FACULTY
OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Shape characteristics of geometrical objects and their use in computer graphics

State of the Art and Concept of Ph.D. Thesis

Martin Prantl

Technical Report No. DCSE/TR-2017-03
July 2017
Distribution: public

Technical Report No. DCSE/TR-2017-03
July 2017

Shape characteristics of geometrical objects and their use in computer graphics

State of the Art and Concept of Ph.D. Thesis

Martin Prantl

Abstract

Shape characteristics play an important part in computer graphics. They help us to better understand various properties of geometrical objects. One of the most important shape characteristics is curvature. However, the curvature itself is not always enough and other characteristics have to be utilized. Those can be based on: angles between triangles, distances of points, topology of models, volume, area etc. Once the 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. There are many different algorithms for shape characteristics with a different quality and performance. In the first part, we focus on curvature-based characteristics and related algorithms. We present a new solution for curvature estimation in the screen space. The second part is focused on other shape characteristics.

This work was supported by Advanced Computing and Information Systems (Project code SGS-2013-029 and SGS-2016-013).

Copies of this report are available on
`http://www.kiv.zcu.cz/en/research/publications/`
or by surface mail on request sent to the following address:

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Acknowledgments

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.

Contents

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 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, we have to propose a solution for a "black-box".

One of the most important shape characteristics is curvature. There are many different algorithms with a different quality and performance. The more triangles the model has, the more exact curvature estimation can be but the calculation becomes slower. However, the curvature itself is not always enough and other characteristics have to be utilized. Those can be based on: angles between triangles, distances of points, topology of models, volume, area etc.

Once the 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

[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.

database from 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 the user some regions of interest that can be highlighted (by color, different rendering style etc.).

In computer graphics, the basic representation of objects are usually triangle meshes. However, they represent only an approximation of the original model and the same triangle mesh can be obtained for different models. Based on that, triangle mesh 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.

In this report, we present shape characteristics mainly for triangulated objects. The main attention is devoted to curvature, its use for lighting in the term of ambient occlusion and curvature-based characteristics. To extend the scope of knowledge, other shape characteristics (based on normal vectors, Euclidean distances etc.) are briefly discussed as well. This part of the report is mainly focused on the proposed future work.

Structure of report

This report begins with a description of basic terms (Section 2) that are used through all sections. Basic theory behind curvature is covered in Section 3. Description of selected state-of-the-art methods in curvature estimation is presented in Section 4.

Shape characteristics, mostly curvature-based, are described in Section 5. However, other characteristics are discussed as well (Fourier transformation, Voxelization, Euclidean distances etc.).

From the visualization topic, the problem of ambient occlusion (Section 6) is presented. It can be partially solved with curvature or further improved using characteristic of the shape or model.

Our contribution is presented in the second part of the report. This section is based on our publication from [PVK16]. The screen space version of curvature estimation algorithm is described in Section 7. This solution is targeted to an interactive curvature estimation in the screen space on a GPU. Application of this algorithm to an estimation of ambient occlusion is presented as well.

The plans for a future research are summarized in Section 8. Section 9 concludes the report.

The used notation is as follows: ***bold italic*** symbols represent vectors, 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.

Chapter 2

Basic terms

First, we will briefly introduce some of the basic terms needed for this report. In this work, we concentrate primarily on the 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.

The geometry is represented as a set of vertices $P = \{P_j\}_{j=1,2...M}$, where M is a total number of vertices. For triangle meshes, each triangle has vertices denoted 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.

Since we use the triangulated geometry, we need to establish the term *neighborhood*. The neighborhood of a vertex P_j is consisting of all vertices adjacent to P_j up to some given distance. This distance can be expressed either by the number of edges k (this is referred as k-ring) or as the Euclidean distance.

2.1 Normal vectors

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

$$\boldsymbol{n} = \frac{\boldsymbol{e_i} \times \boldsymbol{e_{i+1}}}{|\boldsymbol{e_i} \times \boldsymbol{e_{i+1}}|}, \tag{2.1}$$

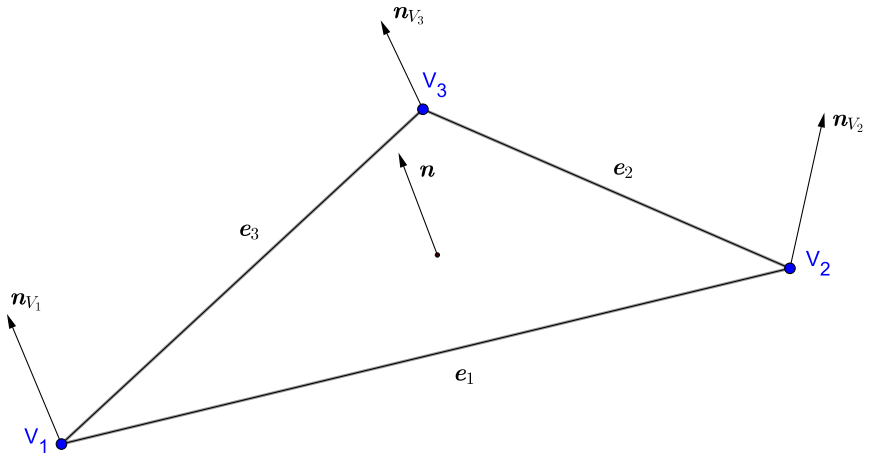


Figure 2.1: Triangle labeling

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

A problem is with normal vectors at the vertices. Since the triangle mesh is a discrete representation of the original model, 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.

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 (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].

2.2 Tangent space

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 a 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).

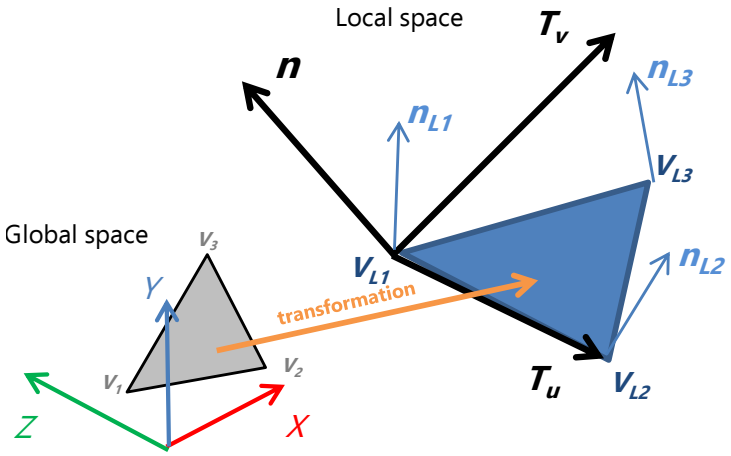


Figure 2.2: Transformation of triangle from Global (world) space to Tangent (local) space.

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):

$$\begin{aligned} \boldsymbol{T_u} &= \frac{V_2 - V_1}{|V_2 - V_1|}, \\ \boldsymbol{n} &= \frac{\boldsymbol{T_u} \times (V_3 - V_1)}{|\boldsymbol{T_u} \times (V_3 - V_1)|}, \\ \boldsymbol{T_v} &= \frac{\boldsymbol{T_u} \times \boldsymbol{n}}{|\boldsymbol{T_u} \times \boldsymbol{n}|}. \end{aligned} \tag{2.4}$$

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 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. 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

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.

3.1 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:

$$\kappa = \frac{d\omega}{dS}, \tag{3.1}$$

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 (3.1) and the osculating circle, see Figure 3.1 and [Rob01].

From Equation 3.1 we can obtain ([Kre91]) the equation

$$\kappa = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{\frac{3}{2}}}. \tag{3.2}$$

If the curve is defined explicitly in a form $y = f(x)$, the Equation (3.2) can

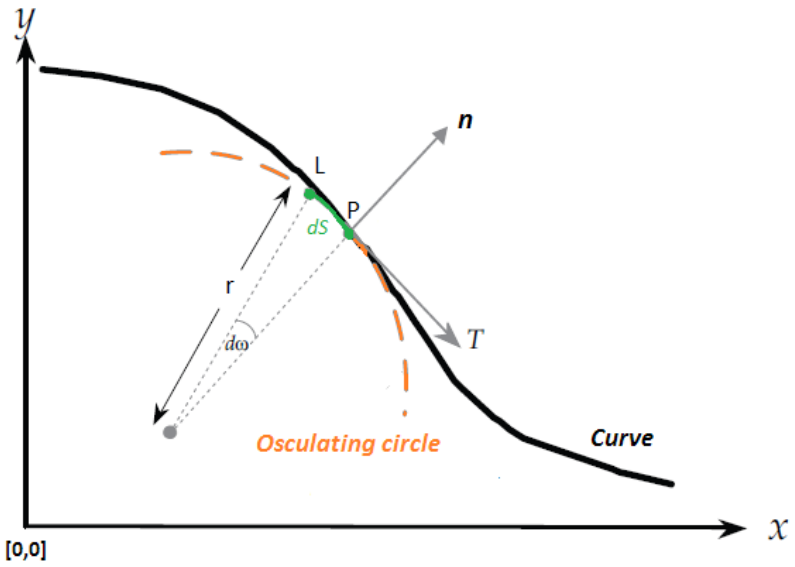


Figure 3.1: Definition of curvature using tangential angle $d\omega$ and arc length dS. [Rob01]

be rewritten as

$$\kappa = \frac{\frac{d^2y}{dx^2}}{\left(1 + \left(\frac{dy}{dx}\right)^2\right)^{\frac{3}{2}}}. \tag{3.3}$$

For more detailed explanations and derivations of Equations (3.1) - (3.3), see [Kre91].

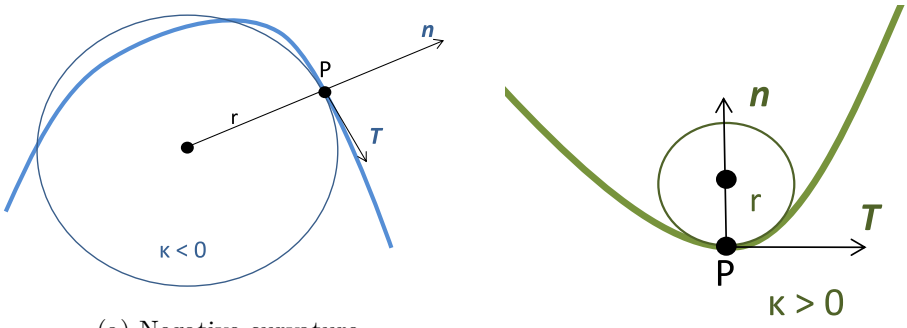
From a geometrical point of view, curvature in a point P is defined by an osculating circle. The sign of curvature is defined by the curve parametrization. See two osculating circles, curvature sign and a curve in Figure 3.2.

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 (3.1) using polar coordinates (see [Kre91]). The final equation is defined as

$$\kappa = \frac{1}{r}. \tag{3.4}$$

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 3.2).



(a) Negative curvature

(b) Positive curvature

Figure 3.2: Plane curve with two osculating circle at points P and curvature sign given by curve parametrization

3.2 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 details, the other can be seen, e.g., in [Rob01].

3.2.1 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 3.3). There is a 2D curve within this slice and we are looking for its curvature at a point P.

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 leads to the maximal (K_{max}, K_1) and the 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

$$k_n = K_1 cos^2\alpha + K_2 sin^2\alpha, \tag{3.5}$$

where α is the angle between the plane of K_1 and the plane for k_n (see Figure 3.4).

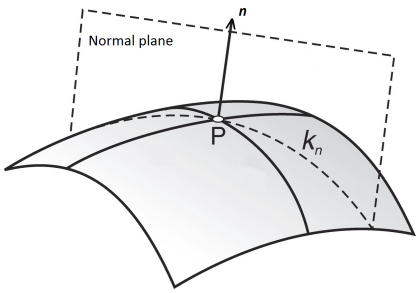


Figure 3.3: Normal curvature k_n within a normal place

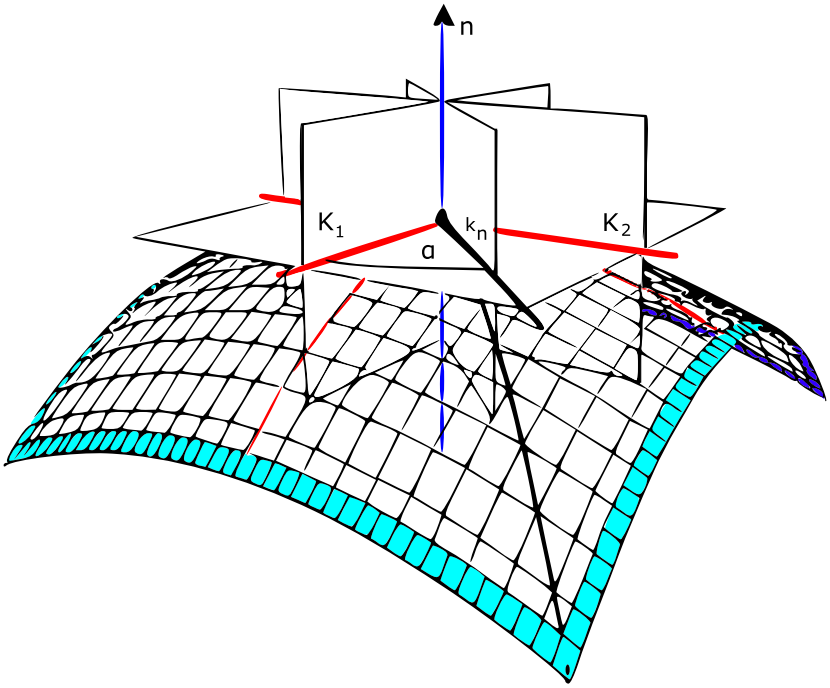


Figure 3.4: Angle between planes K_1 and k_n. [Unk17]

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

Table 3.1: Surface Shape Classes as defined in [Fis89]

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 3.1.

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.

3.2.2 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

$$K_H = \frac{K_1 + K_2}{2}. \tag{3.6}$$

Gaussian curvature is defined as

$$K_G = K_1 K_2. \tag{3.7}$$

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

$$K_{1,2} = K_H \pm \sqrt{K_H^2 - K_G}. \tag{3.8}$$

3.3 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.

