EUROGRAPHICS 2005 / M. Alexa and J. Marks
(Guest Editors)

Volume 24 (2005), Number 3

D-Charts: Quasi-Developable Mesh Segmentation

Dan Julius[†] Vladislav Kraevoy [‡] Alla Sheffer [§]

University of British Columbia, Vancouver, B.C., Canada

Abstract

Quasi-developable mesh segmentation is required for many applications in graphics and CAD, including texture atlas generation and the design of patterns for model fabrication from sheets of material. In this work we introduce D-Charts*, a simple and robust algorithm for mesh segmentation into (nearly) developable charts. As part of our method we introduce a new metric of developability for mesh surfaces. Thanks to this metric, using our segmentation for texture atlas generation, we can bound the distortion of the atlas directly during the segmentation stage. We demonstrate that by using this bound, we generate more isometric atlases for the same number of charts compared to existing state-of-the-art techniques. Using our segmentation algorithm we also develop a technique for automatic pattern design. To demonstrate the practicality of this technique, we use the patterns produced by our algorithm to make fabric and paper copies of popular computer graphics models.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computing Methodologies / Computer Graphics]: Surface representations, J.5. [Computer Applications / Arts and Humanities]: Fine arts

1. Introduction

Mesh segmentation has numerous applications in computer graphics. One of its oldest applications is texture atlas generation. For atlas generation, the mesh is first segmented into charts and then each chart is parameterized onto the plane. The parameter space is covered with an image, which is then mapped onto the model using the parameterization. With the introduction of programmable GPUs, more general attributes can be mapped onto the model in real time (e.g., BRDFs, bump maps, displacement maps, etc.). It is even possible to represent the geometry of the model in parameter space, leading to the *geometry images* representation [GGH02]. To preserve the texture and other attributes during mapping, the parameterization has to be quasi-isometric. This is possible only if the charts are nearly developable. Boundaries between charts lead to undesirable discontinuities in the mapped texture; hence a good segmentation method must segment the mesh into nearly developable charts while minimizing the number of charts and their boundary lengths. Since finding an optimal set of charts

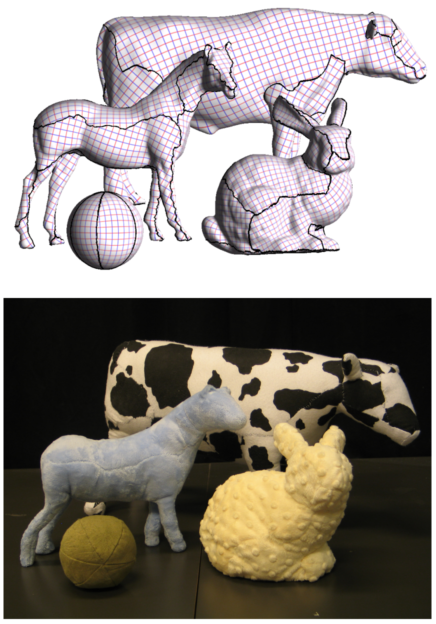
Figure 1: *Texture atlases and stuffed toys generated using D-Charts.*

† djulius@cs.ubc.ca
‡ vlady@cs.ubc.ca
§ sheffa@cs.ubc.ca

is NP-Hard in nature, existing methods typically search for a local minimum.

The need for developable segmentation is not unique to atlas generation. Such segmentation is also required for the design of sewing patterns, for metal forming and forging, and for other fabrication applications where 3D objects are constructed from sheets of material.

In this work we introduce a novel, simple, and intuitive metric of surface developability. Based on this metric we describe an efficient and robust algorithm, which we name *D-Charts*, for segmenting meshes into (nearly) developable charts. The algorithm combines geometry processing techniques with tools traditionally used by pattern designers. As a result, compared to recent mesh segmentation methods, for a given number of charts the D-Charts method segments the mesh into more developable charts, leading to less distortion in the generated texture atlases. Our algorithm successfully segments mechanical models into their developable components, producing isometric texture atlases. For other models for which a compact developable segmentation does not exist, the segmentation is based on a user-prescribed developability tolerance.

Based on our segmentation algorithm we introduce an automatic method for pattern design, focusing on the design of soft, stuffed toys. To demonstrate the practicality of our technique we design patterns for several popular computer graphics models and then use them to construct physical copies of the models out of fabric or paper.

The rest of this paper is organized as follows: Section 1.1 reviews previous work on texture atlas generation and computer-aided pattern design. Sections 2 and 3 describe our D-Charts segmentation algorithm. Section 4 introduces a novel pattern design technique, based on the D-Charts algorithm. Section 5 demonstrates the use of the D-Charts algorithm for texture atlas construction and compares the results with those of previous methods. It also presents the patterns and actual soft toys designed using our pattern design technique. Finally, Section 6 summarizes our work.

1.1. Previous work

Texture atlas generation: Over the last decade, many methods have used mesh segmentation to generate texture atlases [IC01, KL96, MYV93, LPRM02, ZSGS04]. Most existing segmentation algorithms focus on the construction of nearly planar charts [MYV93, GVSS00, SSGH01, GWH01, SWG*03]. While planar charts are obviously developable, most developable surfaces are not planar. Thus planar segmentation is too restrictive and results in more charts than are necessary.

