Hitoshi Yamauchi · Stefan Gumhold · Rhaleb Zayer · Hans-Peter Seidel

Mesh Segmentation Driven by Gaussian Curvature

Abstract Mesh parameterization is a fundamental problem in computer graphics as it allows for texture mapping and facilitates a lot of mesh processing tasks. Although there exists a variety of good parameterization methods for meshes that are topologically equivalent to a disc, the segmentation into nicely parameterizable charts of higher genus meshes has been studied less. In this paper we propose a new segmentation method for the generation of charts that can be flattened efficiently. The integrated Gaussian curvature is used to measure the developability of a chart and a robust and simple scheme is proposed to integrate the Gaussian curvature. The segmentation approach evenly distributes Gaussian curvature over the charts and automatically ensures disc-like topology of each chart. For numerical stability, we use area on the Gauss map to represent Gaussian curvature. Resulting parameterization shows that charts generated in this way have less distortion compared to charts generated by other methods.

1 Introduction

The goal of surface mesh segmentation is to construct a covering of a given mesh, which is composed of surface patches i.e. *charts*. There has been a considerable body of research devoted to this technique in the last few years. Its significance to computer graphics arises in several contexts such as surface parameterization, compression, surface editing, and morphing. This diversity of applications fosters the need for segmentation techniques that best suit specific purposes. The quality of segmentation usually depends on its field of application. As general requirement, it is desirable to maintain clean boundaries between charts as unresolved boundary artifacts reduce the usability of the segmentation to a great

extent. Other quality measures regarding the flatness of the charts or how good they capture the semantics of the surface mesh could be taken into consideration as well.

Our key contribution is a novel efficient segmentation approach generating charts with balanced curvature distribution. The mathematical tool at the heart of the approach is the Gauss map of the surface to the unit sphere, regardless of its genus. This allows us to guide our new chart flooding (t-flooding) in an intelligent manner that automatically cuts along sharp creases and corner points. An additional property of our method is the total control over the timing when a chart can start growing and when it should stop. Furthermore our algorithm considers the shape of the chart boundary during the growing process and tends to avoid jagged boundaries. While our method is especially suited to surface parameterization, this does not prevent using it for applications such as the fitting of subdivision surfaces to the original mesh and compression.

The rest of the paper is organized as follows. Section 2 gives a brief review of the literature relevant to our method. In Section 3, we detail our Gaussian curvature distribution method for surface segmentation. The results are shown in Section 4, and Section 5 concludes the paper.

2 Previous and Related Work

The problem of segmenting a polygonal mesh into charts has been studied in computer graphics with different goals. This problem is fundamentally an ill-posed problem. An optimal solution is often application specific, and quite heavily depends on what is an optimal or what is a *meaningful* segmentation. Without loss of generality, many existing schemes might be classified into one of the following categories, shape analysis [1,5,10,18,20,23,28], shape simplification [2, 9, 14], shape modeling and retrieval [8, 15], and texture atlas generation [16, 17, 26]. We also refer the reader to [27] for a recent survey of this area.

H. Yamauchi · S. Gumhold · R. Zayer · H.-P. Seidel
MPI Informatik, Saarbrücken, Germany
Tel.: +49-681-9325-650   Fax: +49-681-9325-499
E-mail: {hitoshi|sgumhold|zayer|hpseidel}@mpi-inf.mpg.de

Our goal in mesh segmentation is the generation of low distortion charts for surface parameterization. From this point of view, the predominant segmentation criterion is the ability to flatten the generated charts. Applications of our segmentation approach are texture atlas generation, sheet- or plate-metal based industrial design, and so forth. In accordance with this criterion, the next three categories are relevant to our approach: 1. Approximating surfaces with developable parts, 2. Flatness or normal based segmentation, 3. Low distortion parameterization for texture atlas generation.

Approximating surfaces with developable parts: The research dedicated to this technique aims at generating developable surface patches which approximate an original surface with Bézier and B-spline surfaces [6, 24], surfaces of revolution [11], and general triangle mesh [22].

In order to maintain the developability, these methods tend to generate charts composed of polygon- or triangle-strips. In general, this yields charts with rather long boundaries in comparison to their area and may cause considerable artifacts during texture atlas generation.

Flatness or normal based segmentation: Since a plane is the simplest developable surface, it seems intuitive that segmentation based on plane fitting or normal clustering automatically considers the ability to flatten the resulting charts. Cohen-Steiner et al. [2] use normal clustering to subdivide a complex polygonal mesh into regions that can be approximated by planar polygons. Inoue et al. [12] develop a method based on normal scoring where the scores combine the variance of normals and their largest derivatives.

While these methods yield interesting results, they are limited to a smaller class of developable surfaces. For example, a cylinder and a cone are also developable, however, the plane fitting method would not recognize these surfaces as such.