3.3.1 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:

$$H^* = \begin{bmatrix} H_{11}H_{33} - H_{23}H_{32} & H_{23}H_{31} - H_{21}H_{33} & H_{21}H_{32} - H_{22}H_{31} \\ H_{13}H_{32} - H_{12}H_{33} & H_{11}H_{33} - H_{13}H_{31} & H_{12}H_{32} - H_{11}H_{32} \\ H_{12}H_{23} - H_{13}H_{22} & H_{21}H_{13} - H_{11}H_{23} & H_{11}H_{22} - H_{12}H_{21} \end{bmatrix}, \tag{3.9}$$

$$K_G = \frac{\boldsymbol{\nabla F}\, H^*\, \boldsymbol{\nabla F}^T}{|\boldsymbol{\nabla F}|^4}, \tag{3.10}$$

$$K_H = \frac{\boldsymbol{\nabla F}\, H\, \boldsymbol{\nabla F}^T - |\boldsymbol{\nabla F}|^2 trace(H)}{2|\boldsymbol{\nabla F}|^3}, \tag{3.11}$$

$$K_{1,2} = K_H \pm \sqrt{K_H^2 - K_G}, \tag{3.12}$$

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].

3.3.2 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 3.5.

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

$$\mathrm{I} = \begin{bmatrix} E & F \\ F & G \end{bmatrix}, \qquad E = \boldsymbol{T_u} \cdot \boldsymbol{T_u},\ F = \boldsymbol{T_u} \cdot \boldsymbol{T_v},\ G = \boldsymbol{T_v} \cdot \boldsymbol{T_v}. \tag{3.13}$$

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

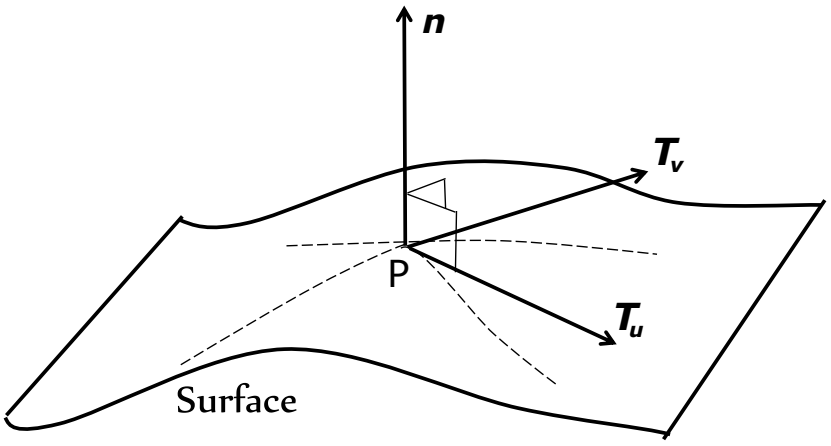


Figure 3.5: Tangent vectors of surface at point P

matrix II are computed as

$$\begin{aligned} \mathrm{II} &= \begin{bmatrix} L & M \\ M & N \end{bmatrix}, \\ \boldsymbol{n} &= \frac{\boldsymbol{T_u} \times \boldsymbol{T_v}}{|\boldsymbol{T_u} \times \boldsymbol{T_v}|}, \\ L = \boldsymbol{T_{uu}} \cdot \boldsymbol{n},\ M &= \boldsymbol{T_{uv}} \cdot \boldsymbol{n},\ N = \boldsymbol{T_{vv}} \cdot \boldsymbol{n}. \end{aligned} \tag{3.14}$$

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:

$$W = \mathrm{I}^{-1}\mathrm{II}. \tag{3.15}$$

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.

3.3.3 Monge patch

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

$$x(u, v) = (u, v, h(u, v)), \tag{3.16}$$

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:

$$K_H = \frac{h_{uu}h_{vv} - h_{uv}^2}{(1 + h_u^2 + h_v^2)^2}, \tag{3.17}$$

$$K_G = \frac{(1 + h_v^2)h_{uu} - 2h_u h_v h_{uv} + (1 + h_u^2)h_{vv}}{2(1 + h_u^2 + h_v^2)^{\frac{3}{2}}}, \tag{3.18}$$

where h_u, h_v, h_{uu}, h_{vv} and h_{uv} are derivatives of the function $h(u, v)$. Based on Equation 3.8, we can obtain principal curvatures.

Chapter 4

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.

4.1 Discrete methods

We start with a description of several discrete-based algorithms.

4.1.1 Monge patch

For a simple representation of 2.5D heightfield, we can use directly Monge patch (see sub-section 3.3.3). Required derivatives can be estimated with a discrete finite difference.

4.1.2 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 this leads to a system of equations:

$$\begin{aligned}
\mathrm{II} \begin{bmatrix} \boldsymbol{e_1} \cdot \boldsymbol{T_u} \\ \boldsymbol{e_1} \cdot \boldsymbol{T_v} \end{bmatrix} &= \begin{bmatrix} (\boldsymbol{n_{V_3}} - \boldsymbol{n_{V_2}}) \cdot \boldsymbol{T_u} \\ (\boldsymbol{n_{V_3}} - \boldsymbol{n_{V_2}}) \cdot \boldsymbol{T_v} \end{bmatrix}, \\
\mathrm{II} \begin{bmatrix} \boldsymbol{e_2} \cdot \boldsymbol{T_u} \\ \boldsymbol{e_2} \cdot \boldsymbol{T_v} \end{bmatrix} &= \begin{bmatrix} (\boldsymbol{n_{V_1}} - \boldsymbol{n_{V_3}}) \cdot \boldsymbol{T_u} \\ (\boldsymbol{n_{V_1}} - \boldsymbol{n_{V_3}}) \cdot \boldsymbol{T_v} \end{bmatrix}, \\
\mathrm{II} \begin{bmatrix} \boldsymbol{e_3} \cdot \boldsymbol{T_u} \\ \boldsymbol{e_3} \cdot \boldsymbol{T_v} \end{bmatrix} &= \begin{bmatrix} (\boldsymbol{n_{V_2}} - \boldsymbol{n_{V_1}}) \cdot \boldsymbol{T_u} \\ (\boldsymbol{n_{V_2}} - \boldsymbol{n_{V_1}}) \cdot \boldsymbol{T_v} \end{bmatrix}.
\end{aligned} \tag{4.1}$$

From the above presented Equations (4.1), 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 weighting 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.

4.1.3 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 dif-

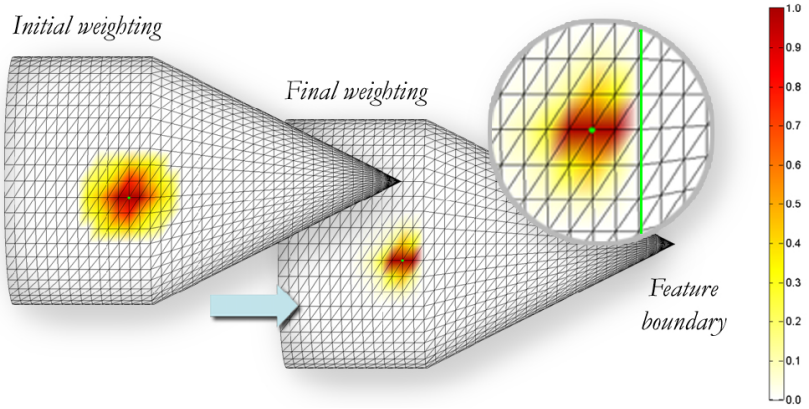


Figure 4.1: Re-estimation of curvature after initial estimation. Re-estimated curvature is correctly cut on the feature boundary (see detailed subimage). [Kal+07]

ferences 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 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 4.1.

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.

4.2 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.

4.2.1 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.

4.2.2 Bézier surfaces

Bézier surfaces (or patches) are one type of mathematical splines. It is given as the Cartesian product of the blending functions of two orthogonal Bézier curves. General Bézier patch is defined as:

$$F(u,v) = \sum_{i=0}^{N}\sum_{j=0}^{N} P_{ij} B_i^N(u) B_j^N(v);\ u,v \in \langle 0,1\rangle\,, \tag{4.2}$$

$$B_i^N(u) = \frac{N!}{i!(N-i)!} u^i (1-u)^{N-i}, \tag{4.3}$$

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

Bézier patches can also been computed for triangles. Every triangle in the mesh can be replaced with a Bézier patch in the form:

$$F(u,v,w) = \sum_{i=0}^{N}\sum_{j=0}^{N-i} P_{ijk} B_{ijk}^N(u,v,w), \tag{4.4}$$

$$B_{ijk}^N(u,v,w) = \frac{N!}{i!j!k!} u^i v^j w^k,\ 0 \le u,v,\ w \le 1, \tag{4.5}$$

$$u+v+w = 1;\ i+j+k = N, \tag{4.6}$$

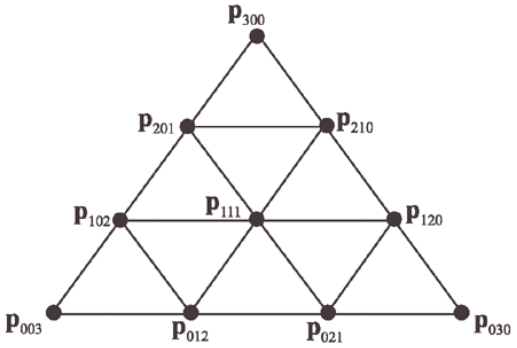


Figure 4.2: Control points of Bézier patch for a single triangle (top view)

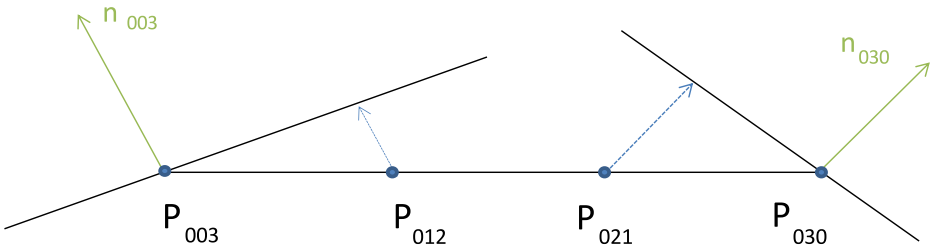


Figure 4.3: Calculation of control points on the edge of the triangle (side view)

where N is the patch degree 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 4.2) for the patch.

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 to 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 4.3 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 4.3). The center control point P_{111} is calculated from all other control points as defined in [Vla+01].

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

Biquadratic patch

Bézier patch with $N = 2$ 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 derivation of Equation (4.4) with $N = 3$. 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].

Blended Bézier surfaces with G^1 continuity

The problem with a 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, the curvature is not directly defined. It is calculated as an average from neighboring edges but there could be a sharp turn leading to a steep change in the curvature. Fünfzig et al. in [Fün+08] proposed a solution called *PNG1* to overcome the problem. See Figure 4.4 for difference between *PNG1* and a classic Bézier patch. *PNG1* patch is created by blending standard Bézier patches from neighboring triangles (see Figure 4.5).

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. The curvature can be directly calculated using an analytic solution as proposed by the same authors in [Bos+12]. A drawback of this method is that the second derivatives are much more complex than for a simple Bézier patch.

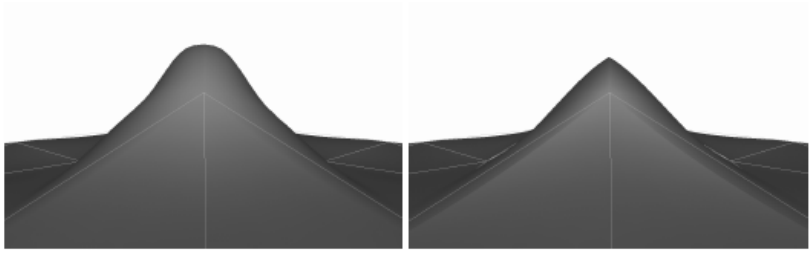


Figure 4.4: Comparison of the *PNG1* [Fün+08] (left) and Bézier patch (right). Sharp change of the surface can be observed for a classic Bézier patch.

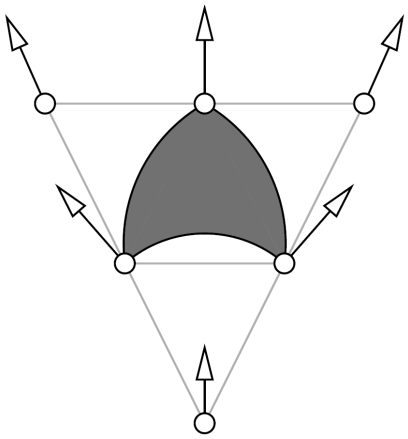


Figure 4.5: *PNG1* patch (gray) and neighboring triangles used for blending in [Fün+08].

4.2.3 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.

4.3 Other methods

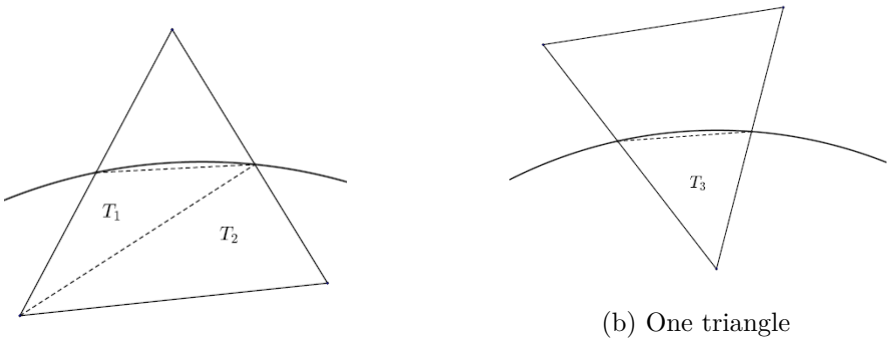
The last section is used for algorithms that cannot be directly assigned to previously mentioned groups. They use combinations of approaches and often take knowledge of both worlds - discrete and surface fitting.