Other segmentation methods, such as [KT03], focus on feature-based segmentation. They segment the mesh into charts corresponding to protrusions and other meaningful model components. Typically these charts are very far from being developable and are thus unsuitable for our needs. Methods such as that of Gelfand and Guibas [GG04] are well suited for segmenting CAD point-sets into kinematic surfaces. However, these surfaces form only a small subset of the developable surfaces, and the method is not appropriate for irregular meshes.

Levy et al. [LPRM02] introduce a segmentation method that detects crease lines and generates charts using these lines as boundaries. Charts are then further segmented if the stretch after the parameterization is too high. The method fails to segment models properly if there are no clear crease lines.

Sorkine et al. [SCOGL02] simultaneously generate both the parameterization and the cuts. Thus this method can successfully discover developable regions and parameterize them using a single chart. Since this method depends on the order of the triangle insertions it tends to form charts with long and complex boundaries.

Gu et al. [GGH02] construct a single chart out of the input model. They first generate cuts that convert the surface into a disk and then iterate between parameterization and cutting, continuing to add cuts from the current boundary to points of maximal distortion, as long as the distortion is deemed to be too high. Sheffer and Hart [SH02] also cut the entire surface into a single chart. They detect points of high curvature and connect these by a Steiner tree of cuts, taking visibility into account, so that the cuts go through less visible parts of the model. While both methods result in the minimal number of charts (i.e., one), the chart boundaries tend to be quite long and complex.

The *Iso-charts* method [ZSGS04] interleaves parameterization and segmentation. Starting with an initial segmentation it repeatedly parameterizes the charts and then segments them if the parametric distortion is high. The authors use a variation of the fuzzy-region cutting approach [KT03] to generate straight boundaries between the charts. In Section 5, we present a comparison of atlases generated by our method with those generated by Iso-charts.

Guthe and Klein [GK03] introduce a method for atlas generation for NURBS surfaces, which is particularly suitable for pattern design. The charts are formed by stitching NURBS patches together based on distortion considerations. The authors employ a cutting approach similar to that used by [GGH02,SH02] using the distortion in the current parameterization as an indicator of where to place cuts. Further cuts are added when the resulting charts overlap in the parameter domain. It is not clear how well these methods will work for irregular meshes.

Pattern design: Much of the research on pattern generation focuses on unfolding, or parameterization, of given charts. McCartney et al. [MHS99] introduce an automatic unfolding method and demonstrate the impact of darts and gussets on the quality of the unfolding. Darts are stitched tapering folds, generated by cutting a sharp corner out of a pattern. Gussets are triangular inserts added into seams to widen the corresponding regions. Wang [WTY05] proposes another method for the unfolding of free-form surface charts for pattern design by fitting a woven mesh model.

Some of the mesh parameterization techniques developed in recent years can be directly applied to the unfolding of patterns. For a survey of recent techniques see [FH05]. The only parameterizations useful for pattern design are those that compute the boundary of the planar domain as part of the solution. For any sewing task the fabric cannot stretch beyond a certain limit based on the properties of the fabric. Shearing, on the other hand, is acceptable in some applications but not in others. For instance, when fabric is wrapped around a hard object, the shearing of the material causes no visible artifacts; thus for these applications stretch minimizing parameterization [SGSH02] would be a natural choice. In contrast, when wrapping the fabric around soft stuffing, such as the soft toys in our examples, shearing leads to visible wrinkles and is therefore extremely undesirable. Thus, for this particular application, conformal parameterization methods that minimize shearing are preferable (e.g., [HG00, LPRM02, DMA02, SdS01]).

Several methods address segmentation for fabrication applications. To generate patterns for metal-work, Elber [Elb95] approximates NURBS surfaces by cross-section aligned developable strips. This method requires a very large number of strips to approximate accurately the input models, since only rectangular, ruled surface strips were allowed. Kolmanic and Guid [KG03] use cross section curves provided as part of the data to segment models into developable strips bounded by these curves in order to generate shoe patterns.

Mitani et al. [MS04] introduce a method for pattern generation for making papercraft toys from meshes using strip-based approximate unfolding. Since basic triangle strips can be extremely long, they first segment the mesh as suggested in [LPRM02] and then compute strips within each chart. This method results in quite a large number of charts with complex boundaries, and so the actual cutting and gluing together of charts is quite challenging. The technique is specific to papercraft; here the core constraint is that paper is incompressible, requiring charts to be truly developable. Since fabric can stretch, the "stripification" approach is too restrictive for fabric pattern design.

Developable surfaces: The research on developable surfaces in the modeling and graphics communities focuses predominantly on modeling with such surfaces [PW99, LP98] and on the approximation of point clouds [CLL*99, PS04, Pet04]. The proposed approximation and modeling techniques require complex manipulation of spline curves and surfaces.

2. D-Charts algorithm overview

Our mesh segmentation algorithm uses a region-growing approach. Like [SWG*03, CSAD04] we use an iterative Lloyd scheme [Llo57], avoiding random initialization issues common to older methods, such as [SSGH01, SCOGL02]. We use a similar notation of proxies and seeds to characterize each chart. The general Lloyd algorithm framework is:

- Grow charts, covering the entire model, based on the current proxies and seeds.
- Compute new proxies and seeds.
- Repeat until the process converges.

As pointed out in [CSAD04] the segmentations, computed using this framework, depend on the initial positioning of the seeds. To improve the segmentation the authors propose the use of manual intervention techniques to increase or decrease the number of charts, or to "teleport" them.

