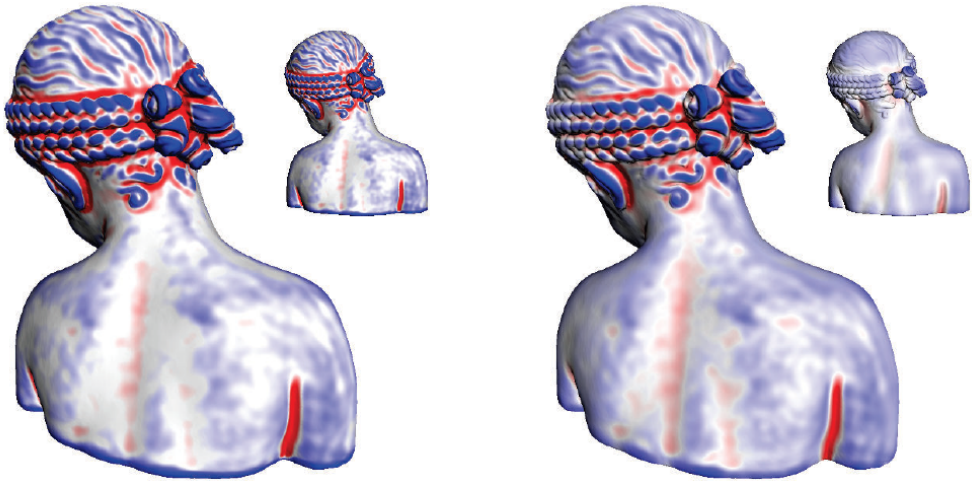


Figure 5.6: Comparison between mean curvature estimated directly from mesh (left) and its screen-space version (right). [Mel+13]

Chapter 6

Other shape characteristics

We have already covered curvature itself as a basic shape characteristic. However, curvature is seldom used in its raw form directly. Each object can have various additional characteristics based on normal vectors, positions, local objects volumes etc. We can also combine several characteristics together and create a different descriptor, but we should look for its important properties. Based on [Guo+16], each descriptor should have several properties. The most important one is mapping of a local geometry to a vector in Euclidean space $\mathbb{R}^D$ which supports efficient nearest-neighbor searches. Based on that, two feature vectors that describe similar neighborhood should be close to each other in the nearest-neighbor space. The size of vector should be compact with a small dimensionality D. Another important property is the robustness to the artifacts (noise, missing parts etc.) that are often found in data.

To select an appropriate descriptor is not a trivial task. There are many factors influencing the quality and precision. Usually, if a descriptor works well for one type of a problem it does not mean it will work well as an universal solution for every input. Several descriptors are implemented in the well-known PCL library [RC11]. It is a cross-platform library created from several smaller parts. Each part can be used separately. An advantage in a form of similar interfaces offers an easy way to change one descriptor to another and perform tests again. Apart from the descriptors, the library containts implementations of variety of other algorithms related to processing of point clouds in general.

Comparisons

In literature, there are many comparisons of different descriptors. An article comparing several algorithms to local descriptors have been presented by Heider et al. [Hei+11]. Their survey work is mainly focused on local descriptors but some information regarding global ones is also provided. Performance-based comparison of several normal-based descriptors have been conducted by Mateo et al. [MGT14]. Some of the interesting local descriptors from these surveys are explained in more detail in the following paragraphs.

A comparison of descriptors implemented in PCL is provided by Holz et al. [Hol+15]. They compare various algorithms and also show the source code snippets presenting the usage of them.

A comprehensive comparison of feature vectors was done by Guo et. al in [Guo+14] and [Guo+16]. Authors compare several properties of feature vectors, like their size, descriptiveness, noise robustness etc. Their goal is to determine which descriptor is suitable for which type of data.

Subdivision of shape descriptors into categories can be found in a survey by Tangelder et al. [TV08]. They divided descriptors into three main categories and each of them contains subcategories, see Table 6.1. Feature based descriptors are based on important parts of the model, usually feature lines or feature areas. Graph based descriptors use graph theory to describe object. They represent triangle mesh as a graph, where vertices are nodes and triangle edges are graph edges. Geometry based descriptors use geometry information, such as volume, normal vectors etc.

Feature -based	*Graph-based*	*Geometry-based*
Global features	Model graph	View based
Spatial map	Skeleton	Volumetric
Local features	Reeb graph	Deformation based

Table 6.1: Descriptors division from [TV08]

6.1 Curvature-based charactetistics

In Chapter 4, the basic background regarding curvature was described. The principal curvatures $K_{1,2}$ themselves, however, are not used as descriptors too often. They are represented by two values (K_1, K_2) where each of them hold one part of the information. A better way is to have a single value that holds a certain information alone, rather than two values, each with a partial information. Many authors create their own descriptors for certain purposes. Their common ground is the use of principal curvatures.

6.1.1 Detection of points of interest

These descriptors are often used for the detection of points of interest - points on faces, such as the nose, eyes, the mouth etc. These descriptors are derived from principal curvatures as follows.

- Mean curvature (K_H)
$$K_H = \frac{K_1 + K_2}{2}$$
- Gaussian curvature (K_G)
$$K_G = K_1 K_2$$
- Shape index (SI)
$$SI = \frac{2}{\pi} atan\left(\frac{K_1 + K_2}{K_1 - K_2}\right)$$

The shape index has been introduced by [KD92]. It describes a local topology of the shape independently on the scale, e.g., a cup has always the same index value, no matter what its size is. The shape index value is always in the range $\langle -1, 1 \rangle$ with -1 being a cup, 0 a saddle and $+1$ a cap. See Figure 6.1.

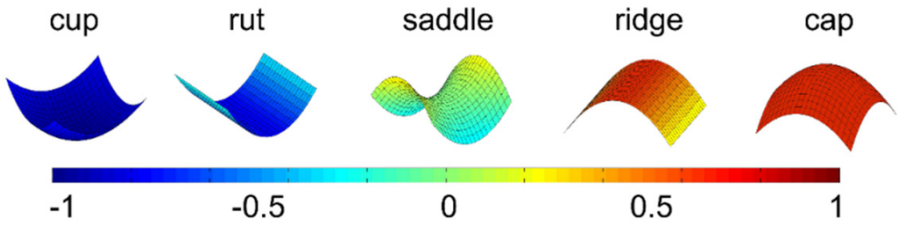


Figure 6.1: Shape Index SI

- Curvedness (C)

$$C = \sqrt{\frac{K_1^2 + K_2^2}{2}}$$

The curvedness has been introduced together with the shape index by [KD92]. It describes the magnitude of the curvature at a point, which is a measure of the extent to which a region deviates from flatness.

- Willmore energy (WE)

$$WE = \frac{(K_1 - K_2)^2}{4}$$

Willmore energy ([KS12]) is a quantitative measure describing the amount of deviation of the surface from a round sphere. A round sphere has a minimal Willmore energy, which is zero. Any other surface has always greater value, in other words, Willmore energy is never negative.

Application in human head models processing

The above mentioned descriptors are frequently seen in many publications regarding registration or description of human heads models. For human faces, it is often important to detect the tip of the nose. Szeptycki et al. in [SAC12] detect tip of the nose candidates based on evaluation of curvature based descriptors. For those points, support vector machine (SVM) classification from trained data is used to detect the correct nose tip. Their solution is, however, targeted only for heightfield-based data (sometimes referred as 2.5D) obtained from a direct scan.

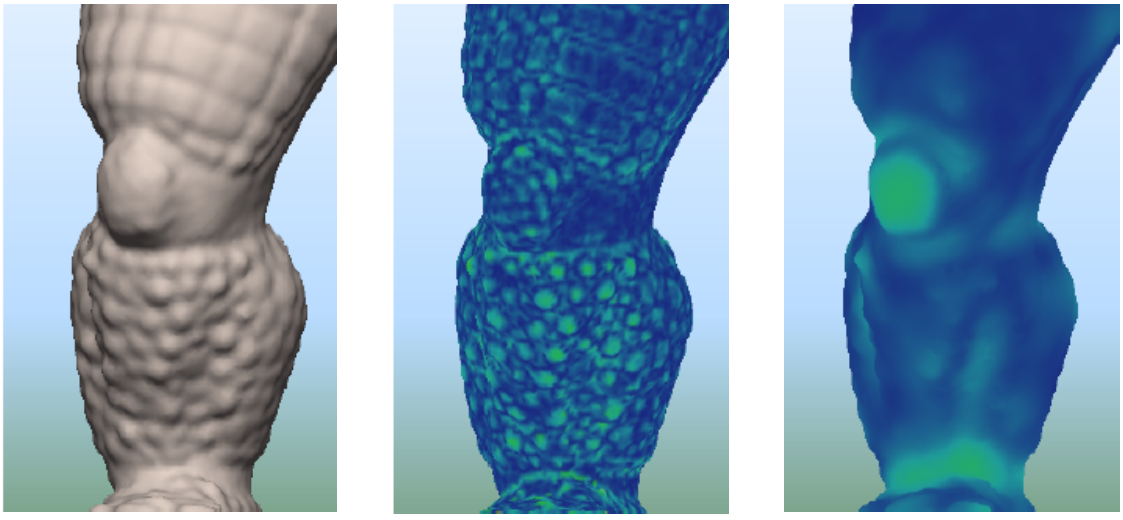
Nose tip and inner corners of eyes are detected with solution from Nair et al. [NC09]. Curvedness and shape index together with thresholds are used to detect nose and eyes. They use detected points for 3D face registration via Point Distribution Model (PDM). However, their solution is not fully automated and requires manual point selection within a training set.

Colbry et al. [CSJ05] use shape index to find feature points on human face. Based on shape index value and thresholding, they detect and recognize different points. They have created two versions of the algorithm - one for faces oriented towards camera and one for rotated ones. Faces oriented towards camera can benefit from this orientation and for example nose is detected as a point closest to the camera. A small deviation in forward orientation in about $\pm 15°$ is, however, possible.

Zhang et al. [ZW07] also use shape index. Based on this descriptor, they have created several regions. To select points from those regions, a graph is constructed and measurements of distances and angles are used to identify different points using fitness function.

6.1.2 Saliency

Lee et al. [LVJ05] do not use curvatures directly because curvatures capture fine details (see Figures 6.2a and 6.2b) that are usually not very interesting in the first phase of object description. They use a method called mesh saliency. Loosely speaking, a salient geometric feature is a region of the surface which has a nontrivial shape. It is computed from the mean curvature by the Gaussian-weighted average. This leads to the effect where fine details are smoothed out and more important parts of the models are highlighted (see Figure 6.2c). Another solution using saliency was presented in [GCO06]. Both of these solutions resemble a smooth version of curvature estimators, such as [CSM03; All+03; Kal+07].



(a) Original part of the mesh (b) Curvature (c) Saliency

Figure 6.2: Comparison of curvature and saliency. [LVJ05]

6.1.3 Curvature maps

A method based on the use of curvature was presented by Gatzke et al. [Gat+05]. Since curvature is a local point metric, it cannot be used directly for a description of points neighborhood. In the presented solution, a descriptor named *Curvature map* is created

from a neighborhood of a point. *Curvature map* accumulates curvatures (calculated by [Mey+03], but any other algorithm can be used) from a k-ring or geodesic neighborhood of a point. The *Curvature map* can be represented by two 1D, 2D or 3D vectors (one vector for mean and one for Gaussian curvature). A 1D version contains just curvature and leads to artifacts. It is not used and higher dimension maps are utilized instead. The dimensionality comparison of *curvature map* can be seen in Figure 6.3. The "*Ear Tip Vertex*" is selected as a reference point. Similarity of this point to other points on the model is color-coded from the least to the most similar points. As can be seen, for 1D vector, a lot of points within the model are identified as "*Ear Tip Vertex*". A 2D version is better, but not still quite correct. For a 3D version, only points within ears are correctly identified to be similar.

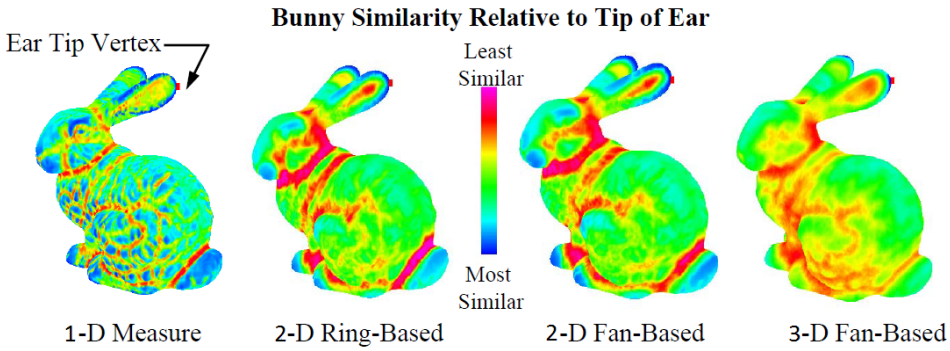


Figure 6.3: Curvature map and similarity measure relative to a selected vertex (ear tip). [Gat+05]

In 2D solution the distances to the points are stored in vectors together with curvature. In the end, curves are generated using each element created vectors. To generate a curve from the curvature map, a set of piecewise linear functions is used (a list of them can be found in article [Gat+05]). An example of one such curve with a distance can be seen in Figure 6.4. Curvature (mean or Gaussian) is expressed as a function of the distance from the point, for which the *curvature map* was created. The curves are further used to compare different points and if the curves are similar, the points are similar as well. The comparison metrics are described in [Gat+05].

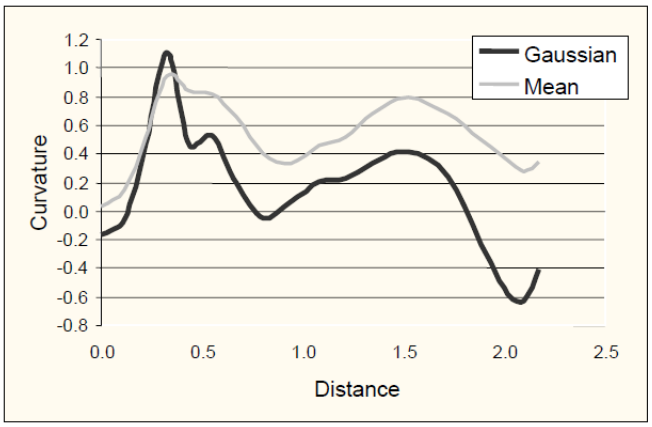


Figure 6.4: Example of curve generated from *curvature map* for a single point. [Gat+05]

A 3D version is similar to 2D. Apart from curvatures and distances, directions are stored as well. From a point, several directions are randomly chosen. In every direction, a 2D map is created as mentioned previously.

6.1.4 Scale independence

A scale-independent local descriptor has been presented by Cipriano et al. [CJG09]. They use a vertex neighborhood with a given radius. Vertices inside this neighborhood are weighted with the area of their nearest triangle. Also, vertices closer to the edge of the surface are given lower weights. Vertices are represented as a heightfield on a surface tangent plane around the central vertex. This is done to simplify further calculations. The final heightfield is described by local descriptors. As the shape of the heightfield can be quite complex, it is simplified by quadratic fitting. For very small areas, this approach will end up with a value of curvature at the center point (basically, it will be the algorithm for curvature estimation). For larger areas, however, this solution will create a surface descriptor averaged from several curvatures and their directions.

Another scale-independent solution was presented by Akagündüz et al. [AU09]. They used mean (H) and Gaussian (K) curvatures to detect points of interest on parameterized 3D surfaces.

6.1.5 Integral-based

Solution based on integral descriptors was presented by Gelfand et al. [Gel+05]. They use curvature computed from the geometry in a way similar to integral invariants from [Pot+09] (a voxelization-like algorithm). The computed curvature, after normalization, is used to obtain descriptors on the model surface (see Figure 6.5).

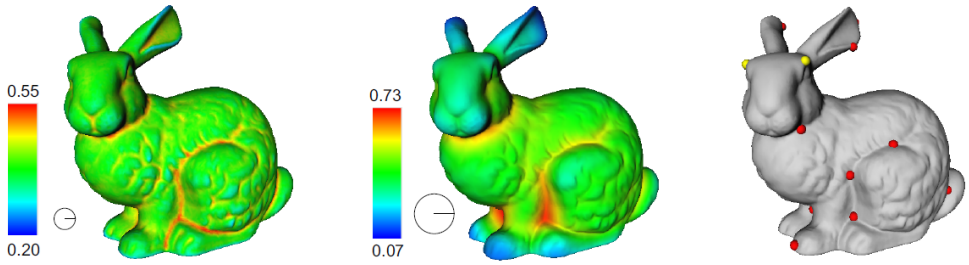


Figure 6.5: Normalized curvature computed with two different radius spheres (left, middle) and the resulting descriptors (right). [Gel+05]

Interesting parts of the models are found using a curvature histogram. The most important parts of the model are the ones, where frequency of the curvature is the lowest (e.g., this curvature value is the fewest in the model). The problem is to select only one point inside a certain neighborhood. For this, distances within a sphere are used. After the first point is selected, the sphere is created around it. If another point should be inside this sphere, this new point is rejected.

A problem with curvature is its scale-dependence. The solution to this problem is to use a different size of the sphere during the voxelization, as can be seen in Figure 6.5. Values of curvature are different, but its characteristics (convex/concave shape) are preserved and that is the important part for the presented solution of the shape descriptor.

6.1.6 Other

Specialized descriptors for certain purposes can be created. An example is a curvature-based descriptor for a face recognition presented by Salazar et al. [SCP10]. They use a statistics-based Fisher coefficients for surface feature description. Fisher´s analysis, instead of principal component analysis (PCA), allows us to find features with the most relevant information.

Relation of Gaussian curvature with *Heat Kernel Signature* proposed by Sun et al. [SOG09] was discussed in [Bro11]. *Heat Kernel Signature* is based on the concept of heat diffusion over a surface. Given an initial heat distribution over the surface, the heat kernel $h_t(x, y)$ relates the amount of heat transferred from one point (x) to another (y) after some time t. Using the transfer between two points directly will lead to a high complexity of computations. The computations are therefore restricted to just using $h_t(x, x)$, which means that they transfer a heat from a point to itself over a time. This descriptor is isometry-invariant, captures local geometric information at multiple scales, is insensitive to noise. A disadvantage is its dependence on the global scale of the shape.

The relation between the heat diffusion and Gaussian curvature K_G for small timesteps t according to [Bro11] can be expressed as:

$$h_t(x, x) = \frac{1}{4\pi t} + \frac{K_G(x)}{12\pi}. \tag{6.1}$$

6.2 Normal-based

Having a normal vector does not mean that we have the curvature. We can compute it, but it may slow things down and we may want to use just normal vectors. However, even if we do not directly require curvature the results of calculations are often of a similar nature as curvature and express the same information in a different way.

A method for point clouds originating from a 2D algorithm was proposed by Tombari et al. [TSS10]. They created a solution based on local histograms. They use a 2D image descriptor *SIFT* [Low04] as a reference and based on this, they have created a 3D modification called *SHOT*. Their algorithm uses normal vectors of points to construct local histograms. Based on a constructed Local Reference Frame (LRF, recall Section 2.4), the spherical neighborhood is divided into several bins (8 divisions along the azimuth, 2 along the elevation, and 2 along the radius - see Figure 6.6). Each bin has its own histogram created from angles between normal vectors of points and a normal at the center point. In the end, the normalization of the descriptor is required to improve robustness.

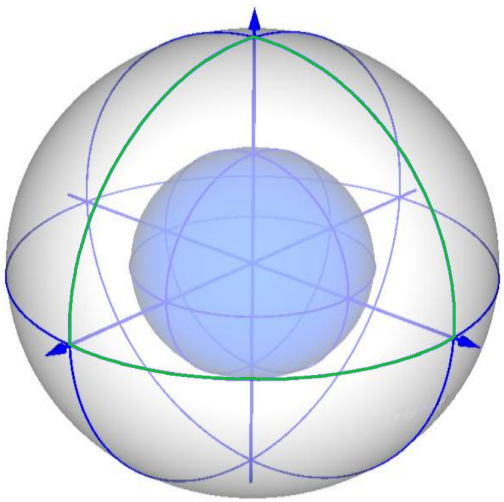


Figure 6.6: Support structure to compute *SHOT*. For clarity, only 4 azimuth divisions are indicated. One of these azimuth divisions is highlighted in green. The division in radius and elevation results in inner sphere. [TSS10]

One of the older descriptors is based on a shape context. *3DSC* was proposed in 2004 by Frome et al. [Fro+04] and later improved by Tombari et al. [TSDS10] (*USC*). The original version *3DSC* uses spherical neighborhood. The sphere is centered in a way that its "north pole" is in a direction of a normal vector. The sphere is divided to several bins. In azimuth and elevation, the spacing is regular, the division along the radius is spaced logarithmically (smaller bins closer to the center point). For each bin, weight from its corresponding points is calculated. This weight depends on bin size and number of points inside the bin. However, the rotation of sphere is not determined. It can be rotated around the normal vector in an infinite number of ways. Original article solves this by trying different sphere orientations. However, solution using local

reference system improves the stability and quality of the descriptor. This improvement was proposed in *USC*. A different algorithm originating in shape context was proposed by Kokkinos et al. [Kok+12]. Their solution is invariant to isometric deformations.

Marton et al. [Mar+10] proposed Radius-Based Surface Descriptor (*RSD*). This descriptor uses radial relationships of the point and its neighbors. Euclidean distances and normal vector differences between central point and other points in the neighborhood are calculated. For each pair of points a sphere is found using the points and their normal vectors. From all the spheres created from the neighborhood, the ones with minimal and maximal radii are used for the descriptor. The idea of this solution is similar to a descriptor based on a curvature of the neighborhood.

Another descriptor for point clouds is Point Feature Histograms (*PFH*) descriptor [Rus+08] created by Rusu et al. They use multi-dimensional histogram created around the point. *PFH* is based on the relationships between the points in the spherical area and their estimated surface normals. The quality of the final descriptor is influenced by the quality of normal vectors. If we have a pair of two points (P_s, P_t) with normal vectors ($\boldsymbol{n}_s$, $\boldsymbol{n}_t$) and local coordinate systems in them ($\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$), they can be described via a quadruple $q = \langle \alpha, \phi, \theta, d \rangle$ (see Figure 6.7), where

$$\begin{array}{rcc} \alpha & = & \boldsymbol{v} \cdot \boldsymbol{n}_t \\ \phi & = & (\boldsymbol{u} \cdot (P_t - P_s))/||P_t - P_s|| \\ \theta & = & arctan(\boldsymbol{w} \cdot \boldsymbol{n}_t, \boldsymbol{u} \cdot \boldsymbol{n}_t) \\ d & = & ||P_t - P_s|| \end{array} . \tag{6.2}$$

Instead of 12 values (two times - 3 values per position, 3 per normal) we have only 4 values that are also rotationally and transitionally invariant.

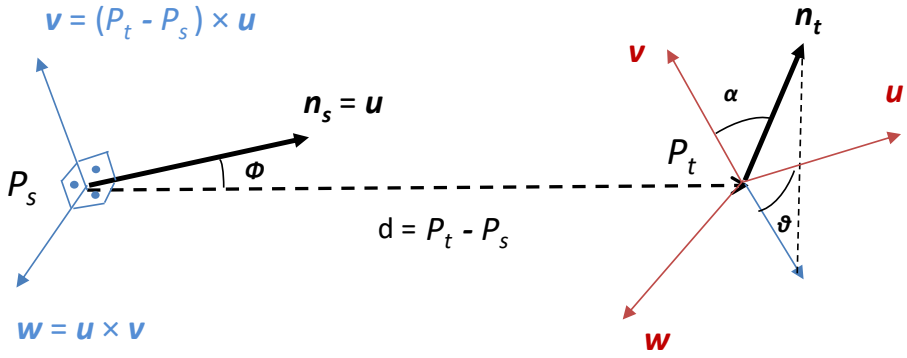


Figure 6.7: Angles from quadruple between two points (P_s,P_t) in *PFH*. [Rus16]

Created tuples are treated as 4D vectors. In some cases, the fourth component (distance) can even be omitted because points can be sampled in some view-dependent manner. From the vectors, 16-bin histogram is created as

$$bin = \sum_{i=0}^{i \leq 4} 2^{i-1} step(s_i, q_i), \tag{6.3}$$

where i is a quadruple index element, s_i is the center of value interval (0 for α, ϕ, θ and $d/2$ for distance d) and *step* is a function that gives 0 if $q < s$ and 1 otherwise. In the

end, each bin contains points based on the neighborhood with a radius $d/2$.

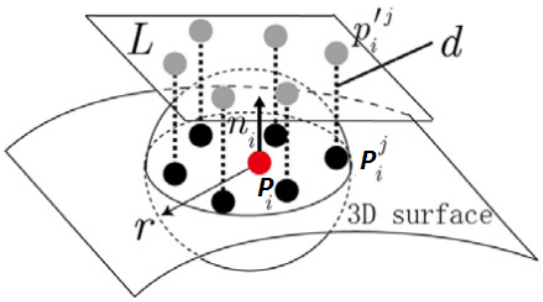
A problem is the complexity of the method, since it computes relations between every two points, leading to complexity $O(nk^2)$, where n is the number of points P_i and k is the number of neighbors for each point P_i.

A solution for the slow performance of the *PFH* method was proposed by Rusu et al. as Fast Point Feature Histograms (*FPFH*) descriptor [RBB09]. This version reduces the time complexity to $O(nk)$, while retaining most of the *PFH* power. The number of points in the neighborhood with a threshold radial distance is limited and weighting is used for a final histogram creation. The simplicity, low number of elements in the final feature vector and a high performance make very often *FPFH* a basic choice when selecting a descriptor. After a modification, the feature vector can also be used for a global description of entire model (see [Rus+10]). This solution is based on the direction of viewpoint vector from camera that is scanning the model.

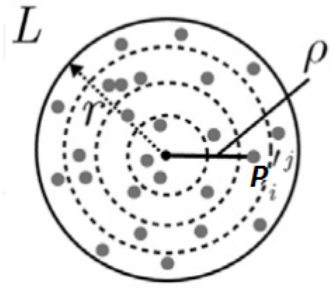
In many cases, solely normal vector-based solutions have disadvantages because their quality often depends on the quality of normal vectors. Local feature descriptor *LFSH* was proposed by Yang et al. [JZQ16]. They use not only normal vectors, but also the density of points in the neighborhood and local depths.

A sphere with a radius r (based on the neighborhood size) is created around the center point P_i. A sphere tangent plane L is created with a help of local reference axis that goes through point P_i. Points from neighborhood are projected into this plane which leads to local depths d. The entire process can be seen in Figure 6.8a. The points in tangents plane are also used for density calculation based on several layers, as depicted in Figure 6.8b.

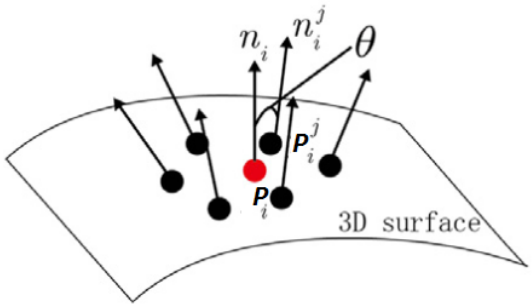
In the last step, deviation angles between normal vectors from neighborhood are calculated (Figure 6.8c).



(a) Creation of local depths



(b) Point distribution in tangent plane



(c) Calculation of deviation angles between normals.

Figure 6.8: Steps of *LFSH* feature vector computation from [JZQ16]

Results from all three steps are used to create an overall histogram. The histogram is used as a feature vector.

6.3 Other

Even though our primal region of interest are curvatures, we have researched other algorithms as well to extend our knowledge and possible combine the different approaches. There are many more ways how to describe geometrical object for further processing. Some of them use a variation of 2D descriptors known from image processing. Skeleton-based methods describe a model with its underlying skeleton. These methods are mainly for global characteristics, but can be used for a part of the model (e.g. a hand with fingers).

Older, but due to its simplicity still used descriptor is *SPIN image* [JH99]. *SPIN image* is created from a projection plane that is being rotated (spinned) around one axis of LRF. For a descriptor, a keypoint is chosen and points in its neighborhood, given by radius, are projected onto the rotated plane. In the plane, points are used to calculate various statistics and their histograms are used as feature vector. However, this descriptor is limited by a data resolution and uniformity of points distribution. If two datasets have different points distributions, the statistics for the same parts are often incorrect. The original version has also a problem with different scales. However, a scale

invariant method extending classic *SPIN image* has been recently proposed by [Lin+17].

The descriptor named *TriSi* based on axes of LRF (recall Section 2.4) has been presented by Guo et al. [Guo+13]. A set of local descriptors is generated based on a triangle mesh surface. For a point on the surface, its radial neighborhood is used. From neighboring points, a matrix is created using continuous PCA algorithm and three eigenvectors ($\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$) of this matrix are computed. They form the LRF system. However, the signs of eigenvectors are ambiguous and a sign disambiguation technique is used. The newly created "eigenvectors" ($\widetilde{\boldsymbol{v}}_1$, $\widetilde{\boldsymbol{v}}_2$, $\widetilde{\boldsymbol{v}}_3$) are used as the description of LRF.

TriSi descriptor is in 3D created from three spin sheets (each of them is considered as a single *SPIN image [JH99]*). Every spin sheet is a plane corresponding to one axis defined by a sign-corrected eigenvector. An example of a spin sheet aligned in the plane given by $\widetilde{\boldsymbol{v}}_1$ and $\widetilde{\boldsymbol{v}}_3$ can be seen in Figure 6.9. Points from the model are projected into these planes in a way similar to *SPIN image*.