4.3.1 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 and integral. To simplify this, a discretization is used and the integral is expressed via summing over an arbitrary region R surrounding the vertex V. 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:

$$\Gamma = \frac{1}{|R|} \sum_{\forall \boldsymbol{e}} \beta(\boldsymbol{e}) \, |e \cap R| \, \overline{\boldsymbol{e}} \, \overline{\boldsymbol{e}}^T, \tag{4.7}$$

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 $\overline{\boldsymbol{e}}$ is the unit vector in the direction of $\boldsymbol{e}$. From the matrix τ, three eigenvectors and their corresponding eigenvalues are calculated. The eigenvectors associated with the minimal-magnitude eigenvalue is a normal vector, the remaining two



(a) Two triangles

(b) One triangle

Figure 4.6: Two possible splitting schemes of triangle with neighborhood intersection

eigenvalues and eigenvectors represent principal curvatures $K_{1,2}$ and their swapped principal directions.

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.

4.3.2 Generalized shape operator

An approach based on a generalized version of the shape operator (see Equation (3.15)) 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 4.6a) or only one triangle T_3 (Figure 4.6b).

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. A generalized version of the shape operator (expressed using a 3×3 matrix $\overline{S}$ 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].

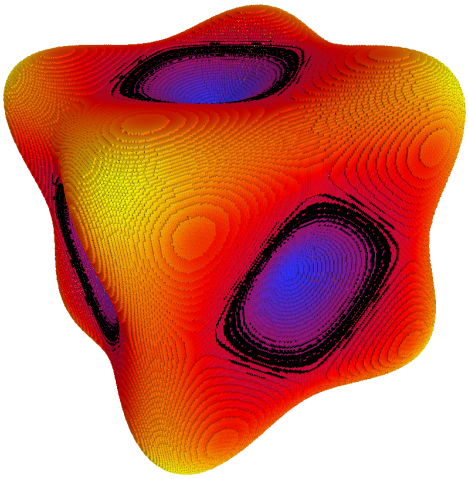


Figure 4.7: Mean curvature calculated on volume data. [Lev15]

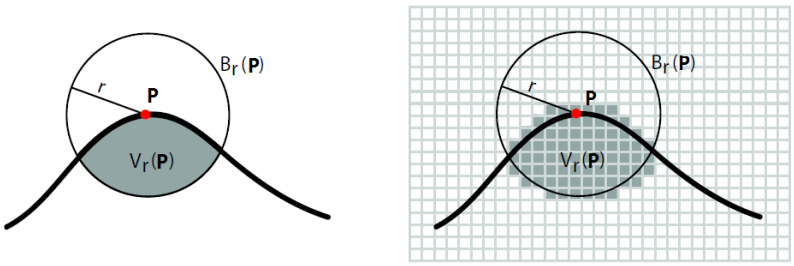


Figure 4.8: Example of rasterization between sphere and curve intersection in 2D. [Gel+05]

4.3.3 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 4.7). The modification of this algorithm by Levallois et al. is a part of DGtal library [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 4.8. 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.

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, the lower precision is used.

4.4 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 paralellize the computation of curvature estimation. 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 4.1.2. 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 es-

timated 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 4.9. The visual quality of both results is quite similar, but the solution by Kalogerakis runs 1.7 ms, while Rusinkiewicz took 91 ms.

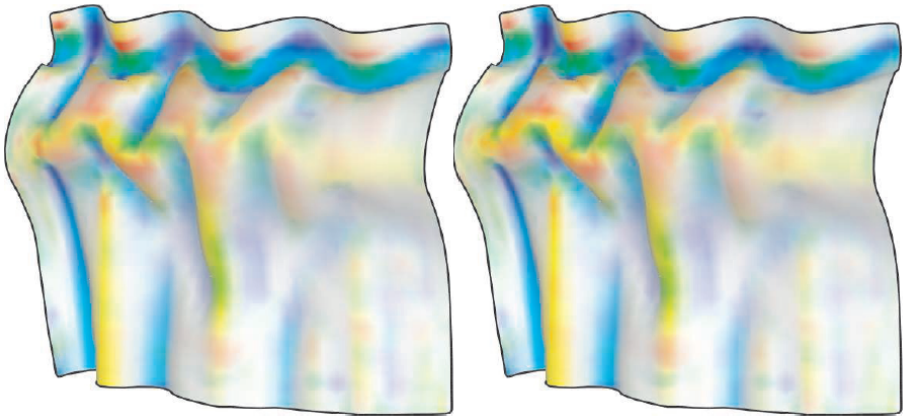


Figure 4.9: Comparison of principal curvatures produced by the method of Rusinkiewicz [Rus04] and the dynamic Kalogerakis [Kal+09]

4.4.1 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 4.10. 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 4.10, where the fine details of the hair are lost.

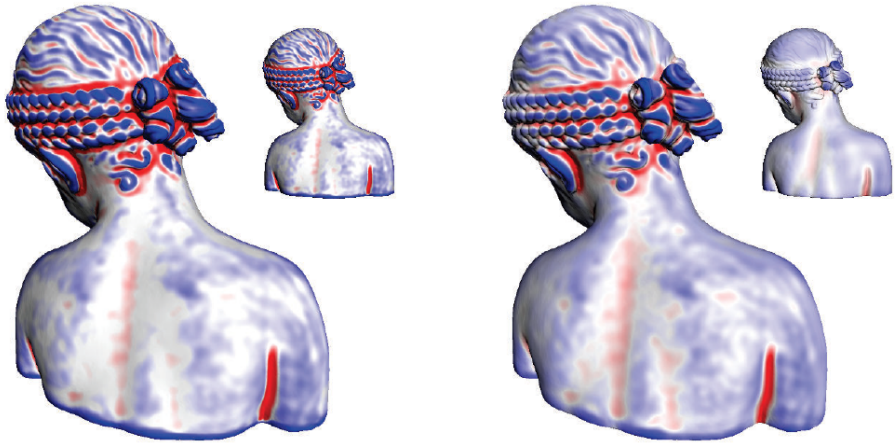


Figure 4.10: Comparison between mean curvature estimated directly from mesh (left) and its screen-space version (right). [Mel+13]

Chapter 5

Other shape characteristics

We have already covered curvature as a basic shape characteristic. However, each object can have various other characteristics, such as normal vectors, positions, local objects volumes etc. We can also combine several characteristics together and create a different solution.

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.

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 5.1. 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 5.1: Descriptors division from [TV08]

5.1 Curvature-based

In Section 3, 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.

5.1.1 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., as can be seen in many publications [CSJ05; ZW07; SAC12; NC09] etc. Descriptors derived from principal curvatures are 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. 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 5.1.

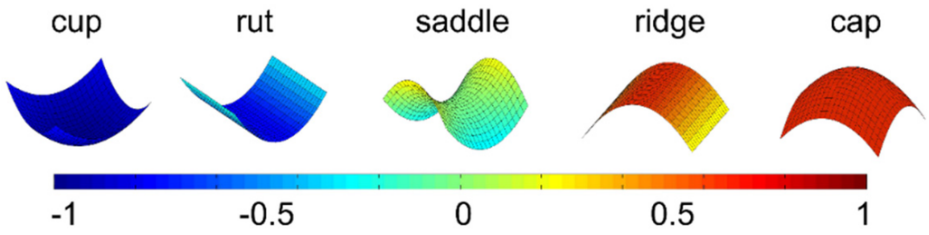


Figure 5.1: Shape Index

- 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.

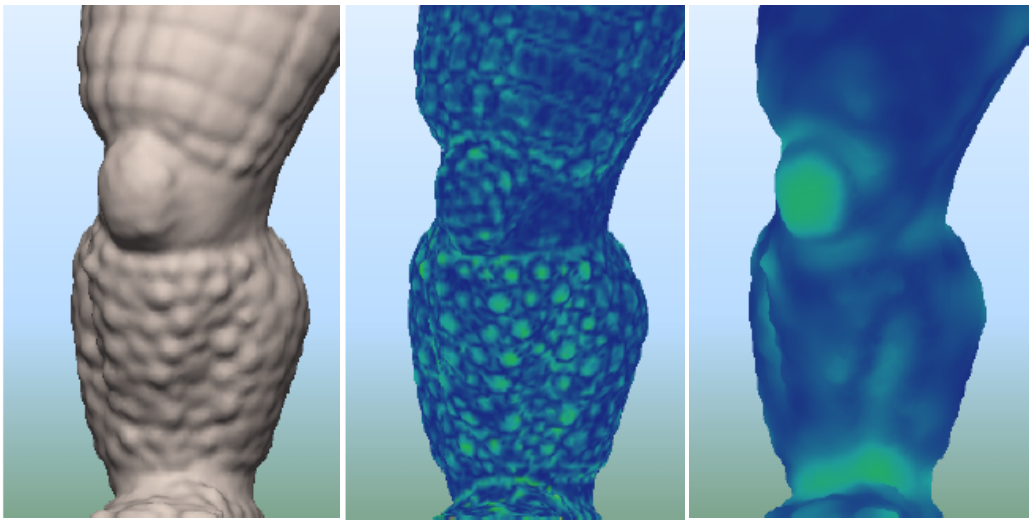
- Willmore energy (WE)

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

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.

5.1.2 Saliency

Lee et al. [LVJ05] do not use curvatures directly because curvatures capture fine details (see Figures 5.2a and 5.2b) 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 5.2c). 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 5.2: Comparison of curvature and saliency. [LVJ05]

5.1.3 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 5.3. The "*Ear Tip Vertex*" is selected as a reference point. Similarity of this point to other points on 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.

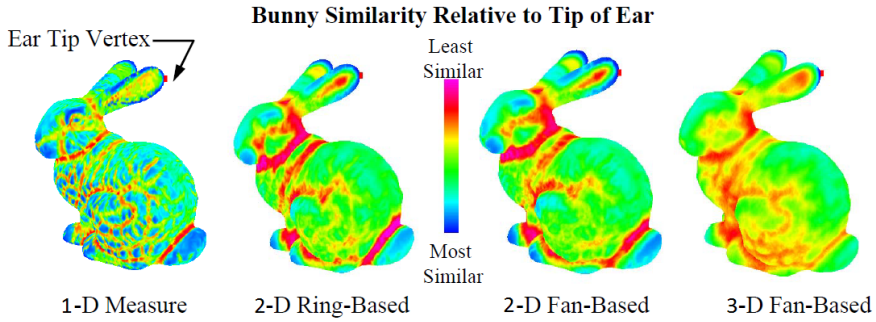


Figure 5.3: Curvature map and similarity measure relative to a selected vertex (ear tip). [Gat+05]

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 5.4. 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].

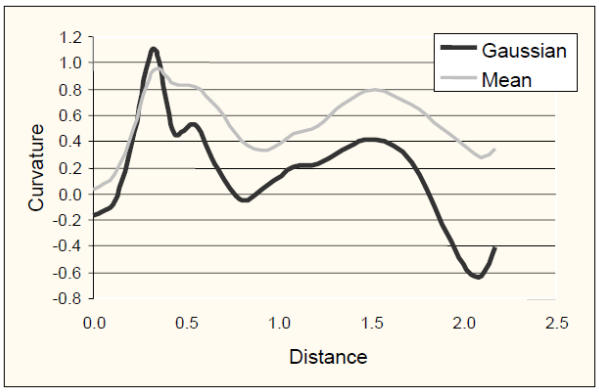


Figure 5.4: Example of curve generated from curvature map for a single point. [Gat+05]

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.

5.1.4 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 parametrized 3D surfaces.

5.1.5 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 5.5).

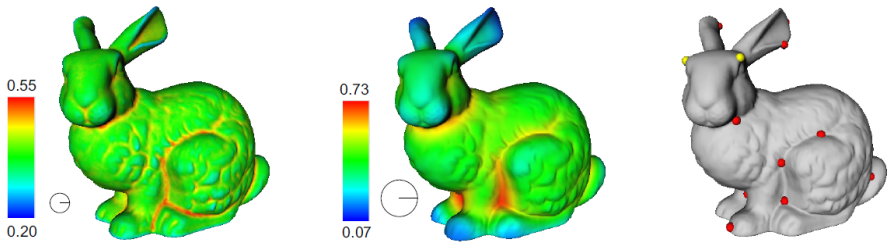


Figure 5.5: Normalized curvature computed with two different radius spheres (left, middle) and the resulting descriptors (right). [Gel+05]

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-dependance. The solution to this problem is to use a different size of the sphere during the voxelization, as can be seen in Figure 5.5. 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.

5.1.6 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" (HKS) 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 for small timesteps according to [Bro11] can be expressed as:

$$h_t(x, x) = \frac{1}{4\pi t} + \frac{K_G(x)}{12\pi}. \tag{5.1}$$

5.2 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.

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 2.3), the neighborhood is divided into several 3D spherical volumes. Each volume 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.

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 neighborhood of radius r 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 5.6), where

$$\begin{array}{rcc} \alpha & = & v \cdot n_t \\ \phi & = & (u \cdot (P_t - P_s))/||P_t - P_s|| \\ \theta & = & arctan(w \cdot n_t, u \cdot n_t) \\ d & = & ||P_t - P_s|| \end{array} . \tag{5.2}$$

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.

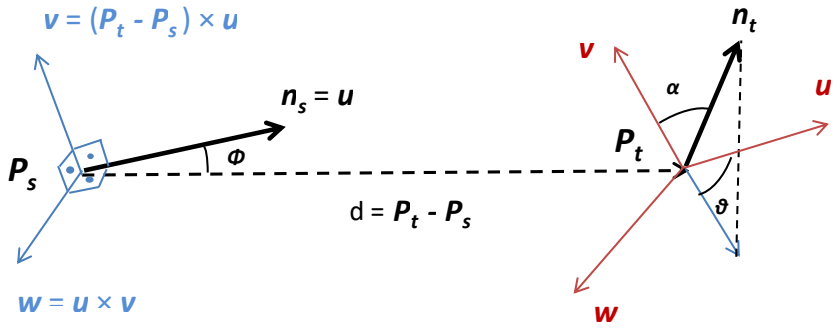


Figure 5.6: Angles from quadruple between two points (P_s,P_t) in PFH. [Rus16]

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

$$bin = \sum_{i=0}^{i \leq 4} 2^{i-1} step(s_i, q_i), \tag{5.3}$$

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 the same authors 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.

5.3 Other

Apart from previously mentioned curvature or normal based methods, 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).