Low distortion parameterization for texture atlas generation: Maillot et al. [19] propose an interactive tool for user guided segmentation based on a distortion energy. Sanders et al. [25] use a hierarchical face clustering method, which was simultaneously proposed by [9], for mesh segmentation by merging triangles driven by a planarity criterion. In order to account for sharp features Lévy et al. [17] first detect sharp features and use these to place chart seeds maximally apart for a chart growing process. Shlafman et al. [28] combine the region growing strategy with a Lloyd-Max iteration that allows for the replacement of the seeds and accounts for geodesic distances as well as dihedral angles. The stability of this method was improved by Sanders et al. [26] by introducing a representative normal which is the average of all triangle normals in a chart. The method of Sorkine et al. [29] grows charts and simultaneously creates a parameterization. The mesh is cut whenever distortion is above a certain threshold. Zhou et al. [31] address the problem using a combination of stretch minimization and multidimensional scaling.

In most of the above-mentioned methods, there is a tight connection between two problems, segmentation and parameterization. It is hence hard to see exactly whether the distortion stems from the segmentation technique or parameterization method. A naïve solution would be to generate all possible segmentations, and parameterize them, then select the one with minimal distortion. While such a solution is impractical, an estimation of distortion without explicit parameterization is the key point to a good segmentation.

Our goal is to segment mesh models based on a measure that is independent of the subsequent parameterization method.

Independently from our work, Julius et al. [13] proposed recently an approach similar to ours. Instead of using the Gaussian curvature to steer the segmentation process, they rely on a developability measure which captures how well a chart approximates a cone or a cylinder. They straighten out the chart boundaries in a postprocessing step and construct a sewing pattern, that allows for making stuffed toys.

3 Segmentation based on Developability

Texture mapping is one of the oldest techniques in computer graphics, still it is one of the most powerful tools today to represent complex objects in low computational cost. Texture mapping usually proceeds on charts with disk-like topology. In practice however, meshes usually have arbitrary topology. A straightforward solution consists of generating a texture atlas which consists of decomposing high-genus meshes into several disk-like charts. In general, two problems arise after segmentation,

1. misalignment of the chart boundary,
2. distortion in a successive parameterization.

In order to address these problems, Lévy et al. [17] observe that the perceived visual quality is better when the chart boundary is aligned with surface features. While the presented results seem to be convincing it is hard however to formalize the relationship between segmentation and human perceptual factor of textured models. Sander et al. [26] propose a method which tends to produce charts with rounded shapes based on similarities of the normals. Benko et al. [1] propose a normal classification based segmentation method for point clouds. However, when applied to meshes, this approach often generates problematic extra charts due to its insensitivity to mesh connectivity. Steiner et al. [2] classify triangles using both normals and mesh connectivity. While this method is primarily proposed for shape simplification, it can be used effectively for producing near planar charts.

A second category of mesh segmentation techniques targets the direct control of the segmentation process using predefined parameterization distortion measures. This requirement induces a partial or full mesh parameterization during the segmentation process. In fact this would restricts the method to the use of an incrementally computable parameterization e.g. [19,29], where the number of charts and their shape are completely driven by the ongoing parameterization, or to an iterative process where several chart parameterizations have to be computed in every step e.g. [31] which may increase the segmentation cost. Furthermore, the lack of a consensus on what distortion measure would be the most appropriate makes the coupling of parameterization and segmentation a difficult choice.

In order to address the chart boundary artifacts and parameterization distortion in a cost effective manner, we introduce a new method based on Gaussian curvature.

3.1 Charts Gauss area distribution

Our approach is motivated by two goals. Firstly, we aim at generating suitable charts for mesh parameterization without any use of parameterization during the segmentation process. Secondly, we aim at segmentation of developable charts as opposed to planar charts, which have been treated in the literature before.

In order to decouple the distortion measure from the parameterization technique, we base our segmentation strategy on a more objective measure which also captures the developability of the charts. For this purpose, we introduce the well known Gaussian curvature $K = \kappa_{\min}\kappa_{\max}$. If $K \equiv 0$ everywhere, the chart is developable and can be parameterized without distortion, independently of the chosen distortion measure. Therefore, our segmentation method aims at an even distribution of the error ϵ in developability, which we define as

$$\epsilon(c_i) = \int_{c_i} |K| \cdot dA \tag{1}$$

where c_i is the i-th chart. Let us consider a small surface patch of area A on a surface. The Gauss map maps the surface points to their normals, which live on the unit sphere. Let A_G be the area of the patch normals on the unit sphere. In the limit for $A \to 0$ the ratio A_G/A converges to the Gaussian curvature K. This can be abbreviated in differential form as $K = dA_G/dA$. Combining this with Equation (1) yields

$$\epsilon(c_i) = \int_{c_i} |dA_G|. \tag{2}$$

Thus the chart error is the integral of the absolute value of the area on the Gauss map (= Gauss area) and the segmentation problem reduces to the problem of distributing the Gauss area evenly among the charts. This

can be achieved by minimizing the standard deviation in the chart error, i.e.

$$\min \sqrt{\sum_i (\epsilon_{ave} - \epsilon(c_i))^2} \tag{3}$$

where the ϵ_{ave} is the average charts error ($\epsilon_{ave} = \frac{1}{n}\sum_i \epsilon(c_i)$ for n charts). In this way, we have defined the objective function for our developability based segmentation. This setup comes in handy as it allows for a simple and intuitive discretization.

