GPU Accelerated Computation of Geometric Descriptors in Parametric Space

Anthousis Andreadis Georgios Papaioannou Pavlos Mavridis

Department of Informatics, Athens University of Economics and Business
76 Patission Str., 10434 Athens, Greece
`{anthousis,gepap,pmavridis}@aueb.gr`
`http://graphics.cs.aueb.gr`

Abstract. We present a novel generic method for the fast and accurate computation of geometric descriptors. While most existing approaches perform the computations directly on the geometric representation of the model, our method operates in parametric space, decoupling the computational complexity from the underlying mesh geometry. In contrast to other parametric space approaches, our method is not restricted to specific descriptors or parameterisations of the surface. By using the parametric space representation of the mesh geometry, we can trivially exploit massive parallel GPU architectures and achieve interactive computation times, while maintaining high accuracy. This renders the method suitable for computations involving large areas of support and animated shapes.

Keywords: Geometric descriptors, Parametric space, Mesh geometry, GPU acceleration

1 Introduction

The computation of *geometric descriptors*, like curvature, is central in a wide range of applications, including object retrieval, registration, texture synthesis, stylized rendering and many more. The computation of these fundamental metrics is usually performed by algorithms that operate directly on discrete polygonal representations of the continuous surface. In the case of static meshes, these geometric descriptors can be computed once without worrying about the performance. In contrast, in the case of moderately dense meshes with large areas of support and especially in the case of animated or dynamic meshes, performance becomes critical and this computation process becomes a challenging task.

In this work we focus on the general class of metrics with finite local support, whose computation depends on the local neighbourhood of an arbitrary point $\mathbf{p}$ on the object's surface. Robustness in the presence of noise is achieved by performing computations at multiple scales [30]. The computation of these types of descriptors often relies on data structures that encode the adjacency information and allow efficient discovery of the neighbouring points on the surface. This is especially true for algorithms that operate on meshes. The computational complexity of such *object-space* approaches is directly proportional to the geometric

Fig. 1. a) "Lucy" model (200K) colourized with mean curvature, computed in 49ms, b) geometry (position) buffer (normalized for visualization), c) surface normal buffer, d) polygon chart identifiers (colourized for clarity) along with the adjacent chart identifiers on border pixels, e) mean curvature in parametric space (colourized for visualization).

density and quadratic with respect to the extent (i.e. radius) of the local area of support. Despite the fact that computing the metric for independent surface points is an inherently parallel task, the use of complex data structures for storing the adjacency information, prevents a trivial and efficient mapping of these computations to massively parallel stream processors, like commodity GPUs, at arbitrary scales. For these reasons, real-time computation is often limited to meshes with relatively low geometric complexity and 1-ring vertex neighbourhoods [6].

In order to alleviate these limitations, we shift all computations from *object-space* to *parametric space*, by transferring all the geometric data of the object to a two-dimensional layout, along with extra adjacency information that allows us to reconstruct the object-space local neighbourhood of a given point on the fly. While this choice is similar to *Geometry Images* [7], we do not restrict our method by requiring a specific parameterisation of the surface, but rather develop a scheme that handles any underlying parameterisation, including multi-chart layouts. The benefits of parametric-space computations are twofold: First, sampling the geometry at arbitrarily large areas of support is much more efficient in parametric space, since the samples can be directly indexed in contrast to a geometry-based estimation, where the traversal of a surface patch is performed via the connectivity information of the vertices. Second, the parametric space computations are directly mapped to the GPU/many-core computing paradigm in a very efficient manner, rendering the approach suitable for real time calculations over deformable or animated objects. Another gain that stems from the utilization of GPUs, is that we have access to linear interpolated data between all the sample points of the surface with minimal impact on the performance, as linear interpolation is natively supported by the hardware. This is very use-

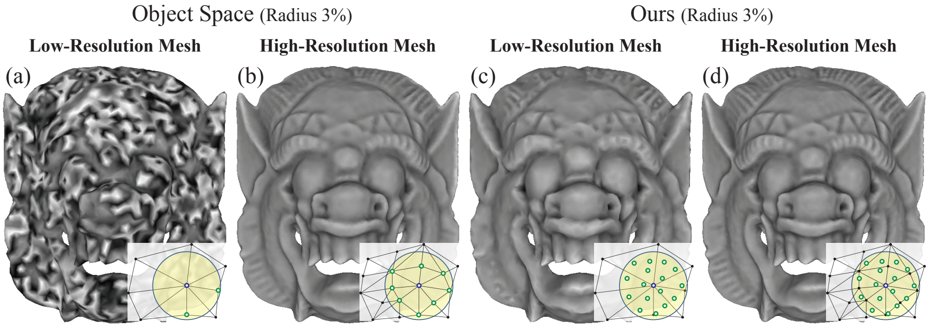

Fig. 2. a) Using an *object-space* approach on a sub-sampled surface (low-resolution mesh) with a relative small support radius (3% of the object's diagonal), results in inaccuracies. b) The same computations applied on the densely sampled surface (high-resolution mesh). c) Using our parametric-space approach on the low-mesh we obtain accurate results without any extra effort, due to the linear interpolated samples of the surface. d) Results of our approach used on the high-resolution mesh.

ful, especially in cases of sub-sampled surfaces, where *object-space* methods give inaccurate results for small support radius (see Figure 2).

2 Related Work

Most of the existing methods in the bibliography concentrate on the computation of a specific geometric descriptor, and do not try to generalize their framework. For our overview, we do not focus on the specific descriptors used in the existing works, but rather focus and classify methods, based on how they sample the geometric information of the object. Existing methods can be classified to those that sample the geometry in *object-space*, in *screen space*, from a *volumetric representation* and those that operate in *parametric space*. In the remainder of this section we will review the main representatives for each category.

Object-space methods operate directly on the discrete mesh representations of a surface. [29] and [18] generalize the differential-geometry-based definition of curvatures to discrete meshes but their computations are limited to 1-ring neighbourhoods, which renders them sensitive to noise. Similarly [23] estimate the curvature over meshes using essentially a 2-ring neighbourhood. For efficient arbitrary neighbourhoods, *object-space* methods require a data structure that encodes the adjacency information between the triangles of the mesh, such as the *half-edge* [2] or a kd-tree data structure. However, as discussed in the introduction, a mapping of this data structure to the GPU is neither trivial nor optimal. Most of the existing methods belong to this category and thus operate on the CPU. GPU-based methods, have been proposed for the computation of specific descriptors, like curvature [6], but these methods do not generalize to the sampling of arbitrary neighbourhoods.

Screen space methods sample the geometric information of a mesh from a 2D pixel buffer, where each pixel encodes the projected surface position of the mesh from a specific point of view. In this form of representation, adjacency information is implied by the pixel grid, and therefore sampling is trivial and can be efficiently mapped to GPUs. This efficiency in sampling is also the main motivation behind our method. The main disadvantage of screen-space methods is that computations are limited to the surface points visible from a particular view, resulting in inaccuracies near occluded points and at the screen-space silhouettes of the object. Such screen-space methods have been proposed for curvature estimation in real-time stylized rendering [17], [13]. Our method retains most of the sampling efficiency of the screen-space methods, but avoids the view-dependence of the results by moving all the computation to the parametric space.