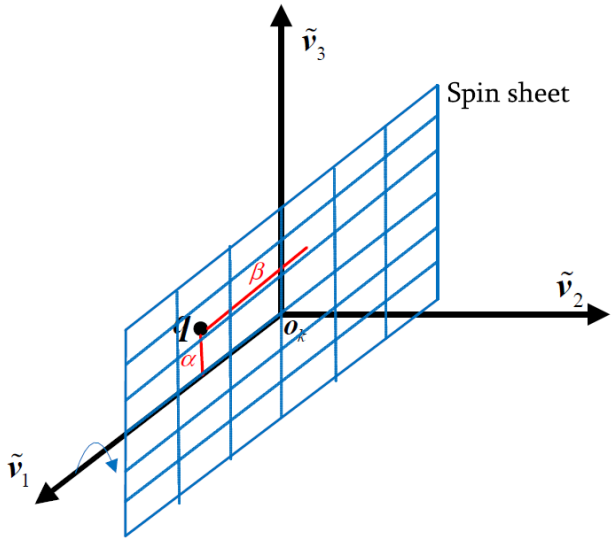


Figure 6.9: Generating a spin sheet for *TriSi*, α and β are distances of origin from projected point q after its perpendicular projection onto axes $\widetilde{\boldsymbol{v}}_1$ and $\widetilde{\boldsymbol{v}}_3$. [Guo+13]

For every projected point, values of α and β are accumulated into a 2D histogram of certain size. To overcome problems caused by noise, histograms can be bilinearly interpolated. The histogram can be quite large (depends on the chosen size). To decrease its size, PCA is used again. From a selected set of training descriptors, a matrix M is calculated as

$$M = \sum_{i=1}^{L} (\boldsymbol{f}_i - \bar{\boldsymbol{f}})(\boldsymbol{f}_i - \bar{\boldsymbol{f}})^T, \tag{6.4}$$

where L is the number of training descriptors, $\boldsymbol{f}_i$ is the selected training descriptor and $\bar{\boldsymbol{f}}$ is the mean vector created from all training descriptors. Using the eigenvalue decomposition, eigenvectors of M are calculated. The final compressed descriptor is created by an approach partially similar to a singular value decomposition (SVD) used for image compression. The resulting TriSi descriptor is robust to noise and a mesh resolution.

Algorithm *3D-Div* was proposed by Shah et al. [Sha+13]. They detect 3D keypoints on the surface of the mesh. For each keypoint, a local surface patch is created using a sphere of the given radius. In the next step, the LRF is constructed for each keypoint the same way as described in previous paragraphs for the *TriSi* method. For every keypoint, its LRF vectors, and local surface patch, the trilinear interpolation is performed to get uniformly sampled points. The normalized 3D vector field is computed from the local surface patch and aligned with the LRF. A gradient is computed for this reoriented field. For a keypoint, Euclidean distances to its neighbors within the local surface patch are calculated. Gradient combined with distances of neighboring points is used as the final descriptor.

The quality and robustness of descriptors is often limited by the construction of LRF. The *BSC* (Binary Shape Context) descriptor with a more robust LRF was proposed by Dong et al. [Don+17]. For LRF construction, covariance matrix is first created and eigenvectors are used as basis vectors. However, the eigenvectors have ambiguous directions. There are two choices for each axis, leaving us with four possible LRF. To solve this issue, they work with all four variants. Once the LRF is constructed, the local neighborhood is projected onto the coordinate planes created by the axes of LRF. The planes are divided into bins and for each bin, density and distance are determined. To improve robustness (noise, point density), the results are weighted using Gaussian kernel. The effect of this solution can be seen in Figure 6.10. The final descriptor consist of LRF vectors, and binarized densities and distances in each plane. The binarization is done by the scheme proposed in [Don+17].

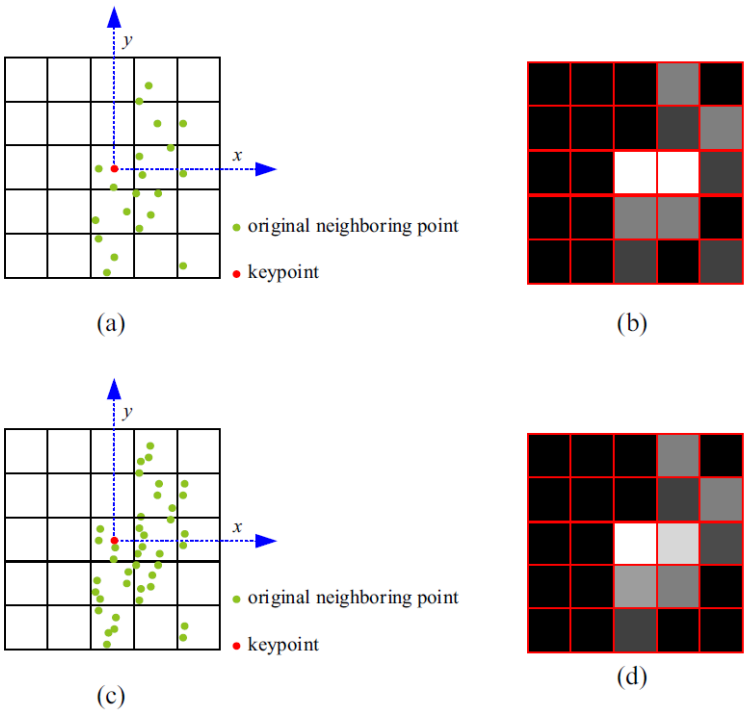


Figure 6.10: One projection plane of *BSC* and two different point densities of neighborhood (a) and (c) and their corresponding weighted histograms (b) and (d) as suggested by [Don+17]

6.3.1 Euclidean distances

Solution presented by Maximo et al. [Max+11] uses local heightmaps with stored Euclidean distances. The tangent plane is created at a vertex from its position and a normal vector. The distance-map (sampled as a grid 16×16) in the tangent plane is aligned with the principal curvature directions at the vertex. Each grid cell has associated one ray perpendicular to the tangent plane. Distances are computed from the plane to the intersection of the mesh surface with the ray. See an example of one such tangent plane for a single vertex in Figure 6.11. This approach is simple, but robust to holes, non-manifoldness etc. since it only computes ray - triangle intersections and store distances.

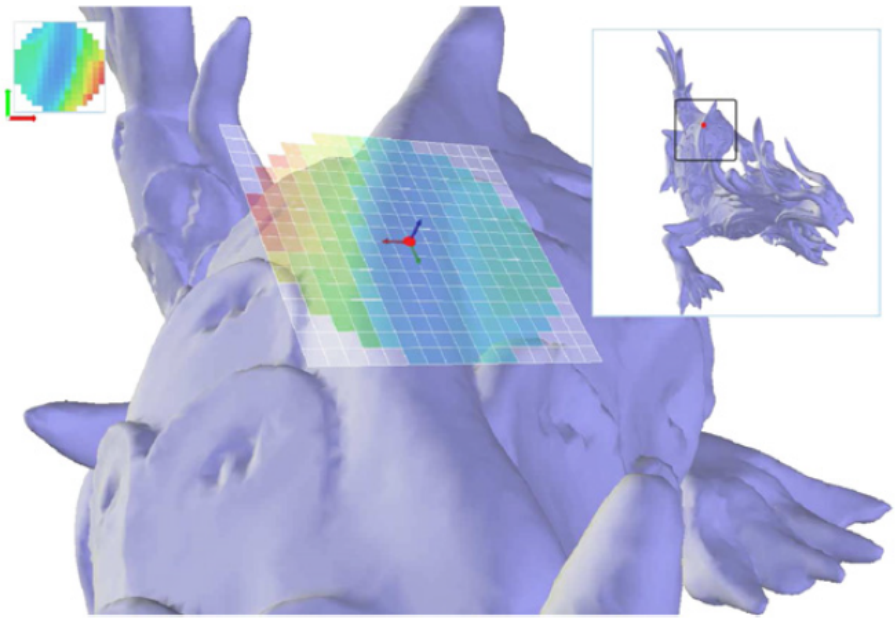


Figure 6.11: A surface descriptor example for a single vertex (red). The descriptor is a distance-map in a tangent plane. Distances are visualized as color-coded values. [Max+11]

We can directly compare meshes from the constructed distance-map at each vertex. However, this solution is not robust in general. This is caused by principal curvature directions, because they are not always correctly aligned with the tangent plane. The solution could be to compute differences for every possible rotation and select the minimal value as the similarity. This brute-force solution is, however, very slow. Authors use Zernike polynomial functions instead. The Zernike polynomials are a set of functions orthogonal over the unit circle. They compare Zernike polynomial coefficients instead of pixels from the distance-map. The theory behind this is quite complex and out-of-scope of this report.

The basic solution uses a descriptor defined at a single vertex. This may cause some problems in ignoring objects features. In the proposed solution, Maximo et al.

use the vertex neighborhood. For each vertex, Gaussian weights are applied to Zernike coefficients. Final coefficients are combined as an average from nearby vertices (within a neighborhood radius).

6.3.2 Voxelization

The solution based on local voxelization has been presented by Knopp et al. [Kno+10]. Their solution is a 3D variant of 2D feature descriptor *SURF* [Bay+08]. The geometry is voxelized into the cube using the intersection of mesh faces with the grid-bins. The saliency measure is computed of each grid-bin and is defined as the absolute value of the determinant of the Hessian matrix that is computed from box filters on the rasterized volume. This is similar to 2D convolution for 2D image. The *3D SURF* descriptor is computed at the maxima of the voxelized grid. Another voxel based solution was created by Song [Son15].

6.3.3 Fourier transform

A descriptor based on Fourier coefficients has been proposed by Foulds et al. [HF11]. To overcome the problem with rotation and translation, the objects are first processed using PCA. The centroid of the object is set based on the results of PCA. In the next step, distances of triangle faces from the centroid are stored in a matrix C. The matrix is indexed with angles in polar coordinates. The distances are stored in the matrix at the positions $[\theta, \varphi]$, where $\theta \in \langle 0, \pi \rangle$ and $\varphi \in \langle 0, 2\pi \rangle$. The angles are used with an increment step, the authors suggest to use the increments of size 1, 4, 9 or 18 degree. The larger the increment is, the smaller matrix and thus the lower precision we have.

From the C matrix, the Fourier transform is calculated, from which a $(2N + 1) \times (2N + 1)$ feature matrix centered on the lowest frequency coefficient is created (N is the number of Fourier coefficients selected by the user). For each element of the feature matrix, its distance to the centroid is calculated. Based on this distance, the elements of the feature matrix are rearranged and sorted into a 1D array. This creates the feature vector (descriptor). Later on, the feature vectors are used to compare the similarity of models. The matching process can be found in [HF11].

Solution based on a 3D curve and its description by Fourier series have been proposed by Elmustapha et al. [Elm+10]. They again use PCA to align models to the initial positions. From the model in its initial position, a closed 3D curve is built. They use a Helix curve (see Figure 6.12) constructed on the unit sphere given by:

$$\begin{array}{rcl} x(t) & = & cos(qt)sin(t) \\ y(t) & = & sin(qt)sin(t) \\ z(t) & = & cos(t) \end{array} , \tag{6.5}$$

where q is a parameter of curve quality (the bigger value, the more points the curve has) and t is a curve parameter, $t \in < 0, \pi >$.

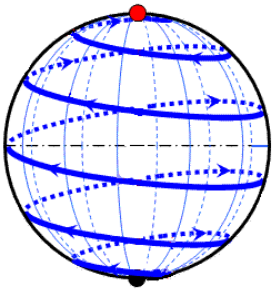


Figure 6.12: A spherical Helix curve

For the descriptor curve extraction, ray-casting is used. The unit sphere with a helix curve is placed in the center of the mass. Rays go from the object's center of mass through the points on the helix curve (points are given by Equation 6.5). The furthest intersections of the ray with the faces of the surface triangles create points of the descriptor curve. To close the curve the first and the last point are set to be the same.

The created curve must be re-parameterized in order to compute the feature vector. The authors have selected the natural parametrization (the arc length parametrization). Fourier series are applied to this parameterized curve. The feature vector is created from the magnitudes of complex quantities. It is a good decision to take the first coefficients, because the later represents high frequencies with noise. The feature vectors are used for objects comparison and searching in large databases, for more details see the article [Elm+10].

6.3.4 Neural Networks

Nowadays, solutions based on neural networks gain interest. Neural networks are able to find relations that may be hidden to other algorithms. The main difference with previous methods is the process of finding the final transformation that will register the parts together. Neural networks are used mainly in this part of the algorithm. The core part of the registration, the description of the neighborhood, can in some cases be a standard solution as described in the previous sections.

The overall problem with these types of solutions are their inputs. Neural networks often require training sets from which they can learn and improve their weights. If the user does not have the input training data, some universal predefined values given by authors of publications can be used. However, in these cases, the quality of registration can be affected.

Zeng et al. [Zen+16] use convolution neural network. As an input, they use volume data created from truncated distance functions from the point neighborhood. Khoury et al. [KZK17] create solution that can be used as a direct replacement of *FPFH*. They use parametrization of model via spherical histograms centered at each point. These histograms describe local neighborhoods of points. Neural network is used to map histograms to a lower-dimensional Euclidean space and thus speed up nearest-neighbor searches.

Ai et al. [Ai+17] select a set of points of interest on the surface of both parts. They

first downsample inputs and for every point, covariance matrix is created. Based on its eigenvalues, the best points for description are selected. Points in their neighborhood are gathered into clusters around already selected best points. The centers of the clusters are used as keypoints. The clusters and the obtained keypoints are often different for each model, because of noise and other inconsistencies in input data. To find the corresponding points on input parts, a convolutional neural network (CNN) is used. The input keypoints are not used directly. Instead of them, a weighted adjacency matrix is computed. Euclidean distances between keypoints are used as weights. From the corresponding points, the transformation matrix is created and in the final step of the algorithm, ICP [BM92] is used to improve the quality.

From now on, the following chapters present our contributions. Their overview can be found in section 1.2. We start with our contributions in curvature estimation algorithms and continue with the use of curvature and shape descriptors.

Chapter 7

Screen space curvature

Curvature estimation can be computationally expensive for geometry objects with a high number of triangles. The existing algorithms are usually not suitable for a real-time curvature estimation if object is changing for example during interactive sculpting.

To partially mitigate this problem, the curvature is not estimated directly from the mesh, but rather from the final rendered image in screen space. In screen space, only data that are currently visible and interesting for the viewer are processed. Calculations are independent of triangle count of the original object, the only limitation is the screen resolution. There is also an advantage that the curvature can be calculated from any possible model representation with the same algorithm. There is no limitation to triangle meshes, the final scene can contain volumetric models, implicit surfaces, procedurally generated geometry and other screen space generated effects, such as a water surface.

Our proposed algorithm, published in [PVK16] and [PVK17], works in the screen space and is therefore independent of the data representation. We are interested only in the final rendered image. The core of the algorithm is similar to the one used in [Rus04] and uses fundamental forms as well.

The screen space techniques have a major advantage to existing rendering software - they can be easily added as post-process methods or replace an existing rendering output. Nowadays, these methods are quite popular due to their simplicity and performance for many problems, such as water rendering, lighting, ambient occlusion (shading technique used to calculate the exposition of a point to an ambient light) and reflections. In screen space, however, some problems may occur, usually on the object edges, where pixel flickering may occur. Another disadvantage comes directly from the screen space itself, where the geometry outside the visible area cannot contribute to the results.

In this section, we present the proposed algorithm for screen space curvature calculation and briefly its ambient occlusion modification. First, a description of the proposed algorithm for a triangle mesh is presented. The screen space modification is discussed next. As last, ambient occlusion modification is mentioned.

7.1 Object space version

7.1.1 Basic algorithm

The main idea is to describe every triangle independently by the shape operator W, recall Equation (4.15). Elements of the shape operator must be calculated in order to find eigenvalues of the matrix and calculate the final curvatures.

The proposed method uses an orthonormal basis. In such a case, the first fundamental form (I) is the identity matrix which means that the second fundamental form (II) is equivalent to the shape operator, i.e. $W = \mathrm{II}$.

To eliminate one dimension, every triangle is transformed to a local coordinate system, also known as the tangent space (see Section 2.2). Once the triangle is in the local space, one of the dimensions is constant and represents the plane of the triangle. In the following calculations, this dimension is not used and the problem is reduced from 3D to 2D.

7.1.2 The curvature calculation

The triangle in the local space is used to build the shape operator, as can be seen in Equation (4.14), where variables L, M, N are unknown.

The shape operator describes the change of the normal over the edge of the triangle. The triangle is in the local space and one of the coordinates is constant. This coordinate is left out, which leads to 2D vectors instead of 3D. The edges of the triangle are expressed as 2D vectors

$$(u_i, v_i)^T = V_{Li} - V_{L(i+1)}, \tag{7.1}$$

and changes of the triangle normals are again as 2D vectors

$$(dNu_i, dNv_i)^T = \boldsymbol{n}_{Li} - \boldsymbol{n}_{L(i+1)}, \tag{7.2}$$

where $i = 1, 2, 3$. Index i denotes the triangle edge index.

Changes of normals along the edges of the triangle are known. These changes together with edge vectors are used to create a system of equations to find the unknown variables L, M, N. For one edge of the triangle, we get the underdetermined system

$$\begin{bmatrix} L & M \\ M & N \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} dNu_1 \\ dNv_1 \end{bmatrix}. \tag{7.3}$$

However, by constructing the same system for every edge of the local space triangle, an overdetermined system is obtained. The system is in the form $A\boldsymbol{x} = b$, the least squares method is used to obtain unknown variables:

$$\boldsymbol{x} = \ (A^T A)^{-1} A^T b \ . \tag{7.4}$$

In this particular case, the matrix A is built from the triangle edge vectors $(u_i, v_i)^T, i = 1, 2, 3$ and b is the vector of changes of the triangle normals $(dNu_i, dNv_i)^T, i = 1, 2, 3$. Index i denotes the triangle edge index. Final matrices are as follows:

$$A = \begin{bmatrix} u_1 & v_1 & 0 \\ 0 & u_1 & v_1 \\ u_2 & v_2 & 0 \\ 0 & u_2 & v_2 \\ u_3 & v_3 & 0 \\ 0 & u_3 & v_3 \end{bmatrix}, b = \begin{bmatrix} dNu_1 \\ dNv_1 \\ dNu_2 \\ dNv_2 \\ dNu_3 \\ dNv_3 \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} L \\ M \\ N \end{bmatrix}. \tag{7.5}$$

calculate. Some optimizations can be done to decrease the total number of numerical operations. Substitution $B = A^T A$ is introduced. The matrix B is symmetric and its elements can be represented by variables p, q, r:

$$B = A^T A = \begin{bmatrix} p & q & 0 \\ q & p+r & q \\ 0 & q & r \end{bmatrix},$$

$$p = u_1^2 + u_2^2 + u_3^2, \tag{7.6}$$

$$q = u_1 v_1 + u_2 v_2 + u_3 v_3,$$

$$r = v_1^2 + v_2^2 + v_3^2.$$

The inverse of the matrix B can be computed using Equation (7.7). Since B is symmetric, the computation is fast and easy.

$$B^{-1} = det(B) \begin{bmatrix} p(r+p) - q^2 & -qr & q^2 \\ -qr & pr & -pq \\ q^2 & -pq & p(r+p) - q^2 \end{bmatrix} \tag{7.7}$$

The final step is to calculate values for the unknown vector x. A part of this step can be simplified, because the inverse of the matrix B is symmetric (see the symmetry pattern in Equation (7.8)) and the matrix A has many zero elements. A simplified multiplication can be seen in Equation (7.9).

$$B^{-1} = det(B) \begin{bmatrix} b_1 & b_2 & b_3 \\ b_2 & b_4 & b_5 \\ b_3 & b_5 & b_6 \end{bmatrix}, \tag{7.8}$$

$$B^{-1} A^T = det(B) *$$

$$\left(\begin{bmatrix} u_1 b_1 & u_1 b_2 & u_2 b_1 & u_2 b_2 & u_3 b_1 & u_3 b_2 \\ u_1 b_2 & u_1 b_4 & u_2 b_2 & u_2 b_4 & u_3 b_2 & u_3 b_4 \\ u_1 b_3 & u_1 b_5 & u_2 b_3 & u_2 b_5 & u_3 b_3 & u_3 b_5 \end{bmatrix} + \begin{bmatrix} v_1 b_2 & v_1 b_3 & v_2 b_2 & v_2 b_3 & v_3 b_2 & v_3 b_3 \\ v_1 b_4 & v_1 b_5 & v_2 b_4 & v_2 b_5 & v_3 b_4 & v_3 b_5 \\ v_1 b_5 & v_1 b_6 & v_2 b_5 & v_2 b_6 & v_3 b_5 & v_3 b_6 \end{bmatrix} \right) \tag{7.9}$$

Having obtained the final vector $\boldsymbol{x}$, we can construct the desired shape operator. From this matrix, the eigenvalues λ_1, λ_2 are computed by solving the characteristic polynomial. These values correspond to the principal curvature estimated to the triangle. The principal curvatures can be used to evaluate the mean and Gaussian curvature (see Equations (4.6) and (4.7)).

The presented algorithm computes the curvature for each triangle. To obtain the curvature at the vertices, we have to use all adjacent triangles at the given point. The final curvature can be estimated as a simple average from all adjacent triangles or the curvature can be further weighted by the triangle area.

In the above calculations, an overdetermined system was constructed from all three edges of the triangle. To solve the system, only two edges are sufficient (values for $i = 3$ will be zero). Differences in both approaches are shown in Section 7.3.

7.2 Screen space variation

The screen space version of the proposed algorithm was designed to fit directly into an existing deferred rendering pipeline. Only the normal and the depth (from which the position is reconstructed) are required for every pixel. There could be probably some quality improvements, if additional information (id of the triangle to which the current pixel belongs, the triangle size in the screen space etc.) was available, but this is not the current target.

The screen space depth buffer can be interpreted as a 2.5D function with an underlying regular grid and function values of the depth. In the screen space, there is a constant step size between neighboring pixels. Those pixels are triangulated and each pixel center is taken as a triangle vertex. One possible subdivision can be seen in Figure 7.1. This screen space triangulation is converted to the world or camera space by reconstruction of the position and the normal for each pixel. This creates a simple triangulated mesh and the curvature is estimated on this mesh using the technique described in Section 7.1.1.

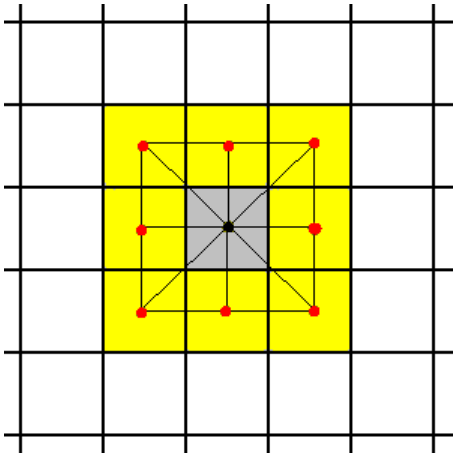


Figure 7.1: The local triangulation of neighborhood pixels. The center pixel is currently calculated, one-ring neighborhood forms triangulation.

The algorithm from Section 7.1.1 can be used directly in the screen space. It can run entirely on the GPU, using a pixel shader. As shown in Section 7.1.1 (see Equation (7.7)), the inverse matrix can be computed very fast and only six values have to be stored due to the matrix symmetry.

There is only one difference in the final calculation step of the vector x. If all three edges of each triangle are used, there is a limitation caused by shaders, where maximal dimension of the native data type can be four, but 3×6 matrix and 6×1 vector are needed. However, if the simplified matrices from Equation (7.9) are used, the calculation can be split into two parts. Each of these parts has a halved dimension (Equation (7.10)) of the original matrix.

$$B_1 = \begin{bmatrix} u_1b_1 + v_1b_2 & u_1b_2 + v_1b_3 & u_2b_1 + v_2b_2 \\ u_1b_2 + v_1b_4 & u_1b_4 + v_1b_5 & u_2b_2 + v_2b_4 \\ u_1b_3 + v_1b_5 & u_1b_5 + v_1b_6 & u_2g + v_2b_5 \end{bmatrix},$$

$$B_2 = \begin{bmatrix} u_2b_2 + v_2b_3 & u_3b_1 + v_3b_2 & u_3b_2 + v_3b_3 \\ u_2b_4 + v_2b_5 & u_3b_2 + v_3b_4 & u_3b_4 + v_3b_5 \\ u_2b_5 + v_2b_6 & u_3b_3 + v_3b_5 & u_3b_5 + v_3b_6 \end{bmatrix}, \tag{7.10}$$

$$\boldsymbol{x} = det(B) \left(B_1 \begin{bmatrix} dNu_1 \\ dNv_1 \\ dNu_2 \end{bmatrix} + B_2 \begin{bmatrix} dNv_2 \\ dNu_3 \\ dNv_3 \end{bmatrix} \right)$$

If only two edges are used, calculations can be computed even more efficiently on the GPU:

$$\boldsymbol{x} = det(B) \begin{bmatrix} u_1b_1 + v_1b_2 & u_1b_2 + v_1b_3 & u_2b_1 + v_2b_2 & u_2b_2 + v_2b_3 \\ u_1b_2 + v_1b_4 & u_1b_4 + v_1b_5 & u_2b_2 + v_2b_4 & u_2b_4 + v_2b_5 \\ u_1b_3 + v_1b_5 & u_1b_5 + v_1b_6 & u_2g + v_2b_5 & u_2b_5 + v_2b_6 \end{bmatrix} \begin{bmatrix} dNu_1 \\ dNv_1 \\ dNu_2 \\ dNv_2 \end{bmatrix}. \tag{7.11}$$

All calculations are based on triangles that need to be reconstructed in the screen space. They are obtained directly from the currently rendered pixel and its neighbors. See again Figure 7.1, where possible subdivision and the triangle reconstruction are shown. However, if the neighborhood width is only one pixel (as the case in Figure 7.1), all of those triangles are not needed to compute the curvature estimation and based on our testing, the use of only one of them is sufficient.

7.2.1 Level of detail

In the screen space, visible details often depend on the camera distance from the scene object. Small triangles in the world space can occupy almost all the pixels of the rendered image if the camera is very close to the surface. On the other hand, if the camera is far away, the same triangle can take only one pixel of the final image. Taking this into consideration, the level of detail can be used to improve the visual quality of the estimated curvature.

If the neighborhood with one pixel width is used, triangles of the original mesh can be seen in the estimated curvature (see Figure 7.2a). The estimated curvature within every triangle is the same. GPU interpolates normals and positions during rendering, leading to a smooth Phong shading, but the proposed method uses differences in the positions and normals. These differences are constant (except for the numerical errors) for a flat geometry, leading to the same curvature at every inner point of each triangle.

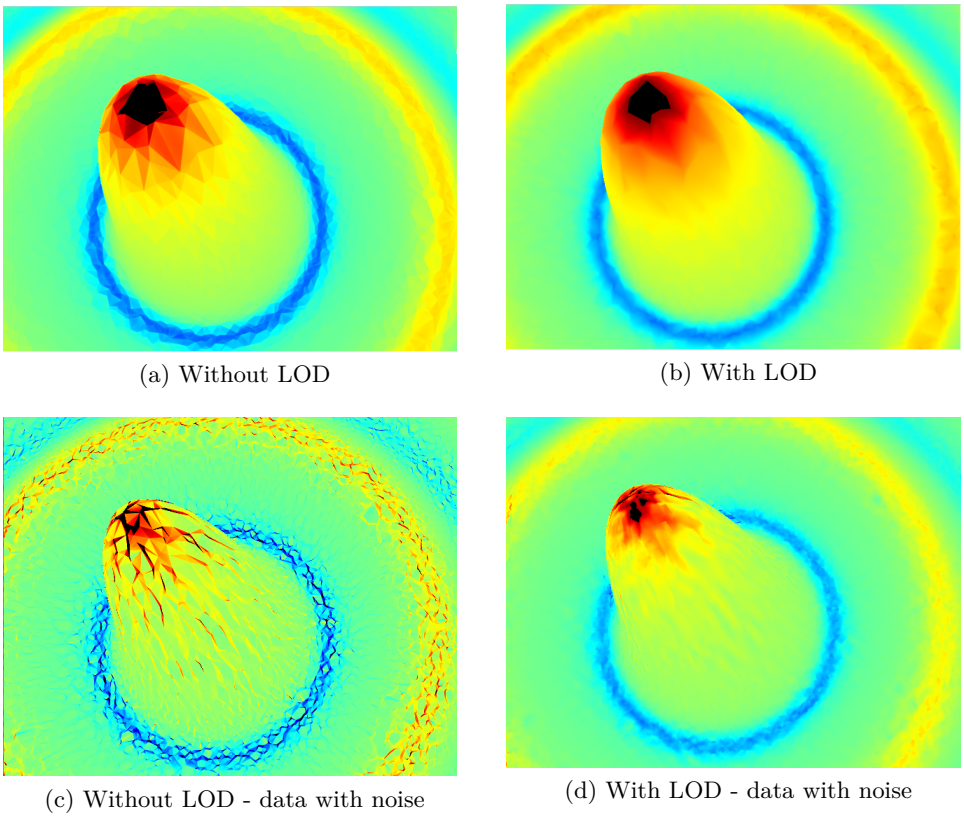
To solve this problem, level of detail (LOD) sampling can be used. For points closer to the camera, triangles are constructed from a wider neighborhood. Our solution is

based on a power function and the final equation is:

$$size = size_{max} \left(\frac{1}{f^2} \right)^d + 1, \tag{7.12}$$

where $size_{max}$ is the maximal size of the neighborhood, f is the distance of the camera far clip plane (in our tests, this value was always set to be $f \geq 100$, smaller values were clamped to this interval) and d is a current pixel depth in interval $\langle 0, 1 \rangle$ where 0 is for the closest points to the camera. The computed $size$ represents the step between pixels that should be converted to integer by omitting the fractional part. However, this could lead to $size = 0$. For this reason, there is the $+1$ term in Equation (7.12). The value of $size_{max}$ can be achieved only for $d = 0$, but this value is very rare in the depth buffer.