The main reason for using Gauss area instead of Gaussian curvature K is numerical stability. This instability of K was also noted in [13]. A robust estimation of K is generally a hard problem while the calculation of the Gauss area offers a simpler alternative and yields a more appealing numerical problem. In fact, using the method of Welch et al. [30], the range of $|K|$, e.g., of the Happy Buddha model is $[2.0 \times 10^{-2}, 2.6 \times 10^6]$, and its standard deviation is 7.3×10^4. This wide range of values is numerically difficult to handle, even with the use of filtering techniques. The Gauss area on the other hand is always within $[0, 4\pi]$.

In the rest of this section we first elaborate on the computation of the Gauss area which is the area per mesh element computed on the Gauss map. Subsection 3.3 details the chart growing process, where the objective function minimization and chart boundary optimization are both taken into consideration. Then we discuss the positioning of seeds for successive iterations and convergence of them. Subsection 3.5 details an important contribution, namely the offset strategy which allows us to perfectly balance the charts in terms of covered Gauss area. We end the section with a description of our approach for controlling the topological integrity of the surface charts.

3.2 Gauss area computation

For the computation of the Gauss area integrals in Equation (2) we regard the polygonal mesh as an approximation to a smooth surface, which may have sharp creases and corners. We use a simple thresholding strategy to extract sharp edges: an edge is tagged as sharp if its dihedral angle is larger than a user defined threshold T_a. One advantage of using the Gauss area integrals is that we can encapsulate smooth surfaces by simply estimating the surface normals at the vertices. To account for sharp features the surrounding of each vertex is split by the sharp edges into smooth regions. For each region we estimate a different vertex normal with weights according to [21]. We note that a more involved normal partial evaluation method was proposed in [3], however we restrict ourselves to our thresholding method for simplicity.

The vertex normals computed this way lead to the following computation of the Gauss area integrals for
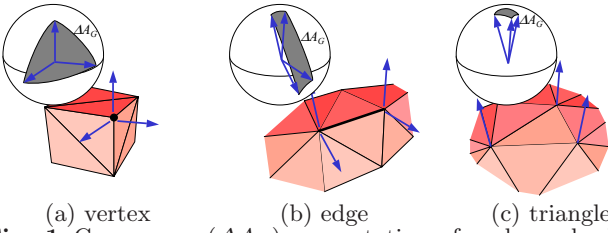
(a) vertex   (b) edge   (c) triangle

Fig. 1 Gauss area (ΔA_G) computation of each mesh element. (a) vertex ΔA_G: The vertex normals (blue arrows) mapped on the unit sphere yield $\Delta A_G > 0$. (b) edge ΔA_G: Gauss area for crease edge case. (c) triangle ΔA_G.

the different mesh elements. For each mesh element, we denote the associated absolute Gauss area by ΔA_G (c.f. Equation (2)). Figure 1 illustrates the computation of the Gauss area for vertices, edges, and faces. For each mesh element the incident vertex normals are projected onto the unit sphere and define a spherical polygon. The area of this polygon is equal to ΔA_G and can be computed using standard formulas from spherical geometry.

We note for non corner points and non crease edges we get only one normal per vertex as we do not apply the splitting procedure of normals. Therefore these vertices and edges have null Gauss area.

During the chart construction the Gauss area assigned to an edge or vertex is added to a chart only when the mesh element is entirely inside the chart, but not when it is still part of the chart boundary. In this way a cut through a bent sharp crease or a corner point does not add up to the Gauss area of all the edges and vertices on the cut. Our chart growing algorithm exploits this fact to direct the cut automatically through sharp features without any special optimization as proposed for example in [17].

3.3 Chart flooding (t-flooding)

The goal of the flooding algorithm is twofold. It aims on the one hand to adapt the chart boundaries to features and on the other hand to distribute the Gauss area evenly over the charts. A minimum spanning tree (MST) approach as described in [26] is not capable of balancing the Gauss area of the charts since the MST is designed for finding a shortest path from a certain node, not for balancing of some integral value. Steering the region growing process only by the normal deviation or the average squared distances to a reference plane as done in [2] leads especially to very jagged boundaries for highly tessellated models.

We therefore designed a new method that floods the mesh simultaneously from the different seeds, where the relative growing rate of the charts in terms of Gauss area is equalized over the charts. As the Gauss area is not evenly distributed over the surface, we introduce an artificial time coordinate t to parameterize the flooding process (*t-flooding*). For reference we defined an arbi-
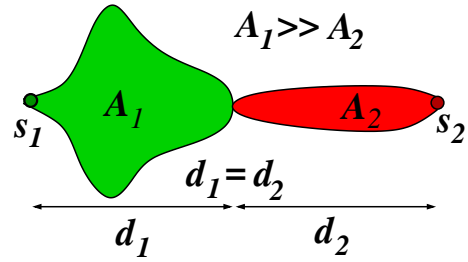

Fig. 2 An example of area imbalance. When charts are flooded from the two seeds (s_1, s_2) with constant speed, the chart on the left covers much more area ($A_1 >> A_2$) although they meet at the mid point ($d_1 = d_2$). The perimeter lengths are also imbalanced in this case.

trary total flooding time t_{total}. Let $A_{G,total}$ be the total integrated Gauss area of the mesh and k the number of charts. The goal is to grow each chart such that the inflow of Gauss area is constant during the growing process, i.e.

$$\alpha := \frac{dA_G}{dt} = const = \frac{A_{G,total}}{k \cdot t_{total}}. \tag{4}$$

We call α the Gauss area inflow or shortly the inflow.