Volumetric data and algorithms can be also employed for the computation of descriptors. In this case, the input mesh is initially converted to a volumetric representation, such as a *level set*, and geometric descriptors are computed by sampling this representation, instead of the original mesh. Finally, the results of these calculations can be mapped back to the original mesh. The advantage of this approach is that the computational complexity does not depend on the underlying geometry but rather on the new volumetric representation, where sampling a local neighbourhood around a surface point is often more efficient than sampling the same neighbourhood on the original geometry. Features, like curvature, can be quickly approximated using the gradient field of the object, as described in the OpenVDB [19] or by using convolutions, which can be accelerated using FFT as shown in [22]. The disadvantage of this approach is that an efficient voxelisation method is required, additional memory is consumed for the storage of the volumetric format and most importantly, the computations are based on a volumetric discretisation, which is a more rough representation of the original surface than the triangular mesh. Furthermore, certain descriptors when computed on volumetric data, are incompatible with the results of the respective surface-based measurements, especially for non-manifold surfaces.

Finally, **parametric space** methods have also been proposed. Methods of this category rely on a the unwrapping of the model's surface on a 2D plane. Using this representation, computational complexity is decoupled from the underlying geometry and additionally, several image analysis techniques can be applied intuitively to 3D data. To our knowledge, so far there has been no practical and generic approach that would allow both geometric and image space descriptors to be computed efficiently, as existing methods focus on applying image space techniques only. [20] propose a method for corner and edge detection that requires a user-driven single chart parameterisation. Furthermore, to handle points lying near the perimeter of charts, the authors construct complementary parameterisations, for which boundary regions are then mapped to internal chart locations. [10] describe another method that locates extrema using a scale space representation. This approach method relies on a specialised conformal mapping and expects pre-computed per-vertex values of mean-curvature and geodesic distance.

In contrast, our method does not rely on a specific parameterisation approach, nor does it require any pre-computed descriptors.

3 Methodology

Our method operates on fully parameterised geometry but does not rely on a specific method for this task. Initially, we perform a pre-processing step in order to locate the surface edges of the polygonal representation, which are mapped to discontinuous regions in parametric space. This is usually part of the model loading process. In real-time, we create the parametric-space representation of the geometry, augmented by the adjacency information and perform the computation of discrete locations in parametric space, i.e. on a texture buffer. During this step, we utilize the information stored in our geometry and adjacency buffers in order to index arbitrary surface samples in the neighbourhood of a point $\mathbf{p}$, regardless of its parametric mapping. The measured metrics can be then queried per vertex, using standard texture look-up operations, or used directly in image space, e.g. to extract salient features and local image-space descriptors. In the rest of this section we each one of the above steps in detail.

3.1 Surface Parameterisation

Surface parameterisation as explained in [5], can be viewed as a one-to-one mapping from a suitable domain to a surface. The parameter domain is also a surface and thus the procedure maps one surface onto another. Our method expects fully parameterised geometry in a normalized 2D domain. This procedure is also known as (bijective) *uv-mapping* and the resulting surface patches are referred to as *charts* or *uv-islands* (see Figure 1*(d)*, and 3). The area of surface parameterisation has been extensively researched in the past years, [5], [26] and the minimization of stretch distortion has been the goal of several works, such as that of [25], [31] and [32]. Therefore, we do not address this part in our work, but rather rely on existing methods and solutions.

3.2 Pre-Processing Operations

The estimation of a local descriptor, requires the calculation of an operator $F(\mathbf{p}, S(\mathbf{p}))$ at a point $\mathbf{p}$, given a neighbourhood $\mathbf{x} \in S(\mathbf{p})$, where $\mathbf{x}$ satisfies a set of criteria, such as a maximum Euclidean or geodesic distance from $\mathbf{p}$. Finally another option, is to use the n-ring adjacency of $\mathbf{x}$ to $\mathbf{p}$ (max. n vertex graph distance), but due to the imposed limitation of uniform triangulation of the surface, this is often impractical. These relations in geometric space are easily represented using data structures with topology. For a review of the existing geometric data representations, see the work of [4].

On the other hand, when operating in 2D parametric space, the connectivity information is implied by the adjacency of neighboring pixels. However, this is not true on the borders of charts, where adjacent geometry is mapped to

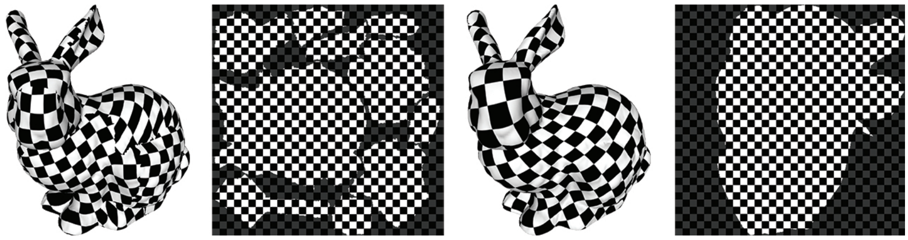

Fig. 3. The "bunny" model with two parameterisations, resulting in different set of *charts*.

discontinuous locations in parametric space (see example in Figure 4). In this case, additional information should be stored at the border pixels to keep track of the hops to geometrically-adjacent pixels in different charts.

In order to appropriately annotate the chart pixels, mesh vertices located at the borders of charts must be first identified and the link to the geometrically adjacent vertices on different charts has to be stored on the affected vertices. Details regarding the information stored can be found in Section 3.3. The complexity of this step is equivalent to the pre-processing stage of all object-space approaches for the adjacency information generation (e.g. half-edge data structure) and even for large meshes, it only takes a few seconds to complete. This stage needs to be performed only once, as the adjacency information for topologically unchanging geometry can be stored in the 3D model file itself.

3.3 Data Buffer Generation

The computation of geometric descriptors requires a set of attributes per sampled surface location, such as the coordinates of $\mathbf{p}$ in the object's local space and the respective normal vector $\mathbf{n}$. These data must be transferred to the parametric space and stored in appropriate buffers, i.e. a set of textures that correspond to the normalized parametric space of the unwrapped geometry. The buffers also store the identifier of the polygon chart that $\mathbf{p}$ belongs to. The object-space position of surface points is stored in a *geometry buffer* $P(u, v)$, the normal vectors are placed in a *normal buffer* $N(u, v)$, whereas the chart identifiers are registered in an ID channel in the geometry buffer $(ID(u, v))$. Another set of textures, comprising the *adjacency buffer*, equal in size to the geometry buffer, store the identifier of the adjacent chart, the *local metric distortion* of the parameterisation (see below) and the corresponding (u, v) coordinates on the adjacent chart. An example of the data channels for the position, normal and current and adjacent chart identifiers is shown in Figure 1.