We avoid the need for manual intervention and develop an automatic procedure, based on an initial estimate of the number of charts and a fitting error threshold value F_{max}. The fitting error measures how well a triangle fits into a chart, and is defined in Section 2.1.1. Our method increases or decreases the number of charts as necessary based on the actual fitting error computed during segmentation.

The steps of our algorithm are as follows:

- **Modified Lloyd Iterations:** Charts are grown using Lloyd iterations while bounding the fitting error by F_{max}. Using this fitting error threshold ensures that charts remain nearly developable (Section 3.1).
- **Hole filling:** As the fitting error is bounded during the growing process, some triangles are not assigned to any chart. During hole filling these triangles are assigned to charts, extra charts are added when necessary (Section 3.2).
- **Post-processing:** During post-processing the resulting charts are further improved. First, the boundaries between charts are straightened, without increasing the fitting error. Then, adjacent charts are merged if the combined chart is developable. Finally, seams are cut toward regions of high error, forming darts and gussets (Section 3.3).

The algorithm stages are explained in detail in Section 3.

2.1. Proxies and cost function

Our main motivation is to segment the mesh into (nearly) developable charts. However, in contrast to segmentation into planar regions, the developability condition alone is typically not sufficient for a meaningful segmentation. We observe that any triangle strip is by definition developable. Meshes can be "stripified" using a very small number of triangle strips. Obviously this segmentation is not particularly useful, as it leads to charts with extremely long boundaries. Hence the challenge we face is to segment the mesh into reasonably compact, developable charts. Below, we introduce a cost function that combines developability and compactness criteria.

2.1.1. Developability and proxies

The standard definition of a developable surface is one with zero Gaussian curvature at all points. In our experience, this definition does not provide a viable tool for detecting developable regions in a mesh, resulting in undesirable charts with very long and complex boundaries. Therefore, we narrow our detection mechanism to a subset of developable surfaces — unions of uni-axial conics. We base our procedure on a simple observation:

A surface is a *union of conics* with aligned axes and the same cone angle if and only if the angle between the normal to the surface at every point and a common axis is constant.

The proof of this observation is trivial. This definition of conics includes both cylinders and planes. In the case of a cylinder, the normal is perpendicular to the axis. In the case of a plane, the normal is aligned with the axis. We use this simple condition to measure developability. The merging procedure (Section 3.3) further broadens the set of developable surfaces that we capture, without increasing the boundary complexity.

This angle-based condition is the basis of our segmentation procedure. We define the proxy for each chart C as the pair $< N_C, \theta_C >$ where N_C is a unit vector representing the axis of the union of cones and θ_C is the constant angle between the normals to the mesh surface and the axis. To measure how well a given triangle t with a normal n_t fits into a given chart C, we define the *fitting error* as:

$$F(C,t) = (N_C \cdot n_t - cos\theta_C)^2. \tag{1}$$

This error metric is significantly simpler than previous metrics used for computing developable regions, such as that of [ZSGS04] where spectral analysis was needed to achieve a similar goal.

2.1.2. Compactness

To test for compactness when adding a triangle to a chart, we introduce two additional metrics, the first aiming at the generation of relatively "round" charts, the second at the generation of charts with straight boundaries. Both metrics measure the suitability of adding a triangle to a chart when the triangle shares one or more edges with this chart. To measure compactness we define

$$C(C,t) = \pi \frac{D(S_C,t)^2}{A_C}$$

where S_C is the seed triangle of the given chart, $D(S_C,t)$ is the length of the shortest path (inside the chart) between the two triangles, and A_C is the area of chart C. For triangles on the boundary of a circle (which is ideally compact) this metric evaluates to one; otherwise, distant triangles are penalized, while close triangles are promoted.

To promote straight boundaries we use the ratio between the length of the triangle's edges that are shared with the chart and those that are not. We define

$$P(C,t) = \frac{l_{outer}(C,t)}{l_{inner}(C,t)}.$$

2.1.3. Combined cost function

The cost function measures the cost of adding a triangle to a chart, where the triangle shares one or more boundary edges with triangles inside the chart. The complete cost metric for the triangle is thus:

$$Cost(C,t) = F(C,t)^{\alpha} C(C,t)^{\beta} P(C,t)^{\gamma}. \tag{2}$$

The weights α, β, and γ control the importance we give to each metric. For all our examples we used $\alpha = 1$, $\beta = 0.7$, and $\gamma = 0.5$. These numbers were found empirically to provide an appropriate balance between developability and compactness. Using lower values of β and γ sometimes resulted in elongated strips or jagged boundaries. Figure 3 demonstrates the effect of β when segmenting the sphere.

3. Algorithm stages

Given the above definitions of a proxy, a fitting error, and a cost function we now describe the stages of the algorithm in more detail (Figure 2).

3.1. Modified Lloyd Iterations

Initialization: Given the initial number of charts, we first select a seed for each chart. Although in theory any random set of triangles should be adequate (since the iterations that follow will eventually move the charts into place), we find that when extremities exist in the mesh a more refined choice of seeds usually leads to faster convergence with better results. Thus, we try to maximize the distance between seeds, using the standard farthest point algorithm.

Next, a proxy is calculated for each of the seeds. For each seed we examine three potential charts — the sets of triangles around each of the three seed-triangle vertices. We calculate a proxy for each chart as explained below (Equation 3) and select the proxy that minimizes the fitting error (Equation 1) for the seed triangle.