In 1D a constant inflow can be easily achieved by a weighted distance based growing algorithm, which can be efficiently implemented by a fast marching method. But on a 2-manifold this is slightly more complicated. The typically used approach is to grow the charts with constant speed of its boundary, where speed is measured. This results in a generalized Voronoi diagram and does not even balance the charts. A simplified example is visualized in Figure 2, where we only considered the unweighted case to evenly distribute surface area among two charts. Although the charts meet at equal distance from the two seeds, the area of the left chart is much larger than the area of the right chart. This is due to the changing perimeter p of the chart. For a flooding speed of v the area inflow computes to $v \cdot p$ and therefore varies with the perimeter. It is obvious that the same thing happens in a weighted approach as for example with the Gauss area inflow, which does not allow a balanced generation of charts. Interestingly, all approaches like [2] or [26] do neglect this fact and just accept the loss of balance.

Our first approach to make the inflow constant over time was to set growing speed v_i of chart i proportional to the inverse of its perimeter p_i on the Gauss map, i.e. $v_i = \alpha/p_i$. To make the whole idea independent of the perimeter, which is hard to interpret for the Gauss area, we defined for each chart boundary edge E_j the local inflow α_j. The quantities defined in the following are illustrated in Figure 3. The chart boundary will be extended by the exterior triangle $T_{ex,j}$ at time $t_{ex,j}$. Suppose the interior triangle $T_{in,j}$ was incorporated to the chart at time $t_{in,j}$. Then the time delay Δt_j at the chart boundary edge E_j is defined to $t_{ex,j} - t_{in,j}$. When the exterior triangle $T_{ex,j}$ is incorporated to the chart, it adds its own
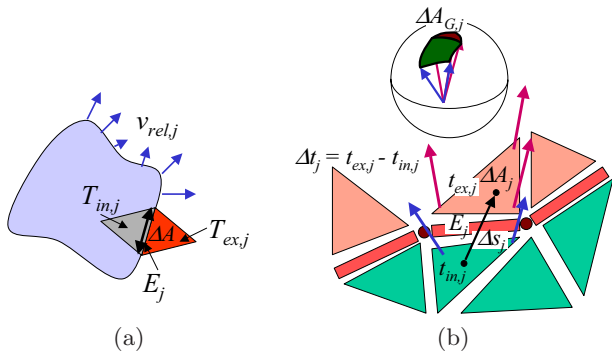
Fig. 3 Growing a chart. (a) A chart boundary edge E_j and its incident triangles $T_{in,j}, T_{ex,j}$, and different relative speeds $v_{rel,j}$ along the boundary. (b) The mesh elements that contribute to $\Delta A_{G,j}$ include the triangle $T_{ex,j}$, the edge E_j and its incident vertices. The top sphere shows Gauss mapped normals of adding triangle (bottom).

Gauss area and the one of the newly incorporated edge E_j. In case $T_{ex,j}$ completes the fan of one of its incident vertices with all triangles from the same chart i, also the Gauss area of this vertex is incorporated into the chart at time $t_{ex,j}$. Summing all the Gauss area contributions together yields the local delta $\Delta A_{G,j}$ in Gauss area, from which the local inflow is computed to $\Delta A_{G,j}/\Delta t_j$.

By summing over all local inflows of a chart, we obtain a relation between the chart inflow and the local time delays Δt_j

$$\alpha = \sum_{j \in B_i} \alpha_j = \sum_j \frac{\Delta A_{G,j}}{\Delta t_j} \tag{5}$$

where B_i is the set of boundary edges of i-th chart c_i, i.e. $B_i = \{E_j | E_j \in \partial c_i\}$. As α is constant and the $\Delta A_{G,j}$ are computed from the mesh, only the local delays Δt_j or equivalently the triangle incorporation times $t_{ex,j}$ are unknown. We sort the exterior triangles incident to the chart boundaries into a heap, which is sorted by $t_{ex,j}$. To achieve a flooding with nice evolution of the chart boundary it is essential to compute the incorporation time of an exterior triangle at the moment when it becomes incident to the chart boundary for the first time. We note that this is quite different from a greedy approach e.g. where the heap is sorted by normal deviation. In our method, the time parameter is related to the distance from the seed and the chart perimeter while in a greedy approach there is usually no consideration for both of them, therefore the boundary generated by such methods can be more jaggy than ours.