The buffer generation process is performed in two steps. First, the geometric information is efficiently generated in the GPU by rasterising the object triangles using orthographic projection, where the normalized texture coordinates $(u, v, 1)$ are used as the vertex coordinates of the mesh. The chart ID is passed as a vertex attribute and copied for all points inside the triangles of a chart. Similarly to [24]

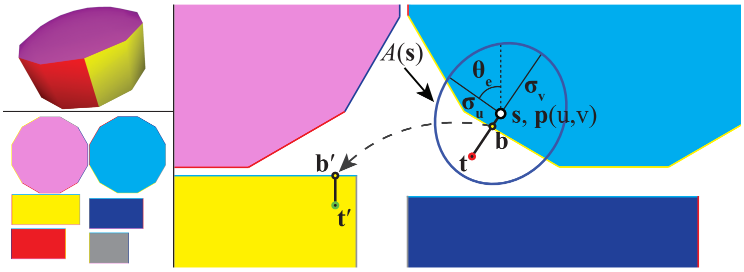

Fig. 4. Indexing a sample inside the neighbourhood of a point. Sample $\mathbf{t}$ does not lie inside the chart of $\mathbf{s}$. Locate the boundary point $\mathbf{b}$, read adjacency buffers and relocate sample to adjacent chart.

we also rasterise each chart's boundary edges in order to avoid the generation of disconnected regions.

In the second step, we compute the *local metric distortion* factors that will be used for the anisotropic adjustment of scale and sampling direction in various calculations. In order to do so we use the *Jacobian* $J_P = (P_u, P_v)$, where P_u and P_v the partial derivatives of the surface. The left-singular vectors of J_P are used in order to get the θ_e angular distortion of the anisotropic ellipse while the singular values of J_P σ_u, σ_v are the stretch factors in the u and v direction. Due to the fact that the singular value decomposition is a tedious task, we use the equivalent eigen decomposition of the 2x2 first fundamental form matrix:

$$J_P^T J_P = \begin{bmatrix} E & F \\ F & G \end{bmatrix}, \tag{1}$$

where $E = (\partial P(u,v)/\partial u)^2$, $G = (\partial P(u,v)/\partial v)^2$ and $F = (\partial P(u,v)/\partial u) \cdot (\partial P(u,v)/\partial v)$. For more information see [9]. Additionally, in this second pass we also store the rest of the adjacency data. These attributes are calculated when setting up the triangle connectivity and are simply copies to the adjacency buffers for the chart border pixels. While for static objects the buffer generation step could be performed only once, we focus on a method suitable for deformable/animated objects, and treat it as a per-frame event. Therefore, the reported timings in the remaining text include the data buffer generation overhead.

3.4 Sampling a Point's Neighbourhood

In order to be able to perform the calculation of a feature $F(\mathbf{p}, S(\mathbf{p}))$ in parametric space, we need to establish a procedure for drawing individual samples from the neighbourhood $S(\mathbf{p})$ of $\mathbf{p}$. Our approach estimates $F(\mathbf{p}, S(\mathbf{p}))$ in image space and therefore, for every pixel (i, j) with a corresponding parameter pair $\mathbf{s} = (u, v)$, $\mathbf{p}(u, v)$ is first retrieved from the geometry buffer: $\mathbf{p} = P(u, v)$. Then, assuming a maximum radius of support r_{max} for the local feature estimator in *object-space* units, a sample $\mathbf{t} = (u', v')$ is generated in a region $A(\mathbf{s})$ in

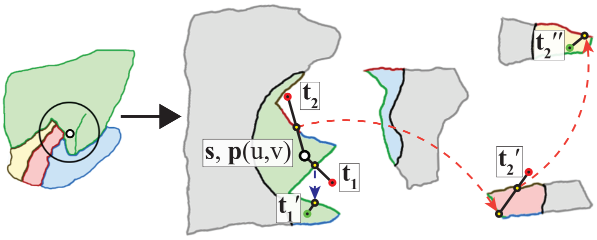

Fig. 5. More examples of indexing samples. $\mathbf{t_1}$ returns to the same chart after a jump. $\mathbf{t_2}$ parametric location is located using two jumps. In the right part, chart adjacencies are colored across the borders.

parametric space so that $\mathbf{x} = P(\mathbf{t})$ satisfies the neighbourhood criterion. $A(\mathbf{s})$ is calculated as an ellipse of radii $(1/\sigma_u(\mathbf{s}), 1/\sigma_v(\mathbf{s})) \cdot r_{max}$ in the parametric domain (upper distance bound) rotated by θ_e, in order to account for local angular distortion and scale, and $\mathbf{x}$ is acquired with rejection sampling according to the selected neighbourhood criterion (Figure 4). The exact pattern or random distribution with which the samples are generated is specific to the feature estimator and the generic sampling approach presented here is agnostic to it. Also, since we perform a random sampling of the neighbourhood of $\mathbf{s}$, no assumption is made about the chart's convexity.

Since the ellipse $A(\mathbf{s})$ may extend beyond the boundary of the chart containing $\mathbf{p}$ (Figure 4), a more elaborate mechanism is required to handle the samples that fall outside the chart. These samples obviously contribute to the result and should not be discarded. Identifying whether the sample $\mathbf{x}$ at $\mathbf{t}$ lands on the same chart as $\mathbf{p}$ is trivially resolved by checking their respective chart identifiers $ID(u, v)$ and $ID(u', v')$.

In the case where $\mathbf{t}$ lands outside the chart of $\mathbf{p}$, we utilize the parametric adjacency data stored in our buffers to find its true location. Initially, we march along the direction $\overrightarrow{\mathbf{st}}$ in pixel-sized increments to locate the first pixel with the chart ID as $\mathbf{p}$ (boundary point $\mathbf{b}$). The adjacency buffer for a border pixel $\mathbf{b}$ of a chart contains the ID of the adjacent chart and the parametric location $\mathbf{b}'$ of the corresponding point on it. For samples across seams of the same chart, the ID of the adjacent buffer is identical to that of $\mathbf{p}$, but the parametric location $\mathbf{b}'$ points safely to the corresponding location on the same chart (see Figure 5-$\mathbf{t_1}$). The adjacency buffer contains also the relative chart edge rotation $\theta(\mathbf{b} \to \mathbf{b}')$ between $\mathbf{b}$ and $\mathbf{b}'$. Finally, a non-uniform scale factor $s(\mathbf{b} \to \mathbf{b}')$ can be calculated, corresponding to the relative scale of the two charts in parametric space at their border locations $\mathbf{b}$ and $\mathbf{b}'$ (this scale factor may vary across a chart):

$$s(\mathbf{b} \to \mathbf{b}') = \left(\frac{\sigma_u(\mathbf{b}')}{\sigma_u(\mathbf{b})}, \frac{\sigma_v(\mathbf{b}')}{\sigma_v(\mathbf{b})} \right). \tag{2}$$

Therefore, we can adjust the location of $\mathbf{t}$ according to the following parametric space transformation to obtain the relocated sampling position $\mathbf{t}'$ on the adjacent chart:

$$\mathbf{t}' = \mathbf{b}' + \mathbf{R}_{\theta(\mathbf{b}\to\mathbf{b}')}\mathbf{S}_{s(\mathbf{b}\to\mathbf{b}')}(\mathbf{t} - \mathbf{b}), \tag{3}$$

where $\mathbf{R}_{\theta(\mathbf{b}\to\mathbf{b}')}$ is the rotation matrix of angle $\theta(\mathbf{b} \to \mathbf{b}')$ and $\mathbf{S}_{s(\mathbf{b}\to\mathbf{b}')}$ is the non-uniform scale matrix of factor $s(\mathbf{b} \to \mathbf{b}')$. In case $\mathbf{t}'$ lands outside the expected chart, the same search is performed similarly in the $\overrightarrow{\mathbf{sb}'}$ direction (see Figure 5-$\mathbf{t_2}$). The sample relocation procedure is shown in Figure 4. Note that the full non-rigid transformation of $\mathbf{t}$ corresponds to the adaptation of the initial sampling ellipse to the new charts. Therefore, if no severe stretching is present, $S(\mathbf{p})$ is properly covered.