Growing Charts: Initially each chart contains only a single triangle — the seed. For each seed, we insert each of its adjacent triangles into a priority queue. For each triangle we specify both its adjacent chart and the cost of assigning it to that chart (Equation 2). As long as the queue is not empty, the triangle with minimal cost is extracted from it and assigned to its specified chart if the following conditions hold:

1. The triangle has not yet been assigned to any other chart (this can occur, since each triangle may have more than one entry in the queue).

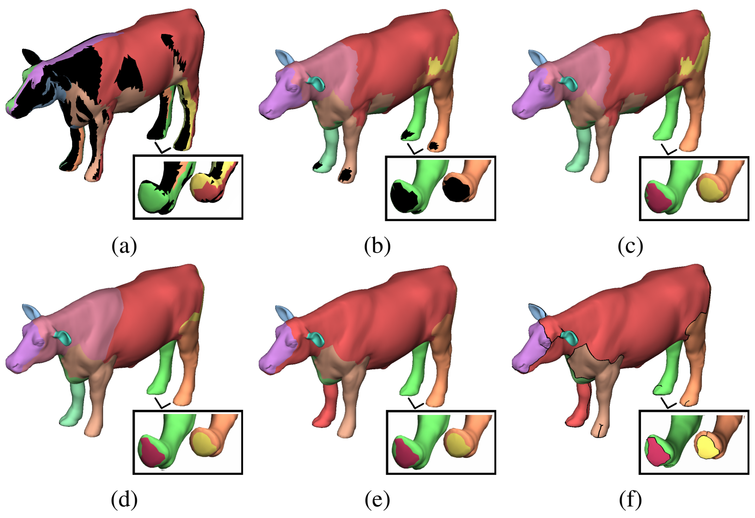

Figure 2: *Stages of D-Charts on a bull model with a bottom view of the back feet: (a) Charts after one growth iteration. (b) Final charts. (c) Hole filling. (d) Boundary straightening. (e) Merged charts. (f) Adding cuts.*

2. The fitting error for the triangle and specified chart is below the threshold $P(C,t) < F_{max}$. The fitting threshold ensures that charts remain nearly developable.

Otherwise, the triangle is skipped and the next triangle in order is processed. After assigning the triangle to the chart, its adjacent triangles, which have not yet been assigned to any chart, are inserted into the queue (specifying the current chart and the cost with respect to it). After the process terminates, proxies are recomputed and the growth procedure is repeated. Figure 2(a) shows the bull after a single iteration.

Note that both $P(C,t)$ and $C(C,t)$ (Section 2.1.2) change as chart C is grown, thus changing the $Cost(C,t)$ for each triangle. To avoid re-sorting the entire priority queue after every step, we maintain a queue per chart. At each step the triangle with minimal cost is found by comparing the first triangle from each queue. This method requires only local updates to each queue after each step. Global sorting is not needed as the order of triangles in each in terms of cost is not altered.

Finding new proxies and seeds: The optimal proxy $< N_C, \theta_C >$ for a given chart contains the normalized axis vector and the angle that minimize the weighted fitting error between the conic surface and the chart triangles. The new proxy is computed by solving the following constrained optimization problem:

$$\min_{N_C,\theta_C} \frac{1}{A_C} \sum_{t \in C} A_t F(C,t) \quad \text{s.t.} \quad \|N_C\|^2 = 1 \tag{3}$$

where A_t is the area of the triangle t. We use a standard Newton solver to obtain the solution. Since only four variables are involved, the solution takes only milliseconds.

After calculating the new proxies, we select a new seed S_C for each chart. The seed must fit the proxy well (i.e., with only a small fitting error), and to the extent that is possible, it should be near the center of the chart. To find such seeds we examine the first k triangles in the chart with minimal fitting error ($k = 10$ in all our examples), and then select the one closest to the center of the chart. This approach improves the method suggested in [CSAD04] since nearly developable surfaces will have a large number of triangles with very low fitting error, resulting in an almost random selection of new seeds.

Termination: The process is terminated when we detect that only a small percentage of triangles (below 5%) have been reassigned from one chart to another between two consecutive iterations. While the convergence of Lloyd iterations is guaranteed in the continuous case, this guarantee does not extend to 3D meshes [CSAD04]. In practice the algorithm converges in a small number of iterations (fewer than 100) given a reasonable initial estimate of the number of charts alongside a small F_{max}. To guarantee termination we specify a bound on the number of iterations. Figure 2(b) shows the bull after the iterations have converged.

3.2. Filling holes

After the Lloyd iterations converge, most of the triangles are classified as belonging to one chart or another. However, due to the use of the error bound F_{max}, some triangles may not belong to any chart (Figure 2(b)). We collect all such triangles into connected components and categorize each component as either a large or a small hole, depending on the area of the component. Small holes are then filled by removing the bound on the fitting error and growing the surrounding charts into the holes (the introduced error is later reduced by adding partial cuts, Section 3.3). Large holes are dealt with by adding additional charts inside the holes. First, all the triangles in the hole are assigned to a new chart; next, an optimal proxy and seed are computed, and finally the chart is "regrown" inside the hole as described above. This may result in additional holes; and these are dealt with recursively. This approach leads to better results than spawning new proxies during the iteration phase. Regions of high Gaussian curvature, such as the tip of the nose on the Horse model (Figure 4), repel the growing charts. Even newly spawned charts are pushed away after a few iterations, resulting in additional ones being spawned and an excessive number of charts in total. Figure 2(c) shows that the small holes on the top side of each of the bull's feet are filled by adjacent charts, while the larger holes on the bottom are filled by creating additional charts.