5.3.1 Local Reference Frame

The descriptor named TriSi based on axes of LRF (recall Section 2.3) 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 S is created using continuous PCA algorithm and three eigenvectors ($\boldsymbol{v_1}$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$) of S are computed. They form the LRF system. However, the sign of eigenvectors is ambiguous and a sign disambiguation technique is used. The newly created "eigenvectors" ($\widetilde{\boldsymbol{v}}_{\boldsymbol{1}}$, $\widetilde{\boldsymbol{v}}_{\boldsymbol{2}}$, $\widetilde{\boldsymbol{v}}_3$) are used as the description of LRF.

TriSi descriptor is in 3D created from three spin sheets (also known as spin images). These are planes, where every one of them corresponds to one axis defined by a sign-corrected eigenvector. An example of one spin sheet aligned in the plane given by $\widetilde{\boldsymbol{v}}_1$ and $\widetilde{\boldsymbol{v}}_{\boldsymbol{3}}$ can be seen in Figure 5.7. Points from the model are projected into these planes.

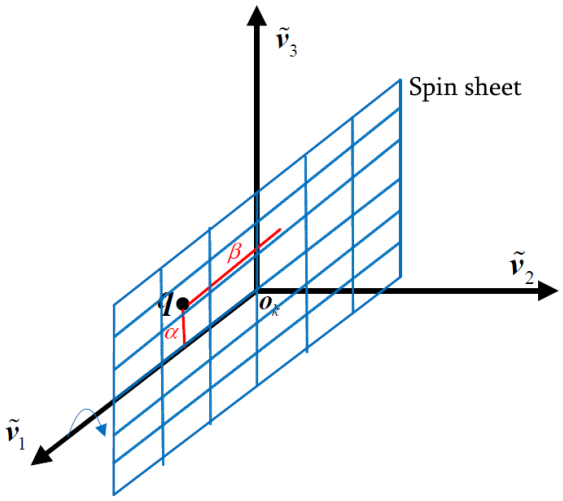


Figure 5.7: Generating a spin sheet for TriSi, α and β are distances of origin from projected point q after its perpendicular projection onto axes $\widetilde{\boldsymbol{v}}_{\boldsymbol{1}}$ and $\widetilde{\boldsymbol{v}}_{\boldsymbol{3}}$. [Guo+13]

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

$$M = \sum_{i=1}^{D} (\boldsymbol{f_i} - \bar{\boldsymbol{f}})(\boldsymbol{f_i} - \bar{\boldsymbol{f}})^T, \tag{5.4}$$

where D 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.

Another algorithm using LRF was proposed by Shah et al. [Sha+13]. They detect 3D keypoints on the surface of the mesh. For each keypoint, 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 method [Guo+13]. 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 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.

5.3.2 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 5.8. This approach is simple, but robust to holes, non-manifoldness etc. since it only computes ray - triangle intersections and store distances.

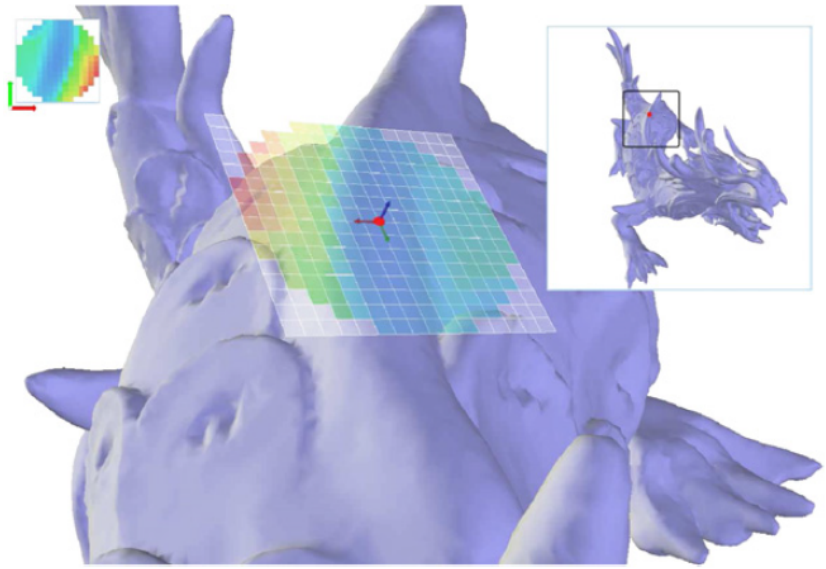


Figure 5.8: A surface descriptor example for a single vertex (red). The descriptor is a distance-map in a tangent plane. Distances are visualized as color-coded values. [Max+11]

We can directly compare meshes from the constructed distance-map at each vertex. However, this solution is not robust. 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).

5.3.3 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 S is computed of each grid-bin. S 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].

5.3.4 Fourier transform

Fourier transforms are very popular in 2D geometry and image processing. Descriptors using Fourier transform are more of a global character. However, instead of their use for the entire geometrical object, they could possibly be used for only a neighborhood as well. In this case, we expect the neighborhood to be of a bigger size then in case of truly local descriptors. This idea is not described in the presented articles, but in the future work we would like to examine this and see, if this assumption is valid or not.

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 3D curve and its description by Fourier series have been proposed by Lmaati 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 5.9) constructed on the unit sphere given by:

$$\begin{aligned} x(t) &= cos(qt)sin(t) \\ y(t) &= sin(qt)sin(t) \\ z(t) &= cos(t) \end{aligned} , \tag{5.5}$$

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

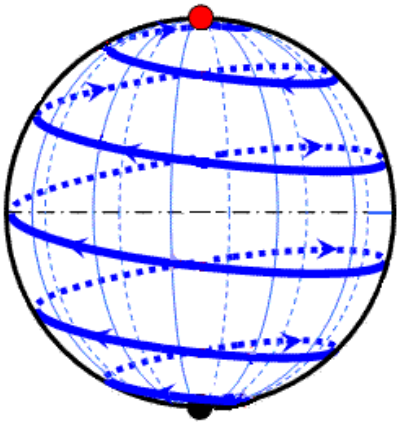


Figure 5.9: A spherical Helix curve

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 5.5). 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-parametrized in order to compute the feature vector. The authors have selected the natural parametrization (the arc length parametrization). Fourier series are applied to this parametrized 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].

Chapter 6

Ambient occlusion

Curvature can be partially utilized for an ambient occlusion (AO) estimation. AO 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 AO 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.

6.1 Basic theory

There are two basic types of the light in the scene - direct and indirect. While the direct light comes from a certain source with a direction (it can be, e.g., a lamp), indirect has no fully defined source and direction. It comes from every direction. Its source does not even have to be a light source itself, but some other surface that only reflects the light. Therefore, to calculate AO for a given point is a very complex task since all objects in the scene can be possible light sources.

The amount of light in the point P (see Figure 6.1 for a simplified 2D case) is given as an integral over the hemisphere Ω.

The hemisphere Ω has its center at the point P and is oriented along a normal vector $\boldsymbol{n}$. To compute AO in a point P, samples from the entire hemisphere must be used. A ray with a direction $\boldsymbol{\omega}$ from the point P is used for every sample with an integration step $d\omega$. This ray is tested by a visibility function $V_{P\omega}$ whether it intersects another object in the scene Ω (called occluder). It can be written as:

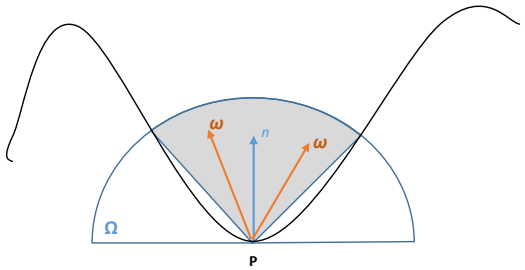


Figure 6.1: A hemisphere Ω at a given point P with a shaded unocluded area

$$AO = \frac{1}{\pi} \int_{\Omega} V_{P\omega}(\boldsymbol{n} \cdot \boldsymbol{\omega}) d\omega, \tag{6.1}$$

where $V_{P\omega}$ is a visibility function defined as

$$V_{P\omega} = \begin{cases} 0 & if\ the\ ray\ \boldsymbol{\omega}\ intersect\ a\ scene\ object \\ 1 & otherwise \end{cases}. \tag{6.2}$$

Equation (6.1) is quite complex and simplifications are often made. The integral can be approximated by Monte Carlo integration (a technique for numerical integration using random sampling) with a randomly selected direction vector $\boldsymbol{\omega_i}$ for every i as

$$AO \approx \frac{1}{N} \sum_{i=1}^{N} V_{P\omega}(\boldsymbol{n} \cdot \boldsymbol{\omega_i}), \tag{6.3}$$

where N is a total number of random samples.

In this simplified case, the problem of the presented visibility function $V_{P\omega}$ (Equation (6.2)) is its binary character. There is no difference how far the occluder is from the point P. With this simplified approach, the final scene is usually too dark. To overcome this problem, visibility function ρ based on the distance $d(P, \boldsymbol{\omega})$ to the first occluder is presented. This solution was first proposed by Zhukov et. al. in [ZIK98] as an Ambient Obscurance (AOb):

$$AOb = \frac{1}{\pi} \int_{\Omega} \rho(d(P, \boldsymbol{\omega}))(\boldsymbol{n} \cdot \boldsymbol{\omega}) d\omega. \tag{6.4}$$

Today, the terms ambient occlusion and ambient obscurance are used interchangeably. They both can be expressed with the same beginning letters, which increases the confusion. In this paper, the notation $AO = AOb$ is used.

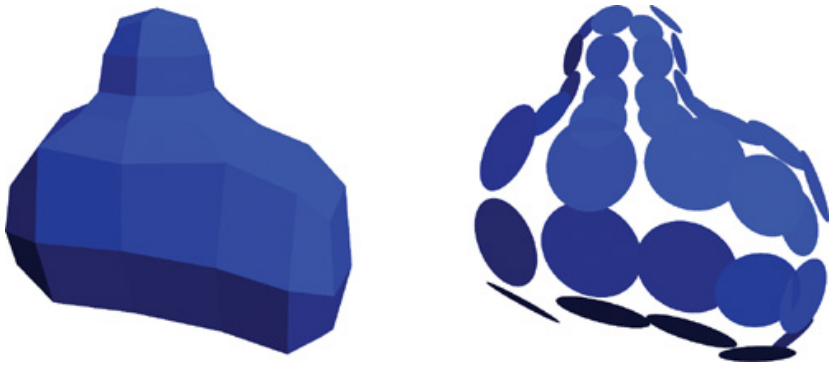


Figure 6.2: Mesh converted to a list of discs [PF05]

6.2 Related work

The best *AO* approximation can be calculated with ray-tracing. This solution is not real-time and even for small scenes it can take hours to compute. Of course, the total time depends on the level of details and geometry quality in the scene. For a real-time computation of *AO*, other faster methods are usually preferred.

There are two main categories of methods - object and screen space. The main difference between these two categories is in computational steps. The screen-space methods are running entirely in the pixel shader on GPU, while the object space methods either use other shaders (vertex, geometry, tessellation) or need some sort of an object-based preprocessing. A very nice comparison of several algorithms for real-time *AO* has been written by Aalund in [FPA13].

Only a minority of methods computes *AO* directly at vertices. These methods are not very well studied because of their special use cases. For a per-vertex ambient occlusion, ray-casting with acceleration structures (such as octree, BSP tree etc.) is usually preferred. These algorithms are not designed for use in real time.

6.2.1 Object space methods

One of the first real-time methods was proposed by Pharr et al. [PF05]. In their approach, all scene objects are converted to set of discs that fill out the object surface (see Figure 6.2). The discs are used to speed up calculations, because to calculate an intersection of rays with the discs is fast. There can (and often are) holes between discs. They are neglected during computations and the intersection is simply not found.

The computation of occlusion is done by mutual comparison of discs. This leads to the time complexity $O(n^2)$. To speed up the process, only

discs within a certain distance are tested. Another speed-up is achieved by geometry simplification, where disc are computed from a simplified model.

The method similar to shadow volumes was presented by McGuire [McG10]. This work is an improvement of Kontkannen et al. [JS05] in combination with shadow volumes. While [JS05] precomputes the occlusion integral (see Equation 6.1) using bounding volume and stores results in a texture, [McG10] computes the same integral on the fly. For each face of the mesh within the visible scene, a bounding volume is calculated in the geometry shader and passed to the pixel shader. An evaluation if a pixel is inside (occluded) or outside this volume is executed for every pixel. There is no need for an additional blurring, because this method does not produce noise artifacts.

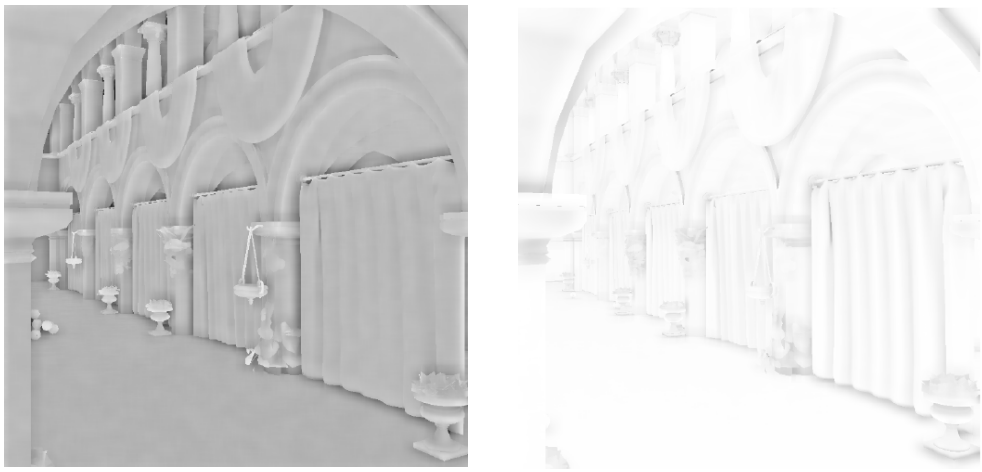
Algorithm based on voxelization of the currently visible region has been proposed by Reinbothe et. al. in [RBA09]. This approach is independent of the current depth buffer used in screen space methods and allows occlusion from the currently invisible parts of the scene. The final occlusion is, however, computed in the screen space, therefore, this technique is marked as a hybrid one.