Equation 5 only fixes one of the Δt_j of each boundary and gives us the freedom to choose the others to best serve our needs. We interpret this degree of freedom with the ability to define arbitrary relative growing speeds $v_{rel,j}$ along the chart boundary, whose scaling is defined by Equation (5). We used the relative growing speeds to adapt the charts to sharp features. For this we defined the relative speed as a function of the absolute value

of the local Gaussian curvature $|K_j|$, which we robustly estimated by $|K_j| = \Delta A_{G,j}/\Delta A_j$, where ΔA_j is just the triangle area on the 3D mesh of the exterior triangle $T_{ex,j}$. From this the local relative speed was defined as

$$v_{rel,j} = \frac{1}{f(|K_j|)}. \tag{6}$$

The function f can be chosen arbitrarily. A good choice turned out to be

$$f(x) = (x + \varepsilon)^p, \tag{7}$$

where p was chosen in $[1, 6]$. The epsilon ε is necessary as the speed would otherwise become infinite in developable regions, which would bring us back to a greedy growing strategy. Epsilon was chosen between 10^{-3} and 10^{-6} of the maximum Gaussian curvature $\max_j |K_j|$, where all triangles, edges and vertices of the triangulation are considered for this maximization. For edges and vertices the area ΔA is chosen as the smallest from the adjacent triangles. The bigger ε is the less adaptive the approach becomes. Using the relative speeds $v_{rel,j}$, the actual local speeds v_j are computed based on a per chart scaling factor λ_i: $v_j = \frac{1}{\lambda_i} v_{rel,j}$. Hence, one can define the time delays to be $\Delta t_j = \Delta s_j/v_j$, and Δs_j is the distance between triangles measured in Euclidean space as shown in Figure 3 (b), i.e.

$$\Delta s_j = \|\mathbf{c}_{T_{in,j}} - \mathbf{c}_{T_{ex,j}}\| \tag{8}$$

where $\mathbf{c}_T$ is triangle T's centroid.

The final step is the computation of the λ_i's for each chart in order to adjust their inflows. This can be achieved by plugging in the expressions for Δt_j into Equation (5) and solving for λ_i. As the sum over the index j in Equation (5) runs over the chart boundary edges, λ_i changes over time. With a changing λ_i also the time delays $\Delta t_j(\lambda_i)$ change over time, what makes it difficult to keep the incorporation times efficiently in the right order.

To avoid this computational burden for the λ_i we propose the use of a multi-heap chart growing approach. For this we make the assumption that the change of a λ_i over time does not re-order the triangle incorporation times inside chart i but only the incorporation order between the different charts. This makes sense as we introduced the λ_i's to balance the growing of the different charts. The idea of the multi-heap approach is to keep one primary heap of charts and for each chart a secondary heap. The primary heap is used to sort the charts according to their accumulated Gauss area. The secondary heap sorts the triangles adjacent to the chart boundary with respect to the local time of the chart in order to determine which triangle will be incorporated next. For a better balancing we did one additional lookahead step by adding the ΔA_G of the triangle that should be incorporated next to the Gauss area weight of the primary heap. The secondary heaps are sorted by local time scales, where the global

time t was scaled by the λ_i of the charts, i.e. $t_i = t/\lambda_i$ such that λ_i is completely dropped from the equations defining the Δt_j. In this way the charts could be flooded with completely balanced Gauss area inflow. Finally, the Δt_j is rewritten as

$$\Delta t_j = \Delta s_j \cdot f(|K_j|) \tag{9}$$

in the multi-heap implementation.

3.4 Seeds positioning and convergence

Our seed placement and re-positioning strategy are similar to [2, 26]. We place the first seed at random, grow one chart and place the next seed furthest apart from the seeds placed thus far. After each t-flooding phase, the seeds need to be repositioned in a way to respect the features even better in the next iteration. We also followed a Lloyd-Max strategy in [2, 26]. For each chart we compute the point of maximal distance from the boundary of the chart. We work on the triangle graph of the charts and weight the Euclidean distances by the Δt_j of the triangles. This approach repulses seeds from each other when they are too close on the surface.

After the number of seeds reaches the user specified number of charts in above iterations, we continue to optimize the seed positions by the Lloyd-Max strategy with a Gauss area offset computation which is detailed in Subsection 3.5. This seed position optimization is repeated until all seed positions are converged. However, discrete optimization procedure sometimes fails to converge because of oscillation as also mentioned in [26]. Therefore, we regard one of the following cases as converged: 1. a seed place oscillation cycle is detected, 2. the number of iterations exceeds a user defined threshold, 3. Gauss area distribution is not improved recent n iterations where n is a user specified threshold, e.g. $n = 10$ in this paper.

3.5 Offset computation of Gauss area

A major problem for a perfectly balanced chart generation is the early blocking of charts during the flooding procedure. It happens quite often that all the boundary edges of a chart become adjacent to different neighboring charts, such that it has no possibility to grow any further. We observed the seed repositioning does not seem to resolve these situations in some cases.