3.3. Post-processing

The post-processing performs three major operations: it straightens chart boundaries, merges adjacent charts when developability conditions allow, and finally introduces partial cuts into the mesh.

Straightening boundaries: After convergence, boundaries between charts often tend to be jagged, especially in areas where fitting errors between the triangles and each of the two adjacent charts are similar. To improve the segmentation we define fuzzy areas between each pair of adjacent charts. Fuzzy areas are the regions along the boundaries in which the triangle fitting error is low with respect to both charts. To find these regions, both charts are grown virtually into one another. Since the fitting error is bounded by F_{max}, only triangles with low fitting errors are reached and marked as belonging to the fuzzy area. Next, each boundary segment between the charts is adjusted. For each segment, first we find its two end-points; then we find the shortest path, within the fuzzy area, between these points, and define this path as the new boundary segment. Figure 2(d) shows the straightened boundaries on the bull.

While our notion of fuzzy regions is somewhat similar to that used in [KT03], the boundary straightening algorithm itself is much simpler. We use the min-cut approach [KT03] only when the boundary is circular, namely when the two charts share an entire boundary loop.

Merging charts: Our developability metric (Equation 1), captures uni-axial unions of cones. To capture more general developable surfaces, we now merge adjacent charts if together they still represent a nearly developable surface. Adjacent charts can be merged if the Gaussian curvature along their shared boundary is zero. To avoid noisy curvature computations we use the same framework as above. For a pair of adjacent charts we test to see if the boundary region between them (formed by triangles near the common boundary) can be approximated by a cylindrical surface. This is a sufficient

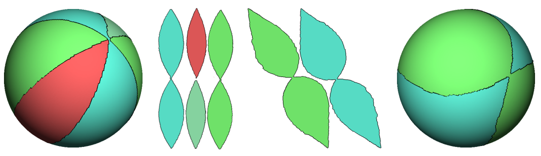

Figure 3: *Segmentation of the sphere. Left — using* $\alpha = 1, \beta = 0, \gamma = 0$. *Right — using* $\alpha = 1, \beta = 0.7, \gamma = 0$.

(but not necessary) condition for the combined chart to be developable.

Given the boundary region B, we compute a proxy $< N_B, \theta_B = \pi/2 >$ (Equation 3). The charts are merged if the average fitting error is below a given threshold η:

$$\frac{1}{A_B} \sum_{t \in B} A_t F(B,t) < \eta.$$

In the mechanical examples (Figure 5) $\eta = 1e-5$ was used to keep the charts strictly developable; in all the other examples we used $\eta = 1e-2$. Figure 2(e) shows the merged charts.

Darts and gussets: Due to hole filling, charts can contain small regions of triangles with large fitting errors. These regions can cause high distortion or "strain" in the surrounding region during parameterization. To reduce the strain, we incorporate the pattern design technique of darts and gussets into our algorithm, introducing partial cuts into the charts. These partial cuts are inserted from the boundaries toward the regions with high fitting error. Each seam relaxes the strain in a circular area centered at the tip of the cut, with a radius equal to the length of the seam; thus, elongated high-error regions may require a number of seams.

During parameterization, cuts corresponding to darts will naturally open up in 2D, while cuts corresponding to gussets will result in overlaps in the 2D parameterization. Therefore, after the parameterization, the overlaps need to be located, the overlapping regions cut off and, when possible, reattached to an adjacent chart (Section 4).

Lastly, to parameterize surfaces such as cylinders isometrically onto the plane, we introduce cuts connecting the chart boundary loops. At the end of this process we have produced a mesh segmentation into quasi-developable charts, which can be used both for texture atlas generation and for model fabrication. Figure 2(f) shows the added seems between charts, as well as darts on the ears and each of the feet.

4. Pattern design

We now describe the use of our segmentation algorithm in the design of patterns for stuffed toys and paper models. We begin by segmenting the model as described in Sections 2 and 3.

When wrapping the fabric around soft stuffing, such as in soft toys, shearing leads to visible wrinkles. Therefore we use free-boundary, conformal (low-shear) parameterization to unfold the charts in 2D. In all our examples we used [SLMB05].

After the parameterization the algorithm optimizes the charts for sewing needs. First, it tests for overlaps, removes those, and generates gussets. It then shortens excessively long darts and simplifies chart boundaries. Finally, it packs the charts into a square domain using the method of Levy et al. [LPRM02] and prints them using standard sewing notations.

We now describe the chart optimization steps in more detail.

Fixing gussets and overlaps: The parameterization we obtain may contain two types of overlaps: gussets (i.e., local overlaps) and regular, global overlaps. The algorithm detects both and cuts the charts to remove them using the method of [LPRM02]. The resulting small charts are merged with neighbors if the merging criterion (Section 3.3) is satisfied and the merge itself does not create new overlaps.

Shortening darts: Since dart creation is based on a local fitting metric, at times more darts are created than are necessary. The parameterization in practice eliminates such darts, stitching boundary vertices where the sum of angles around the vertex is close to 2π. Such darts are easily identified and then shortened by "gluing" the edges together.