Using this approach, the final curvature should be computed from more than one triangle. According to our observations, a maximal number of four triangles for one pixel, creating a triangle fan, is sufficient. The final curvature is calculated as an average value from all triangles. The result with LOD for the same model can be seen in Figure 7.2. There are used two different samplings. In Figures 7.2a and 7.2b, sampling is accurate with exact normals computed directly from the function equation. In Figures 7.2c and 7.2d, the sampling contains noise and normals are computed from mesh geometry using the algorithm from [Max99].



(a) Without LOD

(b) With LOD

(c) Without LOD - data with noise

(d) With LOD - data with noise

Figure 7.2: Screen space curvature

The problem with LOD are discontinuities between neighboring pixels. If the neighborhood has the size of one pixel, they are not very visible and are often not recognizable

during movement. However, with an increased step size, the problem is more serious. We have used the simplest solution with the condition

$$|d - d_{neighbor}| > \frac{1}{f}, \tag{7.13}$$

where d is the depth of the current pixel, $d_{neighbor}$ is the depth of the neighbor with the step $size$ (see Equation (7.12)) and f is the camera far clip plane. If the condition is met, e.g., there is a depth discontinuity, the LOD for the current neighbor is disabled and step is set to $size = 1$.

7.3 Experiments and results

To test the proposed method, a PC with the following configuration was used: Intel Core i7 CPU running at 4GHz, 32GB of RAM memory, NVidia Geforce 960GTX graphics card with 2GB of video memory. The algorithm was implemented in C++ and OpenGL 4.4 with GLSL shaders.

The implementation of the algorithm by [Mel+13], based on [Mel15], has been done using GLSL instead of CUDA used in the original paper.

7.3.1 Curvature error

In this section, comparison of the proposed method for triangle meshes, as defined in Section 7.1.1, and exactly computed curvature from analytic surfaces are provided. Every test uses exact unit-length normals computed from the function itself. In the comparisons, two and three edges were used to create overdetermined system.

The proposed method on the triangle mesh has been also tested against the Bézier triangles algorithm from [Zhi+11].

First, a sphere was tested. A sphere has a constant mean and Gaussian curvature, dependent on the sphere radius r. Curvatures on the sphere can be calculated as $K_H = \frac{1}{r^2}$ and $K_G = -\frac{1}{r}$. As a discrete representation of the sphere, a subdivided (with a step 6) icosahedron sphere with an exact normal and radius 6 was used. The proposed method in both variations has a constant mean square error (MSE) with the value 8.2×10^{-16} for Gaussian and 7.8×10^{-17} for mean curvature. For different radii, MSE has a similar behavior.

Next, two analytic functions were tested (see Figure 7.3). The function f_1 has convex and concave parts, a high peak at its center, and it is undefined at the point $[0, 0]$ (at this point, division by zero occurs). Function f_2 has a saddle shape with minor bumps.

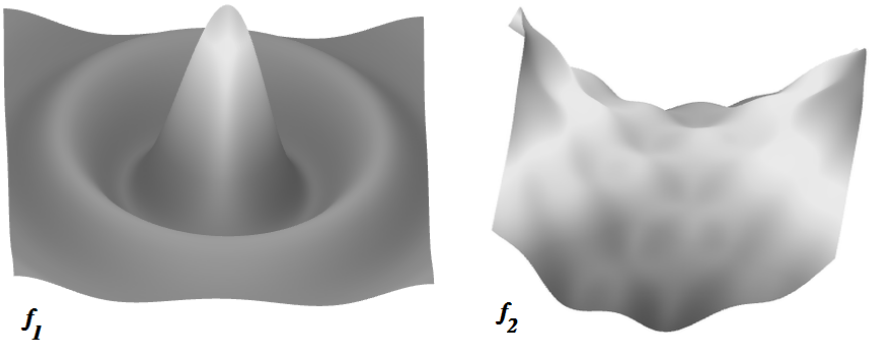
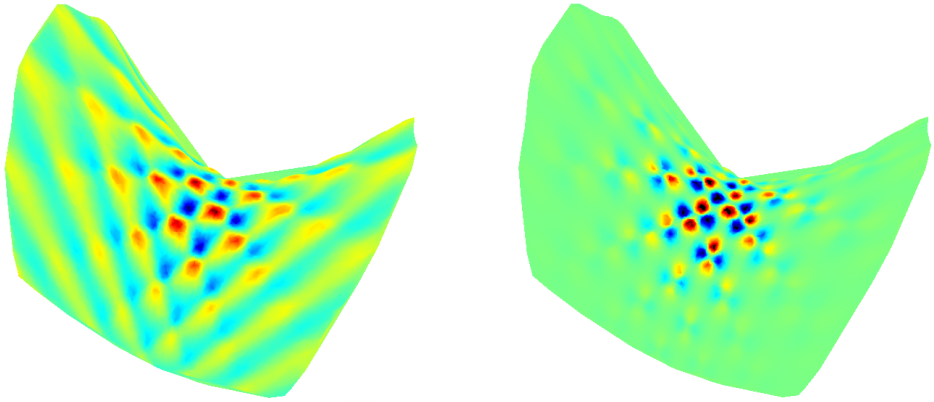


Figure 7.3: Tested functions $f_1 = 10\frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2)}}$, $(x \neq 0), (y \neq 0)$, $f_2 = sin(x)cos(y) + 0.1(x^2 - y^2)$, $x, y \in < -10; 10 >$

To test the proposed algorithm, the functions have been tessellated with Delaunay triangulation in the XY plane using different random point clouds in the interval $[-10, 10]$ in both directions. MSE value gives the error of the proposed method on the triangle mesh in comparison with the analytically computed curvature from the input function. See Figure 7.4 for the result of the curvature for the function f_2.



(a) Mean curvature

(b) Gaussian curvature

Figure 7.4: Curvatures of the function f_2 calculated from the triangle mesh

Result of the comparison is in Figure 7.5. Small peaks in the graph are caused by random distribution of vertices in the underlying triangulation. This is more visible for f_1 due to its peak around the point $[0, 0]$. For more dense tessellation, there is a very small difference in using two or three edges of the triangle to solve the system. In some cases, two edges offer better results, while in other scenarios, three edges are marginally better.

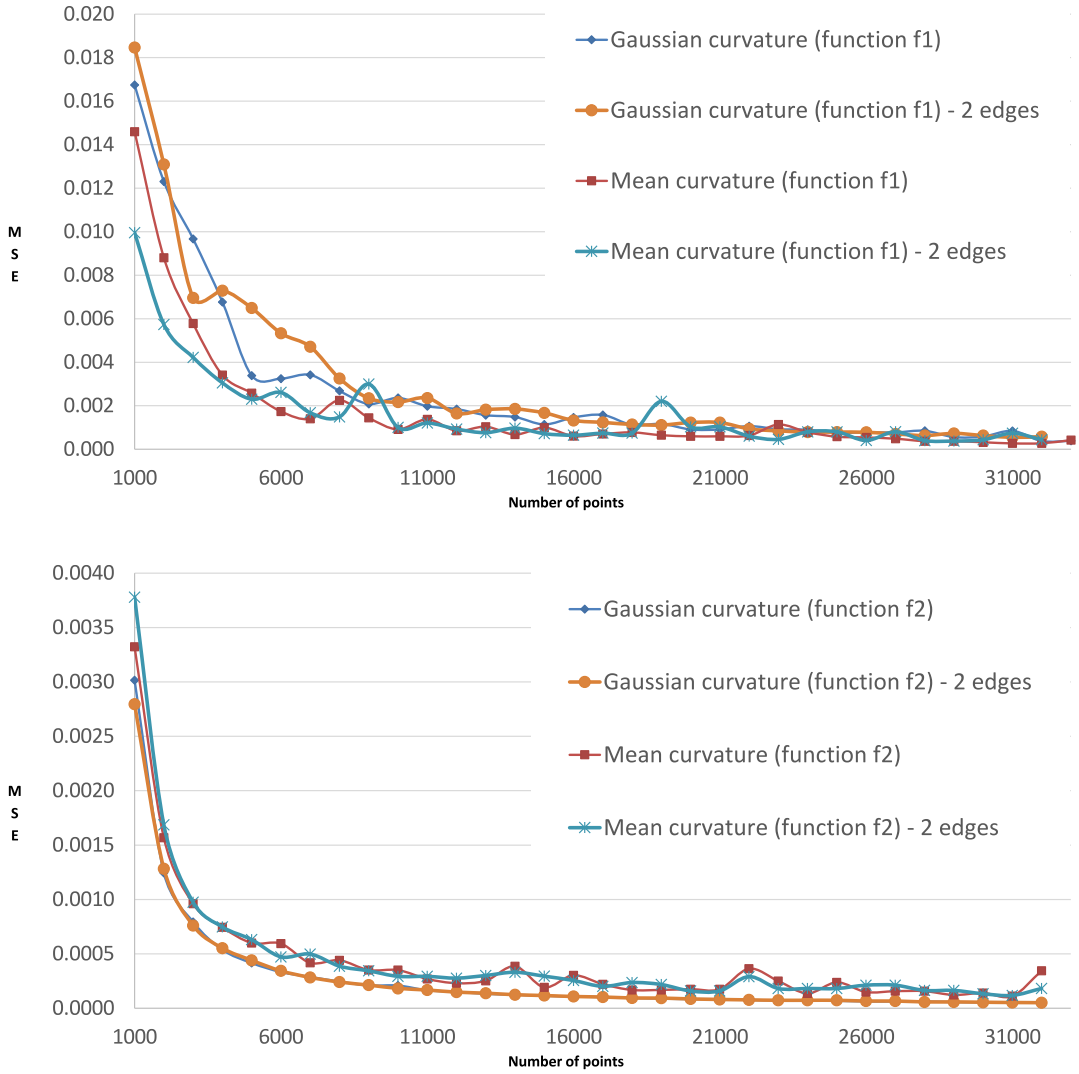
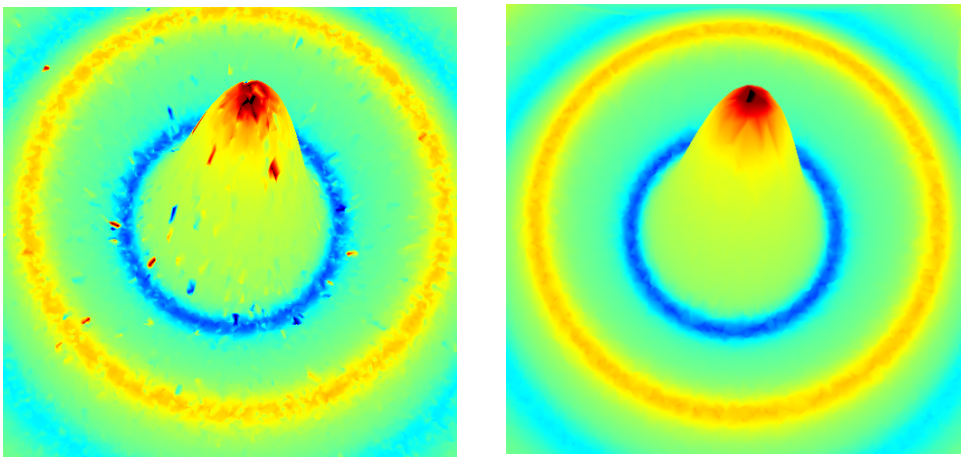


Figure 7.5: MSE of the method for the triangle mesh compared to the analytically computed curvature evaluated directly from the implicit function

Another comparison of the proposed method was done against the algorithm [Zhi+11] using Bézier triangles. This algorithm was chosen according to the promising results in tests and experiments published in the original paper. However, the algorithm has worse quality on the triangle meshes created from the random points (the same random grid has been used for both tests). The result can be seen in Figure 7.6. MSE values for individual triangles were varying from 0.5 to almost 40. For most of the triangles, the calculated curvature gives us the error comparable with our proposed method. However, there were large error values present in results from [Zhi+11], caused by small or sliver triangles. This is because the Bézier triangles in [Zhi+11], constructed from those small or sliver triangles, are too arched. This problem is not present in the proposed method.



(a) Bézier curvature [Zhi+11] (b) Proposed method

Figure 7.6: The comparison of the curvature of f_1, compared on a tessellation created from the random point cloud.

7.3.2 Screen space comparison

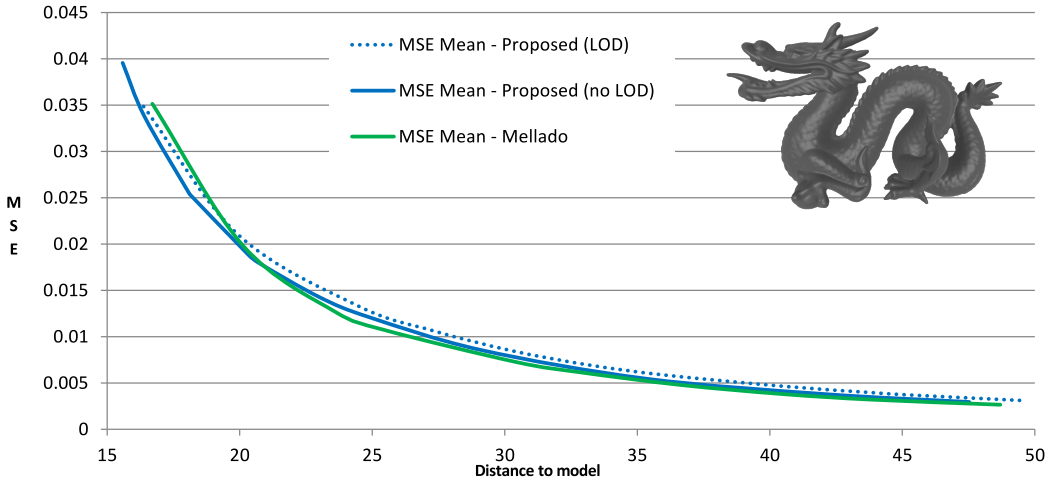
The comparison of the screen space method with and without the LOD active is done against the curvature calculated by the proposed method directly on the triangle mesh. Curvatures inside triangles are linearly interpolated from curvatures at triangle vertices. The proposed algorithm was also compared with [Mel+13], the only other screen space technique known to us.

The tested models are shown in Table 7.1. In the screen space, the quality of the computed curvature depends on the camera distance from the model. If we compute the curvature for the triangle mesh and render the result, with the camera moving away from the model, the triangles become smaller and more triangles can be rendered in the same pixel. This can cause an incorrect curvature to be visualized. In the proposed screen space method the problem associated with rasterization cannot happen because only visible parts are used to calculate the result and only one value is used for the final pixel. In every test, the model was tested as fully visible on the screen and the camera was moving away from the model. The dependency of MSE on the distance between the viewer and the model is shown in the following graphs in Figures 7.7 and 7.8.

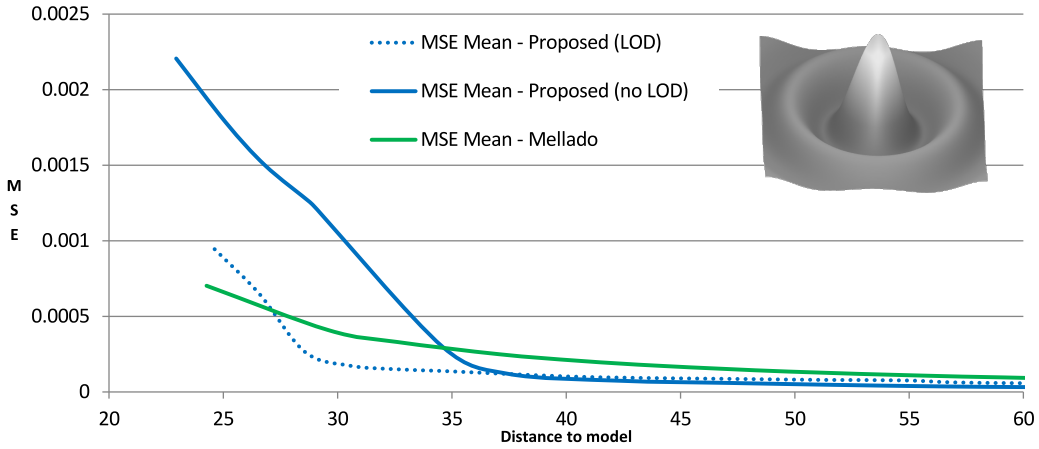
	Vertex count
Stanford Dragon	300 000
MaxPlanck	152 403
Function f_1	15 000
Torus	1 000

Table 7.1: Tested models

From the graphs in Figure 7.7 it can be seen that the quality of both screen space algorithms is comparable for the mean curvature. For the dragon model, using LOD has a little or no effect at all. The original model has a dense tessellation and LOD can skip fine details. On the other hand, for the model of the function, the proposed method with LOD achieves better quality.



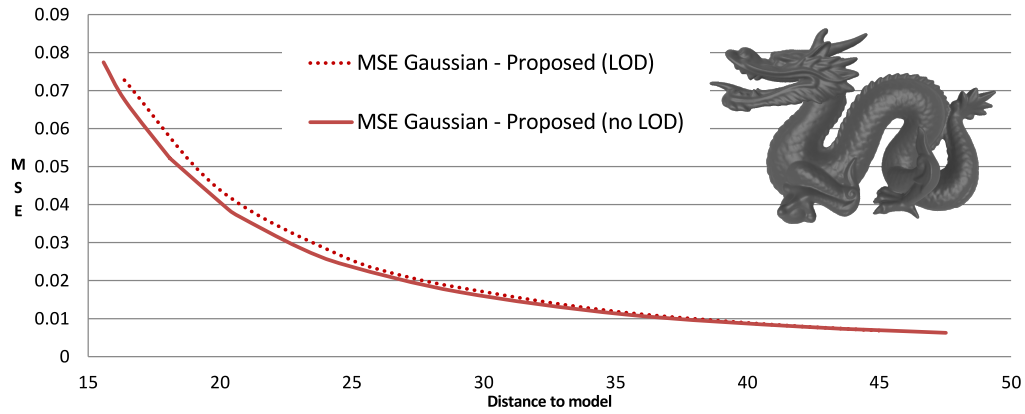
(a) Stanford Dragon



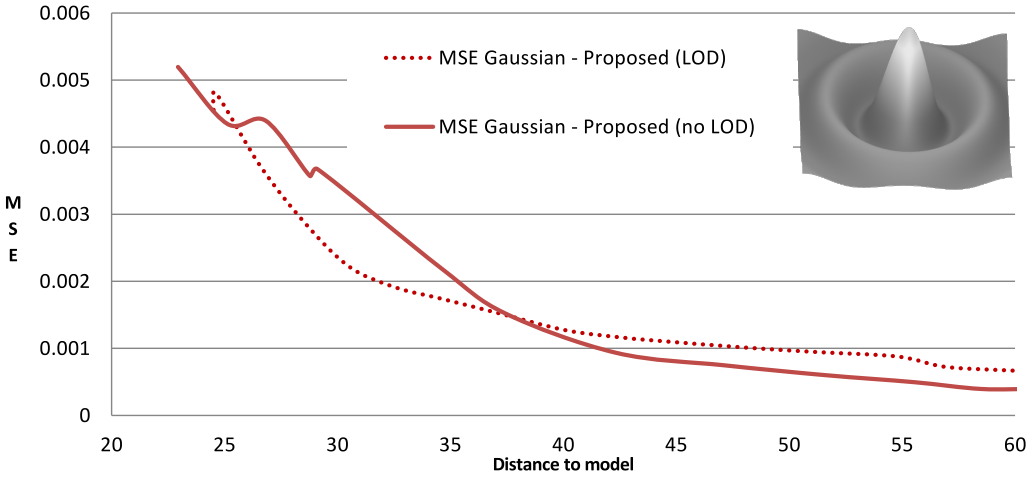
(b) Function f_1

Figure 7.7: Comparison of the screen space MSE for the mean curvature calculated directly from the triangle mesh. The proposed method with and without LOD and algorithm from [Mel+13] (Mellado) were tested.

The Gaussian curvature comparison was done only with and without LOD, since there is no other screen space method known to us that calculates the Gaussian curvature. See results in Figure 7.8. The behavior is similar to Figure 7.7, with a roughly doubled amount of the MSE error. This is caused by the curvature calculation, where the mean curvature is only a sum of the principal ones, while the Gaussian is computed by multiplying principal curvatures. In that case, the errors of both values are multiplied as well.



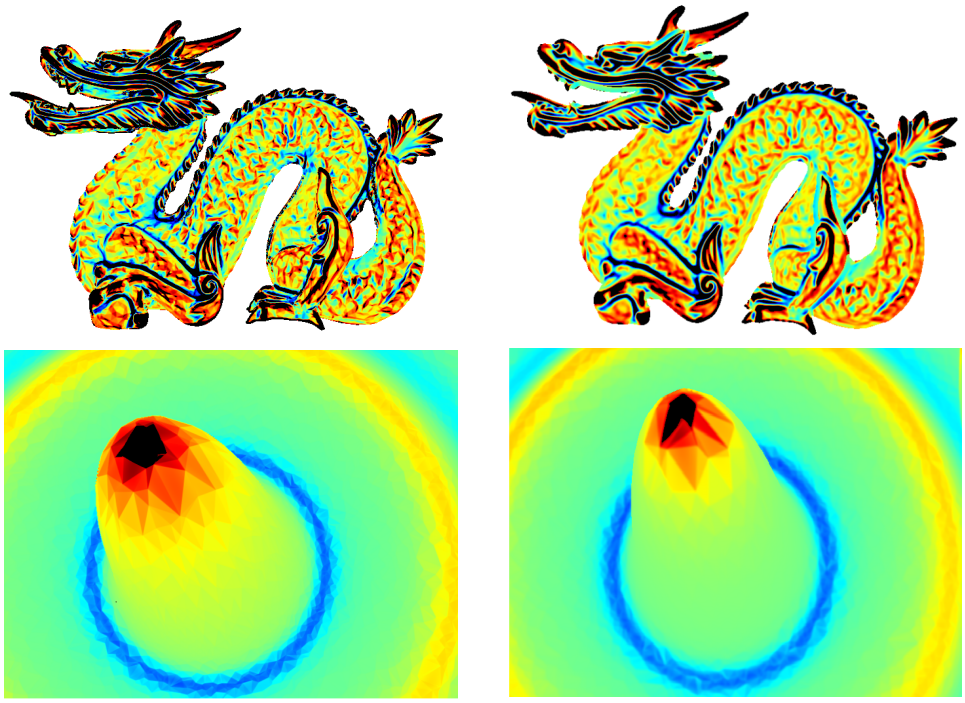
(a) Stanford Dragon



(b) Function f_1

Figure 7.8: Comparison of the proposed screen space MSE for the Gauss curvature calculated directly from the triangle mesh.

The visual comparison of the proposed method with [Mel+13] can be seen in Figure 7.9. Both algorithms have a comparable visual quality. The proposed method results look sharper, [Mel+13] is more blurry.



(a) Proposed method

(b) Algorithm from [Mel+13]

Figure 7.9: Comparison of the mean curvature. Because [Mel+13] has no LOD, the presented comparison also uses none to create comparable images.

No LOD is used to show real differences based on the camera distance. For the camera at a greater distance (full model), there is almost no visible difference. With the camera closer to the surface (detailed parts of the image), the triangles of the mesh begin to appear in the screen space curvature.

For visual comparison of the quality of the proposed method in the screen space against the same method in the object space see Figures 7.11 and 7.10.

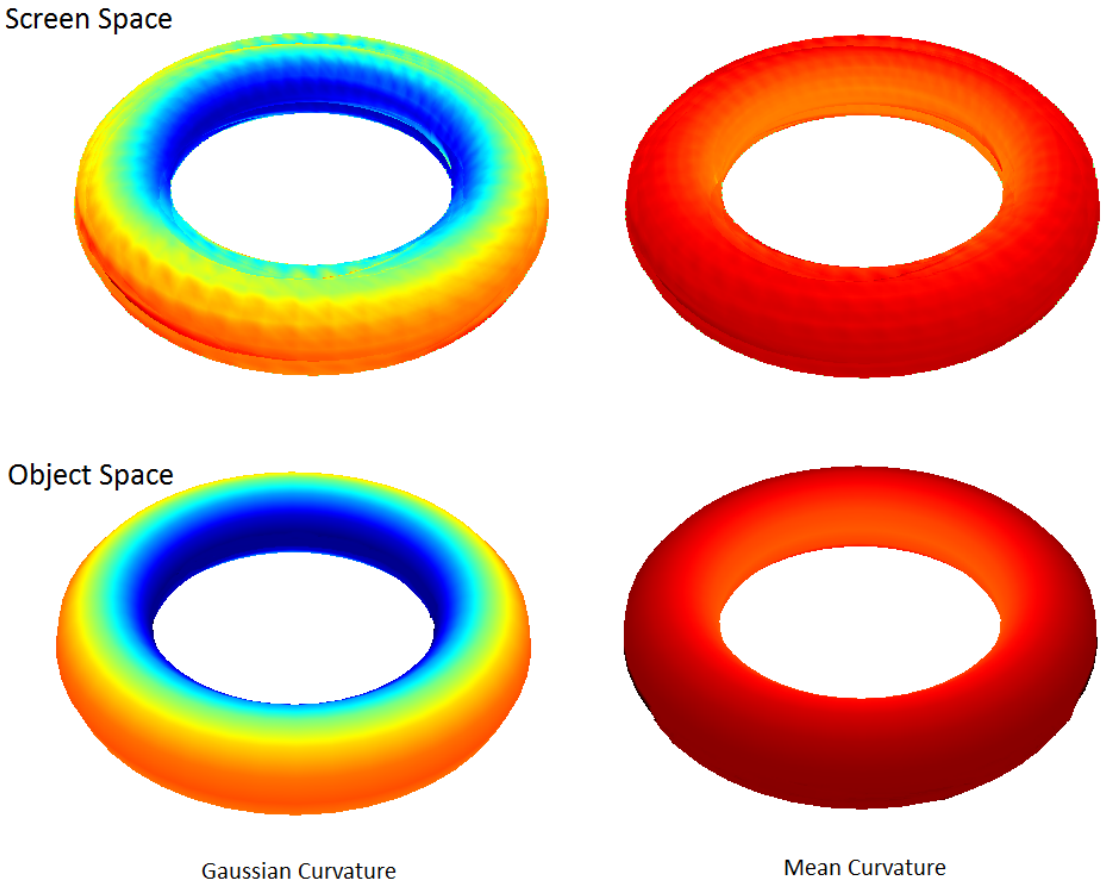
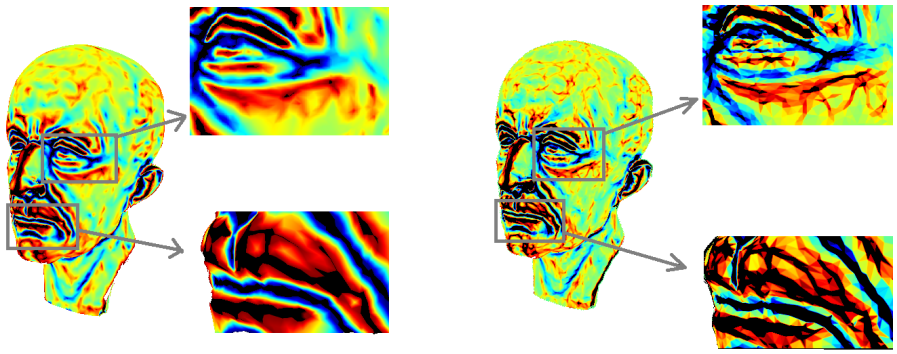


Figure 7.10: Summary comparison of mean and Gaussian curvature computed in the screen space with our solution and ground truth data

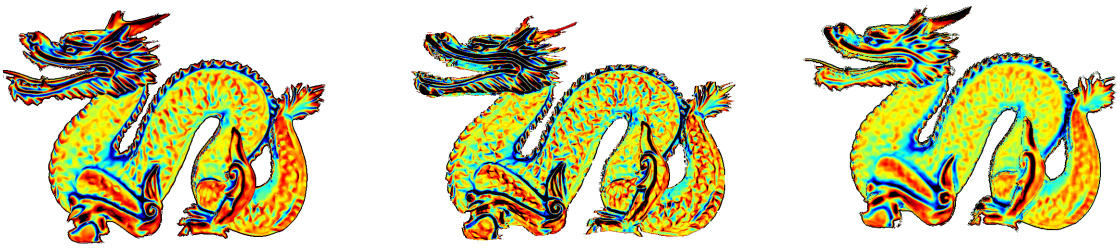


(a) Object space (b) Screen space without LOD

Figure 7.11: Comparison of the mean curvature for the MaxPlanck model.