A useful side-effect of the parametric-space computation is that feature estimation can take into account displacement and normal mapping. In the special case of displacement mapping, our method could be easily adopted in order to handle the changes in the geometry that could break the neighbourhood estimation heuristic. Points lying within the initial Euclidean neighbourhood that stretch out of it due to the displacement are automatically handled by measuring the Euclidean distance from $\mathbf{p}$. The problem arises when point with initial location outside the Euclidean neighbourhood of $\mathbf{p}$ fall within r_{max} *after* displacement. By scaling σ_u and σ_v with the maximum expected displacement distortion, which is usually a user defined parameter, the method successfully handles these points as well.

Finally, we need to clarify that if our method focused only on single chart parameterisations such as *Geometry images* [7] we could avoid highly irregular transitions and in this way reduce the complexity of our operations. On the other hand, multi-chart parameterisations offer an added flexibility that can be used to reduce distortion, particularly for shapes with long extremities, high genus, or disconnected components [24] (see Figure 13).

4 Estimating Integral Geometric Descriptors

Central to many geometric descriptor computations is the estimation of surface and volume integrals in the neighbourhood of $\mathbf{p}$. Integral invariant features for instance, are often used in the formulation of local descriptors [11], or provide the means to estimate differential invariants such as the mean curvature H (see [3] and [15]). Another typical example of an integral operator in $S(\mathbf{p})$) is the estimation of Gaussian curvature K, which can be efficiently computed via the local geodesic area at $\mathbf{p}$.

We estimate integrals in a neighbourhood $S(\mathbf{p})$) using Monte Carlo integration in parametric space and in Section 5 we use this approach to compute a variety of integral and differential features interactively for arbitrary feature scales.

4.1 Monte Carlo Integration

In parametric space, the generated data buffers hold not only the vertex information but also all internal polygon samples, generated by the GPU through linear

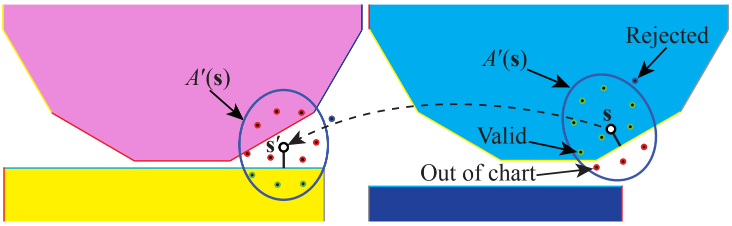

Fig. 6. Monte Carlo Sampling in current and adjacent charts described in Section 4.1.

interpolation during rasterisation. An approach that would use all the per-pixel information inside $S(\mathbf{p})$ would be unnecessarily exhaustive and computationally expensive. Utilizing Monte Carlo integration with a uniform distribution in the parametric domain, any integral $I(\mathbf{p})$ of a function $g(\mathbf{p})$ over $S(\mathbf{p})$ can be approximated by:

$$\langle I \rangle(\mathbf{p}) = \frac{A'(\mathbf{s})}{N} \sum_{i=1}^{N} g(P(\mathbf{t}_i)), \tag{4}$$

where $A'(\mathbf{s})$ is the portion of the elliptical sampling area $A(\mathbf{s})$ centered at parameter pair $\mathbf{s}$ corresponding to the central point $\mathbf{p} = P(\mathbf{s})$ after rejection sampling with the criterion of neighbourhood $S(\mathbf{p})$ (e.g. Euclidean distance of $P(\mathbf{t})$ to $\mathbf{p}$) and N is the number of valid samples. While performing a similar sampling on the geometry itself would require area-weighted probabilities, the parametric-space values can be sampled uniformly, assuming of course a low-distortion parameterisation.

Random samples are generated uniformly using a stratification scheme. Uniform samples in the cells of a planar grid are transformed to disk samples using the concentric mapping of [27]. The disk samples are anisotropically scaled along the u and v axes to form the elliptical region $A(\mathbf{s})$, according to the distortion factors discussed in Section 3.3. The same samples are used at each pixel, randomly rotating them to avoid statistical noise.

The elliptical region $A(\mathbf{s})$ is an approximation that favors fast computations. A more refined but rather more computationally expensive approach would be to pre-compute the maximal distortion for discretised polar coordinates at each pixel and subsequently anisotropically scale each random sample according to the closest distortion term from its conversion to polar coordinates. Nevertheless, as demonstrated in the experiments, the elliptical approximation proved to be both robust and efficient, even for large neighbourhoods.

Given a point $\mathbf{p}$ and its location in parametric space $\mathbf{s}$, initially we perform computations only for the samples that lie on the same chart as $\mathbf{p}$ (Figure 6 - right). At the same time, for all parametric-space samples that fall outside the chart, we mark the ID of the chart they land on. Subsequently, we compute for each marked chart the transformed parametric position $\mathbf{s}'$ of the central parametric pair $\mathbf{s}$ and repeat the sampling procedure on the new location, using the entire sampling pattern (Figure 6 - left). Only samples falling within the

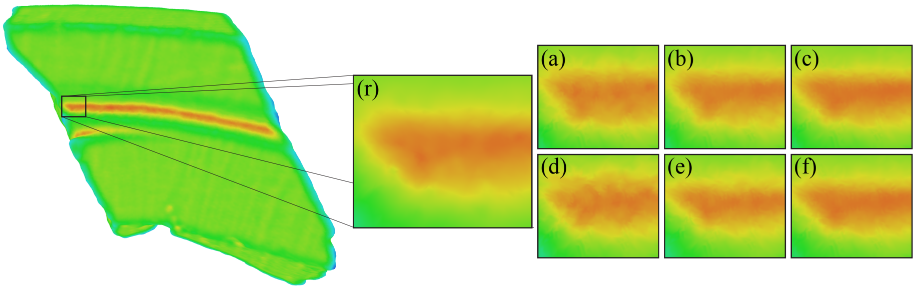

Fig. 7. Comparison of mean curvature for Full and Adaptive Sampling. (r) Reference (a), (b), (c) Full Sampling using 64, 100 and 256 samples in respect. (d), (e), (f) Adaptive Sampling using 32/64, 64/128 and 128/256 samples.

new chart are accounted for and contribute to the final integral. The marking of charts and the central point transformation is done according to the procedure described in Section 3.4.

The sampling scheme described above is generic and can be implemented for an arbitrary number of jumps, excluding each time the already visited charts. In our experiments we noticed that, no more than one jump per sample point was typically required, even for large-scale local feature neighbourhoods. Of course, this also depends on the size of the charts produced by the parameterisation. For example, in Figure 13, where the bunny model is shown in two different parameterisations, for the left one we reported the first missing sample using a support area of 10% the object's diagonal. Conversely, for the one on the right we did not report any missing samples even for neighbourhoods larger than 16% the object's diagonal.

