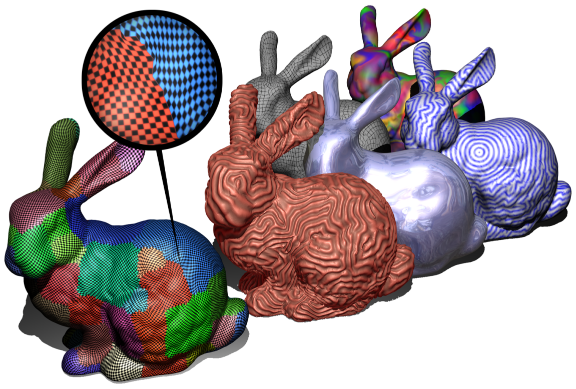
Volume xx (200y), Number z, pp. 1–10

Rectangular Multi-chart Geometry Images



We describe a new mesh clustering method that creates rectangular patches whose texels align across boundaries (above, inset) to conveniently support the implementation of various surface processing applications (above).

Abstract

Many mesh parameterization algorithms have focused on minimizing distortion and utilizing texture area, but few have addressed issues related to processing a signal on the mesh surface. We present an algorithm which partitions a mesh into rectangular charts while preserving a one-to-one texel correspondence across chart boundaries. This mapping permits any computation on the mesh surface which is typically carried out on a regular grid, and prevents seams by ensuring resolution continuity along the boundary. These features are also useful for traditional texture applications such as surface painting where continuity is important. Distortion is comparable to other parameterization schemes, and the rectangular charts yield efficient packing into a texture atlas. We apply this parameterization to texture synthesis, fluid simulation, mesh processing and storage, and locating geodesics.

1. Introduction

Many powerful surface processing operations can be expressed as the solution of partial differential equations (PDEs) on a surface, including feature-sensitive smoothing, reaction-diffusion texturing, texture synthesis, mesh editing, fluid flow, and geodesic tracing. Traditional approaches for solving surface PDEs rely on an irregular mesh over the surface or in space. Recently, geometry images [GGH02] have provided an efficient square domain that allows surface PDEs to be solved through simpler image operations.

The main problem with using geometry images to generate a domain for surface PDEs is that mapping the surface to a single rectangular chart incurs a large amount of distortion. High parametric distortion reduces the precision and efficiency of surface processing applications. Multi-chart geometry images [SWG*03] reduce distortion by cutting the mesh into multiple irregular pieces, each retessellated with a regular triangle mesh and packed into a single atlas. The added flexibility of multiple cuts reduces distortion, but irregular charts do not pack efficiently and require additional processing of the “topological sideband” to determine pixel neighbors across charts.

We present a new surface parameterization scheme that decomposes a triangle mesh into four-cornered quasi-rectangular clusters which map to rectangular charts in parameter space, and whose texels align across shared boundaries. This representation facilitates simple, efficient surface PDE solutions using image processing operations over a set of images with well-defined neighbors at their boundaries.

Many of these operations are easily implemented on the GPU for accelerated surface mesh processing.

A key feature of our approach is the use of *parameterization distance* to control cluster shape. This distance approximates geodesic distance within a cluster-centric coordinate system. We gain flexibility in chart selection by permitting T-junctions and self-neighboring charts, which helps minimize the distortion of the parameterization, as discussed in Sec. 6.

Section 2 compares our approach to other methods for parameterizing meshes and solving PDEs on a surface. Section 3 describes our algorithm, which extends iterative clustering to find developable charts that map naturally into rectangular regions. Section 7 demonstrates the utility of this new data structure with a variety of surface processing applications. Section 6 compares the distortion incurred by this representation to other options, establishing it as one of the most attractive choices for surface PDE processing. Section 8 discusses the limitations of this method, and ideas for its further improvement.

2. Previous Work

2.1. Surface Parameterization

Recent work has demonstrated the utility of decomposing meshes into a regular domain. Geometry images [GGH02] use a regular domain to implicitly encode the connectivity of a mesh, allowing standard image compression to be applied to mesh geometry. Carr and Hart[CH02] and Purnomo *et al.* [PCK04] show how square charts can be efficiently packed (and repacked [CH04]) to eliminate wasted texture space.

Polycube maps warp and project geometry on the quadrilateral faces of a manually-constructed cuberille exoskeleton [THCM04], whereas our approach constructs its charts automatically on meshes, even those containing small intricate features that would significantly challenge Polycube construction.

Purnomo *et al.* [PCK04] merged triangles based on coplanarity into clusters, straightened them into polygons [SSGH01], and used barycentric subdivision to divide them into quadrilaterals. Boier-Martin *et al.* [BMRJ04] constructed a centroidal Voronoi diagram on planar clusters that tended to yield hexagons which were divided and subdivided into a quad mesh. Both techniques yield semi-regular quadrangulations with no T-junctions, whereas we generate rectangular clusters on developable (not necessarily planar) surface segments with T-junctions.