We therefore extend our multi-heap approach. To each chart i we assign a Gauss area offset $A_{G,\text{off},i}$. The offset is kept constant over each flooding step and initialized to zero in the beginning. The weights for the primary heap were simply computed as the sum of the incorporated Gauss area plus the chart offset. Some too fast growing charts have this offset as a penalty. This allows to delay the flooding of some charts with relation to the others.
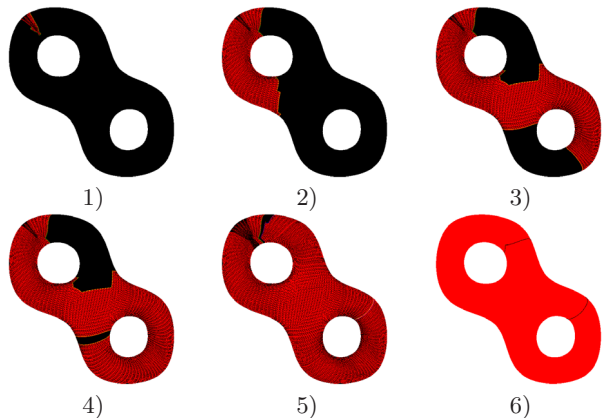

Fig. 4 HandleCutter tool. 1) start growing, 2)-3) split boundary/growing, 4)-5) meets split boundary and detects handle, 6) cut result. The final cutting paths are chosen by keeping the shortest perimeters for each wavefront during the growing.

This very simple mechanism can be efficiently exploited to improve the balancing performance. After each flooding step the Gauss areas of the resulting charts are used to define the offsets for the next flooding round. We increase the offsets of the larger charts and keep the offsets of the smallest chart fixed. If $A_{G,\text{off,min}}$ is the Gauss area of the smallest chart we set

$$A_{G,\text{off},i} := A_{G,\text{off},i} + \beta(A_{G,\text{off},i} - A_{G,\text{off,min}}), \quad \beta \in]0,1]$$

with the damping factor β. In our experiments, we use $\beta = 0.9$. This simple method allowed us to well balance the charts with respect to Gauss area.

3.6 Topology constraints

The final part of our approach deals with the topological consistency of the charts. In a texture mapping context, it is desirable to have chart with disk topology. Thus we implement two tools, a handle cutting tool (HandleCutter) and a tool that cuts a genus zero chart with several boundary loops into a disk-like chart with one boundary loop (LoopMergeCutter). Both tools employ greedy approaches and do not try to generate optimal cuts as this problem is known to be NP-complete [7].

The HandleCutter detects handles and cuts each handle to make a cylinder. This algorithm is based on a region growing algorithm similar to the approximate handle cutting procedure in [7]. We start with a seed, which is usually randomly selected, and grow a region while keeping the region boundary as short as possible. Around a handle the region boundary will split at least once and exactly once per handle two different region boundaries merge again. The two boundary loops that merge are both candidates for cutting the handle. We remember the shortest wavefront boundary loops during the growing, and when we detect a handle, we always select the
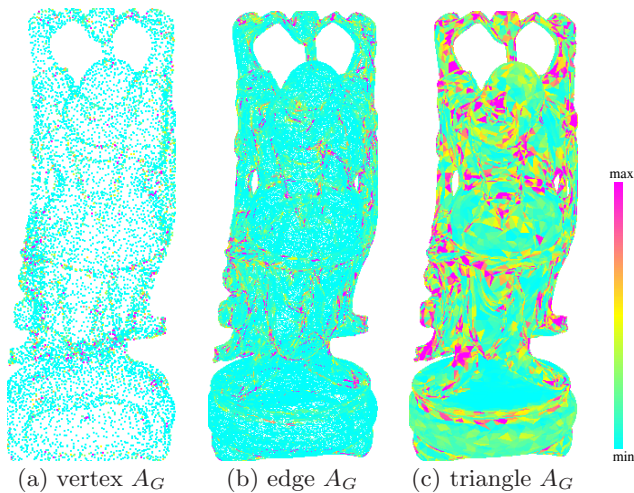
(a) vertex A_G (b) edge A_G (c) triangle A_G

Fig. 5 Gauss area computation. Color coded Gauss area : blue (small Gauss area) to red (large Gauss area). High Gaussian curvature elements are detected as features.

shorter of the two candidate loops as a cutting path. Then we continue to grow until the region grows over all triangles. Figure 4 demonstrates how to cut a double torus.

The LoopMergeCutter traverses the resulting mesh starting from the largest chart boundary loop in a breadth first order storing the shortest distances to the surrounding boundary loop. The interior boundary loop that is hit first to its surrounding boundary is connected with a shortest path to the surrounding boundary. The shortest path can be found by a simple backtracking procedure on the stored shortest distances. The resulting cut and the hit boundary loop are merged with the surrounding loop and the process is continued until all loops have been merged into the surrounding boundary loop. It is additionally possible to cut to the tips cylindrical shapes, which are closed on one side. These typically cause high distortions that are not captured in terms of Gaussian curvature. To cut such tips we detect the local extrema in the distance map resulting from the loop merging process. If an extremum is found that is at least as distant from the final boundary loop as the length of the final boundary loop, we also generate a shortest cut to this local extremal vertex.