4.2 Adaptive Sampling

Since $g(\mathbf{x})$ is a function of the surface geometry, smooth areas of the objects, i.e. areas with smaller variance of the evaluated function $g(\mathbf{x})$, give satisfactory results even when lowering the sampling rate significantly. Given the fact that in our approach the computation time is proportional to the total number of samples drawn, we speed-up our method by exploiting adaptive sampling.

Typically, adaptive sampling methods continue to draw random samples, until the variance of the computed quantity falls below a certain threshold. In our method however, we perform a simplified, two-step adaptive sampling, instead of waiting for the variance to converge: Initially, we compute the integral with $N/2$ samples and measure the variance. For points $\mathbf{p}$ with variance of $g(\mathbf{x})$ greater than a predetermined threshold, we use an additional set of $N/2$ random samples. Using a fixed, two-stage adaptive sampling creates exactly two different GPU execution loads, generally coherent across the output buffer, thus maximizing shader core utilization and performance.

Our experiments show that as the number of samples increases, the difference of % Absolute Error (% AE) between the full and adaptive sampling declines, while at the same time the performance savings increase. (see Table 1 and Figure 7).

5 Performance and Quality Evaluation

In this section we present a number of local geometric descriptor operators using our method and provide a qualitative comparison against respective reference *object-space* CPU algorithms that operate directly on the polygonal geometry using the *Halfedge* data-structure (HE) [2]. Furthermore, we showcase the use of our method in order to exploit 2D image-space interest point detectors over 3D geometry without the need for a specific data structure or implementation. Initially, we briefly present each of the local descriptors used and subsequently evaluate the results against various factors.

5.1 Local Descriptors

Local Bending Energy (LBE). [11] in order to classify a surface as fractured or intact in their fragment reassembly framework define the LBE term $e_k(\mathbf{p})$ for the k nearest vertices to a surface location $\mathbf{p}$. Similarly, given an Euclidean neighbourhood $\mathbf{q}_i \in S(\mathbf{p}, r) : \|\mathbf{q}_i - \mathbf{p}\| \leq r$ with corresponding normal vectors $\mathbf{n}_i$, LBE $e_r(\mathbf{p})$ can be defined as:

$$e_r(\mathbf{p}) = \frac{1}{N}\sum_{i=1}^{N} \frac{\|\mathbf{n} - \mathbf{n}_i\|^2}{\|\mathbf{p} - \mathbf{q}_i\|^2}, \tag{5}$$

where $\mathbf{n}$ is the normal at the central point $\mathbf{p}$ and N is the number o samples taken in the $S(\mathbf{p}, r)$ neighbourhood.

Sphere Volume. [16] presented a stochastic solid angle computation for the approximation ambient occlusion in the hemisphere above a point $\mathbf{p}$. Inspired by this idea, we extend it to a full sphere and compute a fast approximation of the unoccupied volume of a sphere of radius r centered at $\mathbf{p}$. Assuming a smoothly varying tangential elevation around $\mathbf{p}$, the vector $\mathbf{q}_i - \mathbf{p}$ from the central point to any sample $\mathbf{q}_i$ within the Euclidean neighbourhood $S(\mathbf{p}, r)$ approximates the

Samples	Full		Adaptive	
	Time	% AE	Time	% AE
64	17.57ms	1.172	15.94ms	1.331
100	22.17ms	1.035	19.54ms	1.110
256	50.54ms	1.005	41.44ms	1.007
400	74.21ms	0.789	61.75ms	0.824

Table 1. Computation Time and % Absolute Error for Full and Adaptive Sampling over the same metric. Error in comparison to reference *object-space* implementation.

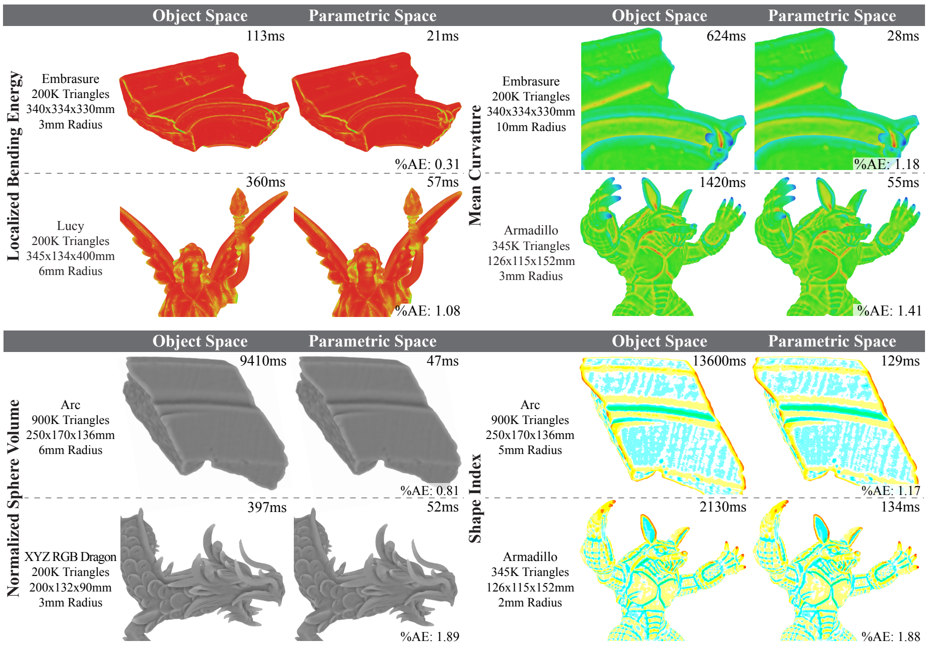

Fig. 8. Comparative visualization, timings and % Absolute Error for the implemented geometric features (Section 5.1).

horizon in this direction with respect to the normal vector $\mathbf{n}$ at $\mathbf{p}$ at a distance scale equal to $\|\mathbf{q}_i - \mathbf{p}\|$. Taking a uniform rotational and radial distribution of samples (direction and scale) $\mathbf{q}_i$ in $S(\mathbf{p}, r)$, we can approximate the open volume $V_o(\mathbf{p})$ above $\mathbf{p}$ by:

$$V_o(\mathbf{p}) = \frac{4\pi r^3}{3N} \sum_{i=1}^{N} \frac{(\mathbf{q}_i - \mathbf{p})\mathbf{n}}{\|\mathbf{q}_i - \mathbf{p}\|}. \tag{6}$$

The *sphere volume integral invariant*, i.e. the part of the sphere volume of radius r ”inside” the surface at $\mathbf{p}$ [21] is the complement of the above integral quantity.

$$V_r(\mathbf{p}) = \frac{4\pi r^3}{3} - V_o(\mathbf{p}). \tag{7}$$

Mean Curvature (MC). [12] derive the relation of MC to the sphere volume integral invariant as:

$$V_r(\mathbf{p}) = \frac{2\pi}{3} r^3 - \frac{\pi H}{4} r^4 + O(r^5), \tag{8}$$

from which we can directly compute MC H at $\mathbf{p}$ for a given radius r.
Shape Index (SI). Introduced by [14], SI is a local descriptor that combines the principal curvatures (PC) in order to classify the locale shape of the surface.