6.2.2 Screen space methods

In 2007, Mittring [Mit07] presented new algorithm to compute *AO* in the screen space. He was the first author ever to do this and his algorithm is ever since known as SSAO. This solution uses only the depth buffer as the scene approximation, no normal vectors are used. Each depth pixel is surrounded by the sphere and sample points are collected within this sphere. If the sample is inside the object (e.g., its depth is greater then the current pixel), it contributes to the final occlusion factor. This method is very simple, its quality depends only on the number of samples within the sphere. This leads to a noisy result that have to be blurred to get a smoother effect. With a bigger count of samples, the noise is less visible, but the algorithm is more computationally expensive. The sampling is done inside a full sphere and it causes a characteristic look of the occlusion. The areas with no occlusion are not white but light-gray (e.g. they are in the middle of the gray-scale range $\langle 0, 255 \rangle$). This can be seen in Figure 6.3a.

Improvement of SSAO was proposed by Fillion et al. in [FM08]. They use a hemisphere (instead of a sphere) oriented along the normal in the given pixel. This overcomes the previously mentioned problem with not having white areas, where there is no occlusion. See Figure 6.3b. The rest of the algorithm is the same as SSAO.

A method that uses multiple resolutions to avoid noise and therefore the need of additional blurring was presented by Hoang et al. [HL10]. They use multiple resolutions and blend them together to obtain the smooth result. Blending is done according to the desired quality. For objects with low details or far away from the camera it is sufficient to use only a low-resolution version



(a) Mittring SSAO ([Mit07])

(b) Fillion ([FM08])

Figure 6.3: Original Mittring SSAO [Mit07] characteristic look (left) vs. Fillion [FM08] with correct shading colors [FPA13] (right)

of AO.

The method similar to the parallax occlusion mapping was presented by Bavoil et al. [BSD08] (known as HBAO). From every point of the scene, the ray is traced to find the limit intersection with the heightfield created by the depth buffer, see Figure 6.4. Found intersections are marked S_i. The horizon angle to every intersection point is calculated. The maximal angle is used to compute occlusion weighted by the length $|P - S_i|$. The weighting creates smaller values for large angles at greater distances. To overcome artifacts and noise, the ray direction is randomized.

Loos et al. [LS10] presented a solution where line samples are used instead of point samples. Lines are sampled inside the sphere around the point. Lengths of visible parts line segments inside the sphere are calculated, see Figure 6.5. Length ratio of the visible (green) and the occluded (red) part of each line is used to calculate AO. The quality and the strength of the resulting occlusion depends on the radius of the used sphere and number of samples inside it.

The majority of the presented methods use a similar solution to the visibility function. They mostly differ in a way, how they sample the depth and reconstruct AO. A method aimed at deferred rendering[1] was presented by McGuire et al. [McG+11]. The use an updated visibility function called the falloff function σ. This is defined as

[1]Deferred rendering is a screen-space technique that stores intermediate results into textures (called a normal buffer, a depth buffer...), then complete the rendering equation by sampling this intermediate data.

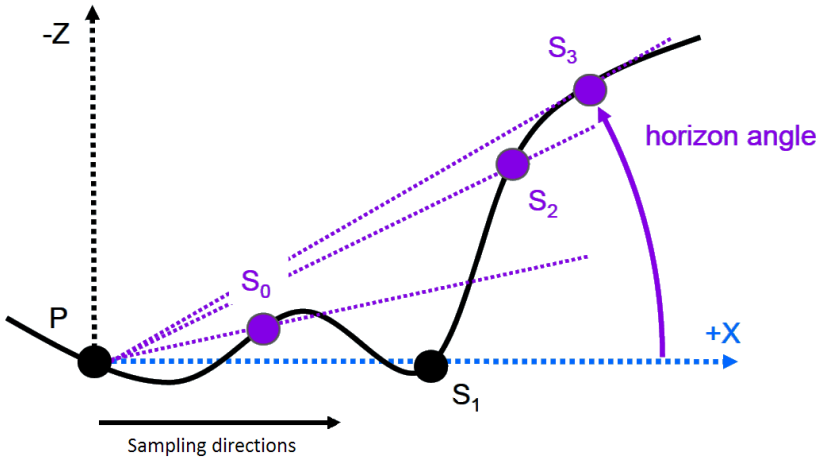
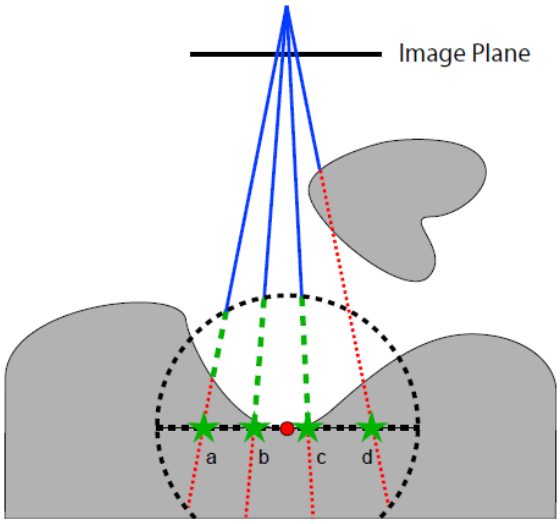


Figure 6.4: HBAO sampling scheme from point P



later

Figure 6.5: Length ratio of visible (green) and occluded (red) are used to compute AO. [LS10]

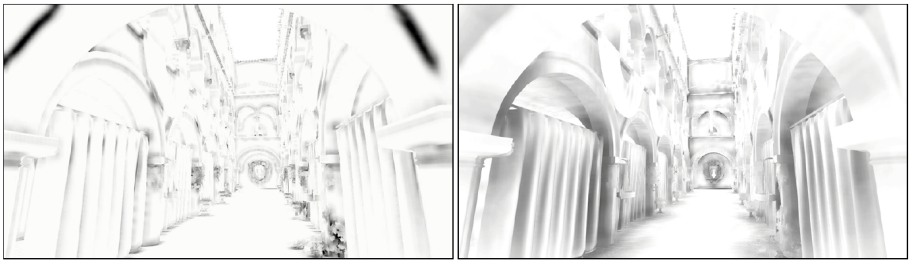


Figure 6.6: Comparison of [McG+11] (left) and its improvement [MML12] (right). The frame time for both methods is the same.

$$\sigma = \frac{ud}{max(u,d)^2}, \tag{6.5}$$

where d is the distance between samples and u is a user-defined parameter to enhance the shape of the falloff function.

The function from Equation (6.5) is used in the integral from Equation 6.4 instead of $\rho(d)$. To solve the integral, Monte Carlo integration over a selected number of samples is used. The sampling is achieved with the approach similar to the one presented in [LS10]. Fewer samples are used for further objects to reduce the computational time.

McGuire et al. presented an improvement of their previous method (see [McG+11]) in [MML12] (known as SAO). See a comparison between both methods in Figure 6.6. The new algorithm is no longer intended only for the deferred rendering. Instead of the normal and the depth buffer, only the depth buffer with an increased precision is needed. Normal vectors and positions can be reconstructed directly from depths. *AO* is reconstructed from the depth using a bilateral filter. It is a non-linear, edge-preserving and noise-reducing smoothing filter for images. The intensity value at each pixel in the image is replaced by a weighted average of intensity values from nearby pixels.

AO computed from a two-layered depth buffer was presented by Mara et al. [Mar+14]. During a depth buffer creation, two textures are used as an output. The first one is a classic depth buffer, the second one is a depth buffer in a certain distance from the camera. The closest objects are not presented in the second buffer. Generated buffers are used to improve the occlusion effect created by [MML12] by removing some artifacts and create smoother transitions between the illuminated and the shadowed area. This solution is not only applicable to *AO* but can also be used for other effects (e.g., screen space reflections, illuminations).

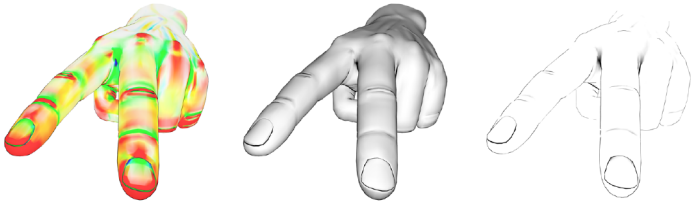


Figure 6.7: A hand model with visualized properties: curvature (left), shaded with ambient occlusion (middle), just ambient occlusion (right). [Gri+12]

Curvature-based

Apart from the traditional approaches, a solution based on curvature has been presented by Hattori et al. in [HKM11] or [HKM12]. The presented solution uses a local neighborhood approximation at a point by principal curvatures ($K_{1,2}$), that are used as terms of Taylor series. A sphere is circumscribed around the same point. The AO is calculated as a volume of an intersection of the sphere and a local approximation of surface. This leads to an integral

$$AO = \int_0^{2\pi} \int_0^{\theta} r^2 \phi sin\theta' d\theta' d\phi, \tag{6.6}$$

$$\theta = acos\left(\frac{-1 \pm \sqrt{1+A^2}}{A}\right), \tag{6.7}$$

$$A = r(K_1 cos^2\phi + K_2 sin^2\phi) \tag{6.8}$$

where r is the radius of circumscribed sphere around the point. The solution of this integral is too complicated to be calculated at real time, so the solution is simplified. AO is computed directly from curvature by an integral simplification

$$AO = \frac{2}{\pi} acos\left(\frac{-1 \pm \sqrt{1+(K_1+K_2)^2 r^2}}{(K_1+K_2)^2}\right). \tag{6.9}$$

The algorithm to compute AO from curvature has also been used by Griffin et al. [Gri+12]. Their solution is partially based on [HKM11], but instead of a full computation of Equation 6.9, they use a mapping function with precomputed coefficients:

$$AO = 1.0 - 0.0022(K_1+K_2)^2 + 0.0776(K_1+K_2) + 0.7369. \tag{6.10}$$

To increase the effect of occlusion and create darker corners, the principal curvatures can be scaled up. The result of the curvature and shading with AO can be seen in Figure 6.7.

Chapter 7

Our contribution - Screen space curvature and Ambient Occlusion

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], works in the screen space and it can also be used for triangle meshes. 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 for many problems, such as water rendering, lighting, ambient occlusion 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.

First, a description of the proposed algorithm for a triangle mesh is presented. The screen space version is discussed next.

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 (3.15). 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 (3.14), 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 $Ax = b$, the least squares method is used to obtain unknown variables:

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

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:

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

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 of the calculation 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).

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

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

$$\left(\begin{bmatrix} u_1b_1 & u_1b_2 & u_2b_1 & u_2b_2 & u_3b_1 & u_3b_2 \\ u_1b_2 & u_1b_4 & u_2b_2 & u_2b_4 & u_3b_2 & u_3b_4 \\ u_1b_3 & u_1b_5 & u_2b_3 & u_2b_5 & u_3b_3 & u_3b_5 \end{bmatrix} + \begin{bmatrix} v_1b_2 & v_1b_3 & v_2b_2 & v_2b_3 & v_3b_2 & v_3b_3 \\ v_1b_4 & v_1b_5 & v_2b_4 & v_2b_5 & v_3b_4 & v_3b_5 \\ v_1b_5 & v_1b_6 & v_2b_5 & v_2b_6 & v_3b_5 & v_3b_6 \end{bmatrix} \right) \tag{7.9}$$

Having obtained the final vector 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 (3.6) and (3.7)).

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 7.5.

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 normal and depth (from which the position is reconstructed) is 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.) were 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.

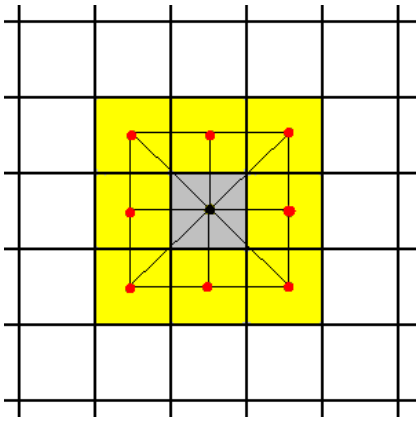


Figure 7.1: The local triangulation of neighborhood pixels. The center pixel is currently calculated, one-ring neighborhood forms triangulation.

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 three 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.

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

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

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

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

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

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.

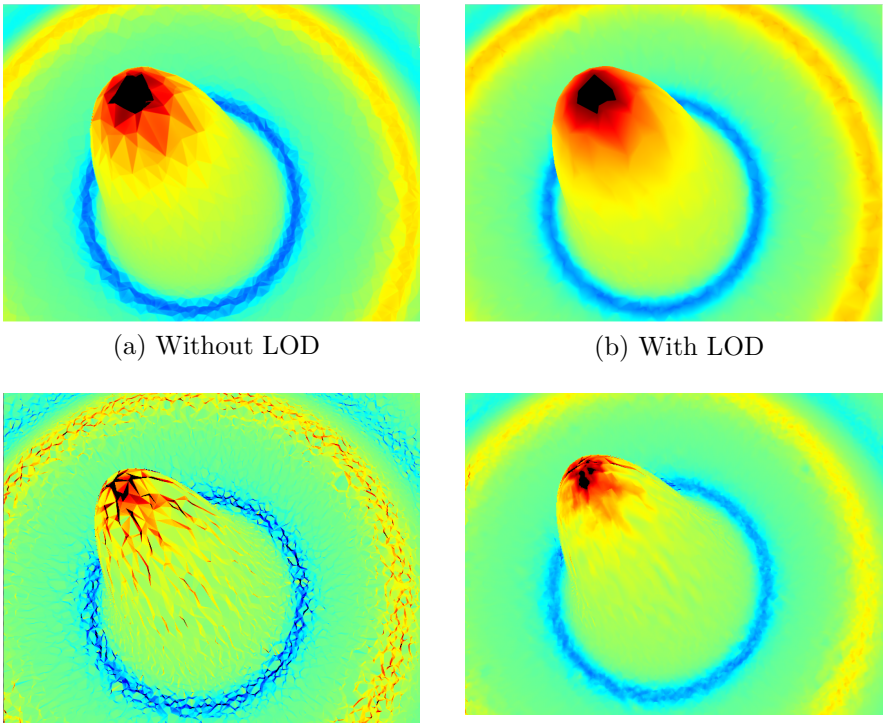
To solve this problem, level of detail (LOD) sampling can be used. For points closer to the camera, triangles are constructed from a wider neighborhood. Our solution is based on a power function and the final equation is:

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

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 $size$ represents the step to the neighboring pixels. For this, the value should be converted to integer by omitting the fractional part. 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 Ambient occlusion

The ambient occlusion can be computed using various algorithms, as stated in Section 6. However, we are interested in curvature-based algorithms (see

Section 6.2.2). The curvature in those solutions is transferred as an additional vertex parameter. In the proposed solution we use the newly created screen space version of the curvature estimation algorithm. Both presented curvature-based AO solutions require the principal curvatures K_1 and K_2 which can be calculated directly from the shape operator W. This approach is also used in our screen space curvature algorithm. Therefore, both of these existing algorithms can be used in the screen space together with our proposed solution for the estimation of the principal curvatures.