Our cluster growth seeks to minimize distortion while constraining growth to a rectangular shape. Our cluster growth is an extension of Sorkine *et al.* [SCOGL02], which parameterized during cluster growth to prevent clusters from exceeding a distortion bound. We likewise parameterize during cluster growth but with additional cluster shape rules.

Iso-charts uses iso-map's geodesic metric extension to multidimensional scaling to grow large contiguous charts within a distortion bound [ZSGS04]. Alternatively such near-developable charts may be formed by iterative clustering based on fitting to a union of conics[JKS05]. Because we parameterize during cluster growth, we use parameterization distance as an approximation of geodesic distance to constrain chart growth to rectangular forms.

Others considerations of cluster shape include forcing their boundaries through regions of high curvature [LPRM02], or (additionally) lower average visibility [SH02]. Sander *et al.* [SSGH01] constrained greedy cluster growth that avoided crossing base domain boundaries, and straightened the boundaries of clusters as a post-process, whereas our greedy cluster growth straightens the sides of its rectangular clusters through the cluster growth rules.

Our proposed method is only guarantees C^0 continuity across charts, whereas techniques such as globally smooth parameterization [KLS03] produce a smoother parameterization. More recent work has extended this method to form globally smooth parameterizations containing nearly uniform sized square charts[RLL*05].

We use a modified chessboard metric to coerce k-means clustering to form rectangles, whereas Hausner [Hau01] used a modified manhattan metric to construct squarish clusters used to artistically tile planar image regions.

2.2. Surface PDEs.

Our goal of establishing a continuous mapping between a surface and an array of rectangular images is a novel contribution among a recent flurry of similar methods that use a parameterization as a basis for solving surface partial differential equations on a surface.

Bertalmio *et al.* [BSCO01] derive formulas for diffusion on a volumetric isosurface by lifting the dynamics to a voxel grid in the embedding space, and applied the results to texture filtering, reaction diffusion texturing and the visualization of the surface's principal curvature flow. Such Eulerian space-grid formulations are more accurate but consume too much space and work at a fixed resolution that ignores surface features.

Sibley and Taubin [ST04] implement diffusion across the irregular chart boundaries of an atlas, whereas the texels of our surface chart images are pre-aligned across chart boundaries by design to overcome these concerns.

Bajaj and Xu [BX03] implemented flow on a loop-subdivided triangulated surface to perform anisotropic (feature-sensitive) smoothing, and Stam [Sta03] implement flows on Catmull-Clark surfaces. They both use the metric tensor to reduce the effects of parametric distortion on flow, and overcome the effects of irregular valence through repeated subdivision about extraordinary points. While the metric tensor is designed to accommodate the distortion of length due to differences in element size, it does not completely eliminate these effects [Sta03]. We too depend on subdivision around extraordinary points, but our method automatically forms an evenly distributed base mesh for the subdivision.

Shi and Yu [**?**] also implemented flow on a mesh by representing fluid fields directly on the mesh topology. This couples fluid detail to tessellation. However, since their implicit Lagrangian technique traces velocity vectors on the mesh surface, it proves robust in the face of extraordinary vertices. Our method handles these points as a special case.

Lui *et al.* [LWC05] solved PDEs on a global conformal parameterization of a surface using the metric tensor to reduce the effects of distortion, with applications in fluid flow, segmentation, denoising and inpainting applied to the surface signal. Conformal parameterization produces quads that meet at right angles, but with wide variances in element size that focuses computation in arbitrary regions unrelated to shape or signal. In contrast, our proposed algorithm parameterizes surfaces in a curvature sensitive manner into uniformly-sized quadrilaterals that reduce the effects of element-size variation on the underlying PDE solution, and distribute computation evenly across the surface.

3. Algorithm Overview

Our algorithm is inspired by the greedy flattening bounded-distortion parameterization algorithm of Sorkine *et al.* [SCOGL02] and the iterative k-means mesh clustering algorithm used by Sander *et al.* [SWG*03], Schlafman *et al.* [STK02], Cohen-Steiner *et al.* [CSAD04], and more recently [JKS05]. An overview of our approach is as follows:

```
Phase I: Iterative Face Clustering (Sec. 4 )
    1. Select an initial set of seed faces to form clusters
    2. Grow clusters outward from seed faces
    3. While large gaps of unassigned triangles remain
        Re-orient each cluster along its local parameter axes
        Regrow clusters outward from seed faces
        Insert a new seed face
    4. Add remaining faces to nearest cluster
Phase II: Chart Parameterization (Sec. 5 )
    5. Determine chart boundaries
    6. Solve each chart's local parameterization
```

Figure 5 illustrates this process. Phase I uses iterative clustering and greedy flattening to determine clusters of faces that parameterize with low distortion into roughly rectangular regions. Phase II forces each cluster into a rectangular region of texture space such that edges shared across chart boundaries sample the same number of texels in both charts. The resulting parameterization provides a seamless set of rectangular domains useful for rendering, storage, and computation.