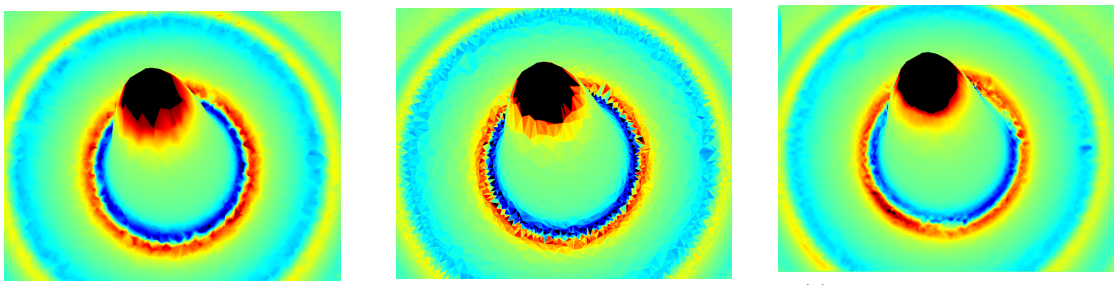
The effect of the used LOD can be seen in Figures 7.12 - 7.14. If the camera is moving away from the mesh, there is a distance, from which further there is a small or no difference between using and not using LOD. In some cases, using LOD can bring worse results as it smooths out fine details (see Figure 7.12). On the other hand, in

the example of the Gaussian curvature in Figure 7.13, the use of LOD improved the result considerably. Another comparison can be seen in the closeup in Figure 7.14. If the camera moves very close to the surface, LOD is required to obtain a smooth result. Without LOD, the computed curvature appears as random colors. In some cases, e.g., in wireframe view, this visualization can be sometimes enough to see the shape. To set a suitable distance for LOD is, however, difficult - the same value does not work for all models.



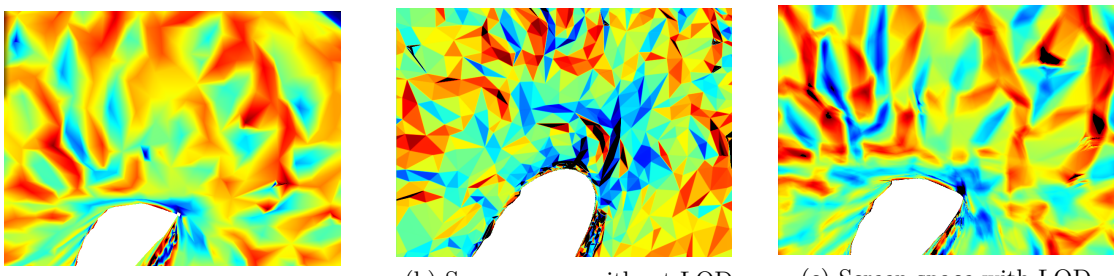
(a) Object space (b) Screen space without LOD (c) Screen space with LOD

Figure 7.12: Comparison of the mean curvature.



(a) Object space (b) Screen space without LOD (c) Screen space with LOD

Figure 7.13: Comparison of Gaussian curvature using function f_1.



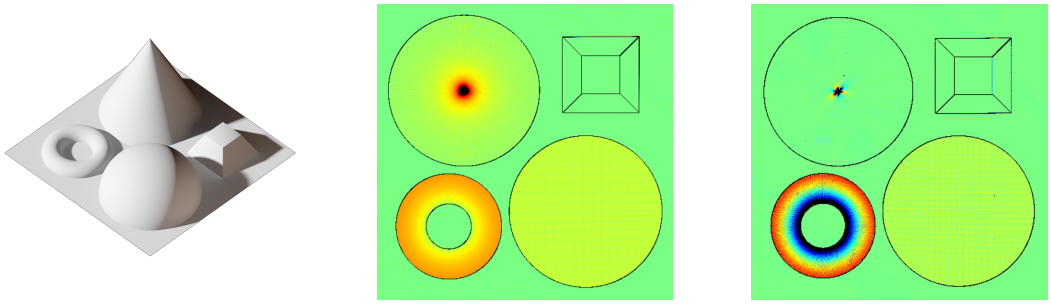
(a) Object space (b) Screen space without LOD (c) Screen space with LOD

Figure 7.14: Detail of the mean curvature

The advantage of curvature estimated directly in the screen space is the possibility to reduce geometry and use normal mapping to add missing details. Our proposed algorithm

can be used together with this approach. We have used a plane with details added from a normal map. Since we used fine details, LOD should be disabled in this case. The results can be seen in Figures 7.15b and 7.15c. They are screenshots of a flat plane as seen from above. The view from sides would result only in a plane with no geometry.

We have used a standard normal mapping test texture that contains a torus, a sphere, a cone and a pyramid. The original test scene with geometry can be seen in Figure 7.15a. The mean curvature (Figure 7.15b) has a lower noise and therefore a higher quality than the Gaussian curvature (Figure 7.15c). This corresponds to the visual quality of tests in Figures 7.12 and 7.13. The resulting curvatures correspond to the curvatures of real objects even if there is no position (we have a flat plane), only a normal vector obtained from a normal map. We have also tested the version with an additional displacement (bump) map, but the results were almost identical.



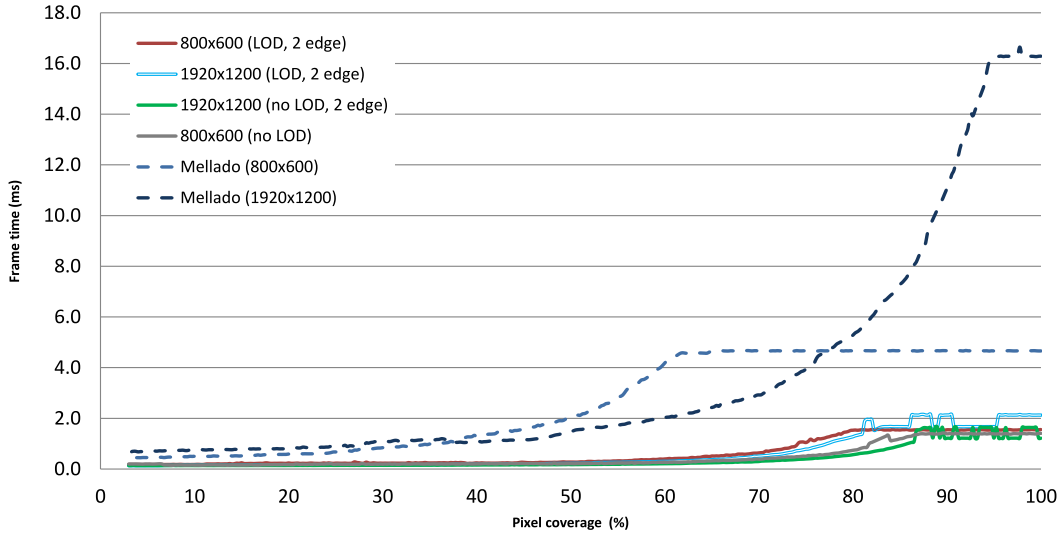
(a) Test scene (b) Mean curvature (c) Gaussian curvature

Figure 7.15: Mean and Gaussian curvature for normal mapped plane. Black borders are caused by a high curvature that is outside the used mapping function.

7.3.3 Performance Evaluation

The proposed method runs at interactive frame rates. Due to the independence of the geometry, all tested models brought nearly the same results. In the tests, the method was computed for pixel coverage of $2-100$ percent of the screen. The depth value of the remaining pixels was set to infinity to discard these pixels. The comparison was done against the screen space method from [Mel+13].

The resulting performance can be seen in Figure 7.16. In all tests, a decrease of performance is partially caused by LOD computation but mostly by the need of branches in the pixel shader to decide if the triangle can be used or will be rejected as described in Section 7.2.1. With a comparable visual quality as [Mel+13] (no LOD used), the proposed algorithm is much faster. The noisy peaks around $80-100$ percent of the screen coverage in Figure 7.16a are caused by camera movements and some possible context switching during measurements due to the graphics driver because the frame time is very low.



(a) Proposed solution

Figure 7.16: Frame time based on screen pixel coverage. The proposed algorithm in versions with only two edges (with and without LOD) against [Mel+13] (Mellado) was tested.

7.3.4 Limitations

Similarly to other screen space techniques, the proposed algorithm has its disadvantages. When two neighboring pixels do not come from the same part of the surface, there appears a surface discontinuity between those pixels and an artifact in the computed curvature may appear. We have proposed one possible solution in Section 7.2.1, but it is not guaranteed to work in every situation. If the depth difference is small and pixels belong to different surfaces, the problem will persist.

Another problem is related to LOD. The estimated curvature depends on the distance of the mesh from the camera, where small details are smoothed if the camera is far away from the surface. The setting of the correct LOD can improve the curvature estimation quality.

In the proposed solution, the LOD comes with a performance lost. Usually, LOD is included to increase the performance by using less samples or to simplify computations. In the proposed solution, the LOD version is less efficient due to the need of sampling more pixels than for a simple neighborhood of size 1.

7.4 Ambient occlusion

Since we already calculated curvature in the screen space, we can use the results for different effects. One of them can be an estimation of ambient occlusion. Ambient occlusion is a shading technique used to calculate the exposition of a point to an ambient light. It is a global method, unlike the well-known local Phong shading, and must be computed as a function of the geometry of the entire scene. Using curvature for ambient

occlusion is not very common and have a disadvantage in missing occlusions from non-connected geometry. However, if curvature of objects is already estimated, it can be used as the first estimation of an ambient occlusion and later, if necessary, the quality can be improved with a traditional approach.

Using curvature for ambient occlusion was already proposed by [Gri+12; HKM11; HKM12]. However, there is a problem with both existing solutions. Neither of them consider convex and concave areas. In both solutions, the sign of curvature is mostly suppressed by the use of quadratic power. This leads to ignoring convexity and concavity, where convex areas should be dark, while concave areas are usually fully lit by light. They use a sphere around a single point to calculate occlusion, but the selection of the radius is difficult to set.

We have modified our screen space curvature algorithm and added the possibility to estimate ambient occlusion as well. This research was presented in [PVK17]. We propose a new function to map curvature to the Ambient occlusion. In the proposed solution, the mean curvature is used. The curvature has to be mapped to the symmetrical interval $\langle -1, 1\rangle$, where 0 is zero curvature. For this, we need the curvature extreme for the mapping. However, this is similar to the need of scaling factor in [Gri+12]. We know that convex areas are usually darker than concave. We have created a statistics-based function that maps the mean curvature to the occlusion based on a threshold. This function consists of two separate parts. One for convex and one for concave areas.

For values below zero (convex areas), we use the Gaussian function

$$AO = a \cdot exp\left(-\frac{(m-\mu)^2}{2\delta^2}\right), \tag{7.14}$$

where m is the negative part of the normalized mean curvature from the interval $\langle -1, 1\rangle$. μ is the expected value and δ^2 is the variance. We have set those two parameters to $\mu = 0$ and $\delta^2 = 0.2$ to get a normalized function centered around zero. The parameter a sets the maximal occlusion value. We use $a = 0.9$.

Values above zero (concave areas) should be lit with a maximal amount of light and therefore $AO = 1$ can be used. However, in some cases, we want a slight occlusion even in these areas to create a smoother transition. For that reason, we use a linear mapping of the interval $\langle 0, 1\rangle$ (positive part of the normalized mean curvature) to the final interval $\langle a, 1\rangle$.

7.4.1 Results

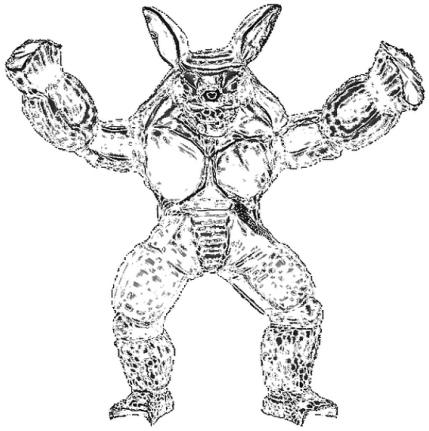
The proposed algorithm can be used to estimate the ambient occlusion in combination with existing techniques from Hattori et al. [HKM11] and Griffin et al. [Gri+12]. Both algorithms can be used with a precomputed curvature, but in the proposed solution, the curvature is estimated directly in the screen space. The comparison of the two methods against occlusion calculated using ray-casting can be seen in Figure 7.17. Both results were computed without LOD because the underlying mesh is of a high quality.

Hattori et al. [HKM11] use the sphere of radius 0.25. We have tested different radii, but the results were too dark or too bright and the occlusion effect was hard to perceive. Solution from Griffin at al. [Gri+12] offers a better visual quality. The curvatures are

scaled up with factor of 5. Different scales result again in a darker or lighter effect, which is similar to Hattori's solution. The problem with the Griffin algorithm are non-white areas with zero occlusion.

In our proposed solution (see Figure 7.17d), the result is not as dark as [Gri+12] and looks more like ray-casted result. On the other hand, solution from [HKM11] keeps correct white areas, but the rest of the model is too dark. We are using the threshold a from Equation 7.14 set to $a = 0.9$. If we set this threshold to $a = 1$, we can obtain white areas as well, however, some details are lost. From our point of view, the configuration we have used offers the best visual appearance.

(a) Ray-casted occlusion



(b) Hattori et al. [HKM11]

(c) Griffin et al. [Gri+12]

(d) Proposed

Figure 7.17: Ambient occlusion estimated using the proposed curvature algorithm

7.4.2 Limitations

Most of the limitations is shared with screen space curvature from subsection 7.3.4.

The main limitation for ambient occlusion is the same as for other curvature based solutions - the inability to calculate occlusion from non-connected parts of the geometry. See Figure 7.18. If we calculate occlusion directly, using ray-casting (rays r_1, r_2, r_3...)

within a half-sphere, there should be an occlusion from ray r_3. However, the curvature at the point P is zero and, therefore, no occlusion will be calculated.

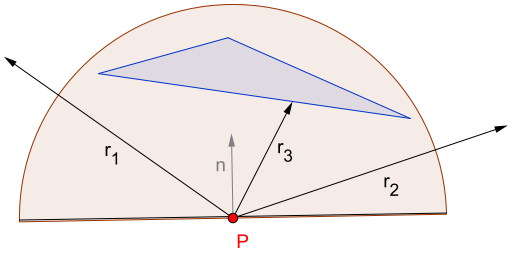


Figure 7.18: Curvature based ambient occlusion limitation

Chapter 8

Estimation of differential quantities using Hermite RBF interpolation

The curvature estimated by algorithm proposed in the previous Chapter 7 is intended only as a fast estimation for the price of a lower estimation quality. However, many times we need precise estimation results close to the exact values. However, the exact curvature can only be calculated for a limited set of surface descriptions. Most of the time, we deal with triangles, point sets or some other discrete representation of the surface. For those, curvature can only be estimated. Luckily, surfaces can be fitted by some kind of interpolation function and from it, curvature can be calculated directly.

To find an ideal algorithm, which produces the best possible results, is a challenging task. In our research [Váš+16], we have reviewed several curvature algorithms. Our main goal was to improve quality of estimation by combining algorithms together because some of them are better for certain scenarios or certain data configurations. Based on statistics of the neighborhood, an algorithm that offers the best result for this kind of setup was used for this neighborhood.

During the research, we have gathered several large datasets of artificially created data (mostly CAD models). These datasets were in a form of triangle meshes or point clouds equipped with normal vectors in both cases, but with no additional information. There is no algorithm directly targeted for this kind of data. We have tested the existing ones, but the results were not satisfying with respect to the data.

Our proposed solution [PV18] is a method for curvature estimation and normal vector re-estimation based on surface fitting using Hermite Radial Basis Function interpolation (HRBF). Hermite variation uses not only control points, but normal vectors at those points as well. This leads to a better and more robust interpolation than if only control points are used. Once the interpolant is obtained, the curvature and other possible properties can be directly computed using known approaches. Unlike our Screen-space-based algorithm from Chapter 7, this solution is designed for precision and high-quality of results instead of speed.

The proposed algorithm can be used for point clouds and triangle meshes as well. It was tested on several explicit and implicit functions and it outperforms current state-of-

the-art methods if exact normals are available. For normals calculated directly from a geometry, the proposed algorithm works on par with existing state-of-the-art methods.

8.1 Basic algorithm

The proposed algorithm uses HRBF (see Section 3.2.2) to construct local interpolants that represent the shape in the vicinity of each single point. These interpolants are then used for the curvature estimation at each point. The algorithm can run in parallel without the need for synchronization primitives, since each point is processed individually. Consequently the computation time depends linearly on the number of points, since no global computation is performed.

For a given point P the neighboring points have to be found. Based on the geometry representation, they can be obtained from Euclidean neighborhood of the point P or from the k-ring neighborhood in case of triangle meshes. The obtained neighboring points, together with the normal vectors are used as input values for the HRBF interpolation (recall Equations (3.9) and (3.11)).

8.1.1 Basis function selection

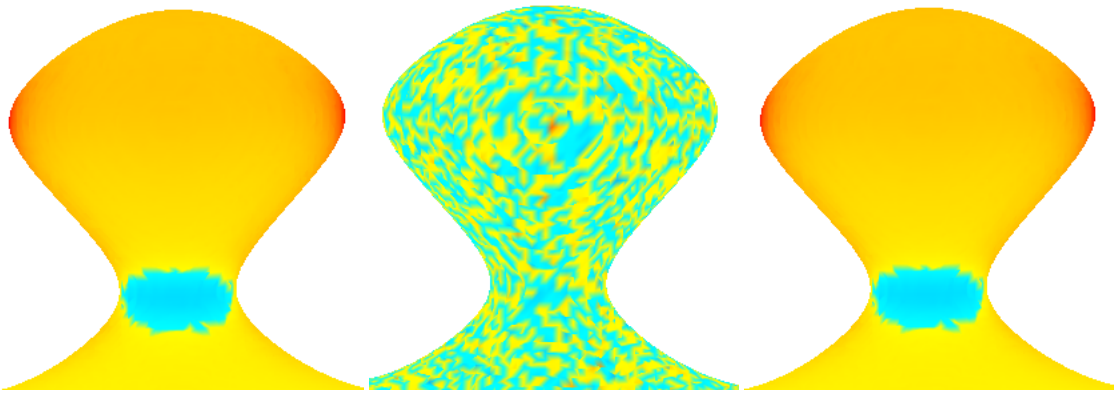
In the proposed algorithm, two functions have been selected based on the performed tests:

- The polyharmonic spline r^3 (also known as tri-harmonic spline) is usually preferred. This basis function is utilized in many RBF interpolation schemes. It is globally supported with C^2 smoothness, leads to a dense symmetric linear system and also works for a highly irregular sampling. Another advantage is its scale independence.

- A non-standard $exp(-\varepsilon r^3)$ is used in some experiments as well. Based on our tests, in some cases this function leads to more precise curvature estimation and it is also more suitable for noisy data or inaccurate normals. The only problem is the shape parameter ε. A constant value can lead to inaccurate results and the loss of the scale independence. In the proposed solution, a dynamic solution based on edge lengths in the neighborhood of a center vertex V is used. The shape parameter is calculated as

  $$\varepsilon = \bar{l}/max^3,$$

  where $\bar{l}$ is an average edge length within the neighborhood and max is the maximal size of the axis-aligned bounding box (AABB). AABB is fast to compute, but this solution is not rotation invariant. The user can manually change the object orientation to be axis-aligned before computing AABB. However, to overcome the need of user input, one can use an approximation of object-oriented bounding box (OBB), for details see [BHP01]. The cubic power for max is used because of the same power in parameter r^3. The difference of the constant vs. dynamic selection of the shape parameter can be seen in Figure 8.1.



(a) Exact curvature (b) Static $\varepsilon = 0.001$ (c) Dynamic ε

Figure 8.1: Comparison of shape parameters

These two basis functions are the most universal and provide reasonable results without any further data knowledge.

8.1.2 The curvature calculation

The principal curvatures are calculated from Equations in Section (4.3.1). For this task, we need to obtain gradient of function $\boldsymbol{\nabla F}$ (first-order derivative) and components of Hessian matrix H (second-order partial derivatives of scalar field) from HRBF implicit function $f(\boldsymbol{x})$.

The first-order derivative (gradient) of $f(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, x_2 ... x_D)$, $\boldsymbol{x} \in \mathbb{R}^D$ is evaluated as

$$\begin{aligned} \nabla f(\boldsymbol{x}) = \quad & \textstyle\sum_{i=1}^{N} \alpha_i \left[\phi'(r_i)\nabla r_i\right] + \boldsymbol{\beta}_i \frac{1}{r_i} \left[\phi'(r_i)\right] + \\ & \left[\boldsymbol{\beta}_i \cdot (\boldsymbol{x} - P_i)\right](\boldsymbol{x} - P_i) \frac{1}{r_i^2} \left[\phi''(r_i) - \frac{\phi'(r_i)}{r_i}\right], \end{aligned} \tag{8.1}$$

where N is the size of neighborhood of the evaluated point, $r_i = ||\boldsymbol{x} - P_i||$, $\phi'(r_i)$ is the first- and $\phi''(r_i)$ is second-order derivative of basis function ϕ, $\alpha_i \in \mathbb{R}$ are the weights for points and $\boldsymbol{\beta}_i \in \mathbb{R}^D$ are the weights for normal vector.

The second-order partial derivative of a scalar field $f(\boldsymbol{x})$ is the Hessian matrix

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \frac{\partial^2 f}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix}. \tag{8.2}$$

Computing this matrix using HRBF, each line can be evaluated similarly to the gradient. For example,

$$\frac{\partial^2 f}{\partial x_1 \partial x_3} = \sum_{i=1}^{N} \left(\frac{\partial^2 f}{\partial x_1 \partial x_3} \right)_i . \tag{8.3}$$

For one particular value of the index $i = 1$, the computation is as follows :

$$\begin{aligned} \left(\tfrac{\partial^2 f}{\partial x_1 \partial x_3}\right)_1 = \quad & \alpha_1 \left[\phi''(r_1) \tfrac{\partial r_1}{\partial x_3} \tfrac{\partial r_1}{\partial x_1} + \phi'(r_1) \tfrac{\partial^2 r_1}{\partial x_1 \partial x_3} \right] + \\ & \boldsymbol{\beta}_1 \tfrac{1}{r_1^2} \left[\phi''(r_1) \tfrac{\partial r_1}{\partial x_3} r_1 - \phi'(r_1) \tfrac{\partial r_1}{\partial x_3} \right] + \\ & \left(\boldsymbol{\beta}_1 \cdot \tfrac{\partial \boldsymbol{d}}{\partial x_3} \right) A + (\boldsymbol{\beta}_1 \cdot \boldsymbol{d}) A' \end{aligned} \tag{8.4}$$

where

$$\boldsymbol{d} = \boldsymbol{x} - P_1,$$

$$\frac{\partial \boldsymbol{d}}{\partial z} = (0, 0, 1),$$

$$A = \frac{\boldsymbol{d}}{r_1^2} C,$$

$$A' = \frac{1}{r_1^4} \left[\frac{\partial \boldsymbol{d}}{\partial x_3} C r_1^2 + \boldsymbol{d} C' r_1^2 - \boldsymbol{d} C \frac{\partial r_1^2}{\partial x_3} \right]$$

and

$$C = \left[\phi''(r_1) - \frac{\phi'(r_1)}{r_1} \right],$$

$$C' = \phi'''(r_1) \frac{\partial r_1}{\partial x_3} - \frac{1}{r_1^2} \left[\phi''(r_1) \frac{\partial r_1}{\partial x_3} r_1 - \phi'(r_1) \frac{\partial r_1}{\partial x_3} \right].$$

To express the other terms of the Hessian matrix H (Equation (8.2)), the computations are similar, only with different partial derivatives.

8.1.3 Data quality

To compute HRBF, we need positions and normal vectors at them. There can be two types of data at the input of our algorithm.

Exact data

In this case, the input data are noise-free and equipped with exact normals. For this ideal scenario, the curvature estimation is obtained directly from the derivatives at the center point P. They are calculated from the HRBF interpolation using Equations (8.1) and (8.2).

Approximated normals and noise

In the more common case of imprecise normals, that are estimated from input discrete data, the curvature calculated directly at the center point P offers poor results, since the HRBF interpolant reflects the imperfections of the input data. Every input value is located on the surface and normal vectors at these points match those of the input. Around these points, the surface is strongly influenced by the noise.

To overcome this problem, we take several samples from the interpolant around the center point and average them. We have tested uniformly distributed random points and points generated by Poisson disc sampling (using algorithm of [Bri07]). It produces points that are tightly-packed, but never closer one to another than a specified minimum distance (see Figure 8.2). Based on our experiments, the Poisson disc sampling offers better results than the uniform distribution.

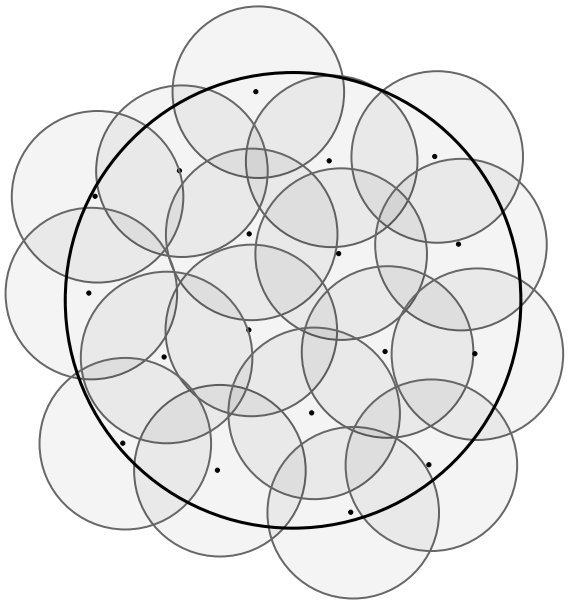


Figure 8.2: Example of Poisson disc sampling

A set of points C_i is randomly generated using a unit disc. This same distribution is used for every point. The minimal distance between samples influences the total number of points in the disc. This distance can be selected from the interval $\langle 0, 1 \rangle$. The experiments with different distances can be seen in Figure 8.3. The good choice which we have used is 0.3 (this corresponds to approximately 30 points). Larger minimum distances produce worse results, while the number of generated points is smaller and thus the estimation runs faster.

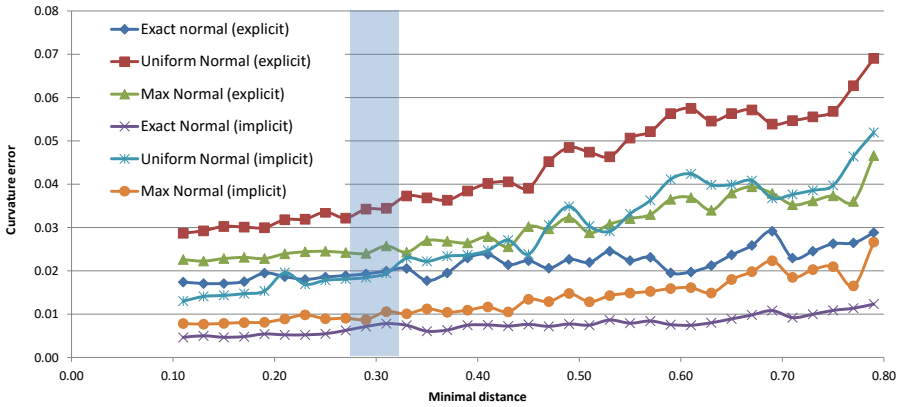


Figure 8.3: Poisson disc sampling and absolute curvature error. Tested with exact, uniform and Max [Max99] normals for explicit and implicit input data. The recommended selection of distance is highlighted.

The radius of the disc is scaled to be equal to the average edge length of the entire geometry. The sampling points C_i are created in a plane which is then mapped onto the tangent plane of each point P, creating mapped points C_{ip}, at which the curvature is computed. Note that these points are generally not located on the HRBF surface, i.e. in fact the curvature of a different isosurface of the implicit function is evaluated. To fix this problem, the points can be pushed onto the actual interpolating surface using binary partitioning. The improvement in curvature estimation using this fix is below 1%, while the computation time grows drastically. Based on this observation, we keep the points in the tangent plane.

The final computation of principal curvatures $\widehat{K_{1,2}}$ in the center point P is:

$$\widehat{K_{1,2}} = \frac{1}{M}\sum_{i=1}^{M} K_{1,2}(C_{ip}),$$

where M is the number of the mapped sampling points C_{ip} and $K_{1,2}(C_{ip})$ is a function to compute principal curvatures $K_{1,2}$ at the point C_{ip} (see Equation (4.12)).

One disadvantage of the presented approach is the need of multiple calculations of Equation (4.12), which can be slow. Instead of averaging curvatures one can compute the average of the gradient $\boldsymbol{\nabla F}$ and the Hessian H as:

$$\overline{\boldsymbol{\nabla F}} = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{\nabla F}(C_{ip}), \tag{8.5}$$

$$\overline{H} = \frac{1}{M}\sum_{i=1}^{M} H(C_{ip}), \tag{8.6}$$

where $\overline{\boldsymbol{\nabla F}}$ is the average gradient, $\overline{H}$ is the average Hessian and M is the number of projected sampling points C_{ip}. The resulting values are used directly in Equations (4.9) - (4.12) to obtain the final curvature estimation.

8.1.4 Time complexity

The time complexity of the proposed algorithm is dictated by the solution of HRBF interpolation.We expect that the solution exists and thus the HRBF matrix is invertible. The system of linear equations is solved by LU decomposition with a partial pivoting. Without any further requirements, this can be solved in times close to $O(n^3)$. The HRBF is not solved globally for all points. For each point P, we need only a limited neighborhood with a number of points equal to m (including central point P). For each point, HRBF interpolation involves solving a system of $n = 4m$ equations (see Equation (3.11) in Section (3.2.2)). The total method complexity is $O(N(4m)^3)$, where N is the total number of points in geometry, i.e. the computation time scales linearly with N. The N part of calculation is also easy to parallelize, since every point can be processed separately. The average size of m varies with the size of used neighborhood.