Simplifying boundaries: When creating sewing patterns we need planar boundaries that are very simple. Hence we simplify the boundaries between pairs of charts in 2D using a vertex collapse method. The cost of collapsing a vertex is defined as the sum of the areas of the two triangles formed by the vertex and its neighbors on the boundary (a single triangle is formed on each side of the boundary). The costs are accumulated by adding half the cost of each collapsed vertex to each of its neighbors.

Sewing notations: To ease the sewing, we add reference points to each chart. These help identify corresponding boundary points on adjacent charts and greatly simplify the alignment of the sewing patterns.

5. Results

To evaluate the segmentation we used it to generate texture atlases for several models (Figures 4 and 5). For the parameterization stage we used the free-boundary stretch minimizing parameterization [SGSH02]. To evaluate the parameterization distortion we measured the $L_2^{Stretch}$ and $L_\infty^{Stretch}$ of the parameterization, as defined in [SSGH01], and the L_2^{Shear} as defined in [SLMB05].

The parameters used to generate each model and the results obtained are summarized in Table 1. All the results

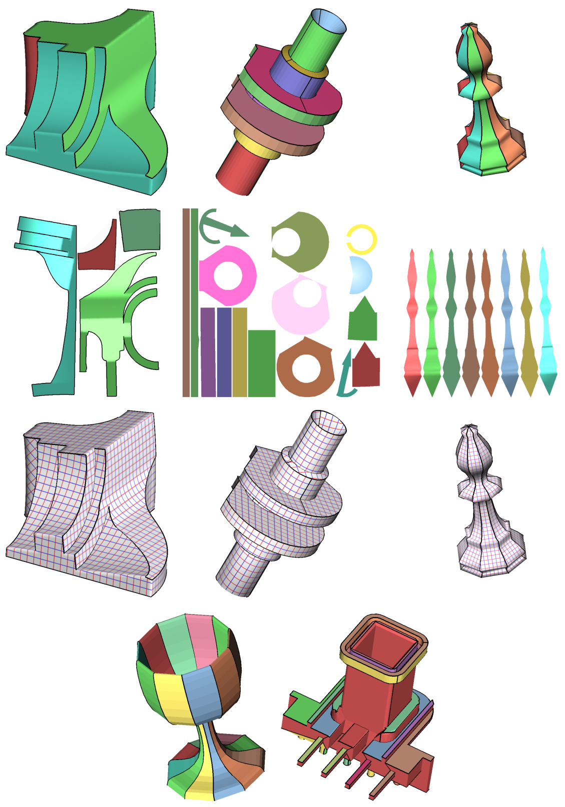
Figure 5: *Segmentation of mechanical models. First and last rows — mesh segmentation. Second row — texture atlases. Third row - parameterization (iso-lines). The $L_2^{Stretch}$ error for all the textured models is less than 1.00075 (it is not 1 as some models contain small blends or fillets).*

were generated on a P4 3GHz machine using the Graphite software package [Gra03]. The runtime is a function of both model size and complexity. For mechanical models such as the fandisk (10K faces) the time is less than 10 seconds. For free-form models the algorithm takes from 100 seconds to segment a medium-sized 20K faces model (gargoyle) and up to 500 seconds to segment the larger and more complex ones (bunny or feline). Most of the time is spent performing the Lloyd iterations; the hole-filling takes a negligible amount of time and the post-processing takes about 10% of the total time. Our times are comparable to those of the MCGIM algorithm [SWG*03]. We note that our method performs well on models of high genus such as the feline, as well as models with boundaries, such as the bunny. The D-Charts algorithm is particularly suitable for segmenting mechanical models (Figure 5) for which it accurately captures the planar, cylindrical, and conic parts.

We show a comparison between our results and those provided to us by the authors of Iso-charts [ZSGS04] and by the authors of MCGIM [SWG*03]. Our segmentation sig-