4. Rectangular Cluster Formation

Standard methods for iterative mesh clustering begin with an initial collection of disjoint seed faces and attach remaining faces to the cluster whose seed face is *closest*, according to a given metric (often a combination of Euclidean distance and orientation). The triangles containing the centroids of these clusters then become the seed faces for the next iteration, and the process repeats until it converges, i.e., when cluster centers stabilize.

With a goal of rectangular shaped clusters, our approach differs from this standard algorithm in several ways. First, we parameterize new triangles during cluster growth to predict and track chart distortion, and evaluate additional rules to avoid jagged boundaries and holes in the cluster. Second, the "distance" metric uses a different coordinate frame for each cluster, and is approximated using the distance in the parameterization domain. Third, even though we use a chessboard metric to encourage rectangular cluster shapes, we can still get charts extending away from the original axes, so we select a new reference frame at the end of each iteration.

4.1. Parameterization Distance

To constrain chart growth to a rectangular shape, we must estimate geodesic distances on the meshed surface. We can take the same parameterization used during cluster growth to bound chart distortion, and use it to approximate geodesic distance. Distances in this domain correspond to geodesic distance altered by the path integral of the parameteric distortion.

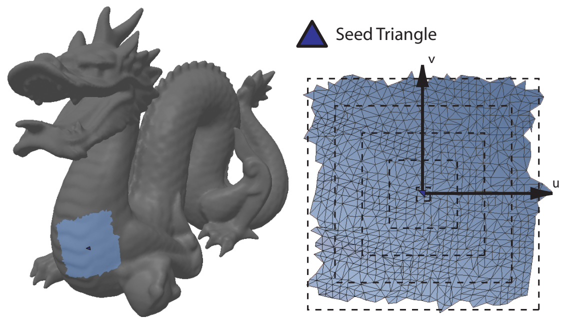


Figure 1: *An isolated cluster grown under the chessboard metric forms a square in the parameterization domain.*

We encourage rectangular clusters by changing our metric from the usual Euclidean L^2 distance that would form round clusters to a chessboard L^∞ distance whose equidistant "circles" are squares, as illustrated in Fig. 1. We further generalize to *rotated* rectangular clusters by formulating the oriented, anisotropic metric $L^\infty_{a,\theta}$ such that magnitude is measured

$$||u,v||^{\infty}_{a,\theta} = \max\left(\frac{|u\cos\theta - v\sin\theta|}{a_u}, \frac{|u\sin\theta + v\cos\theta|}{a_v}\right) \tag{1}$$

for aspect ratio a and orientation θ.

Since each cluster has its own unique distance metric, the notion of distance does not extend globally across the mesh. This certainly defies the definition of a distance metric and somewhat confounds the direct assignment of triangles to their "nearest" cluster, but still assigns triangles to the chart which is in some sense the best.

4.2. Frontier Face Parameterization

Frontier faces lie just outside of a cluster, sharing at least one edge with the cluster's boundary. Before we can include a frontier face to a cluster, it must be flattened into the parameterization domain so we can evaluate the four inclusion criteria: (1) its parameteric distance to the cluster center, (2) the amount of distortion it would contribute to the cluster, (3) its potential for fold-over, and (4) its effect on boundary smoothness. Face parameterization depends on the number of edges shared with the cluster boundary:

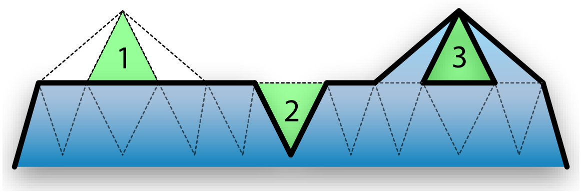


Figure 2: *Three cases of triangles adjacent to the frontier on the corresponding number of its edges.*

Case 1: One shared edge. The two vertices of the shared edge inherit the cluster's parameterization coordinates, but the third vertex is assigned new coordinates. Coordinates can be assigned by a rigid unfolding of the triangle about its shared-edge hinge, but a simple look ahead during flattening can reduce distortion and lead to larger, more efficient clusters. Following Sorkine *et al.* [SCOGL02], we rigidly unfold any frontier faces that share the same free vertex, and assign to the free vertex the average of the resulting parameterization coordinates for it.