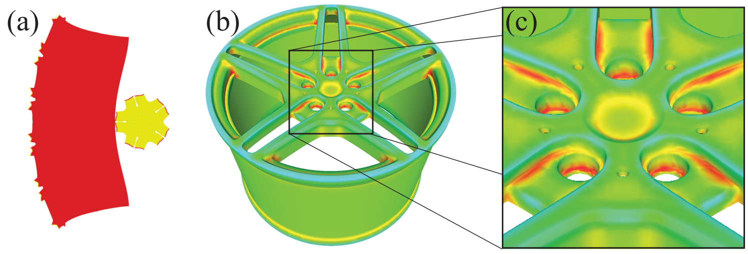

Fig. 9. Genus 20 Rim model. a) Parametric space charts. b) Mean Curvature colourized (generated in 58ms). c) Zoom to detail.

SI is a normalized descriptor and for a given surface point $\mathbf{p}$ is defined as:

$$S(\mathbf{p}) = \frac{2}{\pi} \arctan \frac{K_2(\mathbf{p}) + K_1(\mathbf{p})}{K_2(\mathbf{p}) - K_1(\mathbf{p})}, \tag{9}$$

where $K_1(\mathbf{p})$, $K_2(\mathbf{p})$ are the principal curvatures at $\mathbf{p}$.

In order to calculate the K_1 and K_2, we rely on their relation to mean curvature H and Gaussian curvature (GC) K:

$$K_{1,2} = H \pm \sqrt{H^2 - K}. \tag{10}$$

The computation of H was discussed earlier. For the GC we rely on the work of [1] that relates K with the perimeter and surface area of a geodesic disk on a surface. In particular, we utilize the formula that uses the geodesic area GA of distance r:

$$K = 12 \frac{\pi r^2 - GA^r}{\pi r^4}. \tag{11}$$

The only unknown parameter now is the geodesic area GA^r at a given distance r. In the case of the geometric evaluation, we sum the Voronoi area of each vertex within a neighbourhood of geodesic distance r. For the parametric-space computation of GA^r, we first draw a number of samples N_{tot} in the Euclidean neighbourhood of $\mathbf{p}$ (see Section 4.1). Then, for each sample $\mathbf{q}_i$ at parametric location $\mathbf{t}_i$, the geodesic distance to $\mathbf{p}$ is approximated by a sum of chords at $P(\mathbf{s}_j)$, i.e. at the intermediate parametric space coordinates $\mathbf{s}_j = \mathbf{t}_i + j(\mathbf{s} - \mathbf{t}_i)/N_{steps}$, where $\mathbf{s}$ are the uv coordinates of $\mathbf{p}$ and N_{steps} is the number of chords. Depending on the local distortion of the parameterisation, $P(\mathbf{s}_j)$ may not reside exactly on the same plane. According to the computed geodesic distance between $\mathbf{q}_i$ and $\mathbf{p}$, a final set of N_g samples is retained, $N_g \leq N_{tot}$, and GA^r is estimated by:

$$GA^r = \frac{N_g}{N_{tot}} EA^r, \tag{12}$$

where EA^r is the Euclidean area.

Euclidean area can be approximated in the following way. Let P_{tot} be the total number of pixels in the elliptical region. Given the ratio of the samples

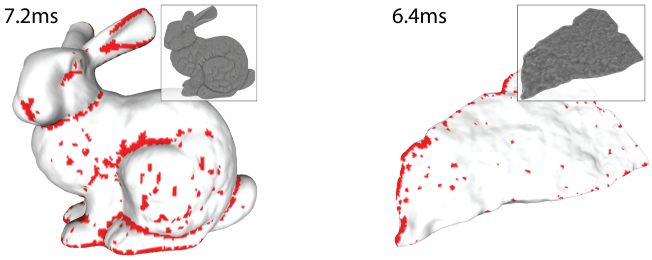

Fig. 10. Harris corner points computed over the Normalized Sphere Volume descriptor and painted (red) on the mesh vertices.

that satisfy the Euclidean criterion to the total samples $N_{A(s)}/N_{tot}$, and $A_{\mathbf{q}_m}$ the mean area represented by each sample in $A(s)$, we approximate EA^r as:

$$EA^r = P_{tot}\frac{N_{A(s)}}{N_{tot}}A_{\mathbf{q}_m}, \tag{13}$$

$A_{\mathbf{q}_m}$ is given by the formula:

$$A_{\mathbf{q}_m} = \frac{1}{N_{A(s)}}\sum_{i=1}^{N_{A(s)}} A_{\mathbf{q}_i}, \tag{14}$$

where $A_{\mathbf{q}_i}$ is the product of the distortion factors $l_u(u,v)$, $l_v(u,v)$.

5.2 Image Descriptors

Harris Corner Detection. Harris and Stephens [8] describe a mathematical operator for the computation of corner points of interest (features) on images, based on the change of intensity. These feature-points are invariant under rotational and intensity changes and can be used for matching. The same computation has been used on 3D meshes to generate feature points for object registration and retrieval [28]. The mathematical formulation of Harris corner response is:

$$R = \det(M) - k \cdot \operatorname{trace}(M)^2, \tag{15}$$

where k is constant ($k \in [0.04, 0.06]$) and M is given by:

$$M = \sum_{x,y} w(x,y)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}. \tag{16}$$

I_x, I_y are the image derivatives, and $w(x,y)$ a Gaussian window function. Using the parametric-space indexing scheme (see Section 3.4) we compute the image-space derivatives and the Gaussian window function. Of course, using our indexing scheme, any other image space descriptor can also be applied over the 3D data.

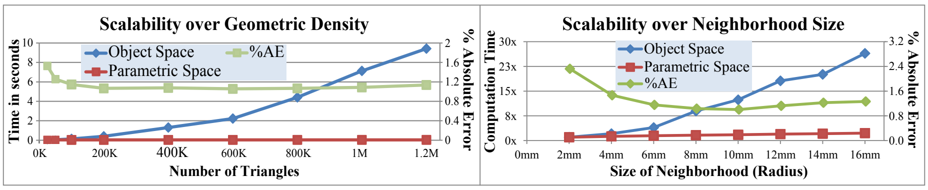

Fig. 11. (Left Graph) Computations using the same metric and neighbourhood size (Left axis). (Right Graph) Computations using the same metric and geometric complexity (Left axis). Green line shows the %AE of the parametric method (Right axis).

5.3 Results and Discussion

We have tested our method using a large variety of objects, ranging from simple geometric shapes to complex and detailed 3D scanned models. Indicative results can be seen in Figure 8 and Figure 9, where we report an average of 49× acceleration and 1.245% Absolute Error (AE) relative to the reference CPU *object-space* method described below. Please note that in such comparisons, reporting maximum error is not indicative of the method's performance, since a slight mismatch in the representation at a single point due to parameterisation can cause an isolated but inconsequential measurement difference. Timings of our method do not include the parameterisation and the charts boundary edge detection. Similarly, timings of the reference method do not include the *Half-Edge (HE)* data structure generation. It is important to mention here that while geometric algorithms for computing features operate on discretised values at a vertex or triangle level, the parametric space calculations can exploit interpolated values at arbitrary surface locations. Therefore, the measurement deviations that are reported here as errors, mostly stem from the different approximation and sampling of the underlying surface (see Figure 2). Finally in Figure 10 we show *Harris corner points* detection over 3D data. Timings for the GPU parametric implementation are shown for an NVIDIA GTX 670 GPU. We use 1024x1024 floating point texture buffers, while metrics are computed over a 512x512 buffer with 256 samples per pixel unless stated otherwise. The reference geometric algorithm results are shown for a Corei7-3820 system (4 cores @ 3.60GHz, 8 threads). Our implementation uses the *OpenMP API* and takes advantage of the current generation multi-core CPU's.