In case of closed triangle manifold mesh with k-ring neighborhood, average values of m are calculated as:

$$m = 1 + \sum_{k_i=1}^{k} 6k_i.$$

In the proposed solution, we use triangle meshes with k-ring of sizes 2 and 4, which gives us 19 and 61 points respectively. For one vertex, the 4-ring version of the algorithm should be 33 times slower for a solver with $O(n^3)$. In the proposed solution, we use Eigen library [GJ+10]. Based on our experiments, their solver runs with a complexity of roughly $O(n^{2.8})$. However, this only holds for large matrices, particularly with sizes over 500×500. For smaller matrices, which are used in the proposed algorithm, the solution complexity seems to be dominated by linear and quadratic terms, yielding results closer in complexity to $O(n^2)$. We have measured that a solution of a 244×244 system is roughly $13.3\times$ slower than a solution of a 76×76 system. As a result, the whole algorithm is is approximately $10.1\times$ slower for the 4-ring neighborhood than for the 2-neighborhood.

8.1.5 Limitations

The accuracy of the proposed curvature estimation depends on the quality of the vertex normals. The proposed method is therefore not directly suitable for noisy data. Such data is interpolated in the course of the algorithm, which leads to an incorrect surface reconstruction. Our experiments with regularization of the system matrix did not bring any significant improvement. The quality of the resulting curvature depends strongly on the selected basis function. Selecting a proper basis function is not trivial for ordinary RBF, and for Hermite variant it is even more complicated. There are no universal guidelines how to select the basis function that will work for every input dataset. However, the presented $\phi = r^3$ and $\phi = exp(-\varepsilon r^3)$ are usually the best overall choice based on our experiments. A possible limitation is the speed of the proposed solution. Based on the size of the vertex neighborhood, the number of control points for HRBF can influence the speed of the surface fitting. This is especially noticeable when using neighborhoods of size over 4. The total algorithm speed can be significantly improved by a parallelization of computations, since every vertex can be processed independently.

8.2 Experiments and results

A comparison of the proposed method on the triangle mesh and the exactly computed curvature from analytic surfaces is provided. We have not used point clouds directly because of more problematic neighborhood definition via its size instead of a simpler k-ring.

The proposed method requires normal vectors as input. The exact unit-length normals computed from the function itself and normals calculated from the triangle mesh are compared in tests. For the normal vectors calculated from the triangle mesh, the average normal (*Avg*) from all adjacent triangles and the method of [Max99] (*Max*) were used. The algorithm was implemented in C++ using the Eigen library [GJ+10]. The basic implementation of HRBF was adapted from [Vai13]. Mean squared errors (MSE) of curvature values estimated by the proposed method and other methods are compared in the following subsections. The results are summarized in tables with the highlighted best result for every tested function.

8.2.1 Experiments settings

All experiments were primarily done with basis functions $\phi = r^3$ and a non-standard $\phi = exp(-\varepsilon r^3)$. As it turns out, $\phi = r^3$ is more general, while $\phi = exp(-\varepsilon r^3)$ provides better results for certain data and has tendency to smooth result. We have tested also other basis functions. For explicit functions and exact normals, an improvement of the results was achieved with $\phi = r^t$ for $t > 3$, where t is odd. However, results using $\phi = r^3$ were more consistent across all tests. The error of the estimation is expressed as a sum of MSE of $\widehat{K1}$ and $\widehat{K2}$. This corresponds to the error of the mean curvature, i.e. $0.5(\widehat{K1} + \widehat{K2})$. However, the Gaussian curvature $\widehat{K1}\widehat{K2}$ cannot be expressed by only taking the error of principal curvatures. The total error of Gaussian and Mean curvature are therefore generally different. Based on our experiments, however, the MSE results for both types of curvature show similar behavior. Therefore all included tests are based on the mean curvature only, computed as a sum of MSE of $\widehat{K1}$ and $\widehat{K2}$. The presented values are rounded to three decimal places. In the tests, Poisson disc sampling is used.

8.2.2 Curvature estimators

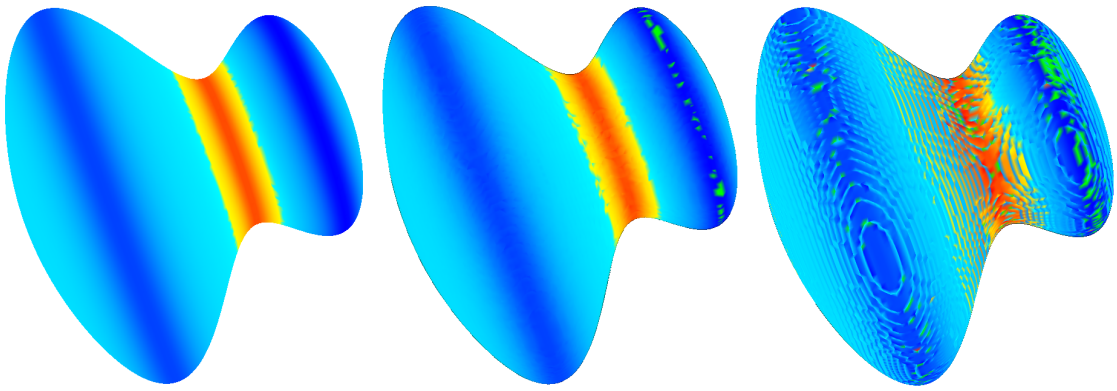
The proposed method has been tested mainly with two types of k-ring neighborhood: $k = 2$ (*HRBF 2*) and $k = 4$ (*HRBF 4*). We have tested Euclidean neighborhood as well. The size of the neighborhood was selected as a multiple of mesh average edge length. The results were consistent with k-ring neighborhood for a similar number of points. Based on this observation, we have included only the results for k-ring neighborhood, since they are faster to compute and also easier to compare with other state-of-the-art methods that use k-rings.

The proposed algorithm has been tested against several state-of-the-art methods. The algorithm from [Rus04] (*Rus*) was used due to its simplicity, popularity and quite accurate results. From surface fitting methods, [GI04] (*GI*) was used. Input to this algorithm is similar to our proposed solution. It also requires normal vectors as input and its results are affected by the accuracy of those normal vectors. For the independence

on the normal vectors, [Kal+07] (*Kal*) and a hybrid algorithm [HP11] (*HP*) have been tested.

We have also tested [CSM03] (*CSM*) and a reference implementation of [CP05] included in the CGAL library [PC16]. However, the results of these algorithms were often much worse than those of other approaches. Therefore we have decided not to include these results in the tables. A sample of the visual quality of mean curvature estimation by the proposed solution (with $k = 2$ and $\phi = exp(-\varepsilon r^3)$ and [CP05] is shown in Figure (8.4). The visual quality of the CGAL solution is rather low.

Some algorithms require additional input settings from the user. Algorithms *GI* and *HP* require the size of the used k-ring neighborhood. In the tests, ranges from $k = 1$ to $k = 4$ were tested and the best result was used for comparison with the proposed method.



(a) Exact (b) *HRBF 2* (c) CGAL

Figure 8.4: Comparison of exact, proposed and CGAL mean curvature

8.2.3 Test data

The proposed method has been tested with explicit functions (see Table 8.1) and implicit functions (see Table 8.2). Apart from exact inputs, we have also tested data with additional artificial noise: uniform and Gaussian. In order to align the results from meshes of different size/tessellation, we choose the standard deviation for noise to be the average edge length. In the experiments with noise, normal vectors were estimated directly from the noisy data using the *Max* method. We have not used *Avg* for noisy data, because the best possible quality of normal vectors is required.

Function	Interval
$f_{1e} = 10\frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2)}},\ (x \neq 0), (y \neq 0)$	$x, y \in \langle -10; 10 \rangle$
$f_{2e} = sin(x)cos(y) + 0.1 * (x^2 - y^2)$	$x, y \in \langle -10; 10 \rangle$
$f_{3e} = 0.1(\sqrt{1 + 100x^2} + \sqrt{1 + 100y^2})$	$x, y \in \langle -1; 1 \rangle$
$f_{4e} = 0.05(sin(25x) + sin(25y))$	$x, y \in \langle -1; 1 \rangle$

Table 8.1: Tested explicit functions

Function	Interval
$f_{1i} = x^4 + y^4 + z^4 - (x^2 + y^2 + z^2) + 0.4$	$x, y, z \in \langle -3; 3 \rangle$
$f_{2i} = x^2 + y^2 + z^2 + 10sin(x) - 17$	$x, y, z \in \langle -3; 3 \rangle$
$f_{3i} = 2sin(x) + sin(y) - x - z^2$	$x, y, z \in \langle -3; 3 \rangle$
$f_{4i} = x^4 + y^4 + z^4 - 1$	$x, y, z \in \langle -3; 3 \rangle$
$f_{5i} = sin(10x) + sin(10y) + sin(10z)$	$x, y, z \in \langle -1; 1 \rangle$

Table 8.2: Tested implicit functions

8.2.4 Explicit functions

The surfaces created from explicit functions have been tessellated using Delaunay triangulation in the XY plane. We have tested a tessellation with 10 000 points in a rectangle area of size and position based on the input function (see Tables 8.1 and 8.2). The points were distributed either on a uniform grid or randomly (using uniform distribution). We have run the experiments several times using a varying random seed and averaging the results from all runs.

The results from the uniform grid were similar to the ones with random points. The only difference for the proposed method was in the size of used neighborhood. For the regular grids, the results for neighborhood $k = 2$ were very similar to those obtained with $k = 4$. Based on this observation, only random vertex positions were used in the presented results.

The MSE values represent the error of the proposed method on the triangle mesh in

comparison with the analytically computed curvature of the input function at the same point.

Noiseless data

The basis function used in the proposed method is $\phi = r^t$. The first test compares the proposed method using exact normals. In this test, curvature is estimated directly at each vertex. We can use recommended value of $t = 3$. However, based on our tests, the results can be further improved with $t > 3$, where t is odd. The choice of t depends on the size of used neighborhood - for k-ring, we have used $t = 2k + 1$. The results can be seen in Table 8.3. The second test uses normals calculated with *Max*. Results can be seen in Table 8.4. The third test uses normals calculated from vertex positions using the *Avg* approach. Results can be seen in Table 8.5.

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1e}	0.007	***0.0001***	0.054	0.039	0.070	0.136
f_{2e}	0.004	***0.0003***	0.046	0.022	0.062	0.090
f_{3e}	0.009	***0.0001***	0.093	0.071	0.121	0.288
f_{4e}	0.346	***0.063***	4.453	2.773	5.962	9.620

Table 8.3: MSE for explicit data with $\phi = r^t$, where $t = 2k + 1$ exact normals

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1e}	0.059	***0.043***	0.091	0.057	0.070	0.136
f_{2e}	0.037	***0.027***	0.067	0.035	0.062	0.090
f_{3e}	0.079	***0.064***	0.148	0.105	0.121	0.288
f_{4e}	***3.160***	3.195	5.472	3.237	5.962	9.620

Table 8.4: MSE for explicit data with $\phi = r^3$ and *Max* normals

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1e}	0.079	***0.066***	0.159	0.087	0.070	0.136
f_{2e}	0.063	***0.054***	0.125	0.062	0.062	0.440
f_{3e}	0.168	0.164	0.362	0.176	***0.121***	0.288
f_{4e}	***3.590***	3.704	6.089	4.476	5.962	9.620

Table 8.5: MSE for explicit data with $\phi = r^3$ and *Avg* normals

We present one sample of the Gaussian curvature estimation MSE in Table 8.6. We only show results for normals estimated with *Max*. The distribution of MSE is similar to the values for mean curvature in Table 8.3.

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1e}	0.017	0.007	0.019	0.012	0.011	0.016
f_{2e}	0.009	0.006	0.014	0.010	0.012	0.021
f_{3e}	0.117	0.095	0.176	0.109	0.104	0.221
f_{4e}	31.796	32.184	56.721	32.305	60.104	86.823

Table 8.6: MSE of Gaussian curvature for explicit data with $\phi = r^3$ and *Max* normals

The results of the proposed algorithm depend on the size of the selected neighborhood. As can be seen from the tests, the difference between the k-ring neighborhoods of sizes $k = 2$ and $k = 4$ can be substantial in some cases. However, using a smaller neighborhood is preferred because of the computational cost. For inaccurate normals, the proposed algorithm performs worse than for exact normals, but it is still one of the best algorithms.

The selection of basis function can improve quality of results for data equipped with exact normals. In Table 8.7 we present MSE for curvature estimation based on different basis functions. All results are created for 4-ring neighborhood. For *Max* normals, there is small or no improvement at all and the basis function $\phi = r^3$ offers better results. Based on this, we have not included the results for this test scenario.

ϕ	f_{1e}	f_{2e}	f_{3e}	f_{4e}
r^3	0.006	0.003	0.010	0.838
r^5	0.0005	0.0003	0.0004	0.154
r^7	0.0002	0.0002	0.0002	0.071
r^9	0.0001	0.0003	0.0001	0.063
$r^4 ln(r)$	0.002	0.001	0.002	0.288
$\frac{1}{\sqrt{1+(\varepsilon r)^2}}$	0.577	0.501	1.141	21.650
$\frac{1}{\sqrt{1+(\varepsilon r)^3}}$	0.011	0.007	0.018	1.347

Table 8.7: MSE of curvature for explicit data with *exact* normals. Tested different basis function on 4-ring neighborhood.

Noisy data

Our proposed method is designed primarily for noiseless data, however, we have also tested how the proposed solution works for data with noise. The used basis function is $\phi = r^3$.

Overall, in over half of the tests, the proposed solution outperformed the next best solution by more than 14. For Gaussian noise with small standard deviation (≤ 0.01 of the body diagonal length), the proposed method was the best in almost all cases. In other scenarios the results vary.

8.2.5 Implicit functions

Implicit functions have been tessellated using Marching Cubes [LC87]. We have selected this algorithm because of its simplicity, which makes it a common first step before implementing more advanced methods. At the same time, Marching Cubes are known for generating meshes of rather problematic character, with elongated triangles etc., which makes the curvature estimation all the more difficult. We have used grids of $128\times128\times128$ samples.

We have also tested our implementation with Dual Contouring [Ju+02]. It is an algorithm similar to Marching Cubes. In this case, the tesselattion points are not located on the cell edges but inside of the cells. Cell corners are values of voxelized grid. For a simplified 2D version see Figure 8.5.

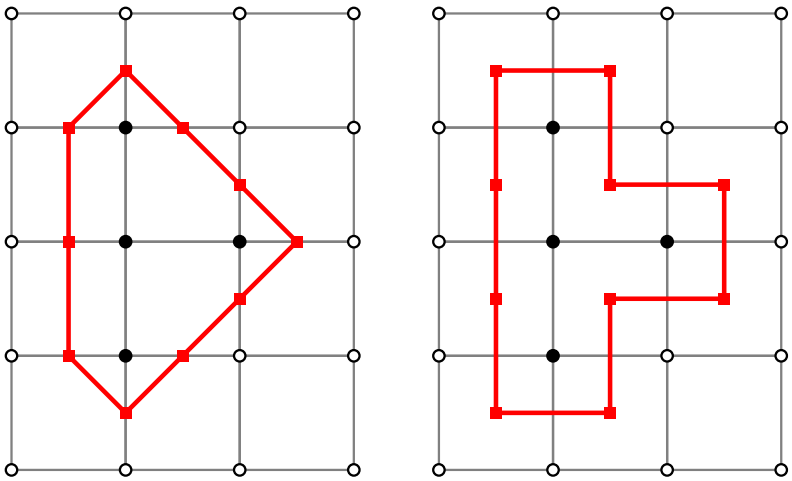


Figure 8.5: Comparison of 2D version of Marching Cubes (left) and Dual Controuring (right). In Marching Cubes, the generated vertices are on the cell edges, while for Dual Contouring they are inside the cells.

The results for Dual Controuring were similar to those obtained with Marching Cubes. Therefore we have chosen to include only the results from the Marching Cubes.

Because Marching cubes linearly interpolates the sample values along the regular grid edges, the resulting mesh vertices generally do not lie on the original implicit surface, although they are usually very close. In this section, two variants are tested. First, this non exact version (called *original*), and second, a modification, where the grid edge intersections are calculated exactly by binary search using the original implicit function. The stopping condition is set for function values to be under 10^{-7}. We denote this version *exact*.

There is a problem with the direct curvature comparison for the *original* version. The curvatures evaluated at sampled vertices are not exactly those of the smooth implicit surface, although the difference is usually small. Taking this into consideration, we have used the same approach as for the exact scenario, where estimated curvatures are compared with curvatures of the input implicit function at the vertex locations. Similarly to the previous section, we have tested different basis functions. However, the results were not improved and therefore we have not included these results.

Original Marching cubes

In this first experiment, the *original* version of the Marching Cubes algorithm was tested. The input can be interpreted as slightly noisy, because the extracted isosurface vertices are not generally located on the actual surface of the implicit function.

We have tested this scenario with interpolant evaluation directly in a point, as well as with the Poisson disc sampling, because of the noisy character of input tessellation. Results can be found in Tables 8.8 - 8.10.

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1i}	0.170	***0.066***	0.199	0.299	0.326	0.290
f_{2i}	0.021	***0.005***	0.021	0.024	0.035	0.033
f_{3i}	0.031	***0.010***	0.018	0.041	0.048	0.053
f_{4i}	0.079	***0.016***	0.043	0.186	0.134	0.077
f_{5i}	0.448	***0.165***	0.317	0.492	0.691	0.508

Table 8.8: MSE for *original* marching cubes with $\phi = exp(-\varepsilon r^3)$ and *"exact"* normals

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1i}	0.426	0.387	0.799	0.375	0.326	***0.290***
f_{2i}	0.061	0.055	0.119	0.037	0.035	***0.033***
f_{3i}	0.082	0.073	0.152	0.059	***0.048***	0.053
f_{4i}	0.201	0.158	0.342	0.199	0.134	***0.077***
f_{5i}	0.889	0.957	1.631	0.737	0.691	***0.508***

Table 8.9: MSE for *original* marching cubes with $\phi = exp(-\varepsilon r^3)$ and *Max* normal

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1i}	0.499	0.465	1.113	0.382	0.326	***0.290***
f_{2i}	0.052	0.049	0.136	0.036	0.035	***0.033***
f_{3i}	0.068	0.063	0.167	0.055	0.048	***0.053***
f_{4i}	0.253	0.216	0.515	0.223	***0.134***	0.077
f_{5i}	0.863	0.818	1.444	0.655	0.691	***0.508***

Table 8.10: MSE for *original* marching cubes with $\phi = exp(-\varepsilon r^3)$ and *Avg* normals

For *"exact"* normals, Poisson disc sampling offered a slightly better estimation. However, to be consistent with the previous chapter, these results are not included. The

size of the neighborhood is important for estimation with *"exact"* normals. The bigger neighborhood offers almost three times better results. However, for estimated normals, the improvement is often marginal and the smaller neighborhood is sufficient if we consider its faster computation. Overall, the algorithm for estimated normals for original marching cubes performs poorly. This is expected, since the data are noisy. Nevertheless, the proposed solution offers better results than the popular *Rus* algorithm.

Exact Marching cubes

The results can be found in Tables 8.11 - 8.13. For exact normals, the algorithm outperforms other methods in a way similar to *orignal* Marching Cubes version from the previous subsection. Estimated normals from *exact* Marching cubes provide generally more precise results than the *orignal* Marching cubes version. The results of the proposed method are comparable with the results of *GI*. Unlike *orignal* Marching cubes, the size of the used neighborhood does not affect the results significantly.

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1i}	0.154	***0.041***	0.195	0.102	0.218	0.301
f_{2i}	0.020	***0.004***	0.021	0.007	0.019	0.033
f_{3i}	0.029	***0.007***	0.018	0.019	0.019	0.053
f_{4i}	0.078	***0.015***	0.041	0.028	0.049	0.070
f_{5i}	0.433	***0.160***	0.324	0.326	0.341	0.521

Table 8.11: MSE for *exact* marching cubes with $\phi = exp(-\varepsilon r^3)$ and *"exact"* normals

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1i}	0.131	***0.111***	0.293	0.154	0.218	0.301
f_{2i}	0.021	***0.017***	0.058	0.019	0.019	0.033
f_{3i}	0.028	0.022	0.064	0.021	***0.019***	0.053
f_{4i}	0.114	***0.023***	0.075	0.044	0.049	0.070
f_{5i}	0.403	***0.330***	0.688	0.495	0.341	0.521

Table 8.12: MSE for *exact* marching cubes with $\phi = r^3$ and *Max* normal

	HRBF 2	HRBF 4	Rus	GI	Kal	HP
f_{1i}	0.341	0.320	0.727	0.290	***0.218***	0.301
f_{2i}	0.046	0.047	0.137	0.30	***0.019***	0.033
f_{3i}	0.049	0.049	0.146	0.045	***0.019***	0.053
f_{4i}	0.170	0.112	0.289	0.096	***0.049***	0.070
f_{5i}	0.605	0.587	1.405	0.600	***0.341***	0.521

Table 8.13: MSE for *exact* marching cubes with $\phi = r^3$ and *Avg* normals

Noisy Exact Marching Cubes

We have tested noise with *exact* version of Marching cubes, since the *original* version contains some additional noise by design and we want to have only noise of a certain type. The used basis function is, again, $\phi = r^3$.

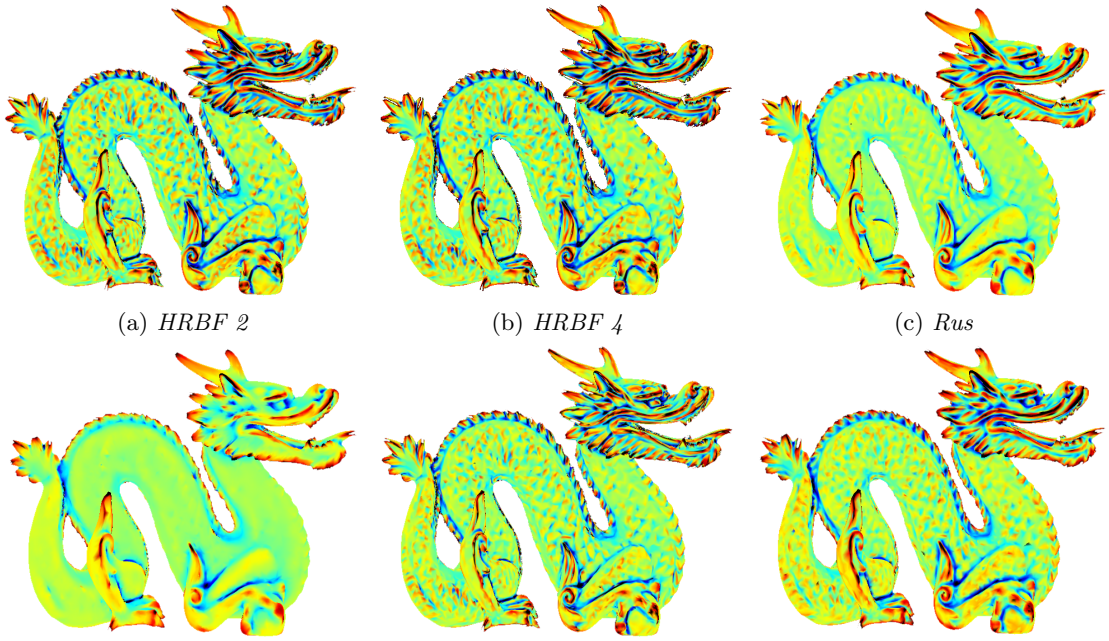
The results are not as good as for explicit data. Our proposed solution was in almost every test outperformed by solutions that do not take the normals into account (*HP*, *Kal*).

Meshes

Apart from numerical comparisons, it is also important to know how the method stands in visual comparison. For these tests, triangle meshes with unknown exact curvature were used. Normals of the model were estimated from the triangle mesh using *Max*.

Curvatures estimated by all methods were mapped to the same interval. This interval was manually selected. An automatic interval selection could be used with one of the estimators as a source; however, a manual selection of the interval overcomes problems associated with outliers. We have tested multiple models from [Sta]. The overall visual results were similar for all models. Therefore only one illustrative sample is presented - the Stanford Dragon (300 000 vertices), where the curvature is mapped to the interval $\langle -10; 10 \rangle$.

For the algorithms with more setup options, the best visual quality result was selected based on our observations. Usually, the results with the best visual quality, which emphasize the fine details of the mesh, used 1-ring neighborhood. These details were smoothed out with a bigger neighborhood. The resulting mean curvature estimations are shown in Figure 8.6.

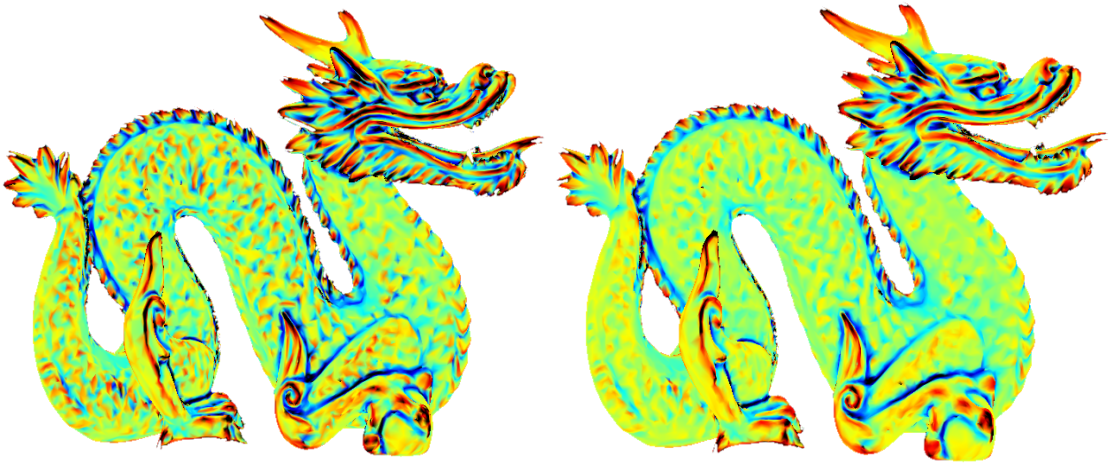


(a) *HRBF 2* (b) *HRBF 4* (c) *Rus*

(d) *Kal* (e) *GI* (f) *HP*

Figure 8.6: Mean curvature comparison

Comparison of the different basis functions for the proposed algorithm can be seen in Figure 8.7. The used neighborhood in this case has size $k = 2$. The selected size did not affect the results and the final visual impression was nearly identical.



(a) $\phi = r^3$ (b) $\phi = exp(-\varepsilon r^3)$

Figure 8.7: Effect of basis function (mean curvature)

With the basis function r^3, the final curvature seems more detailed. The second basis function $exp(-\varepsilon r^3)$ smoothed out some of the fine details. The selection of the

appropriate basis function is a user decision that can be based on the presence of fine details in the model.

Summary

From the above tests it can be observed that with exact normals, HRBF greatly outperforms the other algorithms. With larger neighborhood, the results can be more accurate (this is mainly in case of implicit functions), but the computation is slower.

With normals calculated directly from the triangle mesh, the situation is more complex. The proposed solution for noise-free data (explicit functions and *exact* Marching Cubes) using *Max* outperforms other state-of-the-art solutions. For other scenarios, algorithms that do not require normal vectors as input (like *Kal* or *HP*) offer better results.

However, for a fair comparison, the proposed solution requires normal vectors and should be directly compared with appropriate algorithms. The most similar to our solution is *GI*. The proposed algorithm is slower, but unlike *GI*, it has stable results with the bigger neighborhood. With the increased size of the neighborhood, the results are improved. In the case of *GI*, a smaller neighborhood can sometimes offer a significantly better result and the effect of the neighborhood size is quite unpredictable. This is not the case of the proposed solution.

For noisy data, the proposed solution performs worse. HRBF interpolant created from noisy inputs interpolates the input noise. However, by using Poisson disc sampling, an average value from several samples is calculated, which partially smooths the noise. Still, the proposed algorithm is usually outperformed by the algorithms that do not use normal vectors, such as *Kal* or *HP*. However, among the algorithms that take the normal vectors into account, our proposed solution ranks among the best and the results are considerably better than *GI*.

8.3 Normal vector re-estimation

The curvature estimation algorithms that do not use normal vectors at its input can be used to improve normal vectors. However, only presented solution known to us describing this approach, is the algorithm of Kalogerakis et al. ([Kal+07]) - *Kal*.

Our proposed algorithm can be used to improve the normal vector estimation even though it requires normal vectors at its input. If we have geometry with estimated normals, they can be used to create HRBF interpolation and curvature estimation (see Section 8.1.3). However, an approach similar to curvature estimation can be used to re-estimate normal vector at a vertex.

For the HRBF input, we use normals estimated with [Max99] (any other normal-estimation algorithm can be used as well). The HRBF interpolation and Poisson disc sampling is computed as described in Section 8.1.3. The newly approximated normal vector $\boldsymbol{n}$ is computed as

$$\overline{\boldsymbol{\nabla F}} = \sum_{i=1}^{M} \boldsymbol{\nabla F}(C_{ip}), \tag{8.7}$$

$$\boldsymbol{n} = \frac{\overline{\boldsymbol{\nabla F}}}{||\overline{\boldsymbol{\nabla F}}||}, \tag{8.8}$$

where $\boldsymbol{\nabla F}$ is the gradient, $\overline{\boldsymbol{\nabla F}}$ is the average gradient (see Equation 8.5), C_{ip} are the mapped sampling points (see Section 8.1.3) and $\boldsymbol{n}$ is the new normal vector.