Case 2: Two shared edges. All three vertices inherit the parameterization coordinates from the cluster boundary. If the frontier face fails the inclusion criterion, then it is re-evaluated as a Case-1 frontier face with respect to the edge it shares with the most recently added cluster face, while its second shared edge potentially forms a seam.

Case 3: Three shared edges. An annulus has formed and the frontier face is filling it. The triangle is evaluated as in Case-2 with respect to the two edges it shares with the most recently added contour faces, and the third edge shared with an uncooperative contour face becomes a seam.

The frequency of these Case-3 situations is reduced by biasing the order of cluster growth. When a Case-2 frontier face satisfies the cluster candidacy requirements, it is immediately added to the cluster, bypassing the priority queue usually used to include the best candidates first. This order bias also helps to smooth the cluster boundary.

4.3. Candidacy Criteria

Once a frontier face has been parameterized, the following criteria are evaluated.

Parameterization Distance to Seed Face. We measure distance to the cluster center by evaluating (1) using parameters a, θ of the cluster's coordinate frame. Note that the same triangle may be a frontier face of multiple clusters, but should be added to the closest one.

Incurred Distortion. Numerous parameterization distortion metrics exist, e.g. [SSGH01, SdS00, ZMT05], and nothing in our algorithm precludes the use of any of these. We use the metric of [SCOGL02] which penalizes both stretch and shrinkage for parameterized triangles, and is minimized when the parameterization is perfectly isometric.

Foldover Prevention. To test the parameterization of the frontier face for intersection with the existing cluster parameterization, it is sufficient to test the frontier face's edges against the parameterization of the cluster boundary. We maintain a quad-tree of parameterized boundary edges to hasten intersection queries.

Boundary Smoothness. Since clusters are mapped to rectangular charts, boundary smoothness is very important. We ensure smoothness through the use of boundary constraints. The *final boundary* of a cluster is the subset of its boundary edges which border either another cluster or a frontier face which failed the candidacy test. Given a midpoint m of a final boundary edge, let n denote the nearest position along the cluster's coordinate axes (which are oriented and stretched by θ, a) for which (1) yields the same distance. (For the midpoint m, the vector n is an outward "normal" quantized to be perpendicular to the desired cluster boundary shape.) We exclude from consideration any frontier face whose centroid makes an incidence angle less than $\alpha \approx 80^\circ$ with respect to any final boundary edge midpoint and its associated outward quantized normal.

Since our distance metric is evaluated at triangle centroids, saw-toothed boundaries between patches are prevalent. We prevent sawtooth boundaries by ignoring distance for Case-2 frontier faces, instead relying on the other three criteria for cluster inclusion. In practice this virtually eliminates boundary irregularities.

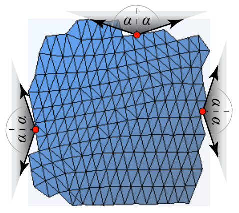


Figure 3: *Constraint regions ensure smooth cluster edges in the parameter domain.*

4.4. Cluster Re-alignment

After each patch growth iteration, clusters may not be rectangular in shape or even centered in their local parameter domain. To guide the patch shape toward its rectangular goal, we perform two refinement operations. Fig. 4 shows the operations of the centering and re-orientation process.

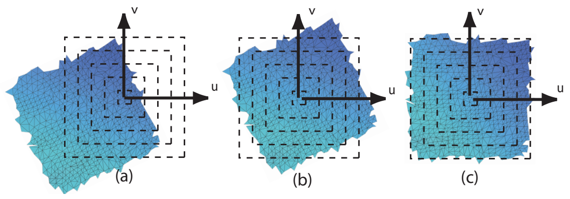


Figure 4: *Patch centering and reorienting process: (a) before, (b) re-centering, (c) reorientation.*

Patch Centering. To center a patch, we offset its coordinates by its (negated) center of mass, computed as follows:

$$\mu = \sum_{t\in T}(A_t \mathbf{c}_t) / \sum_{t\in T} A_t \qquad (2)$$

where T is the set of triangles in the patch, A_t is the area of triangle $t \in T$ in the parameter domain and $\mathbf{c}_t$ is the location of its barycenter.