The efficiency of our method is attributed to the shift of the computations from a topology-detail-dependent representation to two dimensions with application-controlled (sampling) quality settings, which enables very good scaling for multi-core and many-core architectures. The proposed implementation is tailored for (but not limited to) commodity GPUs.

Geometric Detail. In Figure 11(Left) we present comparative results computed over a fixed neighbourhood size (4% of object's diagonal) for a single model (Embrasure) decimated at different geometric detail levels. For small resolutions (25K, 50K triangles) we observe similar computation times between geometric and parametric space approaches, while the %AE is high in comparison to higher

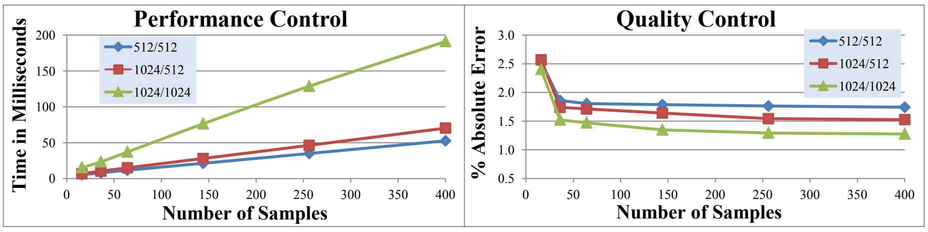

Fig. 12. (Left) Average performance over several models using different number of samples, buffers size and size of texture over which computations are performed. (Right) Average quality over several models using different number of samples, buffers size and size of texture over which computations are performed. Legends show Buffers Texture/Computations Texture Size in square format.

detail versions of the mesh. This is expected as the parametric method uses the position samples as interpolated by the GPU resulting in smoother and therefore slightly different results than the CPU method (see Figure 2). For larger resolutions, we report an acceleration of 3× for the 100K model to 137× for the 1000K model, with a steady AE. Finally, for the original scanned object resolution of 1200K, we report a 181× faster computation with a slight increase in the %AE. This is also expected and attributed to the relative small buffer size for the dense geometric detail. However, this can be trivially addressed by increasing the geometry buffer resolution.

Neighborhood Size. In the measurements of Figure 11(Right) we shift the focus from the geometric detail to neighbourhood size. Results are for the same model (Embrasure) and metric (mean curvature) at 600K resolution. We notice that for small neighbourhoods the %AE is higher. This deviation between the parametric and geometric domain results are due to the inadequate discrete representation of the area of support in the geometric solution. While in the parametric domain due to the interpolation of values we mentioned earlier, an increasing neighbourhood size is directly reflected in a wider selection of samples, the geometric neighbourhood expands in discrete steps, which is actually a deficiency (see Figure 2). For very large neighbourhoods we notice also an increase to the %AE, this time, due to the one jump per sample approach of our implementation (see end of Section 4.1), which starts missing samples. Performance-wise, the parametric space method scales very well and is not significantly affected by the 8× growth of neighbourhood size. More specifically, the computation time for the parametric domain feature estimator grows by 2.25 times in contrast to the 26.45× factor reported by the geometric approach.

Performance and quality control. The number of samples per pixel, buffer size and size of the texture over which computations are performed, are parameters that control the quality/performance of our method. As we can see in Figure 12(Left), increasing the number of samples reduces the %AE and has linear impact on the computation time, regardless of the buffer resolution. The same effect have the buffer size and the size of the texture over which computations are performed (Figure 12(Right)). Using these parameters, performance and quality can be controlled depending on the application requirements.

Memory usage - Texture size and precision. Four RGBA textures are used (see Section 3.3). All the presented results so far were performed using half-float precision textures. In order to evaluate the performance/quality impact of full-float-precision textures (FF), which double the memory requirements, we performed experiments using both resolutions (Table 2). FF buffers present an 8% and 11% performance degradation on 512x512 and 1024x1024 buffers respectively, while the corresponding improvement in AE is 4% and 6%. We can conclude that the minor quality improvement does not justify the performance drop and the doubled memory requirements.

UV Parameterization. In order to evaluate how our method is affected by the underlying parameterisation in terms of speed and quality, we performed several tests. When operating on maps coming from global surface parameterisation (single chart) techniques, we notice faster times, and increased error rates (see Figure 13) compared to multi-chart parameterisations opting for minimal stretching. Single charts, minimize branching operations but at the same time result in greater distortion and less uniform sampling leading thus in loss of representation and measurement accuracy.

6 Limitations

Due to the fact that parameterisation of the objects surface is required, the method is limited to mesh geometries and it cannot be directly applied on point-clouds. Still, most of the local descriptors rely on the notion of a connected neighbourhood. While simple distance queries (without connectivity) can be used in the case of point-clouds, the resulting computations will have inaccuracies, especially when using large areas of support.

7 Conclusion

We presented a novel generic parametric-space approach for the computation of geometric descriptors in multiple scales, that can also be used to trivially apply computer vision algorithms on 3D data. Our method, decouples the computational complexity from the underlying geometry and by taking advantage of

	Buffer Size			
	512x512		1024x1024	
Precision	Time	% AE	Time	% AE
Full-Float	26.5ms	1.662	74.9ms	1.307
Half-Float	24.4ms	1.725	66.8ms	1.384

Table 2. Average Computation time and % Absolute Error over a set of models for the same metric over different resolutions and buffer precision. Error in comparison to the reference CPU implementation.

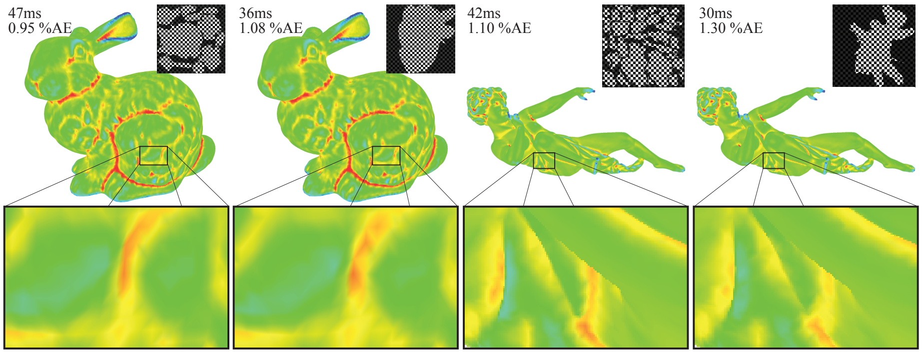

Fig. 13. Mean Curvature (colourized) computed using different parameterisations. Multiple charts result in increased computation times, but smaller error, due to the smaller distortion of the generated charts.

modern multi-core architectures (GPUs), achieves real-time computations even for large areas of support, rendering the method suitable for deformable and animated objects. Finally, despite the focus of our method on efficiency, computations are accurate and equivalent to those of the traditional object-space approaches as shown by our experiments.