8.3.1 Results

We use *Kal* approach for a direct comparison with our proposed solution (*HRBF2* and *HRBF4*). We also compare our solution with *Avg* and *Max* normals. The basis function in the proposed method is $\phi = r^3$. The test data and settings are the same as for curvature estimation tests described in Section 8.2.

In the first test, quality of the normal vector is compared against the exact normal vector calculated directly from the function itself. We have used explicit functions and implicit functions tessellated with *exact* version of Marching Cubes algorithm, since we are able to compare estimated normal vector against the exact normal. For the comparison, average error value ϵ is computed as:

$$\epsilon = 1 - \frac{1}{N}\sum_{i=0}^{N}\left(\boldsymbol{n}_i \cdot \boldsymbol{n}_{i_{est}}\right), \tag{8.9}$$

where N is the number of points, $\boldsymbol{n}_i$ is the exact normal and $\boldsymbol{n}_{i_{est}}$ is the estimated normal. Results of $\epsilon \times 10^{-4}$ are summarized in Table 8.14. There is no visible difference between *Max* and *Kal* in table due to rounding of values. However, the real values are not the same, the difference is approximately at the $10th$ decimal place. This small difference is caused by the nature of *Kal* that can improve normals only for noisy data, while the input data are noise-free. Partially, this effect can be seen for Marching Cubes algorithm that adds some small noise even if we have used *exact* version.

	HRBF 2	HRBF 4	Avg	Max	Kal
f_{1e}	0.979	***0.898***	5.530	1.533	1.533
f_{2e}	0.736	***0.671***	3.369	0.813	0.813
f_{3e}	0.104	***0.097***	0.578	0.132	0.132
f_{1i}	0.130	***0.123***	4.888	0.895	0.894
f_{2i}	0.022	***0.022***	1.675	1.996	1.995
f_{3i}	0.038	***0.036***	1.217	1.362	1.361

Table 8.14: Average normal error $\epsilon \times 10^{-4}$

The second test uses re-estimated normal vector from our proposed solution for curvature estimation. In this test, only estimators that require normal vectors at its input are tested (*Rus* and *GI*). The used functions and their tessellations are the same as in the previous test from Table 8.14. Resulting MSE of mean curvature can be seen in Tables

8.15 (for *Rus*) and 8.16 (for *GI*). One can see that the proposed re-estimated normal vectors significantly improve the curvature estimated with existing state-of-the-art methods that require normal vectors at their input.

		Original	HRBF 2	HRBF 4	Avg	Max	Kal
f_{1e}	Rus	0.054	0.072	***0.068***	0.159	0.090	0.082
f_{2e}	Rus	0.046	0.057	***0.055***	0.125	0.067	0.074
f_{3e}	Rus	0.093	0.124	***0.121***	0.362	0.148	0.148
f_{1i}	Rus	0.195	0.219	***0.216***	0.727	0.293	0.297
f_{2i}	Rus	0.021	***0.024***	***0.024***	0.137	0.058	0.052
f_{3i}	Rus	0.018	***0.025***	***0.024***	0.146	0.064	0.054

Table 8.15: MSE of mean curvature estimated with *Max* for different normal vector estimation algorithms. Columns represent normal estimation algorithm. Original normal vector is calculated directly from the input data equation.

		Original	HRBF 2	HRBF 4	Avg	Max	Kal
f_{1e}	GI	0.039	0.056	***0.054***	0.087	0.057	0.056
f_{2e}	GI	0.022	0.035	***0.034***	0.062	0.035	0.045
f_{3e}	GI	0.071	0.097	***0.096***	0.176	0.105	0.110
f_{1e}	GI	0.102	0.110	***0.109***	0.290	0.154	0.160
f_{2i}	GI	0.007	***0.009***	***0.009***	0.030	0.019	0.018
f_{3i}	GI	0.019	***0.024***	***0.024***	0.045	0.024	0.031

Table 8.16: MSE of mean curvature estimated with *GI* for different normal vector estimation algorithms. Columns represent normal estimation algorithm. Original normal vector is calculated directly from the input data equation.

The normal vector re-estimation part of the algorithm improves normal vectors and outperforms other algorithms. In addition, this solution can also be used for point clouds, while the other algorithms work only for triangulated geometry. In case of triangulated geometry, there is only a small difference between 2 and 4-ring neighborhoods, therefore the smaller neighborhood is preferred. Similar observation can be applied to point clouds as well.

Chapter 9

Curvature-Based Feature Detection for Human Head Modeling

In the field of 3D head modeling and animation, the models are often required to be enhanced by a set of labels or parameters. Feature points are often needed to mark important regions of the face. This additional information can be used to animate or deform the input model.

We have focused on 3D models of faces. They have been widely used in many areas of computer graphics, e.g., animation, virtual reality, computer games, etc. The models can be created manually in a modeling software or obtained by 3D scanners. However, these models are static and need to be equipped with additional properties if they are to be modified or animated.

A common approach for face modeling is based on feature detection. The features of the model are areas or points with interesting topological or geometric properties. Once detected, these areas or points can be used to control the deformation of the model to achieve the desired output. For human beings, the task to detect features is quite simple and straightforward. However, to detect the features automatically can be a challenging task. Algorithms are sensitive to noise and do not have the ability to exclude less important areas without human interaction or a complex learning process.

In our research, published in [Pra+17], we have found a solution suitable for the automatic feature detection on triangulated models of human head. The algorithm is expected to detect not only the main feature regions such as eyes, nose or mouth, but also the important points within these regions. These points can be selected manually, but it can be time-consuming, especially if a large database of input models needs to be processed. An automated solution is required in such a case. The detected features are further used in deformation-based modeling that was presented by Martínek and Kolingerová in [MK14].

9.1 Algorithm

The feature points to detect are selected not based on their anthropology meanings, but their positions with respect to the further processing of the mesh. The automatic detection has to be able to find points lying approximately in the given region, but the detection does not need to be exact, as the deformation method ([MK14]) uses multiple points at once and works on a Gaussian neighborhood.

9.1.1 Preprocessing

As a preprocessing step, principal curvatures at every vertex of the model have to be estimated. Any algorithm for curvature estimation can be used. In our solution, we have tested our Hermite-based solution (see Chapter 8) and Russinkiewicz method [Rus04].

Once the curvatures are obtained, other shape descriptors are then calculated from the estimated curvature values. The proposed algorithm uses shape index, curvedness, mean curvature and Willmore energy. Apart from shape index, all the descriptors are scale-related. To overcome this, the values of the descriptors are linearly mapped to the interval $\langle 0, 1 \rangle$. To reduce the influence of possible noise in the input data, all descriptors are averaged within an Euclidean neighborhood of size 0.01. This setup have outperformed other scenarios (including average edge length) for the tested models with varying resolution.

The presented solution uses a notation with the nose pointing in a positive direction of the z-axis and the top of the head pointing in a positive direction of the y-axis. The positive x-axis direction is referred to as the right, while the negative as the left side of the head. The entire head is also scaled to fit into the box with dimensions $1 \times 1 \times 1$, without the deformation of the shape. All distances and sizes in the following text are scaled accordingly. All the values and distance used in the text are obtained via a cross-validation from several head models.

A prerequisite of the proposed solution is an axis-aligned 3D model of a human head. However, in many cases, these models are not available. To overcome this, we can create an axis-aligning ourselves. In the first step, symmetry axis of the head is computed using Sipiran et al. [SGS14]. Their solution is based on Gaussian curvature which is available to us. The resulting symmetry plane is not enough to align the head. With only the plane, we are able to obtain alignment in YZ plane, but the nose tip is incorrectly positioned (not pointing in a positive direction of the z-axis). To detect the nose tip (P_{nose}), we use solution from Yang et al. [Yan+09]. They can have some false-positive results, but we are able to reject them based on the detected symmetry plane. The nose tip must be located on the plane (or very close to it). This assumption will reject false nose tips candidates that were detected in areas of ears or eyes.

9.1.2 Detection of main anchor points

Three main anchor points have to be detected. Those are: the top of the head P_{top}, the tip of the nose P_{nose} and the chin P_{chin} (see Figure 9.1). These points are later used as anchors for the detection of other points.

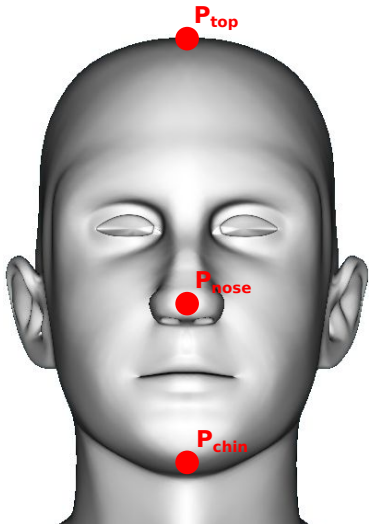


Figure 9.1: Anchor points on a head model.

If we have used axis-aligning, we already have a detected nose tip. Otherwise, for the axis-aligned input model, the nose tip can be detected using extremal values in a certain axis. The nose tip P_{nose} is detected as the point with the maximal z-coordinate.

Next, we need to locate the top of the head P_{top}. This point has the maximal y-coordinate.

The center of the chin P_{chin} cannot be detected using only extrema. This point is expected to be one of the points within a cone that is centered at the nose tip and facing in the direction of the negative y-axis. From all the points that satisfy this condition, the chin center is located in a close neighborhood of the point with the maximal distance from the nose tip. A spherical neighborhood with radius 0.6 is used. The center of the chin is a point from within this neighborhood that has the biggest z-coordinate.

Once the anchor points are detected, they can be used to facilitate the search of the feature regions.

9.1.3 Detection of other points and regions

The detection of the features can be divided into two parts. The first part detects important areas of the head – nose, mouth, eyes and ears. For pair organs, it is capable of detecting separate regions for the left and right organ. The order in which the regions are detected is mandatory, as the already detected parts are used to improve the detection of the next ones. This part of the algorithm can be used independently if only the feature regions have to be detected.

The second part of the algorithm can detect specific feature points inside the areas found in the first step. From the specific points in the areas, we have selected feature

points that are further used in [MK14].

To speed up the algorithm, not all points belonging to the feature regions are detected. A detection radius similar to Poisson disc sampling is used in the proposed solution. If a newly detected point lies within the radius of an already detected point, the new one is not added to the feature region.

Eyes

At first, the eye regions and the corresponding feature points have to be detected. These points are later used to detect other important parts of the head.

The eye regions are detected based on their shape and their relative position to the already detected anchor points. The eyes are located above the nose, so the vertices with the y-coordinate below the nose can be automatically rejected. A better approximation of the position of the eye regions can be achieved by limiting the estimation of the position to the region between the nose tip and the forehead.

The estimation of the forehead has been set to be 60 percent of the Euclidean distance between the nose tip P_{nose} and the top of the head P_{top}. From the vertices that satisfy this condition, eye areas are selected based on the shape of the mesh. Eye areas have a spherical shape, so their Willmore energy should be minimal. For all vertices V_i that satisfy the previous conditions, the measure of resemblance to eyes m_{E_i} is calculated as follows:

$$m_{E_i} = W_i |P_{nose} - V_i|. \tag{9.1}$$

where W_i is Willmore energy at the vertex V_i and P_{nose} is the tip of the nose. The first N points with the biggest value of m_{E_i} belong to the eye regions. The value of N is selected by the user with respect to the tessellation of the model around the eyes. The left and right eye region can be easily distinguished based on the value of the x-coordinate (in our setup, the points with $x < P_{nose_x}$ belong to the left region). Once the eye regions are detected, the required feature points can be extracted (see Figure 9.2). Eye corners E_1 and E_2 are found as the points with extremal values in the x-axis.

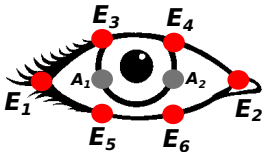


Figure 9.2: Eye feature points.

Once the eye corners are obtained, other eye points can be detected based on the position of the eye corners. In the proposed solution, two points on the upper and lower eyelid are searched for. To find these points, two auxiliary points A_1 and A_2 are inserted in equidistant positions to the line segment connecting E_1 and E_2. Eyelid points E_3 to E_6 are then selected as the points with the minimal difference in the x-coordinate from the points A_1 and A_2 respectively.

Nose

Nose region detection is simplified by the fact that the tip of the nose P_{nose} has already been detected during the detection of the anchor points. The feature region of the nose is represented by a strip of points located above the nose tip with a certain width in the x-axis. In the proposed solution, this value is set to be 0.1. To discard the points that should not belong to the nose region, all the vertices V_i within the strip are assigned the measure of resemblance to nose m_{N_i}:

$$m_{N_i} = S_i C_i. \tag{9.2}$$

where S_i stands for the shape index at the vertex V_i and C_i is the curvedness at the vertex V_i. The first M points with the smallest value of m_{N_i} belong to the nose region. The value of M is selected by the user with respect to the tessellation of the nose.

After the nose region is detected, feature points of the nose can be extracted (points N_1 to N_8 in Figure 9.3).

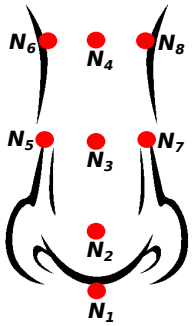


Figure 9.3: Nose feature points.

The point N_2 coincides with the tip of the nose P_{nose} that has already been located in the first step of the algorithm (see Subsection 9.1.2). Another important point to demarcate is the nose root (point N_4 in Figure 9.3). The root of the nose is located approximately between the eyes, so the position can be predicted as a center point between the two inner eye corners that have already been detected using the approach described in the previous section. To detect the root of the nose more precisely, the measure of resemblance to nose root m_{NR_i} is calculated for each vertex V_i from the nose feature region:

$$m_{NR_i} = S_i C_i |P_{pred} - V_i|. \tag{9.3}$$

where P_{pred} is the predicted position, S_i stands for the shape index at the vertex V_i and C_i is the curvedness at the vertex V_i. The distance from the predicted position is used to prevent finding points that are located too far away. The point with the minimal value of m_{NR_i} is marked as the nose root N_4.

Similarly, the nose center (point N_3 in Figure 9.3) is found using the predicted position. The prediction is calculated as the average of the nose tip N_2 and the root N_4. The actual center is then the point with the minimal distance from the predicted position.

The point N_1 has to be located under the nose tip N_2, so the region that is searched can be limited to points with lower y-coordinate. Also, the point lies approximately under the tip of the nose and so only a thin strip of points has to be searched. In the presented solution, a strip of width 0.004 in the x-coordinate is used. To find the correct position of the point N_1, the measure of resemblance to nose base m_{NB_i} is calculated for each vertex V_i from the predefined strip of points:

$$m_{NB_i} = \frac{K_{H_i}}{d_i}, \tag{9.4}$$

where K_{H_i} is mean curvature at the vertex V_i and d_i is the distance between the vertex V_i and the tip of the nose P_{nose} in the z-coordinate. The point with the maximal value of m_{NB_i} is the desired point N_1.

The points on the side of the nose (points N_5 to N_8 in Figure 9.3) are localized using mean curvature and the distance from the points N_3 or N_4. The measure of resemblance to the side point m_{side_i} is calculated for each vertex V_i as:

$$m_{side_i} = \frac{K_{H_i}}{d_i} \tag{9.5}$$

where K_{H_i} is mean curvature at the vertex V_i and d_i is the distance between the vertex V_i and the corresponding feature point (point N_3 for the detection of points N_5 and N_7; point N_4 for points N_6 and N_8).

For each of the feature points N_3 and N_4, two points with the maximal value of m_{side_i} are the desired points. To prevent finding points on the same side of the nose, the x-coordinate of the detected points is compared to the x-coordinate of the corresponding feature point.

Mouth

Detection of the mouth and its feature points is very important for the model deformation, as they are crucial for the changes of expression of the modeled face. In the proposed approach, four points necessary for basic mouth control are located.

The mouth area is detected in a different manner than the other feature regions. The region of the mouth is found using four feature points (Figure 9.4) - the corners of the mouth M_1 and M_2 and the centers of the upper and lower lip M_3 and M_4, respectively.

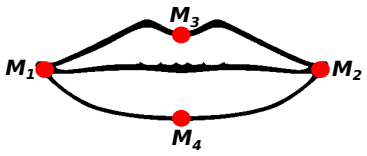


Figure 9.4: Mouth feature points.

Mouth corners are considered to be spherical regions, therefore Willmore energy can be used to detect them. To avoid the detection of points belonging to other feature

regions (i.e., eyes, mouth), points must be under the nose tip and above the chin in the y-coordinate. To detect if a corner is left or right, comparison of the x-coordinate against the nose tip is used.

The points on the upper and lower lip are detected using a predicted position. The prediction is defined as an average of the mouth corners M_1 and M_2. The predicted point lies between the lips. To obtain the correct points on the lips, maximal position in the z-coordinate is used. The points that are closest to the predicted position and have extremal value of the z-coordinate are the centers of the lips. The upper M_3 and lower M_4 lip points are distinguished based on the comparison of the y-coordinate and the predicted center.

Once the four feature points are detected, an axis-aligned bounding box (AABB) is created around them. Mouth corners have minimal, while lip points maximal depth, so the whole mouth can be enclosed within the created AABB.

During the detection of the mouth area, four feature points have been found. More points can be located in a way similar to the location of eyelids, as described in *Eyes* Section, by using predicted anchor points.

Ears

The detection of the ears is usually less important than the detection of other regions, as the ears can often be hidden under hair or some kind of headwear. However, in deformation-based face modeling, the detection of ears may be important to allow the modification of the position of the ears.

An ear is detected based on its helix that has the maximal curvedness. To eliminate other possible points with large curvedness, such as the points on the front part of the face, distance in the x-axis from eye corners is used. Ears are the most distant points with the maximal curvedness. For the left ear, the right eye inner corner is used (and vice versa for the right ear). The final weight m_{R_i} is calculated for each vertex V_i as:

$$m_{R_i} = C_i |V_{i_x} - E_x|, \tag{9.6}$$

where E_x is the x-coordinate of the corresponding eye corner, C_i is the curvedness at the vertex V_i, and V_{i_x} is the x-coordinate of the vertex V_i. The first N points with the maximal value of m_{R_i} belong to the ear region. The N is selected by the user with respect to the tessellation of the ear.

To detect the feature points of the ear (see Figure 9.5), an axis-aligned bounding box (AABB) enclosing the ear feature region is used. The points R_1 to R_4 are detected as the closest points to the corresponding corners of the outer side of the AABB. The point inside the ear (point R_5) is estimated as the point with the minimal distance from the average of the points R_1 to R_4.

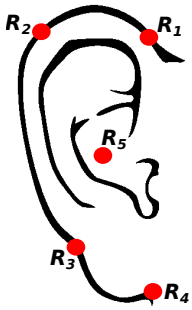


Figure 9.5: Ear feature points.

9.2 Experiments and results

In this section, the results of the proposed method will be shown. For the tests, head models created using a 3D modeling software are used. This is due to the further use of the points detected by the proposed solution for deformations, that can be utilized for example in computer games.

The comparison with an existing state-of-the-art methods cannot be directly conducted. Existing methods are mostly designed to detect different feature points. Our feature points are based on the need of further deformations. Many existing method detect points based on anatomical properties or they detect only certain points. Solution suitable for our needs could be one of the algorithms based on training data (such as [GSM15]). However, we were not able to train the algorithm because of the small set of training data. We are not in possession of hundred of heads, that have manually selected feature points required for the training process.

The presented results compare automatically detected points with manually selected ones that were designed to fit the needs for further use in deformations. The deformations with the detected points are presented as well.

Similar to [CSJ05], the proposed method is capable of handling a small deviation from the axis-aligned state (approx. 10°). Based on our tests, this is sufficient if the input model is not aligned and alignment must be computed as proposed. The required maximal deviation is in the limit of error produced by the proposed align method.

9.2.1 Regions

The detected regions are based on the vertices that belong to them. We use a Poisson disc sampling to influence the number of points in each region; different sampling radii are used for different regions. Smaller radii are used for the ears and the eyes, greater for the nose and the mouth. This is due to the tessellation of the mesh in the given region. The ears and eyes usually have a very detailed geometry to record eyelids and large curvatures of ears, while the mouth and the nose consist of relatively simple regions.

The results of the region detection are shown in Figure 9.6 and Figure 9.7. It can be

seen that the detected regions correctly capture the position of the desired features of a human head.

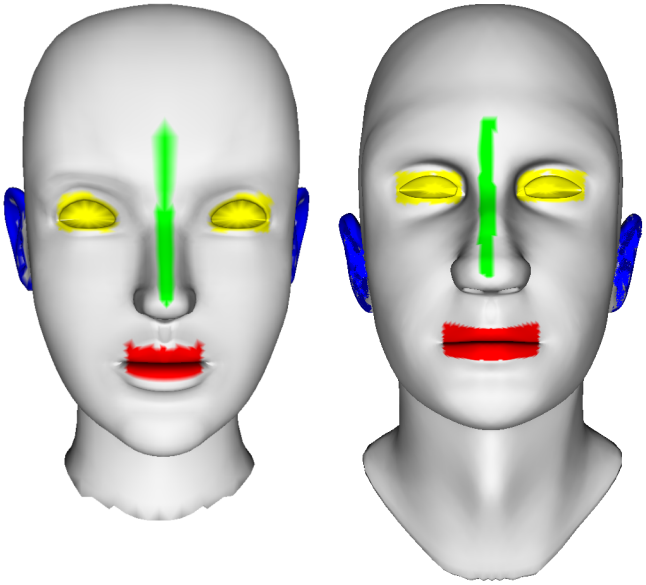


Figure 9.6: Detected feature regions (front view).

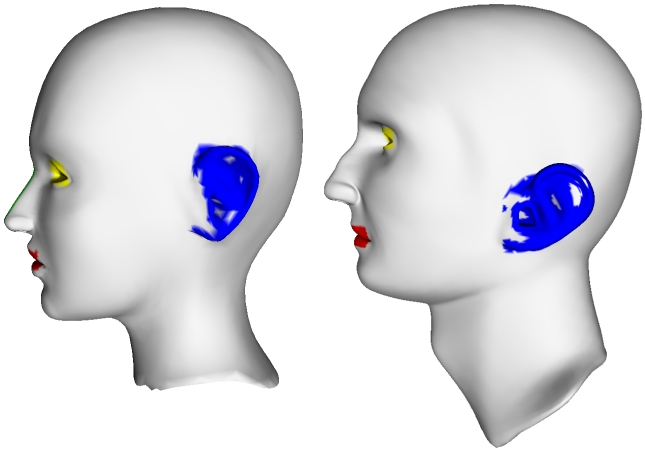


Figure 9.7: Detected feature regions (side view).

9.2.2 Feature points

To verify the results of the proposed solution, we compare the results of the proposed algorithm with the points manually placed by an expert user. The results are captured in Figure 9.8 .

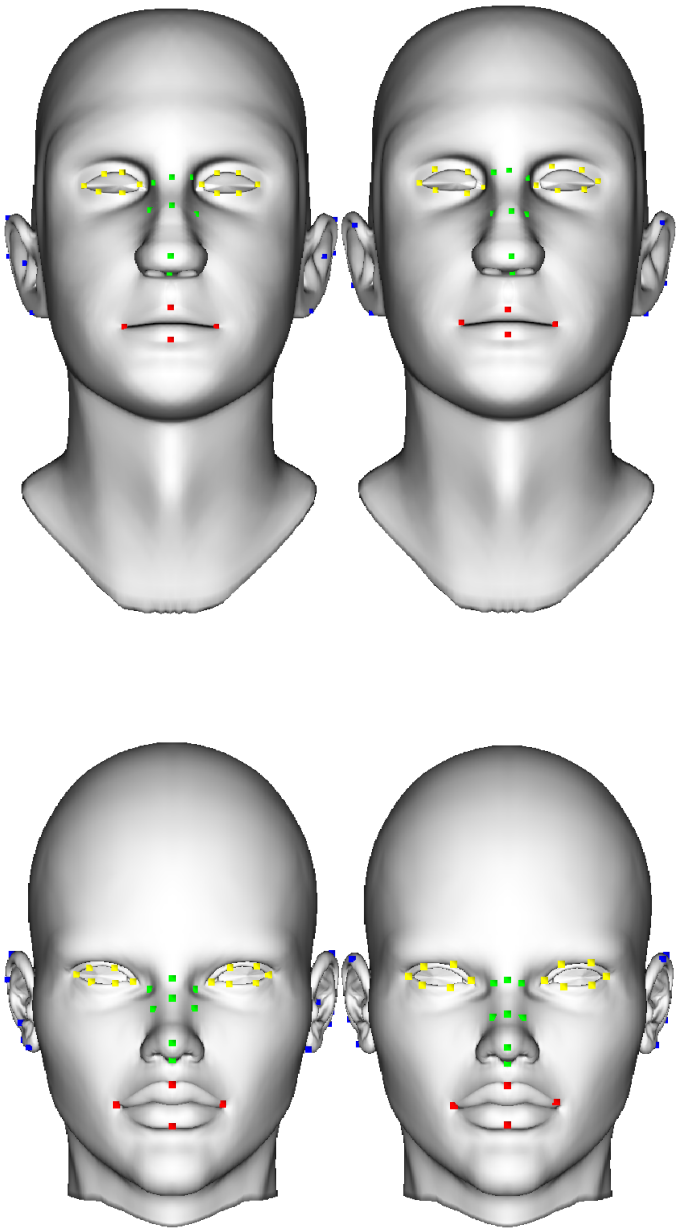


Figure 9.8: Manually inserted (left) and automatically detected (right) feature points.

It can be seen that the positions of the detected and the manually inserted points slightly differ. However, as the intended use of the detected points is for the use in the deformation-based modeling, the accuracy of the detection is sufficient.

9.2.3 Deformations

The purpose of the feature points extraction using the proposed method is to facilitate the initialization of the model for the use in deformation-based head modeling [MK14].

To validate our results, we compare the results of the mesh deformation using the detected feature points to the models deformed via feature points defined manually by an expert. Figure 9.9 shows several examples of a deformation of the mouth, using both the detected and the manually defined feature points. Similarly, Figure 9.10 shows examples of deformations of nose, chin and ears. Each of the transformed models was created using a single deformation of the same magnitude for both sets of feature points.

Figure 9.9: Head model deformation using feature points. Models created using manually inserted (top row) and automatically detected points (bottom row). Face expression (from left to right): original model, happy face, sad face, wide mouth.

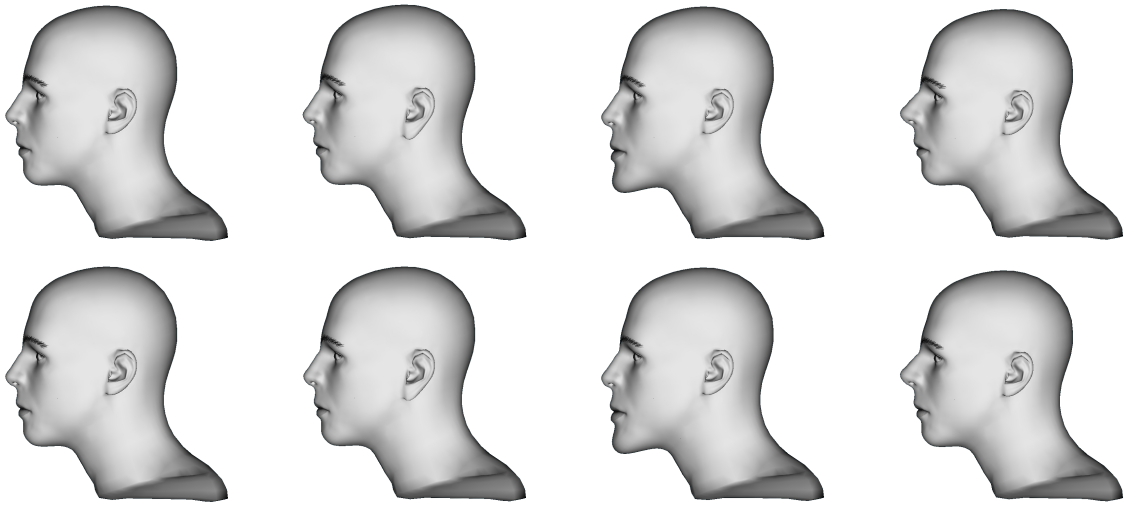


Figure 9.10: Head model deformation using feature points. Models created using manually inserted (top row) and automatically detected points (bottom row). Head deformation (from left to right): original model, prolonged ear lobe, prolonged chin, enlarged nose.

Figure 9.11 shows an example of a complex mesh deformation using multiple deformations of multiple face parts. The same sequence of transformations was applied to the model using manually defined points and to the model using the automatically detected points. The results of the deformations are presented in Figure 9.11. As can be seen in Figure 9.11, the results are very similar, although not exactly the same (e.g., note the length of the nose). This comes from the fact that the positions of some of the feature points are not strictly defined, but are just an approximation of an important region. To define a position of such points (e.g., the points on the side of the nose N_5 to N_8) can be a challenging task even for an expert placing the points manually.

Figure 9.11: Example of a transformed head model. Original model (left), model deformed using manually inserted points (middle) and model deformed using automatically detected feature points (right).