Patch Re-orientation and Size Estimation. We reorient the patch to better align it with the target rectangular shape. Similar to Gottschalk [GLM96], we use principle component analysis to perform area integrals over the triangles in the patch. We express the texture coordinates of a triangle t using barycentric coordinates $\mathbf{u}^t(\alpha,\beta) = \mathbf{u}_0^t\alpha + \mathbf{u}_1^t\beta + \mathbf{u}_2^t(1-\alpha-\beta)$, where $\mathbf{u}_0^t$, $\mathbf{u}_1^t$, and $\mathbf{u}_1^t$ are texture coordinates of triangle t. The symmetric 2×2 covariance matrix C is given by

$$C_{ij} = \sum_{t\in T}\int_0^1\int_0^1 (\mathbf{u}_i^t - \mu_i)(\mathbf{u}_j^t - \mu_j)A_t d\alpha d\beta \qquad (3)$$

(where $\mathbf{u}_i$ denotes the ith coordinate of $\mathbf{u}$) and has a closed-form expression. The eigenvectors of C define the new coordinate frame. Additionally, by comparing the ratio of the eigenvalues of C, we can compute the target size a for the of the patch to guide the L_a^∞ metric in the next iteration.

4.5. Seed Faces

The clustering process involves the introduction of new seed faces both at the start of algorithm and between growth process iterations. Careful choice of seed face locations can improve the resulting cluster quality and convergence properties of our algorithm. Ideally a new seed face should be placed as far way from mesh boundaries, high curvature regions, and existing cluster boundaries as possible, to maximize the size of its potential cluster.

To do this, we grow an advancing front starting with faces that are adjacent to feature regions (i.e. mesh boundaries, high curvature, or an existing cluster). The front moves over only the faces currently unassigned to any cluster. The last face reached by this front becomes the new seed face to start a new cluster. Our implementation for finding such a face follows the algorithmic technique described in [SWG*03] for finding new cluster centers, where distance between faces is measure by the Euclidean distance between their centroids.

Once a new seed face has been found, we parameterize it with its centroid sitting on the origin of its own local parameter domain. Initially we do not know the appropriate aspect scale for this new cluster, so we cannot immediately introduce it into the iterative clustering process. To solve this problem, we perform an outward growth around this seed while leaving existing clusters in place. This growth is performed with a cluster target size of $a = (1,1)$. Following this growth the cluster is re-centered and re-oriented, and its aspect scale a is updated. At this point the new cluster is ready to take part in the iterative growth process.

4.6. Termination

Though we provide no proof of the convergence, we do report that the method tends to settle and approaches a good solution once enough seed faces have been added. In our current implementation we continue adding seed faces until the the gaps between patches becomes small. This process could be easily automated, however, we currently do this by visual inspection.

Due to the strictness of the cluster smoothness criteria, some faces remain unassigned to any cluster as shown in the second figure of Fig. 5. We assign these remaining faces to their nearest clusters in a final outward cluster growth phase, ignoring the distortion, smoothness and even fold-over criteria.

5. Chart Parameterization

The goal of our final patch parameterization is to force each patch's parameter domain into a rectangle while achieving

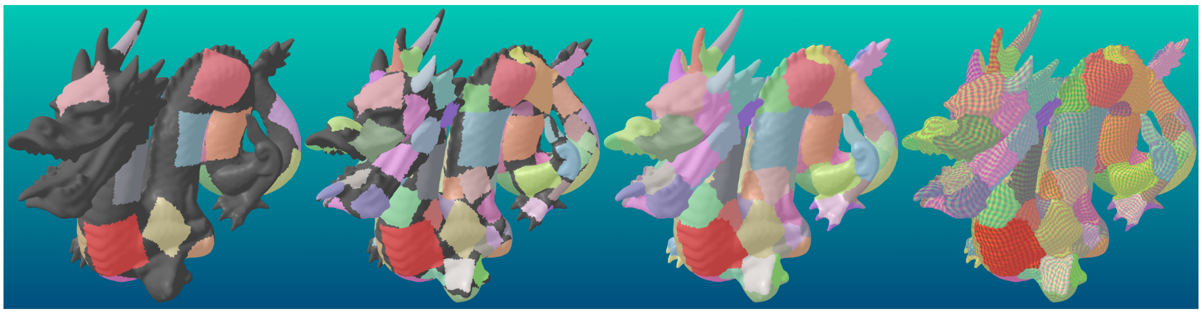


Figure 5: *Dragon model undergoing the iterative clustering process followed by final mapping.*

perfect texel continuity between patches. The latter condition makes our method a *texture resolution dependent* parameterization; texture coordinate assignment is done with knowledge of the underlying texel grid. While our final parameterization computes a specific texture map resolution for each patch, the parameterization supports continuity for texture map resolutions that are successive powers of two larger.

5.1. Chart Boundaries and Fixed Vertices

To achieve perfect texel continuity between patches, careful attention must be paid to the shared boundary segments occurring within and between charts. Each shared boundary segment must be assigned a discrete texel length, and the *fixed* end-points of the boundary segments must be assigned texel coordinates that lie on texel boundaries.