4 Results and Discussions

We applied our algorithm to several data sets and compared the result to some of the existing methods.

Figure 5 visualizes the Gauss area of each mesh element type, (a) vertex, (b) edge, and (c) triangle. Large Gauss area coincides with high Gaussian curvature. The elements which have high Gaussian curvature are detected as they are usually recognized features.

Figure 6 compares the segmentation results of (a) Multi-chart geometry image (MCGIM) [26], (b) Varia-

tional shape approximation (VSA) [2], and (c) our approach. The method of [2] is a shape approximation method, and is not presented as a segmentation method. However, this method usually generates high quality segmentation.

Table 1 shows L^2 geometric stretch [25] measurement results. To obtain these results we first cut the patches into topological discs, then we projected each chart boundary to an unit circle, parameterized each chart with [4] and measured L^2-geometric stretch. The results show that our method gave clearly lowest distortion compared to others. We segmented the meshes into 20 charts in all experiments. For the VSA method, both teleportation and merge of charts were activated. For our method (t-flooding), the dumping factor was set to $\beta = 0.9$ during offset calculation, $\varepsilon = 0.0001$, the power p of the speed function was 3.0 for Happy model and 1.0 for Santa and Rocket. As partial normal evaluation threshold we used $T_a = 45$ degrees for all examples. From Figure 6 and Table 1 one can see that t-flooding distributes distortions equally over the charts, while other methods have both low distortion and high distortion charts.

Table 1 also shows elapsed time of t-flooding and number of triangles of each mesh. Currently, no optimization has been carried out in our implementation. All timings were measured on a 1.7 GHz Pentium 4 Linux machine.

Table 2 shows the distribution of Gauss area over charts and total Gauss area of the models. Since our method is designed for minimizing this standard deviation, it has the lowest standard deviation of Gauss area distribution.

5 Conclusions and Future Work

In this work we presented a new criterion for mesh segmentation, which is based on the even distribution of Gaussian curvature over the resulting charts. For numerical stability we used Gauss area on the Gauss map to estimate Gaussian curvature. This method generates close to developable charts. We showed that the created charts are suited better for parameterization than charts generated with approaches known from the literature. Our approach is especially suited for the cutting of higher genus models into a small number of charts. The generated cuts are nicely shaped and adapt to sharp surface features that are not developable. Our approach generates charts that can be parameterized with low distortion without directly measuring the distortion during segmentation.

For the Gauss area equalization we introduced the new paradigm of flooding with constant inflow, which we implemented with an efficient multi-heap strategy. We also proposed an offset scheme for the repeated flooding process allowing for a much better balanced mesh segmentation.

Fig. 6 Segmentation effect for texturing of several models. (a) Segmentation by MCGIM [26], (b) by VSA [2], (c) by our t-flooding. Number of charts is 20 for all examples. The green line shows chart boundary. A conformal parameterization method [4] is applied for all charts with the same parameters. Each parameterized chart is mapped to $[0, 1] \times [0, 1]$. The size of checker board texture is 256×256 pixels.

Table 1 Parameterization distortion. Distortion is measured by L^2-geometric stretch [25]. The values are averaged over all patches. Standard deviation is given in parentheses. Number of triangles and t-flooding elapsed time (sec.) are also shown.

model	Happy	Rocket	Santa
MCGIM [26]	8.1(7.8)	29.1(39.3)	22.9(11.6)
VSA [2]	12.4(12.6)	28.2(21.5)	60.1(47.1)
t-flooding	7.3(4.7)	17.9(7.3)	17.2(8.6)
# of tris.	19976	80354	151558
Elapsed time	20.6	91.5	363

Table 2 Standard deviation (Equation (3)) of the Gauss area distribution.

model	Happy	Rocket	Santa
MCGIM [26]	64.3	11.0	14.3
VSA [2]	69.0	17.3	18.9
t-flooding	25.6	4.55	2.48

In future work we would like to apply the t-flooding approach to other region growing based mesh segmentation approaches in order to achieve better error equalization. We also would like to improve the Gauss area estimation since current our method is based on simple angle thresholding.