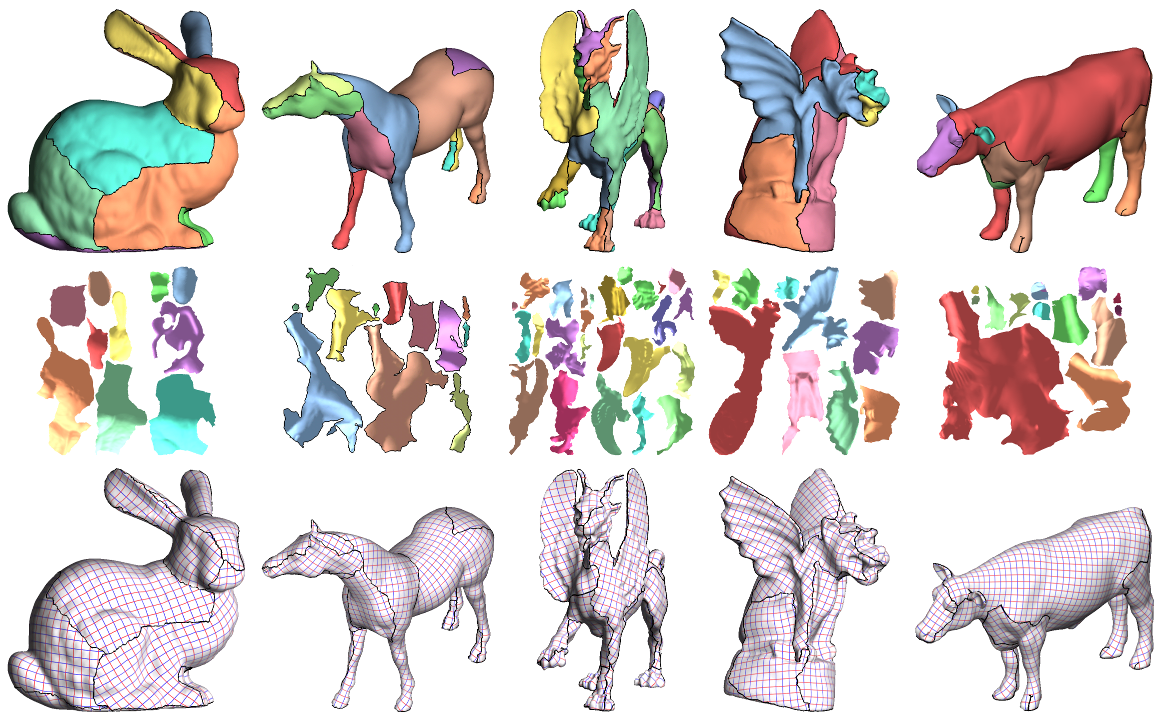

Figure 4: *Model segmentation and atlas generation. Top row — mesh segmentation. Middle row — texture atlases. Bottom row — parameterization (iso-lines). The statistics for most of the models are given in Table 1.*

nificantly improves all three error metrics for all the models, using the same or a smaller number of charts. The improvement is most significant on mechanical models such as the fandisk, where our method achieves the minimal $L_2^{Stretch}$ error of 1. This indicates that the charts we generate are much closer to being developable than those generated using previous methods. Since there is a tradeoff between developability and compactness, as expected, our algorithm generates charts with slightly longer boundaries. The difference in length is on average about 10% to 15%.

5.1. Soft toys

Figures 1 and 6 show examples of physical models we have created together with images of the original virtual models. For mechanical models, such as the fandisk and the bishop, the segmentation results in developable charts. Thus it is possible to reproduce accurately those models from paper (Figure 6) or from any other sheet material. For the toy models, distortion must inevitably be introduced. Nevertheless, the generated stuffed toy models capture the shape of the objects. Predictably, minor details are lost, mostly due to fabric resilience.

6. Conclusions and future work

Our work provides a simple and robust algorithm for quasi-developable mesh segmentation. As demonstrated by the examples, it segments meshes into compact developable charts. It is more suitable for texture atlas generation than previous segmentation techniques. As part of our work we introduced a new, simple to calculate, developability metric for mesh surfaces. We also presented an automatic method for fabric pattern design, an important problem which previous research did not seriously address.

An important consideration, particularly for machine based fabric cutting or stamping, is the symmetry of patterns. If several charts have the same shape (up to mirroring), this will make the cutting simpler. We plan to address this consideration in future research, using some of the available knowledge on shape similarity and symmetry detection.

Acknowledgments

We would like to thank the Graphite team for the software framework; Pedro V. Sander (ATI) and Kun Zhou (Microsoft Reseach Asia) for the comparison results; Stanford University for the Bunny data-set; and all the volunteers for sewing the soft toys. This project was partially funded by NSERC.

	Bunny	Horse	Feline	Gargoyle	Fandisk
#faces	69450	19996	41262	20000	9926
D-Charts					
Input #charts	12	15	25	12	6
Total time (sec.)	468	130	482	91	9
Lloyd (sec.)	417	115	446	73	3
Post (sec.)	51	15	36	17	6
Final #charts	10	12	25	10	4
$L_2^{Stretch}$	1.004	1.01	1.01	1.006	1
$L_\infty^{Stretch}$	1.429	2.315	3.488	1.645	1.017
L_2^{shear}	0.006	0.001	0.012	0.008	0
Iso-charts					
#charts	16	13	26	11	4
$L_2^{Stretch}$	1.023	1.035	1.052	1.019	1.021
$L_\infty^{Stretch}$	2.831	2.766	3.401	2.153	2.272
L_2^{shear}	0.021	0.038	0.056	0.022	0.018
MCGIM					
#charts		15	25	10	5
$L_2^{Stretch}$		1.014	1.018	1.009	1.008
$L_\infty^{Stretch}$		2.803	3.563	2.221	2.092
L_2^{shear}		0.014	0.019	0.011	0.012

Table 1: *Segmentation statistics. For all the models we used $F_{max} = 0.2$. When available we provided the statistics for models segmented with MCGIM and Iso-charts.*

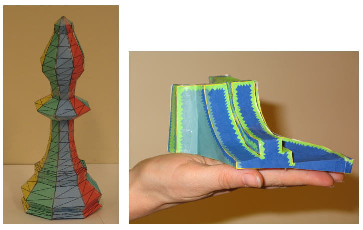

Figure 6: *Papercraft bishop and fandisk.*

References

[CLL*99] Chen H.-Y., Lee I.-K., Leopoldseder S., Pottmann H., Randrup T., Wallner J.: On surface approximation using developable surfaces. *Graphical Models and Image Processing 61*, 2 (1999), 110–124. 3

[CSAD04] Cohen-Steiner D., Alliez P., Desbrun M.: Variational shape approximation. *ACM Trans. Graph. 23*, 3 (2004), 905–914. 3, 5

[DMA02] Desbrun M., Meyer M., Alliez P.: Intrinsic parameterizations of surface meshes, 2002. 3