Two different types of vertices may start or end a boundary segment. A *fixed* vertex is any vertex that resides on the start of a seam where one to many patches come together. A *corner* vertex is any vertex that has been chosen to be placed at one of the four corners of a rectangular parameter domain, which may also be at a fixed vertex location. At the end of the clustering phase, the fixed vertices are determined by the layout of the patches, and the corner vertices may be freely chosen. Figure 6 shows the possible cases for fixed and corner vertices and the resulting boundary segments.

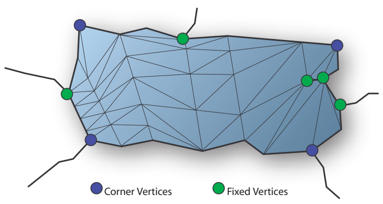


Figure 6: *Fixed and corner vertices for a patch.*

5.2. Patch Size and Corner Determination

For each patch, we walk the boundary in counter-clockwise order and collect fixed and corner vertices into an ordered list ℓ. Corner vertices at this point arise from adjacent patches that have already been visited and thus have their corners assigned.

Discrete texel lengths are assigned to each of the boundary segments. The texel length for any boundary segment shared with an already visited patch is set to be equal to the length of the corresponding segment in the adjacent patch. The remaining boundary segment lengths are directly computed by multiplying the 3D arc length of the segment by a user specified parameter γ to convert to texel length. This value is then rounded up or down to the closest non-zero texel length divisible by two. The last patch visited on a closed mesh has all of its boundary segment lengths determined. Requiring that all segments between fixed vertices are of even texel length ensures that the texel perimeter p of every patch is even and can therefore be mapped to a rectangular texel grid. If the user specifies a γ value too low, it is possible to have a patch with a boundary length of two, which cannot be mapped to a rectangle. In such cases, we repeat the process with a larger γ so that all patches have the necessary resolution.

Corner vertices for the patch are chosen by first selecting a vertex v_{ll} on the boundary of the patch to be the lower left corner of the parameterization. We select v_{ll} by computing the bounding box for the already flattened patch, and choose the vertex closest to any of the bounding box corners. The ordered list ℓ is traversed to find the segment that contains v_{ll}. If v_{ll} is not already in ℓ, then we insert the vertex, which divides a boundary segment into two pieces. To compute the texel distance of v_{ll} along the divided segment, we use compute v_{ll}'s fractional arc length along the segments and snap it to the closest integer texel location.

To insert the remaining three corner vertices into the ordered list, we start by computing an appropriate texel width and height for the patch. The bounding box aspect ratio α and the texel perimeter p allows us to compute the texel width w of our rectangular patch: $w = \lfloor p/(2.0 + 2.0 * \alpha) \rfloor$.

We can now search for a vertex v_{lr} to assign to the lower right corner. This vertex must be placed at w units along the boundary in counter clock-wise order from vertex v_{ll}. We search the ℓ for a vertex that is w units away from v_{ll}. If one is not found we search the containing segment for a vertex that closest matches this distance and then insert it into ℓ, making it exactly w units away from v_{ll}. The remaining two corner vertices are found in a similar manner. In rare cases, no free vertex exists within a segment needing to be subdivided. In such cases we insert a new vertex into the mesh, splitting the two adjacent triangles into four.

5.3. Patch Parameterization and Crossing Edges

After computing the texel size, boundary segment lengths, and corner positions for every patch we can parameterize each patch into a rectangle. The boundary segment lengths and corner vertices determine the location of the boundary of each patch in its local parameter domain. Vertices interior to the patch are solved using a linear spring model[Flo97] which guarantees a valid embedding of the mesh.

Forcing patches into square parameter domain regions requires special handling to avoid degenerate triangles from forming in the parameter domain. We refer to any edge that is not on the boundary of the patch, but whose end vertices sit on the boundary of the patch as a *crossing edge*. Crossing edges form parameter domain degeneracies if the vertices of this edge are mapped to the same edge of the rectangular parameter boundary.

One solution to this problem is to carefully assign patch boundary vertices in such a way that the crossing edges do not result in degeneracies. In general, this is impossible. Furthermore, the location of the crossing edges may require choosing an edge assignment that results in high distortion effectively undoing the work of forming rectangular target patches. To avoid this problem altogether, we perform local re-meshing of any crossing edge that leads to a degeneracy by inserting a new vertex along the edge.

5.4. Parameterization Optimization

To improve the parameterization, we apply non-linear optimization using the stretch metric from [SSGH01]. Our current optimization strategy uses the fast relaxation scheme proposed by [SY04]. Stretch error is then further minimized by visiting vertices one at a time, optimizing their location in their one-ring by performing a conjugate gradient descent. Non-fixed vertices on the border between patches may be optimized with one degree of freedom. This may be accomplished by transforming the two pieces of the one-ring straddling the boundary into a consistent space, applying the optimization, and transforming each piece of the ring back to their local coordinate system.