Chapter 10

Symmetry-aware registration of human faces

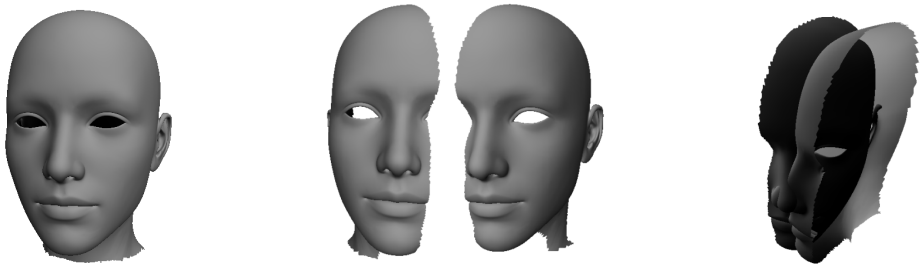
Global registration of partially overlapping parts of the same model is a well-known problem in the field of computer graphics and geometry processing. An object is scanned from several viewpoints, which leads to several partially overlapping data sets. The goal is to find transformations that align these data sets and obtain a complete model which can be further edited or directly printed. To register data sets, overlapping areas have to be found. This task may be simple for a human being, because our brain connects the parts based on previous knowledge of the shapes and types of objects we are registering. However, to automate this process is not straightforward, and many algorithms were proposed for this task.

A commonly used approach consists of two main steps - first, a simplified description of local neighborhoods is created, and then it is used to find a transformation that aligns the parts. The creation of simplified description is essential for the speed of algorithms. Usually, we do not calculate the description for every point, but only for a subset of points. This description is often called a feature vector (or descriptor). It is a vector with a fixed number of elements, which depends on the used algorithm. The feature vector essentially holds condensed information about the neighborhood. For example, the vector can be created using information from local angles, curvatures, normal vectors, distances, colors etc.

In the second step, the created feature vectors are used to find similar parts, based on the difference of the corresponding feature vectors (euclidean distance or dot product of normalized feature vectors can be used for this purpose). Identified similar parts are then used to find a transformation that aligns them. The alignment is often not accurate, and may be further improved by a local registration algorithm, such as Iterative Closest Point (ICP) [BM92].

We have presented several state-of-the-art solutions for a registration of partially overlapping parts in Chapter 6. However, for symmetrical models, such as human heads, these solutions yield the same feature vector for the symmetric, yet not identical parts of the same object, leading to an incorrect registration - see Figure 10.1. We have came across this problem while working on our research presented in Chapter 9. To overcome this problem, we have proposed a solution based on a symmetry-aware feature vector.

Our solution was presented in [PVK19].



(a) (b) (c)

Figure 10.1: a) Original data of a human head; b) Left and right part of the human head; c) An incorrect registration of the two parts.

10.1 Proposed solution

Based on our observations, we propose an algorithm to distinguishing left and right parts in models involving symmetries. The proposed algorithm is based on the current state-of-the-art method Fast Global Registration by Zhou et al. [ZPK16]. This approach uses the FPFH descriptor [RBB09], which is widely used and easy to implement. We have improved this feature vector by adding the symmetry information. The following registration process of finding the transformation itself is adopted from [ZPK16]. The proposed feature vector modification can be used with other registration algorithms as well, not only FPFH within FGR. Our proposed solution fits directly into other existing pipelines, since only the feature vector is changed and the rest of the registration process remains unaffected.

In our proposed solution, we have created two different approaches for differentiation of symmetry. The first approach is based on interaction of the local shape with a strongly orientation-dependent vector field. The second approach is based on curvature estimation and a local coordinate system created in the directions of the normal vectors and the extremal curvatures. Both versions lead to a signed value that represents direction of flux (a volume that goes through a surface in a certain direction) or a signed volume. Based on the sign, we can distinguish symmetric geometry parts.

10.1.1 Vector field

Our first approach is based on analyzing the local shape using a symmetry-aware vector field. Having a surface and a vector field that goes through it, we can calculate the flux. It describes the quantity which passes through a surface and based on the direction of the vector field and orientation of the surface, it has either positive or negative sign. We can utilize this knowledge to distinguish orientation of the local surface. Symmetric parts will have opposite signs and this can be used for the feature vector modification.

The initial derivation is based on a triangle mesh, however, this approach can be used for point clouds as well. First, we describe the solution based on a triangle mesh and later explain its modification for a point cloud.

Triangle mesh

We want to calculate the feature vector at a vertex P (equipped with the normal vector $\boldsymbol{n}_P$) of an input triangle mesh. We define a vector field as a vector function

$$\boldsymbol{v}(X) = (P - X) \times \boldsymbol{n}_P \tag{10.1}$$

The entire setup with a few points X_i can be seen in Figure 10.2.

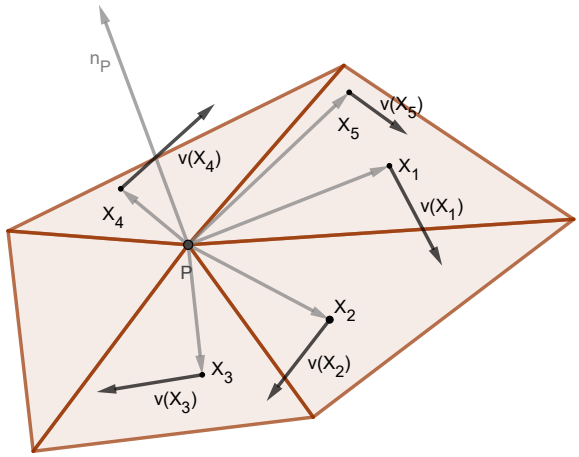


Figure 10.2: A local neighborhood with a few sample points X_i and vector field values at them.

Our goal is to integrate the dot product of this field with triangle normals over all neighboring triangles. For i-th triangle, we compute the following integral:

$$F_i = \int_{\triangle_i} \boldsymbol{v}(X) \cdot \boldsymbol{n}_i d\triangle_i \tag{10.2}$$

where $\boldsymbol{n}_i$ is the normal of the i-th triangle. Finally, we sum these values over all triangles in a local neighborhood as

$$F_s = \sum_i F_i \tag{10.3}$$

Points X in a triangle can be parameterized by scalars r and s as:

$$X = A_i + r(B_i - A_i) + s(C_i - A_i) \tag{10.4}$$

where A_i,B_i and C_i are vertices of the triangle that contains X, $r \in \mathbb{R}$ and $s \in \mathbb{R}$ are parameters fulfilling the conditions $r + s = 1$ and $r, s \geq 0$.

To calculate the integral (10.2), we have to change the bounds of the integral. From the integral over the area of triangle $\triangle_i$, we get the integral based on the previously defined parametric function in Equation (10.4) with parameters r and s:

$$F_i = \int_{r=0}^{1} \int_{s=0}^{1-r} \boldsymbol{v}(A_i + r(B_i - A_i) + s(C_i - A_i)) \cdot \boldsymbol{n}_i J_i \, dsdr \tag{10.5}$$

where J_i is the Jacobian associated with the change of the integral bounds and it is equal to $J_i = ||(B_i - A_i) \times (C_i - A_i)||$. The integral from Equation 10.5 can be rewritten using Equations 10.1 and 10.4 to:

$$\begin{aligned} F_i = J_i \int_{r=0}^{1} \int_{s=0}^{1-r} (&P \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i - A_i \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i \\ &-r(B_i - A_i) \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i \\ &-s(C_i - A_i) \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i) dsdr. \end{aligned} \tag{10.6}$$

To simplify Equation 10.6, we use a substitution:

$$\begin{aligned} \alpha_i &= (P - A_i) \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i, \\ \beta_i &= -(B_i - A_i) \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i, \\ \gamma_i &= -(C_i - A_i) \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i, \end{aligned} \tag{10.7}$$

which leads to

$$F_i = J_i \int_{r=0}^{1} \int_{s=0}^{1-r} (\alpha_i + \beta_i r + \gamma_i s) dsdr \tag{10.8}$$

that is rewritten as:

$$F_i = \frac{1}{6} J_i (3\alpha_i + \beta_i + \gamma_i). \tag{10.9}$$

After the removal of substitution from Equation 10.9, we end up with a solution

$$F_i = \frac{1}{2} J_i (P - T_i) \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i, \tag{10.10}$$

where T_i is a centroid of a triangle $A_i B_i C_i$, i.e. calculated as $T_i = \frac{1}{3}(A_i + B_i + C_i)$.

If we compute the triangle normal $\boldsymbol{t}_i$ as $(B_i - A_i) \times (C_i - A_i)$, then Equation 10.10 can be further simplified to

$$F_i = \frac{1}{2} (P - T_{A_i B_i C_i}) \times \boldsymbol{n}_P \cdot [(B_i - A_i) \times (C_i - A_i)]. \tag{10.11}$$

Point cloud modification

Application to point clouds without connectivity is possible if we interpret the data as a dual representation. We can think of the input points as centroids of virtual triangles, while the actual vertices are unknown. The idea is visualized in Figure 10.3: Figure 10.3a shows the standard way, where points P_i (blue color) of the cloud are taken as vertices of triangles and centroids (green color) are calculated from them, Figure 10.3b shows the dual interpretation, where the points P_i (blue color) are used as centroids directly.

(a) (b)

Figure 10.3: a) Point cloud P_i (blue points) as triangulation with green centroids of triangles; b) Points P_i of cloud (blue points) are used as centroids of "virtual" triangles. The yellow points are just for illustration of one possible "virtual" triangulation of the neighborhood.

For the point cloud modification, the overall triangulation is not required. We use only points P_i with their normal vectors that are used as normal vectors of the "virtual" triangles. The result can be improved if we have available non-normalized normal vectors that hold the information about the area of virtual triangle. This is similar to the Jacobian in the triangle mesh solution. If unit-length normal vectors are used, the virtual triangles are considered to have equal area. Based on this, we can directly use the Equation 10.11 and get the final solution for a point cloud as:

$$F_i = \frac{1}{2}(P - P_i) \times \boldsymbol{n}_P \cdot \boldsymbol{n}_i. \tag{10.12}$$

The result is influenced by the size of the neighborhood. A comparison of different neighborhood sizes is shown in Figure 10.4, which shows the point cloud modification. Near similar results were acquired using a triangle mesh, and therefore they are not included.

It can be seen that with the increasing neighborhood size, the separation of parts is more visible. The sign of the obtained values does not unambiguously identify the right/left side even for large neighborhoods. However, this is actually not our goal. More importantly, parts differing only by symmetry have a different sign and can therefore be distinguished.

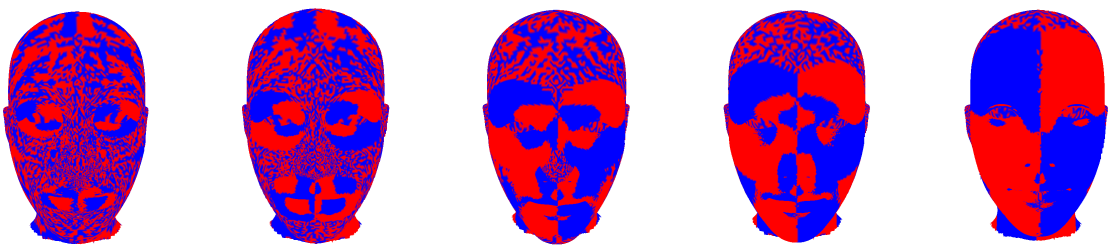


Figure 10.4: Comparison of different neighborhood sizes for vector field flux. Blue color indicates negative flux, red is used for positive values. Neighborhood sizes are taken in percents of bounding box size. From left to right: 7%, 14%, 21%, 28% and 42%

10.1.2 Curvature

The second proposed approach is based on curvature estimation. It can be used for a triangle mesh or a point cloud. Curvature is used to identify "interesting" points (minimal or maximal curvature). These points are used to create triplet of vectors (a local coordinate system) that defines a parallelepiped. Sign of its volume is used to identify symmetrical parts.

There are many algorithms for a curvature estimation of sampled surfaces (for a recent survey, see [Váš+16]) and the selection of one depends on estimation circumstances, such as type of input data, quality of the input, required performance and others.

Given the points of an input point cloud, where we want to calculate a feature vector, for every point P (equipped with the normal vector $\boldsymbol{n}$) we find its neighborhood. The size of the neighborhood can be selected. In our proposed solution, we have used the same size as for the feature vector calculation algorithm (for example [RBB09]).

Points with the maximal (P_{max}) and minimal (P_{min}) value of mean or Gaussian curvature are found within the neighborhood. These points are used to create a triplet of vectors: $\boldsymbol{n}$,$\boldsymbol{u} = P_{max} - P$,$\boldsymbol{v} = P_{min} - P$. The triplet can be used to calculate the signed volume $V_{s_{mean/Gauss}}$ of a parallelepiped, see Figure 10.5, as :

$$V_{s_{mean/Gauss}} = \boldsymbol{n} \cdot (\boldsymbol{u} \times \boldsymbol{v}), \tag{10.13}$$

where the *mean* or *Gauss* index is used to distinguish the type of extrema used for calculation of $\boldsymbol{u}$ and $\boldsymbol{v}$.

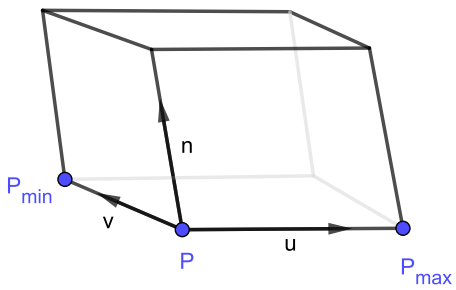


Figure 10.5: The triplet of vectors $\boldsymbol{n}$, $\boldsymbol{u}$ and $\boldsymbol{v}$ and indicated signed volume of a parallelepiped

The sign of the volume is swapped for reflected parts, because the points P_{min} and P_{max} are reflected as well. However, due to the differences in the neighborhoods for left and right part, the positions of extrema are not guaranteed to be the same. which can lead to incorrect results. To overcome this problem, a possible solution is to divide the space into bins, find the bins with the extremal averages of curvature and use the

centroids of the bins to construct the vector triplet. We have created bins from the volumetric sphere centered around the point P. A simplified 2D scenario can be seen in Figure 10.6.

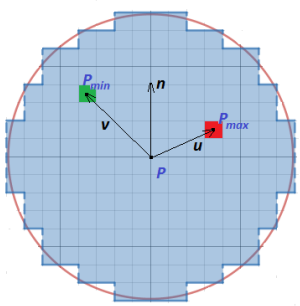


Figure 10.6: Bins around the center point P. Two bins with the extremal average curvatures are highlighted

The curvature based solution for distinguishing left and right part behaves differently from the vector field based approach. The comparison of detected left/right parts for different neighborhood sizes can be seen in Figure 10.7. We have used the same sizes as for vector field comparison in Figure 10.7 and both figures can be compared to each other. In case of the curvature-based solution with the increasing neighborhood size, the quality of the result is lower. For sizes above 25%, the results are generally incorrect and not usefull for symmetry-aware registration. This incorrectness is caused by the large smooth areas with few details where small numerical error can change the vector triplet and results in incorrectly signed area.

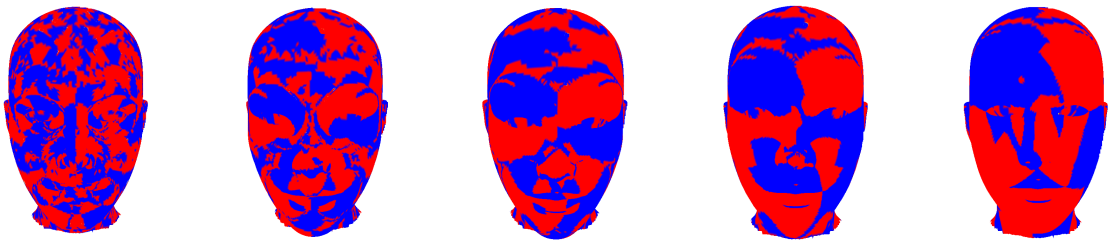


Figure 10.7: Comparison of different neighborhood sizes for signed volume. Blue color indicates negative volume, red is used for positive values. Neighborhood sizes are taken in percents of bounding box size. From left to right: 7%, 14%, 21%, 28% and 42%

10.1.3 Symmetry-aware feature vector

Once we have calculated the signed representation of the neighborhood, as presented in subsections 10.1.1 and 10.1.2, we can use it to modify the feature vector. For the simplest

modification, we can use only a sign of the calculated value, because we only need the sign to distinguish shape orientation. However, in our experiments, this approach was not very accurate. For detailed symmetric parts of a head, such as the ears, one side can contain both positive and negative values. This is often influenced by the size of the neighborhood used to calculate the signed representation. However, the same parts on the left and right part of the head have opposite signs.

There are many ways how to modify the feature vector. We have tested several possibilities that either combine together the two representations (presented in subsections 10.1.1 and 10.1.2) or use them separately. Attaching the value to the end (or beginning) of the feature vector offered only a small amount of new information and did not improve the results. Therefore we have also considered multiplying the feature vector to include the sign together with the value. In some cases, further attaching other calculated value to the end of already multiplied feature vector further improved the results.

From all the tests, we have selected the ones that provide the best overall quality. In every modification, we compute the FPFH [RBB09]) feature vector and use either F_s from the Vector Field (subsection 10.1.1), $V_{s_{mean}}$ from mean curvature or $V_{s_{Gauss}}$ from Gaussian curvature (subsection 10.1.2) to modify the vector as:

1. multiply FPFH by F_s and append F_s to the end
2. multiply FPFH by F_s and append $V_{s_{mean}}$ to the end
3. multiply FPFH by F_s and append F_s, $V_{s_{mean}}$ and $V_{s_{Gauss}}$ to the end
4. multiply FPFH by F_s and append $V_{s_{mean}}$ and $V_{s_{Gauss}}$ to the end

10.2 Experiments and results

We have used one of the current state-of-the-art methods [ZPK16] as a baseline. This method uses FPFH feature vector to obtain the final transformation and in some cases, it leads to an incorrect alignment of two parts if they contain symmetric parts. For example, if we try to register two overlapping parts of the human head, the ears are detected in both parts with similar feature vectors, which affects the final transformation. See again Figure 10.1 for such an example.

10.2.1 Test data

We need an automatic evaluation of registration quality. For this purpose, the correct registration has to be known. Therefore, instead of using scanned data, we have recreated partially overlapping parts from complete models. In our tests. We have used point clouds as well as their triangulations.

To create overlapping parts from a complete model and simulate a scanning device, we use the following steps.

- The plane of symmetry is found.
- The plane is randomly rotated around the coordinate system axes in the interval $< -30°, 30° >$.

- The plane is shifted in the positive or negative direction of its normal vector. Shift distance depends on the model size and the size of the overlap we want to achieve.
- Vertices in one half-space of the plane are discarded.

However, in this scenario, data can be matched 1:1 - e.g. there are points in both halves that can be exactly matched. To overcome this, we have added noise to the data. For triangle meshes, Loop [Loo87] subdivision scheme followed by a mesh simplification with added Gaussian noise was used. For point clouds, we have only added Gaussian noise.

If used data were a point cloud, only points within the tested radius were used for calculations. For triangle meshes, the radius usually intersects triangles. In this case, we have split the triangles and used only the triangle parts fully included in the neighborhood radius.

10.2.2 Comparison metric

To compare the registration results, we have used a metric based on [Pot+06]. From the two parts that are used for the registration, one (called a model P) is at a fixed position and the other (called model Q) is transformed with a random translation and rotation. Inverse of this transformation will put the model to a position Q_1 in which it is correctly registered with the model P.

We have obtained the registration matrix from the registration algorithm ([ZPK16] in our case) and transformed the model Q with this matrix which led to the position Q_2. For a perfect registration, Q_1 and Q_2 are the same. To measure the deviation from a perfect registration, we compute distances between the corresponding vertices of Q_1 and Q_2. The lower the distance, the better the alignment of Q_2 with respect to its true position Q_1 is. The sum of distances is divided by the number of vertices and normalized by the data radius, making the sum scale-independent, although not necessarily in the $< 0, 1 >$ interval.

Based on our observations, the resulting values can be interpreted as follows: values under 0.1 can be considered a correct registration result, values between 0.1 and 0.6 are registered very roughly but the overall shape can be recognized and, finally, values above 0.6 indicate an incorrect registration and the result is on par with a random matrix.

10.2.3 Tests

The core of the tests is based on the FPFH descriptor. We have used several radii that are based on the average number of points that fall within. We have started with 10 points and ended with a neighborhood of the size 160 points, using a step of 10 points. From the number of points, we have calculated the average radius size of the geometry and this radius was used for FPFH. The radius was different for each tested model, depending on the model scale and sampling density.

In our experiments, we have used the same radius for both FPFH calculation and for the calculation of neighborhoods for the proposed methods from subsections 10.1.1 and 10.1.2. We have tested different sizes of neighborhoods as well. However, for smaller neighborhoods the results were often similar to a flat surface. There were certain precision

improvements with neighborhoods of a larger size, but the computation times were longer. As a trade-off between quality and speed we have selected using neighborhoods of the same size.

We have created an automated test scenario, where the input model is randomly divided into two overlapping parts (see subsection 10.2.1). For every split, we have computed metrics for every variation of the feature vector modification proposed in subsection 10.1.3. The basic solution taken from [ZPK16] was able to correctly register only about 40% of our input test cases, while the rest was registered incorrectly (with metric values being above 0.1).

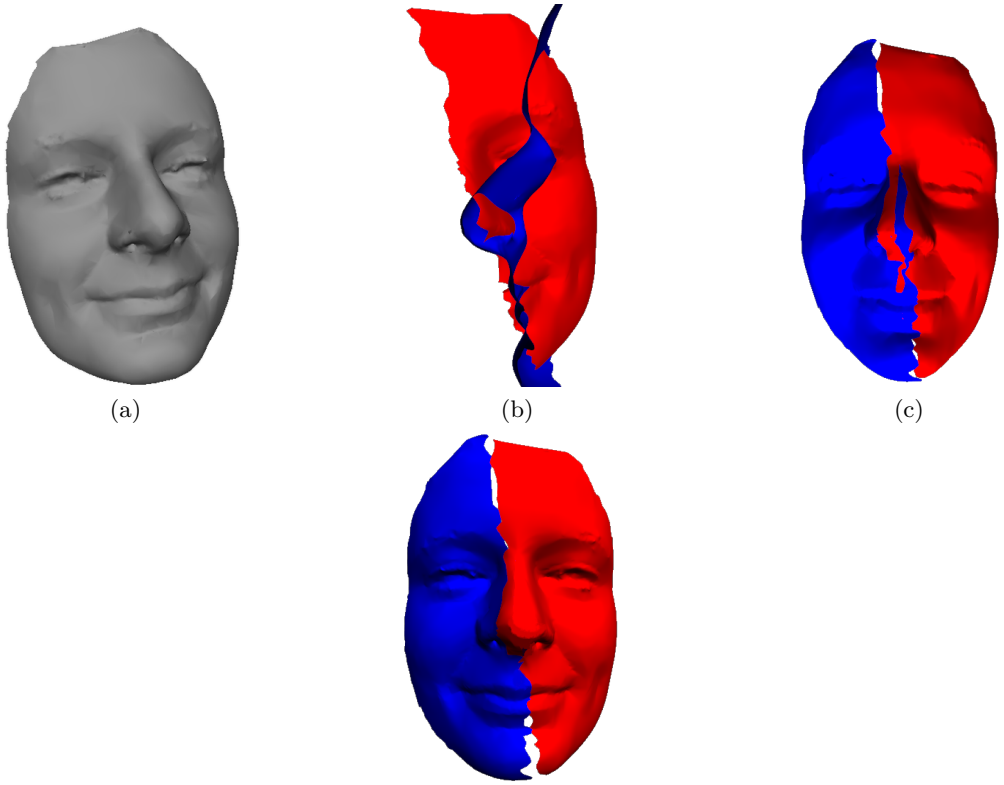
We have conducted several thousand tests with different models of human heads and faces. For the tests, we have used a triangle mesh and a point cloud representation of the input model. The overall results were roughly the same for both approaches and correspond with averaged results. These overall averaged results are presented in Table 10.1. The method number in the first column of the table corresponds with the feature vector modification method number from subsection 10.1.3. The symbol " − " marks the original [ZPK16] method without any modification. The table shows the percentage of correct registrations for three different neighborhood sizes (10, 60 and 150). Note that the globally low success rate in general is caused by an automated data creation, which often leaves only a small overlap.

Method	Size 10	Size 60	Size 150
-	41%	43%	34%
1	44%	42%	28%
2	48%	45%	33%
3	51%	50%	31%
4	55%	49%	36%

Table 10.1: Comparison of successfully registered input cases based on different points count in the neighborhood

From the tests, we have observed that with the increasing neighborhood size, the results were improving only to a certain threshold. The quality of the registration began to decrease for a neighborhood calculated from around 60 points, probably because the descriptor was constructed from a too large area. For a human head, it often means that a large part of the descriptor is based on smooth surfaces located on cheeks, forehead or chin. In these cases, the basic FGR algorithm offers better results and the proposed improvement often leads to a worse registration outcome.

The visual comparison of results can be seen in Figure 10.8. The original data (Figure 10.8a) differs from the best registration result (Figure 10.8d). This difference is caused by the data modification described in Subsection 10.2.1. The difference between Figure 10.8c and Figure 10.8d is mainly in the nose area. The two parts in Figure 10.8c are registered very roughly and there are many intersections of the two parts along their overlap. In Figure 10.8d the registration is correct with a smooth transition from one part to another. The missing parts at the top and bottom of the model are caused by the automated data creation.



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

Figure 10.8: Visual comparison of results; a) Original data; b) Incorrect registration using only [ZPK16] with $error = 1.578$; c) The proposed registration with $error = 0.192$; d) The proposed registration with $error = 0.074$

10.2.4 Limitations

The proposed algorithm has certain limitations. Some of them are globally related to the registration itself. If the overlap of two parts is too small, the resulting registration is often incorrect. The same goes for too much noise in the data.

The main problem of the proposed solution is selecting optimal feature vector modification. Depending on the data, one choice may be considerably better than others. This is currently solved manually, when the user has to check if the registration is correct. If not, the user must dismiss the registration result and another version of feature vector is automatically used. In the future research, we would like to focus on this part and create an automated system that can distinguish an incorrect registration automatically and eventually choose another feature vector. We have already experimented with solutions based on projections of geometry to 2D plane and comparison of the projected depth values, but the results are currently not reliable.

Another problem are the input data themselves. Our current solution is only suitable for a limited set of data - scans of human heads. We have also tested general models with symmetries, however, the quality of registration was not globally improved with our algorithm. In some cases, the results were correctly registered with results similar to the ones for human heads, but most of the time there was no improvement.

Chapter 11

Conclusion

This thesis presents the problem of shape characteristics with the main focus on curvature-based solutions. Curvature is one of the basic geometry descriptors and is widely used. Based on curvature we are able to better understand the surface properties and use them for various applications. These can be visualizations, registration, geometry deformations etc.

One of the contributions of this thesis is an algorithm for real-time curvature estimation that allows an interactive data editing with an immediate result. This field is not widely researched but still important. During interactive geometry editing, users prefer fast estimation over the overall quality. Our proposed algorithm achieves the best performance with respect to the quality among the existing solutions.

From our point of view, the most important contribution is a high-quality curvature estimation. This solution is targeted for a high quality data and for them, the results are significantly improved over existing algorithms. This algorithm can be used as a last step after the user has finished the interactive geometry editing and wants the highest possible curvature estimation at a price of a longer time.

A practical use of the proposed curvature estimation algorithms is presented in our remaining methods. Curvature can be used for a detection of feature points. In our presented algorithm, we use curvature to detect points on a human face. These points are used as a base for anchor points for deformations. These deformations have a wide range of use cases, such as an interactive editing of players characters in computer games, 3D modeling, an automatic aging process etc.

At last, a symmetry-aware method for the registration of a human head model was presented. This approach is based on our previous research where the data were obtained from a scanning device. However, in many cases, the scanned parts have to be registered manually, since the automatic solutions failed for symmetrical parts. The proposed algorithms deal with this problem by the use of curvature and vector fields. The major advantage of this approach is that the improvement can be included as an additional information to existing state-of-the-art feature vectors and the rest of the registration process can be preserved.

There is a lot of space for a future work. Each of the presented algorithms has some weaknesses and limitation. These issues are discussed in separated sections of each algorithm description. The main issues are usually connected with a narrow specialization

of methods. For example, the symmetry registration is targeted to human head datasets, the precise curvature estimation requires robust normal vectors etc. These limitations are partially based on the problem definition where the proposed method is a working solution for the given type of input data.

Bibliography

[Ai+17] S. Ai, L. Jia, C. Zhuang, and H. Ding. "A Registration Method for 3D Point Clouds with Convolutional Neural Network". In: *Intelligent Robotics and Applications*. Ed. by Y. Huang, H. Wu, H. Liu, and Z. Yin. Cham: Springer International Publishing, 2017, pp. 377–387. ISBN: 978-3-319-65298-6.

[All+03] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. "Anisotropic Polygonal Remeshing". In: *ACM Trans. Graph.* 22.3 (July 2003), pp. 485–493. ISSN: 0730-0301.

[AMHH08] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering 3rd Edition*. Natick, MA, USA: A. K. Peters, Ltd., 2008, p. 1045. ISBN: 987-1-56881-424-7.

[AU09] E. Akagündüz and Í. Ulusoy. "Scale and orientation invariant 3D interest point extraction using HK curvatures". In: *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. Sept. 2009, pp. 697–702.

[Bay+08] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. "Speeded-Up Robust Features (SURF)". In: *Computer Vision and Image Understing* 110.3 (June 2008), pp. 346–359. ISSN: 1077-3142.