[Elb95] Elber G.: Model fabrication using surface layout projection. *Computer-Aided Design 27*, 4 (1995), 283–291. 3

[FH05] Floater M. S., Hormann K.: Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*, Dodgson N. A., Floater M. S.,, Sabin M. A., (Eds.), Mathematics and Visualization. Springer, Berlin, Heidelberg, 2005, pp. 157–186. 3

[GG04] Gelfand N., Guibas L. J.: Shape segmentation using local slippage analysis. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), ACM Press, pp. 214–223. 2

[GGH02] Gu X., Gortler S. J., Hoppe H.: Geometry images. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 355–361. 1, 2

[GK03] Guthe M., Klein R.: Automatic texture atlas generation from trimmed NURBS models. *Computer Graphics Forum 22*, 3 (2003), 453–453. 2

[Gra03] Graphite:, 2003. http://www.loria.fr/∼levy/Graphite/index.html. 7

[GVSS00] Guskov I., Vidimc;e K., Sweldens W., Schröder P.: Normal meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 95–102. 2

[GWH01] Garland M., Willmott A., Heckbert P. S.: Hierarchical face clustering on polygonal surfaces. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), ACM Press, pp. 49–58. 2

[HG00] Hormann K., Greiner G.: MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*, Laurent P.-J., Sablonnière P.,, Schumaker L. L., (Eds.), Innovations in Applied Mathematics. Vanderbilt University Press, Nashville, TN, 2000, pp. 153–162. 3

[IC01] Igarashi T., Cosgrove D.: Adaptive unwrapping for interactive texture painting. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), ACM Press, pp. 209–216. 2

[KG03] Kolmanic S., Guid N.: A new approach in cad system for designing shoes. *Journal of Computing and Information Technology 11*, 4 (2003). 3

[KL96] Krishnamurthy V., Levoy M.: Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM Press, pp. 313–324. 2

[KT03] Katz S., Tal A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22*, 3 (2003), 954–961. 2, 6

[Llo57] Lloyd S. P.: Least squares quantization in PCM. *Bell Telephone Laboratory Memorandum (1957), reprint*

IEEE Transactions on Information Theory IT-28, 2 (Mar 1982), 129-137 (1957). 3

[LP98] LEOPOLDSEDER S., POTTMANN H.: Approximation of developable surfaces with cone spline surfaces. *Computer-aided Design 30*, 7 (1998), 571–582. 3

[LPRM02] LÉVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 362–371. 2, 3, 7

[MHS99] MCCARTNEY J., HINDS B. K., SEOW B. L.: The flattening of triangulated surfaces incorporating darts and gussets. *Computer-Aided Design 31*, 4 (1999), 249–260. 2

[MS04] MITANI J., SUZUKI H.: Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Trans. Graph. 23*, 3 (2004), 259–263. 3

[MYV93] MAILLOT J., YAHIA H., VERROUST A.: Interactive texture mapping. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM Press, pp. 27–34. 2

[Pet04] PETERNELL M.: Recognition and reconstruction of developable surfaces from point clouds. In *Proceedings of 'Geometric Modeling and Processing 2004'* (2004), Elsevier Science, pp. 301–310. 3

[PS04] PETERNELL M., STEINER T.: Reconstruction of piecewise planar objects from point clouds. *Computer-aided Design 36* (2004), 333–342. 3

[PW99] POTTMANN H., WALLNER J.: Approximation algorithms for developable surfaces. *Comput. Aided Geom. Design 16* (1999), 539–556. 3

[SCOGL02] SORKINE O., COHEN-OR D., GOLDENTHAL R., LISCHINSKI D.: Bounded-distortion piecewise mesh parameterization. In *VIS '02: Proceedings of the conference on Visualization '02* (2002), IEEE Computer Society. 2, 3

[SdS01] SHEFFER A., DE STURLER E.: Parameterization of faceted surfaces for meshing using angle-based flattening. *Eng. Comput. (Lond.) 17*, 3 (2001), 326–337. 3

[SGSH02] SANDER P. V., GORTLER S. J., SNYDER J., HOPPE H.: Signal-specialized parametrization. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (2002), Eurographics Association, pp. 87–98. 3, 7

[SH02] SHEFFER A., HART J. C.: Seamster: inconspicuous low-distortion texture seam layout. In *VIS '02: Proceedings of the conference on Visualization '02* (2002), IEEE Computer Society, pp. 291–298. 2

[SLMB05] SHEFFER A., LEVY B., MOGILNITSKY M., BOGOMYAKOV A.: Abf++ : Fast and robust angle based flattening. *ACM Transactions on Graphics - In print* (Apr 2005). 7

[SSGH01] SANDER P. V., SNYDER J., GORTLER S. J., HOPPE H.: Texture mapping progressive meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 409–416. 2, 3, 7

[SWG*03] SANDER P. V., WOOD Z. J., GORTLER S. J., SNYDER J., HOPPE H.: Multi-chart geometry images. In *SGP '03: Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), Eurographics Association, pp. 146–155. 2, 3, 7

[WTY05] WANG C. C., TANG K., YEUNG B. M.: Freeform surface flattening based on fitting a woven mesh model. *Computer-Aided Design* (2005). 3

[ZSGS04] ZHOU K., SYNDER J., GUO B., SHUM H.-Y.: Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004), Eurographics Association. 2, 4, 7