Acknowledgments

This work was supported by EC FP7 STREP Project PRESIOUS, grant no. 600533. Armadillo, Lucy, Bunny and XYZ RGB Dragon models are from Stanford 3D Scanning Repository. Angel model is from the Large Geometric Models Archive of Georgia Institute of Technology. Rim model is from TurboSquid. All other models used are from the PRESIOUS project data collection.

References

1. Bertrand, J., Diquet, C., Puiseux, V.: Démonstration d'un théorème de Gauss. Journal de Mathématiques 13, 80–90 (1848)
2. Campagna, S., Kobbelt, L., Seidel, H.P.: Directed edges—a scalable representation for triangle meshes. J. Graph. Tools 3(4), 1–11 (Dec 1998)
3. Connolly, M.L.: Measurement of protein surface shape by solid angles. J. Mol. Graph. 4(1), 3–6 (Mar 1986)
4. De Floriani, L., Hui, A.: Data structures for simplicial complexes: An analysis and a comparison. In: Proc. of the Third Eurographics Symp. on Geometry Processing. SGP '05, Eurographics Association (2005)
5. Floater, M., Hormann, K.: Surface parameterization: a tutorial and survey. In: Advances in Multiresolution for Geometric Modelling, pp. 157–186. Mathematics and Visualization, Springer (2005)

6. Griffin, W., Wang, Y., Berrios, D., Olano, M.: GPU curvature estimation on deformable meshes. In: Symp. on Interactive 3D Graphics and Games. pp. 159–166. I3D '11, ACM (2011)
7. Gu, X., Gortler, S.J., Hoppe, H.: Geometry images. In: Proc. of the 29th Annual Conference on Computer Graphics and Interactive Techniques. pp. 355–361. SIGGRAPH '02, ACM (2002)
8. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proc. of the 4th Alvey Vision Conference. pp. 147–151 (1988)
9. Hormann, K., Polthier, K., Sheffer, A.: Mesh parameterization: Theory and practice. In: ACM SIGGRAPH ASIA 2008 Courses. pp. 12:1–12:87. SIGGRAPH Asia '08, ACM (2008)
10. Hua, J., Lai, Z., Dong, M., Gu, X., Qin, H.: Geodesic distance-weighted shape vector image diffusion. IEEE Trans. Vis. Comput. Graph. 14(6), 1643–1650 (Nov 2008)
11. Huang, Q.X., Flöry, S., Gelfand, N., Hofer, M., Pottmann, H.: Reassembling fractured objects by geometric matching. ACM Trans. Graph. 25(3), 569–578 (Jul 2006)
12. Hulin, D., Troyanov, M.: Mean curvature and asymptotic volume of small balls. The American Mathematical Monthly 110(10), 947–950 (2003)
13. Kim, Y., Yu, J., Yu, X., Lee, S.: Line-art illustration of dynamic and specular surfaces. In: ACM SIGGRAPH Asia 2008 Papers. pp. 156:1–156:10. SIGGRAPH Asia '08, ACM (2008)
14. Koenderink, J.J., van Doorn, A.J.: Surface shape and curvature scales. Image Vision Comput. 10(8), 557–565 (Oct 1992)
15. Manay, S., Hong, B.W., Yezzi, A., Soatto, S.: Integral invariant signatures. In: Computer Vision - ECCV 2004, Lecture Notes in Computer Science, vol. 3024, pp. 87–99. Springer (2004)
16. McGuire, M., Osman, B., Bukowski, M., Hennessy, P.: The alchemy screen-space ambient obscurance algorithm. In: Proc. of the ACM SIGGRAPH Symp. on High Performance Graphics. pp. 25–32. HPG '11, ACM (2011)
17. Mellado, N., Barla, P., Guennebaud, G., Reuter, P., Duquesne, G.: Screen-space curvature for production-quality rendering and compositing. In: ACM SIGGRAPH 2013 Talks. pp. 42:1–42:1. SIGGRAPH '13, ACM (2013)
18. Meyer, M., Desbrun, M., Schrder, P., Barr, A.: Discrete differential-geometry operators for triangulated 2-manifolds. In: Visualization and Mathematics III, pp. 35–57. Mathematics and Visualization, Springer (2003)
19. Museth, K.: Vdb: High-resolution sparse volumes with dynamic topology. ACM Trans. Graph. 32(3), 27:1–27:22 (Jul 2013)
20. Novatnack, J., Nishino, K.: Scale-dependent 3D geometric features. In: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on. pp. 1–8. IEEE (Oct 2007)
21. Pottmann, H., Wallner, J., Huang, Q.X., Yang, Y.L.: Integral invariants for robust geometry processing. Comput. Aided Geom. Des. 26(1), 37–60 (Jan 2009)
22. Pottmann, H., Wallner, J., Yang, Y.L., Lai, Y.K., Hu, S.M.: Principal curvatures from the integral invariant viewpoint. Computer Aided Geometric Design 24(8), 428–442 (2007)
23. Rusinkiewicz, S.: Estimating curvatures and their derivatives on triangle meshes. In: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2Nd International Symposium. pp. 486–493. 3DPVT '04, IEEE Computer Society (2004)

24. Sander, P.V., Wood, Z.J., Gortler, S.J., Snyder, J., Hoppe, H.: Multi-chart geometry images. In: Proc. of the 2003 Eurographics/ACM SIGGRAPH Symp. on Geometry Processing. pp. 146–155. SGP ’03, Eurographics Association (2003)
25. Sander, P.V., Snyder, J., Gortler, S.J., Hoppe, H.: Texture mapping progressive meshes. In: Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques. pp. 409–416. SIGGRAPH ’01, ACM (2001)
26. Sheffer, A., Praun, E., Rose, K.: Mesh parameterization methods and their applications. Found. Trends. Comput. Graph. Vis. 2(2), 105–171 (Jan 2006)
27. Shirley, P., Chiu, K.: A low distortion map between disk and square. J. Graph. Tools 2(3), 45–52 (Dec 1997)
28. Sipiran, I., Bustos, B.: Harris 3D: A robust extension of the harris operator for interest point detection on 3D meshes. Vis. Comput. 27(11), 963–976 (Nov 2011)
29. Taubin, G.: Estimating the tensor of curvature of a surface from a polyhedral approximation. In: Proceedings of the Fifth International Conference on Computer Vision. pp. 902–. ICCV ’95, IEEE Computer Society (1995)
30. Yang, Y.L., Lai, Y.K., Hu, S.M., Pottmann, H.: Robust principal curvatures on multiple scales. In: Symp. on Geometry Processing. pp. 223–226 (2006)
31. Yoshizawa, S., Belyaev, A., Seidel, H.P.: A fast and simple stretch-minimizing mesh parameterization. In: Proc. of the Shape Modeling International 2004. pp. 200–208. SMI ’04, IEEE Computer Society (2004)
32. Zhou, K., Synder, J., Guo, B., Shum, H.Y.: Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In: Proc. of the 2004 Eurographics/ACM SIGGRAPH Symp. on Geometry Processing. pp. 45–54. SGP ’04, ACM (2004)