[BHP01] G. Barequet and S. Har-Peled. "Efficiently Approximating the Minimum-Volume Bounding Box of a Point Set in Three Dimensions". In: *Journal of Algorithms* 38.1 (2001), pp. 91 –109. ISSN: 0196-6774.

[BM92] P. J. Besl and N. D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992), pp. 239–256. ISSN: 0162-8828.

[Bos+12] M. Boschiroli, C. Fünfzig, L. Romani, and G. Albrecht. "{G1} rational blend interpolatory schemes: A comparative study". In: *Graphical Models* 74.1 (2012), pp. 29 –49. ISSN: 1524-0703.

[Bri07] R. Bridson. "Fast Poisson Disk Sampling in Arbitrary Dimensions". In: *ACM SIGGRAPH 2007 Sketches*. SIGGRAPH '07. San Diego, California: ACM, 2007. ISBN: 978-1-4503-4726-6.

[Bro11] A. M. Bronstein. "Spectral descriptors for deformable shapes". In: *CoRR* abs/1110.5015 (2011).

[BSD08] L. Bavoil, M. Sainz, and R. Dimitrov. "Image-space Horizon-based Ambient Occlusion". In: *ACM SIGGRAPH 2008 Talks*. SIGGRAPH '08. Los Angeles, California: ACM, 2008, 22:1–22:1. ISBN: 978-1-60558-343-3.

[BW07] C. H. Batagelo and S. Wu. “Estimating curvatures and their derivatives on meshes of arbitrary topology from sampling directions”. In: *The Visual Computer* 23.9 (2007), pp. 803–812. ISSN: 1432-2315.

[Car+01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. “Reconstruction and Representation of 3D Objects with Radial Basis Functions”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. New York, NY, USA: ACM, 2001, pp. 67–76. ISBN: 1-58113-374-X.

[CJG09] G. Cipriano, G. N. P. Jr., and M. Gleicher. “Multi-Scale Surface Descriptors”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (Nov. 2009), pp. 1201–1208. ISSN: 1077-2626.

[CP05] F. Cazals and M. Pouget. “Estimating differential quantities using polynomial fitting of osculating jets”. In: *Computer Aided Geometric Design* 22.2 (2005), pp. 121 –146. ISSN: 0167-8396.

[CSJ05] D. Colbry, G. Stockman, and A. Jain. “Detection of Anchor Points for 3D Face Verification”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) - Workshops*. June 2005, pp. 118–118.

[CSM03] D. Cohen-Steiner and J. M. Morvan. “Restricted Delaunay Triangulations and Normal Cycle”. In: *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*. SCG ’03. San Diego, California, USA: ACM, 2003, pp. 312–321. ISBN: 1-58113-663-3.

[Don+17] Z. Dong, B. Yang, Y. Liu, F. Liang, B. Li, and Y. Zang. “A novel binary shape context for 3D local surface description”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 130 (2017), pp. 431 –452. ISSN: 0924-2716.

[Elm+10] A. L. Elmustapha, A. E. Oirrak, D. Aboutajdine, M. Daoudi, and M. N. Kaddioui. “A 3-D Search engine based on Fourier series”. In: *Computer Vision and Image Understanding* 114.1 (2010), pp. 1 –7. ISSN: 1077-3142.

[Fis89] R. B. Fisher. *From Surface To Objects: Computer Vision and Three Dimensional Scene Analysis*. John Wiley and Sons, 1989.

[Fro+04] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik. “Recognizing Objects in Range Data Using Regional Point Descriptors”. In: *Computer Vision - ECCV 2004*. Ed. by T. Pajdla and J. Matas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 224–237. ISBN: 978-3-540-24672-5.

[Fün+08] C. Fünfzig, K. Müller, D. Hansford, and G. Farin. “PNG1 Triangles for Tangent Plane Continuous Surfaces on the GPU”. In: *Proceedings of Graphics Interface 2008*. GI ’08. Windsor, Ontario, Canada: Canadian Information Processing Society, 2008, pp. 219–226. ISBN: 978-1-56881-423-0.

[Gat+05] T. Gatzke, C. Grimm, M. Garland, and S. Zelinka. “Curvature Maps for Local Shape Comparison”. In: *Proceedings of the International Conference on Shape Modeling and Applications 2005*. SMI ’05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 246–255. ISBN: 0-7695-2379-X.

[GCO06] R. Gal and D. Cohen-Or. “Salient Geometric Features for Partial Shape Matching and Similarity”. In: *ACM Trans. Graph.* 25.1 (Jan. 2006), pp. 130–150. ISSN: 0730-0301.

[Gel+05] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann. "Robust Global Registration". In: *Proceedings of the Third Eurographics Symposium on Geometry Processing*. SGP '05. Vienna, Austria: Eurographics Association, 2005. ISBN: 3-905673-24-X.

[GI04] J. Goldfeather and V. Interrante. "A Novel Cubic-order Algorithm for Approximating Principal Direction Vectors". In: *ACM Trans. Graph.* 23.1 (Jan. 2004), pp. 45–63. ISSN: 0730-0301.

[GJ+10] G. Guennebaud, B. Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[Gol05] R. Goldman. "Curvature formulas for implicit curves and surfaces". In: *Computer Aided Geometric Design* 22.7 (2005). Geometric Modelling and Differential Geometry, pp. 632 –658. ISSN: 0167-8396.

[Gra97] A. Gray. "Surfaces in 3-Dimensional Space via Mathematica". In: *Modern Differential Geometry of Curves and Surfaces with Mathematica*. 2nd. Boca Raton, FL, USA: CRC Press, Inc., 1997. Chap. 17, pp. 394–401. ISBN: 0849371643.

[Gri+12] W. Griffin, Y. Wang, D. Berrios, and M. Olano. "Real-Time GPU Surface Curvature Estimation on Deforming Meshes and Volumetric Data Sets". In: *IEEE Transactions on Visualization and Computer Graphics* 18.10 (2012), pp. 1603–1613. ISSN: 1077-2626.

[GSM15] S. Z. Gilani, F. Shafait, and A. Mian. "Shape-based automatic detection of a large number of 3D facial landmarks". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 4639–4648.

[Guo+13] Y. Guo, F. Sohel, M. Bennamoun, M. Lu, and J. Wan. "TriSI: A Distinctive Local Surface Descriptor for 3D Modeling and Object Recognition". In: *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications (VISIGRAPP 2013)*. 2013, pp. 86–93. ISBN: 978-989-8565-46-4.

[Guo+14] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, and J. Wan. "3D Object Recognition in Cluttered Scenes with Local Surface Features: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (Nov. 2014), pp. 2270–2287. ISSN: 0162-8828.

[Guo+16] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, J. Wan, and N. M. Kwok. "A Comprehensive Performance Evaluation of 3D Local Feature Descriptors". In: *International Journal of Computer Vision* 116.1 (Jan. 2016), pp. 66–89. ISSN: 1573-1405.

[Hei+11] P. Heider, A. Pierre-Pierre, R. Li, and C. Grimm. "Local Shape Descriptors, a Survey and Evaluation". In: *Proceedings of the 4th Eurographics Conference on 3D Object Retrieval*. 3DOR '11. Llandudno, UK: Eurographics Association, 2011, pp. 49–56. ISBN: 978-3-905674-31-6.

[HF11] G. R. D. H. Foulds. "Three-dimensional shape descriptors and matching procedures". In: *WSCG '2011: Communication Papers Proceedings: The 19th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, p. 1-8.* 2011, pp. 1–8.

[HKM11] T. Hattori, H. Kubo, and S. Morishima. “Real Time Ambient Occlusion by Curvature Dependent Occlusion Function”. In: *SIGGRAPH Asia 2011 Posters*. SA ’11. Hong Kong, China: ACM, 2011, 48:1–48:1. ISBN: 978-1-4503-1137-3.

[HKM12] T. Hattori, H. Kubo, and S. Morishima. “Curvature-approximated estimation of real-time ambient occlusion”. In: *GRAPP 2012 IVAPP 2012 - Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications*. GRAPP 2012 IVAPP 2012 - Proceedings of the International Conference on Computer Graphics Theory et al., 2012, pp. 268–273. ISBN: 9789898565020.

[Hol+15] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu, and S. Behnke. “Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D”. In: *IEEE Robotics Automation Magazine* 22.4 (Dec. 2015), pp. 110–124. ISSN: 1070-9932.

[HP11] K. Hildebrandt and K. Polthier. “Generalized shape operators on polyhedral surfaces”. In: *Computer Aided Geometric Design* 28.5 (2011), pp. 321 –343. ISSN: 0167-8396.

[JH99] A. E. Johnson and M. Hebert. “Using spin images for efficient object recognition in cluttered 3D scenes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.5 (May 1999), pp. 433–449. ISSN: 0162-8828.

[Jin+05] S. Jin, R. B. Fisher, R. Lewis, and D. West. “A comparison of algorithms for vertex normal computation”. In: *The Visual Computer* 21.1 (2005), pp. 71–82. ISSN: 1432-2315.

[Ju+02] T. Ju, F. Losasso, S. Schaefer, and J. Warren. “Dual Contouring of Hermite Data”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 339–346. ISSN: 0730-0301.

[JZQ16] Y. Jiaqi, C. Zhiguo, and Z. Qian. “A fast and robust local descriptor for 3D point cloud registration”. In: *Information Sciences* 346-347 (2016), pp. 163 –179. ISSN: 0020-0255.

[Kal+07] E. Kalogerakis, P. Simari, D. Nowrouzezahrai, and K. Singh. “Robust Statistical Estimation of Curvature on Discretized Surfaces”. In: *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*. SGP ’07. Barcelona, Spain: Eurographics Association, 2007, pp. 13–22. ISBN: 978-3-905673-46-3.

[Kal+09] E. Kalogerakis, D. Nowrouzezahrai, P. Simari, J. Mccrae, A. Hertzmann, and K. Singh. “Data-driven Curvature for Real-time Line Drawing of Dynamic Scenes”. In: *ACM Trans. Graph.* 28.1 (Feb. 2009), 11:1–11:13. ISSN: 0730-0301.

[KD92] J. J. Koenderink and A. J. van Doorn. “Surface shape and curvature scales”. In: *Image and Vision Computing* 10.8 (1992), pp. 557 –564. ISSN: 0262-8856.

[Kla+09] K. Klasing, D. Althoff, D. Wollherr, and M. Buss. “Comparison of surface normal estimation methods for range sensing applications”. In: *2009 IEEE International Conference on Robotics and Automation*. May 2009, pp. 3206–3211.

[Kno+10] J. Knopp, M. Prasad, G. Willems, R. Timofte, and L. V. Gool. "Hough Transform and 3D SURF for Robust Three Dimensional Classification". In: *Proceedings of the 11th European Conference on Computer Vision: Part VI*. ECCV'10. Heraklion, Crete, Greece: Springer-Verlag, 2010, pp. 589–602. ISBN: 3-642-15566-9, 978-3-642-15566-6.

[Kok+12] I. Kokkinos, M. M. Bronstein, R. Litman, and A. M. Bronstein. "Intrinsic shape context descriptors for deformable shapes". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. June 2012, pp. 159–166.

[Kre91] E. Kreyszig. "Differential Geometry". In: 1st. Dover, NY: Dover Books on Methematics, 1991, pp. 34–36.

[KS12] E. Kuwert and R. Schätzle. "The Willmore functional". In: *Topics in Modern Regularity Theory*. Ed. by G. Mingione. Pisa: Edizioni della Normale, 2012, pp. 1–115. ISBN: 978-88-7642-427-4.

[KZK17] M. Khoury, Q.-Y. Zhou, and V. Koltun. "Learning Compact Geometric Features". In: *International Conference on Computer Vision (ICCV)*. 2017.

[LC87] E. W. Lorensen and H. E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 163–169. ISBN: 0-89791-227-6.

[Lev15] J. Levallois. *Integral invariant curvature estimator 2D/3D*. `http://dgtal.org/doc/0.9/moduleIntegralInvariant.html`. Dec. 2015.

[Lin+17] B. Lin, F. Wang, Y. Sun, W. Qu, Z. Chen, and S. Zhang. "Boundary points based scale invariant 3D point feature". In: *Journal of Visual Communication and Image Representation* 48 (2017), pp. 136 –148. ISSN: 1047-3203.

[Loo87] C. Loop. "Smooth Subdivision Surfaces Based on Triangles". PhD thesis. Department of Mathematics, The University of Utah, Masters Thesis, Jan. 1987.

[Low04] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 0920-5691.

[LVJ05] C. H. Lee, A. Varshney, and D. W. Jacobs. "Mesh Saliency". In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. Los Angeles, California: ACM, 2005, pp. 659–666.

[Mar+10] Z. C. Marton, D. Pangercic, N. Blodow, J. Kleinehellefort, and M. Beetz. "General 3D modelling of novel objects from a single view". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2010, pp. 3700–3705.

[Max+11] A. Maximo, R. Patro, A. Varshney, and R. Farias. "A robust and rotationally invariant local surface descriptor with applications to non-local mesh processing". In: *Graphical Models* 73.5 (2011), pp. 231 –242. ISSN: 1524-0703.

[Max99] N. Max. "Weights for Computing Vertex Normals from Facet Normals". In: *J. Graph. Tools* 4.2 (Mar. 1999), pp. 1–6. ISSN: 1086-7651.

[Mel+13] N. Mellado, P. Barla, G. Guennebaud, P. Reuter, and G. Duquesne. "Screen-space Curvature for Production-quality Rendering and Compositing". In: *ACM SIGGRAPH 2013 Talks*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 42:1–42:1. ISBN: 978-1-4503-2344-4.

[Mel15] N. Mellado. *Screen Space Curvature using Cuda/C++ (algorithm implementation from Patate library)*. English. `http://patate.gforge.inria.fr/html/grenaille_example_cxx_ssc_page.html`. Inria, July 2015.

[Mey+03] M. Meyer, M. Desbrun, P. Schröder, and A. Barr. "Discrete Differential-Geometry Operators for Triangulated 2-Manifolds". English. In: *Visualization and Mathematics III*. Ed. by H.-C. Hege and K. Polthier. Mathematics and Visualization. Springer Berlin Heidelberg, 2003, pp. 35–57. ISBN: 978-3-642-05682-6.

[MGT14] C. M. Mateo, P. Gil, and F. Torres. "A performance evaluation of surface normals-based descriptors for recognition of objects using CAD-models". In: *Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on*. Vol. 02. Sept. 2014, pp. 428–435.

[MGV09] I. Macedo, J. P. Gois, and L. Velho. "Hermite Interpolation of Implicit Surfaces with Radial Basis Functions". In: *Computer Graphics and Image Processing (SIBGRAPI), 2009 XXII Brazilian Symposium on*. Oct. 2009, pp. 1–8.

[MGV11] I. Macedo, J. P. Gois, and L. Velho. "Hermite Radial Basis Functions Implicits". In: *Computer Graphics Forum* 30.1 (2011), pp. 27–42. ISSN: 1467-8659.

[Mit07] M. Mittring. "Finding Next Gen: CryEngine 2". In: *ACM SIGGRAPH 2007 Courses*. SIGGRAPH '07. San Diego, California: ACM, 2007, pp. 97–121. ISBN: 978-1-4503-1823-5.

[MK14] P. Martínek and I. Kolingerová. "Deformation method for 3d identikit creation". In: *Computer Graphics Theory and Applications (GRAPP), 2014 International Conference on*. IEEE. 2014, pp. 1–8.

[NC09] P. Nair and A. Cavallaro. "3-D Face Detection, Landmark Localization, and Registration Using a Point Distribution Model". In: *IEEE Transactions on Multimedia* 11.4 (June 2009), pp. 611–623. ISSN: 1520-9210.

[PC16] M. Pouget and F. Cazals. "Estimation of Local Differential Properties of Point-Sampled Surfaces". In: *CGAL User and Reference Manual*. 4.8. CGAL Editorial Board, 2016.

[Pot+06] H. Pottmann, Q. X. Huang, Y. L. Yang, and S. M. Hu. "Geometry and Convergence Analysis of Algorithms for Registration of 3D Shapes". In: *International Journal of Computer Vision* 67.3 (May 2006), pp. 277–296. ISSN: 1573-1405.

[Pot+07] H. Pottmann, J. Wallner, Y. L. Yang, Y. Lai, and S. M. Hu. "Principal curvatures from the integral invariant viewpoint". In: *Computer Aided Geometric Design* 24.8 - 9 (2007). Discrete Differential Geometry, pp. 428 –442. ISSN: 0167-8396.

[Pot+09] H. Pottmann, J. Wallner, H. B.Q.-X. Huang, and Y. L. Yang. "Integral invariants for robust geometry processing". In: *Computer Aided Geometric Design* 26.1 (2009), pp. 37 –60. ISSN: 0167-8396.

[PP18] D. Prantl and M. Prantl. “Website traffic measurement and rankings: competitive intelligence tools examination”. In: *International Journal of Web Information Systems* 14.4 (2018), pp. 423–437. eprint: `https://doi.org/10.1108/IJWIS-01-2018-0001`.

[Pra+17] M. Prantl, V. Skorkovská, P. Martínek, and I. Kolingerová. “Curvature-Based Feature Detection for Human Head Modeling”. In: vol. 108. International Conference on Computational Science, {ICCS} 2017, 12-14 June 2017, Zurich, Switzerland. 2017, pp. 2323 –2327.

[PV18] M. Prantl and L. Váša. “Estimation of differential quantities using Hermite RBF interpolation”. In: *The Visual Computer* 34.12 (Dec. 2018), pp. 1645–1659. ISSN: 1432-2315.

[PVK16] M. Prantl, L. Váša, and I. Kolingerová. “Fast Screen Space Curvature Estimation on GPU”. In: *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016)*. Vol. 1. Best student paper award. Feb. 2016, pp. 151–160. ISBN: 978-989-758-175-5.

[PVK17] M. Prantl, L. Váša, and I. Kolingerová. “Screen Space Curvature and Ambient Occlusion”. In: *Computer Vision, Imaging and Computer Graphics Theory and Applications*. Ed. by J. Braz, N. Magnenat-Thalmann, P. Richard, L. Linsen, A. Telea, S. Battiato, and F. Imai. Cham: Springer International Publishing, 2017, pp. 51–71. ISBN: 978-3-319-64870-5.

[PVK19] M. Prantl, L. Váša, and I. Kolingerová. “Symmetry-aware Registration of Human Faces”. In: *Proceedings of the 14th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2019)*. Vol. 1. Feb. 2019, pp. 185–192. ISBN: 978-989-758-354-4.

[RB05] A. Razdan and M. Bae. “Curvature estimation scheme for triangle meshes using biquadratic Bézier patches”. In: *Computer-Aided Design* 37.14 (2005), pp. 1481 –1491. ISSN: 0010-4485.

[RBB09] R. B. Rusu, N. Blodow, and M. Beetz. “Fast Point Feature Histograms (FPFH) for 3D registration”. In: *Robotics and Automation, 2009. ICRA ’09. IEEE International Conference on Robotics and Automation*. May 2009, pp. 3212–3217.

[RC11] R. B. Rusu and S. Cousins. “3D is here: Point Cloud Library (PCL)”. In: *2011 IEEE International Conference on Robotics and Automation*. May 2011, pp. 1–4.

[Rob01] A. Roberts. “Curvature attributes and their application to 3D interpreted horizons”. In: *First Break* 19.2 (2001), pp. 85–100. ISSN: 1365-2397.

[Rus+08] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz. “Persistent point feature histograms for 3d point clouds”. In: *In Proceedings of the 10th International Conference on Intelligent Autonomous Systems*. 2008.

[Rus+10] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. “Fast 3D recognition and pose using the Viewpoint Feature Histogram”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2010, pp. 2155–2162.

[Rus04] S. Rusinkiewicz. “Estimating Curvatures and Their Derivatives on Triangle Meshes”. In: *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2Nd International Symposium*. 3DPVT ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 486–493. ISBN: 0-7695-2223-8.

[Rus16] R. B. Rusu. *Point Cloud Library Tutorials - PFH and FPFH*. `http://www.pointclouds.org/documentation/tutorials/fpfh_estimation.php`. Mar. 2016.

[SAC12] P. Szeptycki, M. Ardabilian, and L. Chen. “Nose tip localization on 2.5D facial models using differential geometry based point signatures and SVM classifier”. In: *Biometrics Special Interest Group (BIOSIG), 2012 BIOSIG - Proceedings of the International Conference of the*. Sept. 2012, pp. 1–12.

[SCP10] A. Salazar, A. Cerón, and F. Prieto. “3D Curvature-based Shape Descriptors for Face Segmentation: An Anatomical-based Analysis”. In: *Proceedings of the 6th International Conference on Advances in Visual Computing - Volume Part III*. ISVC’10. Las Vegas, NV, USA: Springer-Verlag, 2010, pp. 349–358. ISBN: 3-642-17276-8, 978-3-642-17276-2.

[SGS14] I. Sipiran, R. Gregor, and T. Schreck. “Approximate Symmetry Detection in Partial 3D Meshes”. In: *Computer Graphics Forum* 33.7 (2014), pp. 131–140. ISSN: 1467-8659.

[Sha+13] S. A. A. Shah, M. Bennamoun, F. Boussaid, and A. A. El-Sallam. “3D-Div: A novel local surface descriptor for feature matching and pairwise range image registration”. In: *Image Processing (ICIP), 2013 20th IEEE International Conference on*. Sept. 2013, pp. 2934–2938.

[Sko+17] V. Skorkovská, M. Prantl, P. Martínek, and I. Kolingerová. “Erosion-Inspired Simulation of Aging for Deformation-Based Head Modelingdownload this paper”. In: vol. 108. International Conference on Computational Science, {ICCS} 2017, 12-14 June 2017, Zurich, Switzerland. 2017, pp. 425 –434.

[SOG09] J. Sun, M. Ovsjanikov, and L. Guibas. “A Concise and Provably Informative Multi-scale Signature Based on Heat Diffusion”. In: *Proceedings of the Symposium on Geometry Processing*. SGP ’09. Berlin, Germany: Eurographics Association, 2009, pp. 1383–1392.

[Son15] P. Song. “Local voxelizer: A shape descriptor for surface registration”. In: *Computational Visual Media* 1.4 (2015), pp. 279–289. ISSN: 2096-0662.

[Sta] *The Stanford 3D Scanning Repository*. `https://graphics.stanford.edu/data/3Dscanrep/`. Feb. 2016.

[Tau95] G. Taubin. “Estimating the tensor of curvature of a surface from a polyhedral approximation”. In: *Computer Vision, 1995. Proceedings., Fifth International Conference on*. June 1995, pp. 902–907.

[The+04] H. Theisel, C. Rossi, R. Zayer, and H. P. Seidel. “Normal based estimation of the curvature tensor for triangular meshes”. In: *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*. Oct. 2004, pp. 288–297.

[TSDS10] F. Tombari, S. Salti, and L. Di Stefano. “Unique Shape Context for 3D Data Description”. In: *Proceedings of the ACM Workshop on 3D Object Retrieval*. 3DOR ’10. Firenze, Italy: ACM, 2010, pp. 57–62. ISBN: 978-1-4503-0160-2.

[TSS10] F. Tombari, S. Salti, and L. D. Stefano. "Unique Signatures of Histograms for Local Surface Description". In: *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*. ECCV'10. Heraklion, Crete, Greece: Springer-Verlag, 2010, pp. 356–369. ISBN: 3-642-15557-X, 978-3-642-15557-4.

[TV08] J. W. Tangelder and R. C. Veltkamp. "A Survey of Content Based 3D Shape Retrieval Methods". In: *Multimedia Tools Appl.* 39.3 (Sept. 2008), pp. 441–471. ISSN: 1380-7501.

[Unk17] Unknown. *Normal, Gaussian and Mean Curvatures at the Regular Point of a Surface*. `http://www.grad.hr/itproject_math/Links/sonja/gausseng/introduction/introduction.html`. Feb. 2017.

[Vai13] R. Vaillant. *Recipe for implicit surface reconstruction with HRBF*. `http://rodolphe-vaillant.fr/?e=12`. Mar. 2013.

[Vla+01] A. Vlachos, J. Peters, C. Boyd, and J. L. Mitchell. "Curved PN Triangles". In: *Proceedings of the 2001 Symposium on Interactive 3D Graphics*. I3D '01. New York, NY, USA: ACM, 2001, pp. 159–166. ISBN: 1-58113-292-1.

[Váš+16] L. Váša, P. Vaněček, M. Prantl, V. Skorkovská, P. Martínek, and I. Kolingerová. "Mesh Statistics for Robust Curvature Estimation". In: *Computer Graphics Forum* 35.5 (2016), pp. 271–280. ISSN: 1467-8659.

[Wen95] H. Wendland. "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree". English. In: *Advances in Computational Mathematics* 4.1 (1995), pp. 389–396. ISSN: 1019-7168.

[Yan+09] J. Yang, Z. W. Liao, Z. W. Liao, and Z. D. Wu. "A Method for Robust Nose Tip Location across Pose Variety in 3D Face Data". In: *2009 International Asia Conference on Informatics in Control, Automation and Robotics*. Feb. 2009, pp. 114–117.

[YQ07] P. Yang and X. Qian. "Direct Computing of Surface Curvatures for Point-Set Surfaces". In: *Eurographics Symposium on Point-Based Graphics*. Ed. by M. Botsch, R. Pajarola, B. Chen, and M. Zwicker. The Eurographics Association, 2007. ISBN: 978-3-905673-51-7.

[Zen+16] A. Zeng, S. Song, M. Nie, M. C. Fisher, and J. Xiao. "3DMatch: Learning the Matching of Local 3D Geometry in Range Scans". In: *CoRR* abs/1603.08182 (2016).

[Zhi+11] M. Zhihong, C. G., M. Yanzhao, and K. Lee. "Curvature estimation for meshes based on vertex normal triangles". In: *Computer-Aided Design* 43.12 (2011), pp. 1561 –1566. ISSN: 0010-4485.

[ZPK16] Q. Y. Zhou, J. Park, and V. Koltun. "Fast Global Registration". In: *Computer Vision – ECCV 2016*. Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling. Cham: Springer International Publishing, 2016, pp. 766–782. ISBN: 978-3-319-46475-6.

[ZW07] G. Zhang and Y. Wang. "A 3D Facial Feature Point Localization Method Based on Statistical Shape Model". In: *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*. Vol. 2. Apr. 2007, pp. II–249–II–252.

Appendix A

Professional Activities

Publications

Impacted Journals

[PV18] M. Prantl and L. Váša. "Estimation of differential quantities using Hermite RBF interpolation". In: *The Visual Computer* 34.12 (Dec. 2018), pp. 1645–1659. ISSN: 1432-2315.

[Váš+16] L. Váša, P. Vaněček, M. Prantl, V. Skorkovská, P. Martínek, and I. Kolingerová. "Mesh Statistics for Robust Curvature Estimation". In: *Computer Graphics Forum* 35.5 (2016), pp. 271–280. ISSN: 1467-8659.

Impacted Journals (other)

[PP18] D. Prantl and M. Prantl. "Website traffic measurement and rankings: competitive intelligence tools examination". In: *International Journal of Web Information Systems* 14.4 (2018), pp. 423–437. eprint: `https://doi.org/10.1108/IJWIS-01-2018-0001`.

International Conferences (WoS and Scopus)

[Pra+17] M. Prantl, V. Skorkovská, P. Martínek, and I. Kolingerová. "Curvature-Based Feature Detection for Human Head Modeling". In: vol. 108. International Conference on Computational Science, {ICCS} 2017, 12-14 June 2017, Zurich, Switzerland. 2017, pp. 2323 –2327.

[PVK16] M. Prantl, L. Váša, and I. Kolingerová. "Fast Screen Space Curvature Estimation on GPU". In: *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016)*. Vol. 1. Best student paper award. Feb. 2016, pp. 151–160. ISBN: 978-989-758-175-5.

[PVK17] M. Prantl, L. Váša, and I. Kolingerová. "Screen Space Curvature and Ambient Occlusion". In: *Computer Vision, Imaging and Computer Graphics Theory and Applications*. Ed. by J. Braz, N. Magnenat-Thalmann, P. Richard, L. Linsen, A. Telea, S. Battiato, and F. Imai. Cham: Springer International Publishing, 2017, pp. 51–71. ISBN: 978-3-319-64870-5.

[PVK19] M. Prantl, L. Váša, and I. Kolingerová. "Symmetry-aware Registration of Human Faces". In: *Proceedings of the 14th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2019)*. Vol. 1. Feb. 2019, pp. 185–192. ISBN: 978-989-758-354-4.

[Sko+17] V. Skorkovská, M. Prantl, P. Martínek, and I. Kolingerová. "Erosion-Inspired Simulation of Aging for Deformation-Based Head Modelingdownload this paper". In: vol. 108. International Conference on Computational Science, {ICCS} 2017, 12-14 June 2017, Zurich, Switzerland. 2017, pp. 425 –434.

Talks

- M. Prantl: Curvature estimation, VŠB - Technical University of Ostrava, Czech Republic, February 2018

Talks (other)

- M. Prantl: Information retrieval based on localization data, Archivní rešerše v 21. století, Klášter Teplá, Czech Republic, November 2018

Participation in Scientific Projects

- Advanced Computing and Information Systems. Project code SGS-2013-029. (2013 - 2015)
- Advanced Computing and Information Systems. Project code SGS-2016-013. (2016 - 2018)
- 1st Internal grant scheme of DCSE+P2, 2015
- 2nd Internal grant scheme of DCSE+P2, Metody registrace 3D modelů, 2016
- 2nd Internal grant scheme of DCSE+P2, Metody pro změnu věku 3D identikitu, 2016