Param. Method	Quality Metric	Models			
		Gargoyle	Horse	Dragon	Feline
TMPM	SE		80.0%		
	PE		70.0%		
	TE		56.0%		
GI	TE	67.8%	32.4%	42.4%	33.3%
MCGI	SE	98.7%	99.2%	92.7%	99.1%
	PE	72.7%	75.6%	73.1%	75.6%
	TE	71.8%	75.0%	67.8%	74.9%
RMCGI	SE	72.7%	75.0%	64.8%	67.6%
	PE	83.6%	81.7%	82.9%	80.1%
	TE	60.9%	61.3%	53.7%	54.4%

Table 1: *Comparisons of efficiency metrics: stretch (SE), packing (PE) and texture (TE = SE×PE), for Texture Mapping Progressive Meshes (TMPM), Geometry Images (GI), Multi-chart Geometry Images (MCGI) and our Rectangular Multi-chart Images (RMCGI). PE for RMCGI was measured for packing into a single texture, whereas for per-chart texture maps TE = SE.*

Model	Tris Charts	Cluster Time/Rate	Param. Time/Rate	Packing Time/Rate
Horse	97K	1m:22s	2m:15s	4.23s
	68	1190 ∆/s	720.2 ∆/s	16.08 □/s
Gargoyle	200K	6m:35s	5m:30s	5.76s
	128	505.7 ∆/s	606.3 ∆/s	22.22 □/s
Feline	100K	3m:58s	1m:39s	4.78s
	120	419.5 ∆/s	1011 ∆/s	25.10 □/s
Dragon	150K	6m:30s	3m:46s	7.98s
	164	384.7 ∆/s	663.8 ∆/s	20.55 □/s
Jerry	94860	1m:30s	1m:50s	3.11s
	64	1060 ∆/s	863.5 ∆/s	20.58 □/s
Bunny	69451	1m:31s	1m:24s	3.69s
	64	765.6 ∆/s	822.7 ∆/s	17.34 □/s

Table 2: *Clustering, parameterization, and packing performance of our algorithm for a variety of models.*

6. Results and Discussion

Distortion Forcing each patch into a perfect rectangle requires additional distortion when compared with algorithms which permit natural chart boundaries. However, increased distortion is balanced well by the high packing efficiency of rectangular charts.

Table 1 measures and compares the distortion of our rectangular multi-chart geometry images (RMCGI) using the efficiency metrics of Sander *et al.* [SSGH01] (including *stretch efficiency:* the area weighted average of the inverted stretch of each surface patch). RMCGI's mapping of each quasi-rectangular cluster into a perfectly rectangular parameterization incurs additional distortion when compared to algorithms which allow natural chart boundaries (such as multi-chart geometry images). On the other hand, the packing efficiency of these rectangular charts is quite high, and

Param. Method	CPU	Param. Rate	Packing Rate
BDPMP (fast)	1.0 GHz P3	7833 Δ/s	N/A
BDPMP (relax)	1.0 GHz P3	657 Δ/s	N/A
FBSPTM	2.4 GHz PC	17 Δ/s	N/A
GI	N/A (2002)	19 Δ/s	N/A
Iso-Charts	3.0 GHz Xeon	1576 Δ/s	1897 c/s
LSCM	1.3 GHz Pentium	643 Δ/s	96,968 c/s
MCGI	N/A (2003)	58 Δ/s	19 c/s
RMCGI	2.0 GHz Athlon	297 Δ/s	22,923 c/s

Table 3: *Comparisons of parameterization and packing performance (in elements per second), for Bounded-distortion Piecewise Mesh Parameterization (fast and relaxation methods), Feature-based Surface Parameterization and Texture Mapping, Geometry Images, Iso-Charts, Least Squares Conformal Mapping, Multi-chart Geometry Images and our Rectangular Multi-chart Geometry Images.*

PE=100% if each chart is assigned its own texture map, which results in competitive overall texture efficiency.

We grow surface clusters using a stretch bound of 2.0. Fig. 7 shows that the distortion of RMCGI patches peaks near their boundaries, due in part to the final addition of straggler triangles to the rectangular clusters.

Performance Table 2 decomposes the RMCGI execution time per task and per element. The performance of the shape-constrained k-means clustering algorithm is influenced by a complicated combination of mesh size and shape. Parameterization is more clearly a function of the number of charts, modulated by the complexity of the cluster boundaries which in turn is affected by curvature distribution and tesselation frequency. We pack charts into a single texture atlas using the CompaSS rectangular packing software [CM04]. Packing rectangles is generally easier and more efficient than other shapes.