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 concave areas should be dark, while convex areas are usually fully lit by light. Another problem is linked with the process of obtaining the final Equations (6.9) and (6.10) from Section 6.2.2. The sphere with a certain radius is used, but the selection of the radius is difficult to set.

Based on previously mentioned information, we propose a new function to map curvature to the AO. 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 $\langle a, 1\rangle$.

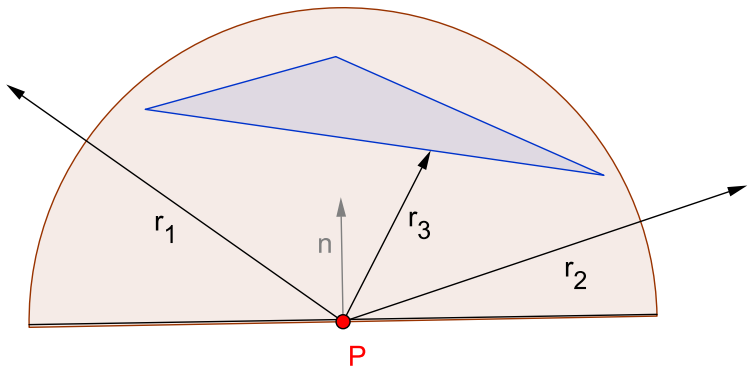


Figure 7.3: Curvature based AO limitation

7.4 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.

The limitation connected to the 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 7.3. 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.

7.5 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.

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 7.4.

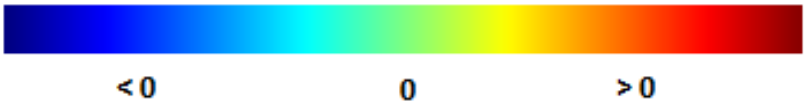


Figure 7.4: The color gradient used in all presented visualizations

7.5.1 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 7.5). 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.

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

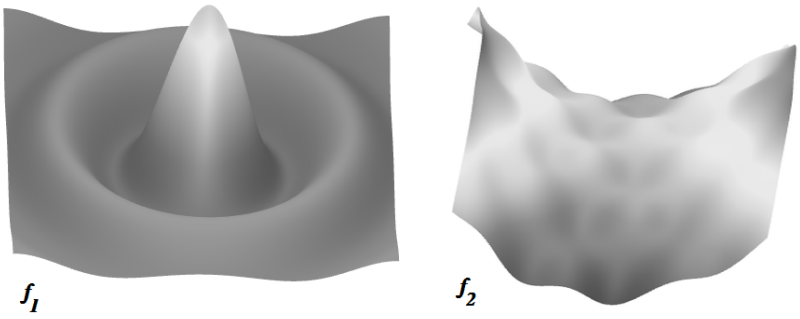
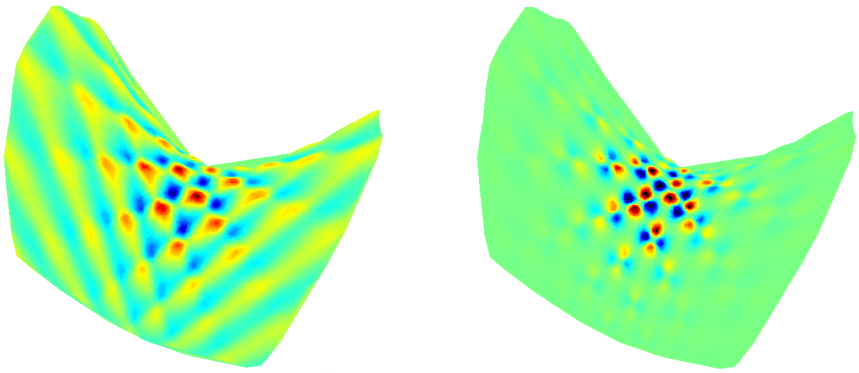


Figure 7.5: Tested functions $f_1 = 10\frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2)}}$, $(x \neq 0), (y \neq 0)$, $f_2 = sin(x)cos(y) + 0.1(x^2 - y^2)$, $x, y \in < -10; 10 >$



(a) Mean curvature (b) Gaussian curvature

Figure 7.6: Curvatures of the function f_2 calculated from the triangle mesh

computed curvature from the input function. See Figure 7.6 for the result of the curvature for the function f_2.

Result of the comparison is in Figure 7.7. 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.

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

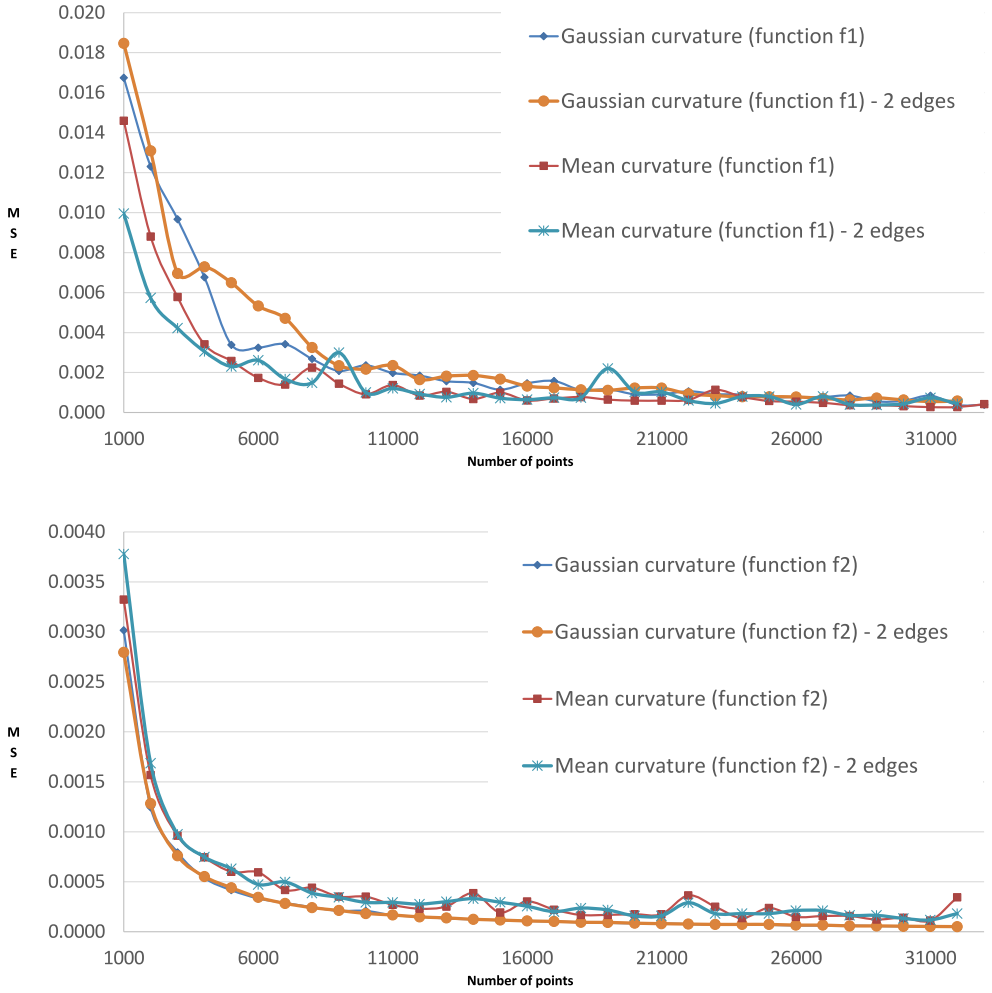
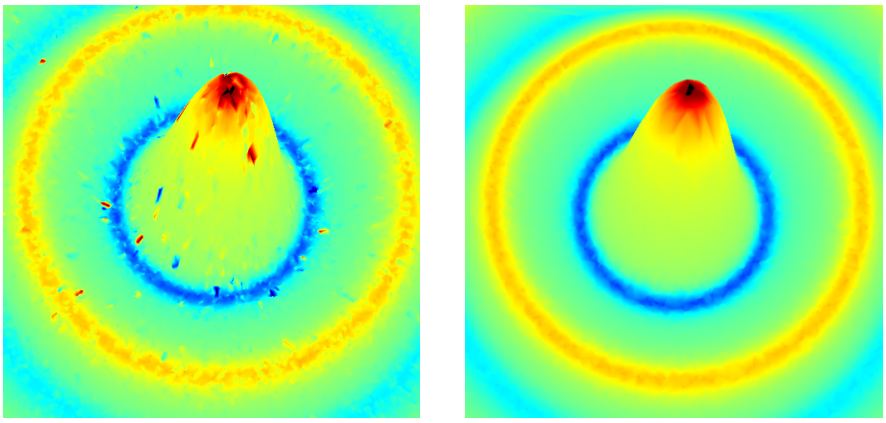


Figure 7.7: MSE of the method for the triangle mesh compared to the analytically computed curvature evaluated directly from the implicit function



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

Figure 7.8: The comparison of the curvature of f_1, compared on a tessellation created from the random point cloud.

tests). The result can be seen in Figure 7.8. 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.

7.5.2 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 7.9 and 7.10.

	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 7.9 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.

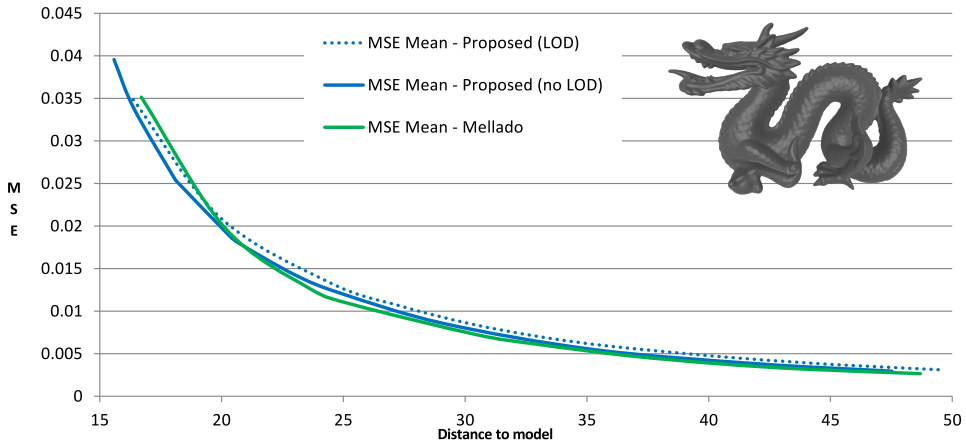
The Gaussian curvature comparison was done only with and without LOD, since there is no other screen space method known to us that calculates the Gaussian curvature. See results in Figure 7.10. The behavior is similar to Figure 7.9, with a roughly doubled amount of the MSE error. This is caused by the 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.

The visual comparison of the proposed method with [Mel+13] can be seen in Figure 7.11. Both algorithms have a comparable visual quality. The proposed method results look sharper, [Mel+13] is more blurry.

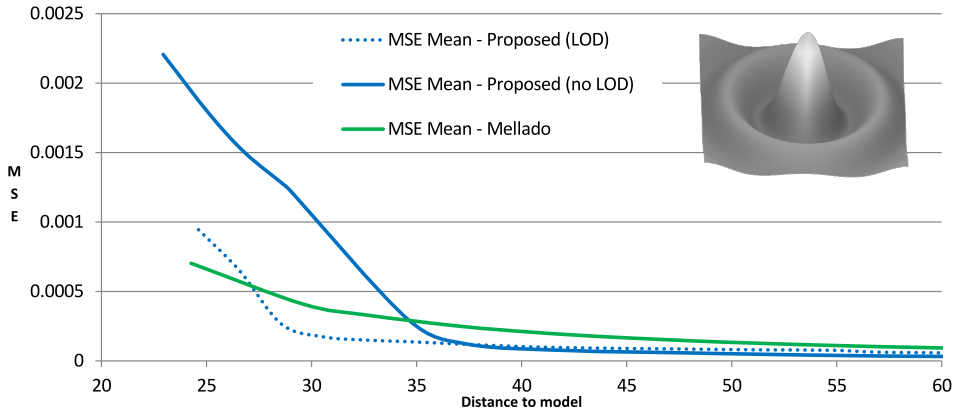
No LOD is used to show real differences based on the camera distance. For the camera at a greater distance (full model), there is almost no visible difference. With the camera closer to the surface (detailed parts of the image), the triangles of the mesh begin to appear in the screen space curvature.

For visual comparison of the quality of the proposed method in the screen space against the same method in the object space see Figures 7.13 and 7.12.