References

1. Benkő, P., Várady, T.: Direct segmentation of smooth, multiple point regions. In: Proc. of the Geometric Modeling and Processing theory and applications (GMP '02), pp. 169–178. IEEE Computer Society (2002)
2. Cohen-Steiner, D., Alliez, P., Desbrun, M.: Variational shape approximation. ACM Trans. Graph. **23**(3), 905–914 (2004)
3. Cohen-Steiner, D., Morvan, J.M.: Restricted delaunay triangulations and normal cycle. In: Proc. of the 19th Symp. on Computational Geometry, pp. 312–321 (2003)
4. Desbrun, M., Meyer, M., Alliez, P.: Intrinsic parameterizations of surface meshes. In Eurographics 2002 Conference Proc. **21**(3), 209–218 (2002)
5. Dey, T.K., Giesen, J., Goswami, S.: Shape segmentation and matching with flow discretization. In: Proc. Workshop Algorithms Data Structures (WADS 03), LNCS 2748, pp. 25–36 (2003)
6. Elber, G.: Model fabrication using surface layout projection. Computer-Aided Design **27**(4), 283–291 (1995)
7. Erickson, J., Har-Peled, S.: Optimally cutting a surface into a disk. In: Proc. of the 18th ACM Symp. on Computational Geometry, pp. 244–253 (2002)
8. Funkhouser, T., Kazhdan, M., Shilane, P., Min, P., Kiefer, W., Tal, A., Rusinkiewicz, S., Dobkin, D.: Modeling by example. ACM Trans. Graph. **23**(3), 652–663 (2004)
9. Garland, M., Willmott, A., Heckbert, P.: Hierarchical face clustering on polygonal surfaces. In: Proc. of ACM Symp. on Interactive 3D Graphics, pp. 49–58 (2001)
10. Gelfand, N., Guibas, L.J.: Shape segmentation using local slippage analysis. In: Proc. of the Eurographics Symp. on Geometry Processing (SGP-04), pp. 219–228 (2004)
11. Hoschek, J.: Approximation of surfaces of revolution by developable surfaces. Computer-Aided Design **30**(10), 757–763 (1998)
12. Inoue, K., Itoh, T., Yamada, A., Furuhata, T., Shimada, K.: Clustering large number of faces for 2-dimensional mesh generation. In: Proc. of the 8th International Meshing Roundtable, pp. 281–292 (1999)
13. Julius, D., Kraevoy, V., Shaffer, A.: D-charts: Quasi-developable mesh segmentation. In: Eurographics (2005). to appear
14. Kalvin, A.D., Taylor, R.H.: Superfaces: Polygonal mesh simplification with bounded error. IEEE Computer Graphics and Applications **16**(3), 64–77 (1996)
15. Katz, S., Tal, A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. ACM Trans. Graph. **22**(3), 954–961 (2003)
16. Lee, A.W.F., Sweldens, W., Schröder, P., L. Cowsar, L., Dobkin, D.: MAPS: Multiresolution adaptive parameterization of surfaces. In: SIGGRAPH '98 Conf. Proc., pp. 95–104 (1998)
17. Lévy, B., Petitjean, S., Ray, N., Maillot, J.: Least squares conformal maps for automatic texture atlas generation. ACM Trans. Graph. **21**(3), 362–371 (2002)
18. Liu, R., Zhang, H.: Segmentation of 3D meshes through spectral clustering. In: Pacific Graphics, pp. 298–305 (2004)
19. Maillot, J., Yahia, H., Verroust, A.: Interactive texture mapping. In: SIGGRAPH '93 Conf. Proc., pp. 27–34 (1993)
20. Mangan, A.P., Whitaker, R.T.: Partitioning 3D surface meshes using watershed segmentation. IEEE Trans. on Visualization and Computer Graphics **5**(4), 308–321 (1999)
21. Max, N.: Weights for computing vertex normals from facet normals. J. Graph. Tools **4**(2), 1–6 (1999)
22. Mitani, J., Suzuki, H.: Making papercraft toys from meshes using strip-based approximate unfolding. ACM Trans. Graph. **23**(3), 259–263 (2004)
23. Page, D.L., Koschan, A., Abidi, M.: Perception-based 3d triangle mesh segmentation using fast marching watersheds. In: Proc. Intl. Conf. on Computer Vision and Pattern Recognition, Vol. II, pp. 27–32 (2003)
24. Pottmann, H., Farin, G.E.: Developable rational bézier and b-spline surfaces. Computer Aided Geometric Design **12**(5), 513–531 (1995)
25. Sander, P.V., Snyder, J., Gortler, S.J., Hoppe, H.: Texture mapping progressive meshes. In: SIGGRAPH '01 Conf. Proc., pp. 409–416 (2001)
26. Sander, P.V., Wood, Z.J., Gortler, S.J., Snyder, J., Hoppe, H.: Multi-chart geometry images. In: Proc. of the Eurographics Symp. on Geometry Processing (SGP-03), pp. 146–155 (2003)
27. Shamir, A.: A formulation of boundary mesh segmentation. In: 3DPVT, pp. 82–89 (2004)
28. Shlafman, S., Tal, A., Katz, S.: Metamorphosis of polyhedral surfaces using decomposition. Computer Graphics Forum **21**(3), 219–228 (2002)
29. Sorkine, O., Cohen-Or, D., Goldenthal, R., Lischinski, D.: Bounded-distortion piecewise mesh parameterization. In: IEEE Visualization 2002, pp. 355–362 (2002)
30. Welch, W., Witkin, A.: Free-form shape design using triangulated surfaces. In: SIGGRAPH '94 Conf. Proc., pp. 247–256 (1994)
31. Zhou, K., Snyder, J., Guo, B., Shum, H.Y.: Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In: Proc. of the Eurographics Symp. on Geometry Processing (SGP-04), pp. 47–56 (2004)