The effect of the used LOD can be seen in Figures 7.14 - 7.16. 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 7.14). On the other hand, in the example of the Gaussian curvature in Figure 7.15, the use of LOD improved the result considerably. Another comparison can be seen in the closeup in Figure 7.16. 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

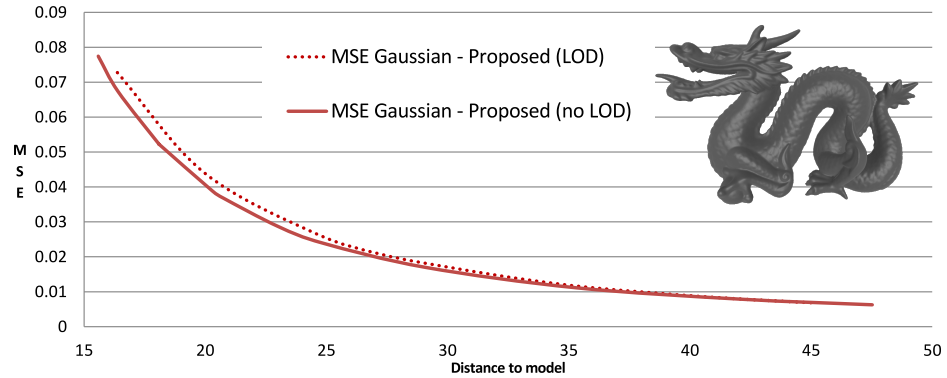


(a) Stanford Dragon

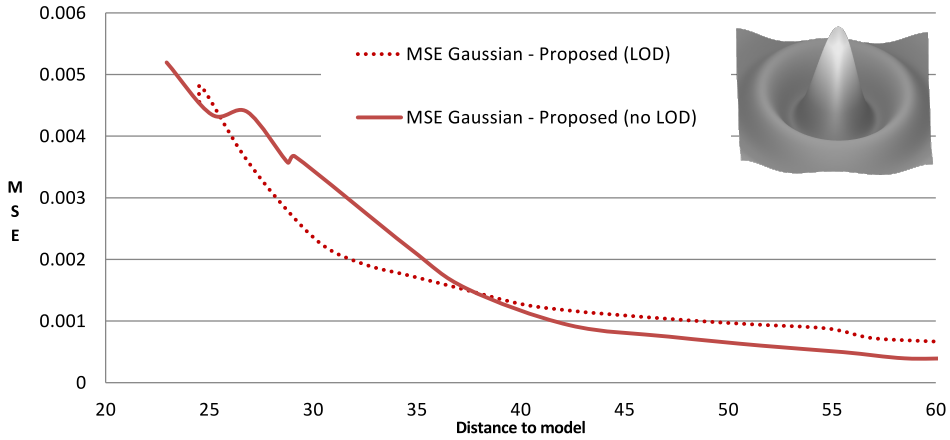


(b) Function f_1

Figure 7.9: 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.

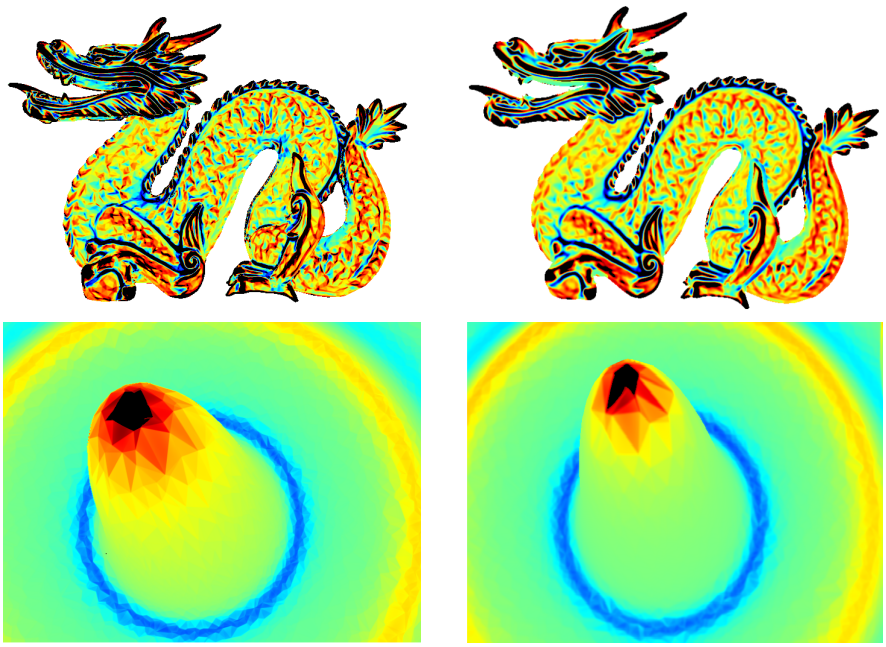


(a) Stanford Dragon



(b) Function f_1

Figure 7.10: Comparison of the proposed screen space MSE for the Gauss curvature calculated directly from the triangle mesh.



(a) Proposed method (b) Algorithm from [Mel+13]

Figure 7.11: Comparison of the mean curvature. Because [Mel+13] has no LOD, the presented comparison also uses none to create comparable images.

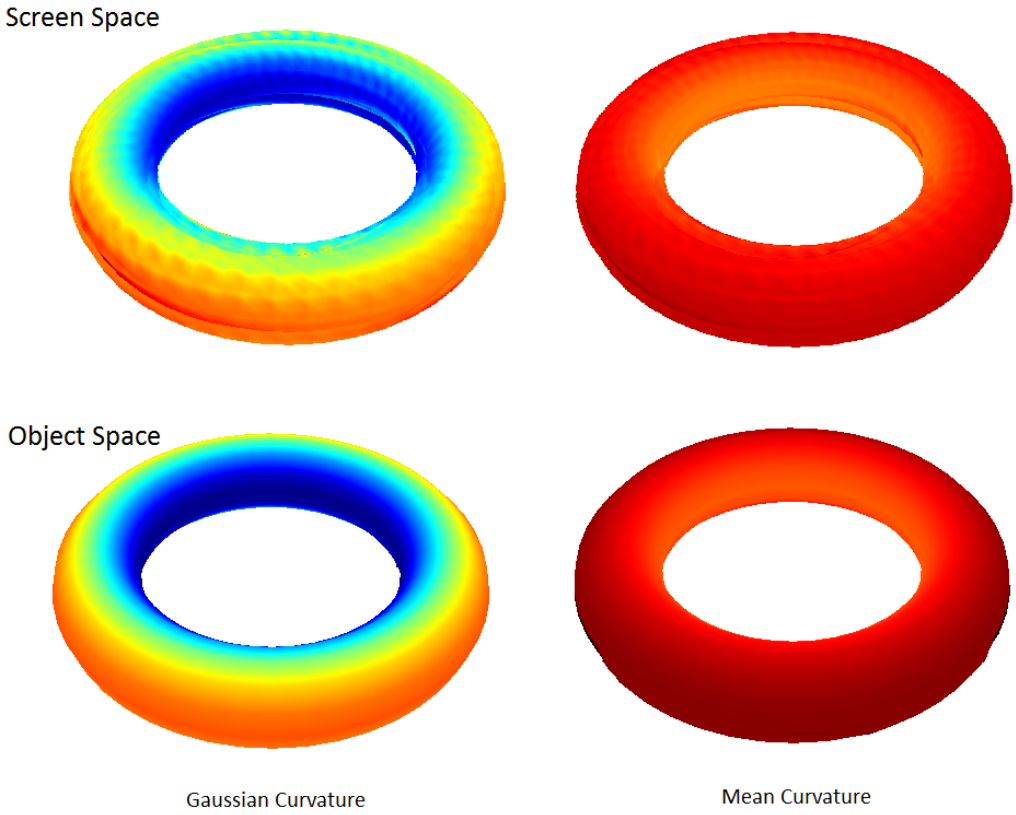
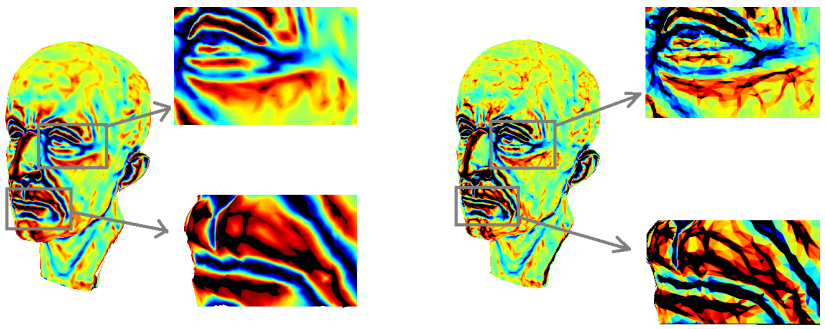


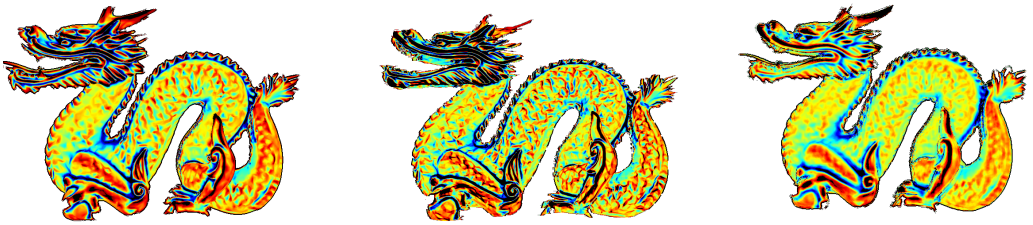
Figure 7.12: Summary comparison of mean and Gaussian curvature computed in the screen space with our solution and ground truth data



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

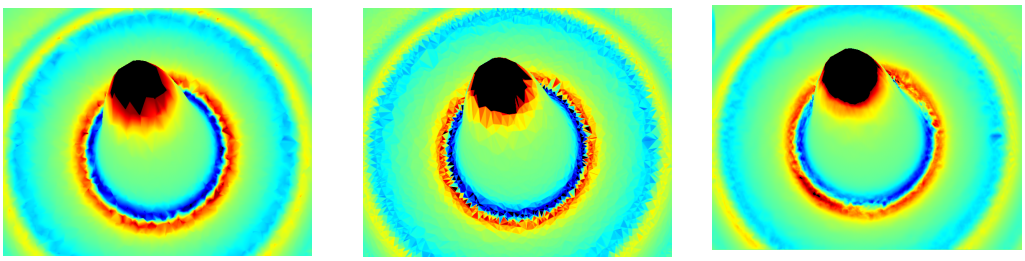
Figure 7.13: Comparison of the mean curvature for the MaxPlanck model.

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 7.14: Comparison of the mean curvature.

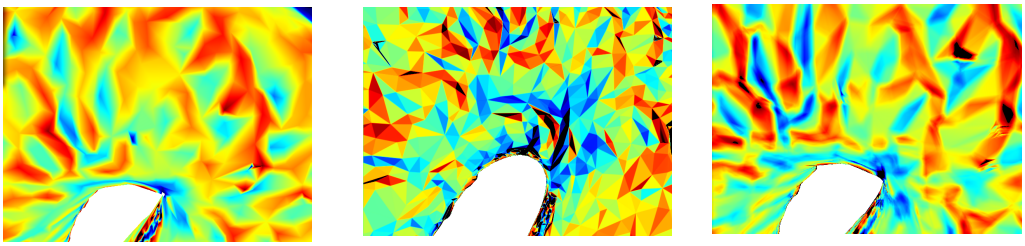


(a) Object space (b) Screen space without LOD (c) Screen space with LOD

Figure 7.15: Comparison of Gaussian curvature using function f_1.

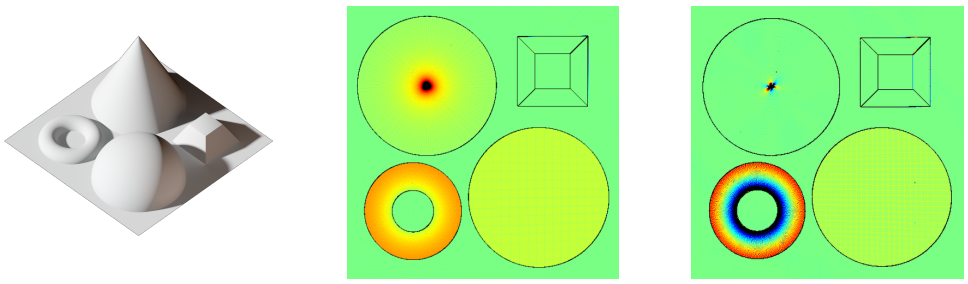
The advantage of curvature estimated directly in the screen space is the possibility to reduce geometry and use normal mapping to add missing details. Our proposed algorithm can be used together with this approach. We have used a plane with details added from a normal map. Since we used fine details, LOD should be disabled in this case. The results can be seen in Figures 7.17b and 7.17c. 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 7.17a. The mean curvature (Figure 7.17b) has a lower noise and therefore a higher quality than the Gaussian curvature (Figure 7.17c). This corresponds to the visual quality of tests in Figures 7.14 and 7.15. 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) Object space (b) Screen space without LOD (c) Screen space with LOD

Figure 7.16: Detail of the mean curvature



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

Figure 7.17: Mean and Gaussian curvature for normal mapped plane. Black borders are caused by a high curvature that is outside the used mapping function.

(a) Ray-casted occlusion (b) Hattori et al. [HKM11] (c) Griffin et al. [Gri+12] (d) Proposed

Figure 7.18: Ambient occlusion estimated using the proposed curvature algorithm

7.5.3 Ambient Occlusion

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 7.18. Both results were computed without LOD because the underlying mesh is of a high quality.

Hattori's solution uses the 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. Griffin's solution 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.

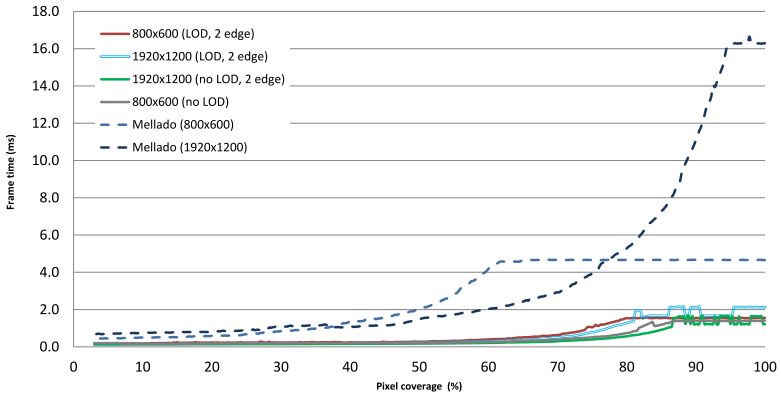
In our proposed solution (see Figure 7.18d), the result is not as dark as [Gri+12] and looks more like ray-casted result. On the other hand, solution from [HKM11] keeps correct white areas, but the rest of the model is too dark. We are using the threshold a from Equation 7.14 set to $a = 0.9$. If we 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.

7.5.4 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 $2 - 100$ per cent 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 7.19. 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 $80 - 100$ per cent of the screen coverage in Figure 7.19a 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 7.19: 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.

The frame times for the ambient occlusion are equal to the version without it. The occlusion computations itself are very fast due to the simplicity of the used Equation (7.14).

Chapter 8

Future work

Our future work will be divided into two main categories - curvature estimators and the use of shape characteristics.

8.1 Curvature estimators

We have already proposed one curvature estimator [PVK16]. This one was designed for an interactive curvature estimation, e.g. during editing of geometry, but the accuracy of the estimation was not crucial to us. Our next target is to create a curvature estimator with an improved quality of estimation for certain types of data. We are going to focus on data with exact normal vectors, since these data can be easily obtained. The possible availability of exact normals for curvature estimation is rarely discussed in existing curvature estimation algorithms. With exact normals, we are able to estimate curvature with higher quality but none of the state-of-the-art algorithms is primary designed to exploit exact normals.

There are many softwares which use analytical tools to describe data during modeling. In the end, they are able to export these data as a discrete representation with added exact normal vectors at vertices. These exported data can be used elsewhere without the need of original modeling software or its proprietary data format.

Based on our experiments, we have decided to create a surface-fitting-based estimator. It can highly benefit from normal vectors, since they are related to the gradient of such a surface. Based on this knowledge, the fitting surface can be improved.

Our proposed solution is based on Radial Basis Functions (RBF) for surface fitting. Description of a surface with RBF was already presented by Carr et. al. [Car+01]. However, simple RBF presented in [Car+01] is not a very stable algorithm for surface reconstruction. The need for off-surface points is a limitation and if these points are incorrectly chosen, the surface is noisy or self-intersecting. A better surface reconstruction can be obtained

with [MGV09]. They use not only vertices, but also normal vector at them. This suits our needs, because we are targeting to data equipped with exact normal vectors. Once the surface is fitted with an interpolant, curvature can be directly estimated using equations from Section 3.3.1. The same solution can also be used in combination with approximation for non-exact or noisy data.

There are other state-of-the-art algorithms based on surface fitting. A similar approach to our proposed solution has already been proposed by Goldfeather and Interrante [GI04]. However, based on our experiments, their solution is hard to configure and there are many ways, how to choose the size of k-ring neighborhood or the final fitting method from the ones described in the article.

8.2 Use of shape characteristics

Another area, which we would like to investigate, is the use of shape characteristics for registration of objects created from point clouds or triangulated geometry. This is an important and interesting topic due to the boom of 3D scanning and printing. The object is scanned from several directions which leads to several partially overlapping data sets. The goal is to register these sets and obtain the whole model which can be further edited or directly printed. To register data sets, overlapping parts have to be found.

One of the latest methods that tries to solve object alignment problem was proposed by Mellado et. al. [MAM14]. However, their solution is not using shape characteristics that are quite useful for this kind of problems. Based on the shape, we are able to identify feature points on both sets and later decide which points should be paired together. Recently, Zhou et al. [ZPK16] proposed a solution based on FPFH [RBB09]. They calculate descriptors for both models and reduce their numbers by removing false pairs based on several conditions. The registration computation from this reduced set is very fast, since the number of input descriptors is decreased. However, the accuracy of the method is still based on the quality of underlying descriptor (FPFH in this case) because they still have to compute them for both models.

Shape characteristics based on curvature can be used directly. However, using curvature directly is not recommended, as explained in Section 5.1. We would like to focus on creating a hybrid algorithm for objects registration that combines curvature with another approach to create a robust solution. A potential problem is a symmetry of models. The state-of-the-art shape characteristics and descriptors are not performing well for data with a certain degree of symmetry. Local descriptors create usually the same output for the symmetrical geometry because they neglect the position of the registered parts against the whole model. Based on this, the registration of symmetrical

Figure 8.1: Incorrectly aligned symmetrical parts of a human head

models with the use of these descriptors ends up with incorrect position. This problem can be observed if we use e.g. [ZPK16] for human heads. The left and right ear are incorrectly paired together, resulting in a reflected part of the model (see Figure 8.1).

Chapter 9

Conclusion

Shape characteristics are an important topic in computer graphics. They can be used for many tasks, including objects description, registration, feature points detection etc. We have primary focused on one shape characteristic - curvature. It is quite easy to obtain and can be used with some minor modifications to create geometry objects descriptors.

In the Chapter 3, selected state-of-the-art methods for curvature estimation are presented. Based on the research in this area, we have proposed (see Chapter 7) a simple, yet effective and easy to implement solution for curvature estimation in the screen space. This proposed algorithm can estimate curvature in real-time. It can be easily added to existing rendering pipelines. The algorithm can also be used for a low polygonal geometry where fine details are added from a normal map. The proposed method can be also used for estimation of ambient occlusion.

However, our proposed solution has some limitations that are similar to other screen-space algorithms. There are possible problems with depth discontinuities caused by depth-buffer. This problem is difficult to solve, because in screen-space techniques we usually have no information about original data and the surface reconstruction cannot be done correctly. Another problem is related to LOD. Small details are smoothed and lost if camera is far away from the surface.

The findings from state-of-the-art in curvature estimation (see Chapter 8) will also be used in our proposed future work. Based on the reviewed literature, there is no algorithm targeted primarily for data equipped with exact normals. These data are easy to obtain for objects created in a modeling software. In the first part of our future work, we are going to propose an algorithm based on these assumptions, as stated in Chapter 8.

State-of-the-art in shape characteristics is presented in Chapter 5. We have mainly focused on methods based on curvature, but some other solutions are discussed as well. The knowledge summarized in this section will be used in our future work for creating symmetry aware registration algorithm, as was discussed in Chapter 8.

Bibliography

[All+03] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. "Anisotropic Polygonal Remeshing". In: *ACM Trans. Graph.* 22.3 (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*. 2009, pp. 697–702.

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

[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.

[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 (2009), pp. 1201–1208. ISSN: 1077-2626.

[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*. 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.

[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.

[FM08] D. Filion and R. McNaughton. “Effects & Techniques”. In: *ACM SIGGRAPH 2008 Games*. SIGGRAPH '08. Los Angeles, California: ACM, 2008, pp. 133–164.

[FPA13] A. B. F. P. Aalund. *A Comparative Study of Screen-Space Ambient Occlusion Methods*. Bachelor Thesis. 2013.

[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 (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 (2004), pp. 45–63. ISSN: 0730-0301.

[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.

[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.

[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.

[HL10] T. D. Hoang and K. L. Low. "Multi-resolution Screen-space Ambient Occlusion". In: *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*. VRST '10. Hong Kong: ACM, 2010, pp. 101–102. ISBN: 978-1-4503-0441-2.

[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.

[HPW05] K. Hildebrandt, K. Polthier, and M. Wardetzky. "Smooth Feature Lines on Surface Meshes". In: *Proceedings of the Third Eurographics Symposium on Geometry Processing*. SGP '05. Vienna, Austria: Eurographics Association, 2005. ISBN: 3-905673-24-X.

[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.

[JS05] J. Janne and L. Samuli. "Ambient Occlusion Fields". In: *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*. ACM Press, 2005, pp. 41–48.

[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 (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.

[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.

[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.

[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`. 2015.

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

[LR11] F. Lindemann and T. Ropinski. “About the Influence of Illumination Models on Image Comprehension in Direct Volume Rendering”. In: *Visualization and Computer Graphics* 17.12 (2011), pp. 1922–1931. ISSN: 1077-2626.

[LS10] B. J. Loos and P. P. Sloan. “Volumetric obscurance”. In: *I3D ’10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. Washington, D.C.: ACM, 2010, pp. 151–156. ISBN: 978-1-60558-939-8.

[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.

[MAM14] N. Mellado, D. Aiger, and N. J. Mitra. “Super 4PCS Fast Global Pointcloud Registration via Smart Indexing”. In: *Computer Graphics Forum* 33.5 (2014), pp. 205–215. ISSN: 1467-8659.

[Mar+14] M. Mara, M. McGuire, D. Nowrouzezahrai, and D. Luebke. *Fast Global Illumination Approximations on Deep G-Buffers*. Tech. rep. NVR-2014-001. NVIDIA Corporation, 2014, p. 16.

[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 (1999), pp. 1–6. ISSN: 1086-7651.

[McG+11] M. McGuire, B. Osman, M. Bukowski, and P. Hennessy. “The Alchemy Screen-Space Ambient Obscurance Algorithm”. In: *High-Performance Graphics 2011*. Vancouver, BC, Canada, 2011.

[McG10] M. McGuire. “Ambient Occlusion Volumes”. In: *Proceedings of High Performance Graphics 2010*. Saarbrucken, Germany, 2010.

[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, 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. 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*. 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.

[MML12] M. McGuire, M. Mara, and D. Luebke. “Scalable Ambient Obscurance”. In: *High-Performance Graphics 2012*. Paris, France, 2012.

[Mor+03] A. B. Moreno, A. Sánchez, J. F. Vélez, and F. J. Díaz. "Face recognition using 3d surface-extracted descriptors". In: *In Irish Machine Vision and Image Processing Conference (IMVIP 2003), Sepetember*. 2003.

[MSR07] E. Magid, O. Soldea, and E. Rivlin. "A comparison of Gaussian and mean curvature estimation methods on triangular meshes of range image data". In: *Computer Vision and Image Understanding* 107.3 (2007), pp. 139 –159. ISSN: 1077-3142.

[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 (2009), pp. 611–623. ISSN: 1520-9210.

[OBS04] Y. Ohtake, A. Belyaev, and H. P. Seidel. "Ridge-valley Lines on Meshes via Implicit Surface Fitting". In: *ACM Trans. Graph.* 23.3 (2004), pp. 609–612. ISSN: 0730-0301.

[PF05] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Chapter 14. Dynamic Ambient Occlusion and Indirect Lighting. Addison-Wesley Professional, 2005. ISBN: 0321335597.

[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.

[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.

[PV17] M. Prantl and L. Váša. "Estimation of differential quantities using Hermite RBF interpolation". In: *Visual Computer* (2017). After the 2nd round of reviews.

[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. 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: *Submited to CCIS – Communications in Computer and Information Science*. Extended version of "Fast Screen Space Curvature Estimation on GPU" send based on request from GRAPP comitee. Springer Berlin Heidelberg, 2017.

[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.

[RBA09] C. K. Reinbothe, T. Boubekeur, and M. Alexa. "Hybrid Ambient Occlusion". In: *Eurographics 2009 - Areas Papers, Munich, Germany, March 30 - April 3, 2009*. 2009, pp. 51–57.

[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*. 2009, pp. 3212–3217.

[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 (IAS-10*. 2008.

[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`. 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*. 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.

[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*. 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/`. 2016.

[Sur+03] T. Surazhsky, E. Magid, O. Soldea, G. Elber, and E. Rivlin. “A comparison of Gaussian and mean curvatures estimation methods on triangular meshes”. In: *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*. Vol. 1. 2003, 1021–1026 vol.1.

[Tau95] G. Taubin. “Estimating the tensor of curvature of a surface from a polyhedral approximation”. In: *Computer Vision, 1995. Proceedings., Fifth International Conference on*. 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*. 2004, pp. 288–297.

[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 (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`. 2017.

[Vai13] R. Vaillant. *Recipe for implicit surface reconstruction with HRBF*. `http://rodolphe-vaillant.fr/?e=12`. 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.

[VP17] L. Váša and M. Prantl. “Surface Registration Using Data-Induced Rigid Transformation Distance Metric”. In: *Submited to SGP 2017* (2017).

[Vá16] 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.

[Wol06] E. Wolfgang. *Shader X5: Advanced Rendering Techniques*. Rockland, MA, USA: Charles River Media, Inc., 2006. ISBN: 1584504994.

[YBS05] S. Yoshizawa, A. Belyaev, and H. P. Seidel. “Fast and Robust Detection of Crest Lines on Meshes”. In: *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*. SPM ’05. Cambridge, Massachusetts: ACM, 2005, pp. 227–232. ISBN: 1-59593-015-9.

[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.

[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.

[ZIK98] S. Zhukov, A. Iones, and G. Kronin. “An ambient light illumination model”. English. In: *Rendering Techniques 98*. Ed. by G. Drettakis and N. Max. Eurographics. Springer Vienna, 1998, pp. 45–55. ISBN: 978-3-211-83213-4.

[ZPK16] Q. Zhou, J. Park, and V. Koltun. “Fast Global Registration”. In: *Computer Vision – ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II*. 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. 2007, pp. II–249–II–252.

Appendix A

Professional Activities

Publications

- published

  [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. 2016, pp. 151–160. ISBN: 978-989-758-175-5.

  [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.

  [Vá16] 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.

- submitted

  [PV17] M. Prantl and L. Váša. "Estimation of differential quantities using Hermite RBF interpolation". In: *Visual Computer* (2017). After the 2nd round of reviews.

[PVK17] M. Prantl, L. Váša, and I. Kolingerová. “Screen Space Curvature and Ambient Occlusion”. In: *Submited to CCIS – Communications in Computer and Information Science*. Extended version of "Fast Screen Space Curvature Estimation on GPU" send based on request from GRAPP comitee. Springer Berlin Heidelberg, 2017.

Related Talks

- M. Prantl: Vizualizace volumetrických dat, University of West Bohemia in Pilsen, Czech Republic, December 2013
- M. Prantl: Výpočet křivosti ve screen space, University of West Bohemia in Pilsen, Czech Republic, May 2015
- M. Prantl: Křivosti, University of West Bohemia in Pilsen, Czech Republic, May 2016
- M. Prantl: Deskriptory, University of West Bohemia in Pilsen, Czech Republic, June 2017

Participation on Scientific Projects

- Advanced Computing and Information Systems. Project code SGS-2013-029.
- Advanced Computing and Information Systems. Project code SGS-2016-013.
- 1st Internal grant scheme of DCSE+P2, 2015
- 2nd Internal grant scheme of DCSE+P2, Metody registrace 3D modelu, 2016
- 2nd Internal grant scheme of DCSE+P2, Metody pro změnu věku 3D identitiku, 2016