Table 3 compares RMCGI's performance to the median per-triangle parameterization and per-chart packing rates reported for other approaches. While such averages do not compare algorithmic complexity, they do show RMCGI runs slower than pure parameterization methods but faster than other regular retessellations that produce geometry-image structures.

Limitations On low-polygon meshes with irregular tessellation, our method produces jagged boundaries, due to a limited number of available decompositions. One might implement a triangle splitting scheme to assist smoothing of chart boundaries, which could also reduce parametric distortion. Continuity near boundaries could also be improved by a computing per-texel metric from a smooth surface corresponding to the mesh. Figure 7 illustrates minor artifacts near a stretch discontinuity while simulating reaction-diffusion, though these kinds of artifacts are not universally apparent (e.g., 9). Metric discontinuity along chart boundaries is not unique to our approach.

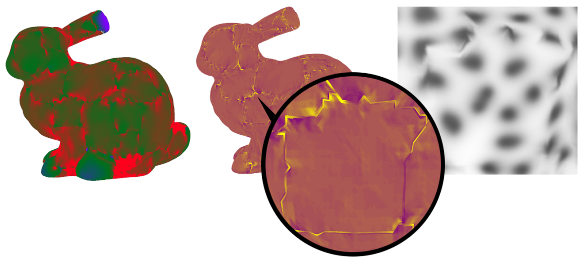


Figure 7: *Parametric stretch is plotted over the bunny (upper-left); red/blue indicate over/under-sampling. Its gradient (right) reveals distortion discontinuities at patch boundaries that can interfere with some PDE processes if special care is not taken to account for distortion.*

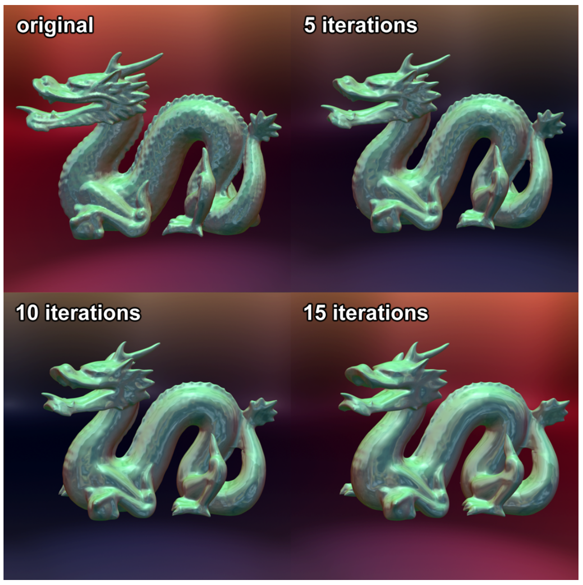


Figure 8: *The Dragon after various iterations of feature-preserving smoothing implemented as a bilateral chart image filter. Smoothing for a 1.4M triangle Dragon was performed at about 7 iterations per second in graphics hardware (GeForce 7800).*

Because our charts are rectangular, we could achieve higher memory efficiency by storing each chart in its own texture, avoiding the wasted texture memory due to gaps between charts in the texture atlas. Current graphics cards only allow sixteen textures to be bound simultaneously, but we expect future hardware will lift this restriction.

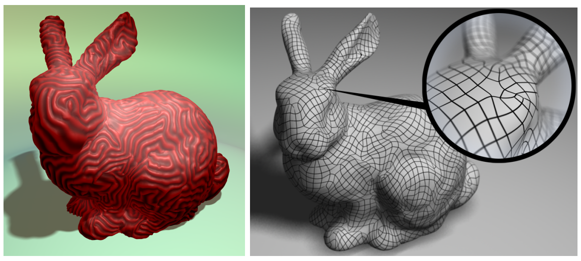


Figure 9: *Reaction diffusion texture synthesis (left) and Catmull-Clark subdivision (right) over the surface of the Stanford bunny.*

7. Applications

We have implemented several surface processing applications, using a GPU's fragment pipelines to convolve data stored in rectangular regions of a single texture. To ensure well-defined neighbors at patch boundaries, we copy a border of texels from neighboring patches between each iteration. Vector data is reoriented into the local parameteric basis with a simple 2x2 matrix multiply. Handling of texels near extraordinary vertices varies depending on the application.

Real-Time Fluid Flow on Surfaces. Our final application solves on the surface the incompressible Navier-Stokes equations for fluid flow. The implementation is based on Stam's stable fluid method [Sta99] and the GPU implementation detailed in [Har04]. Figure 11 shows our results. We achieved 20 fps on a simulation grid with 206,632 cells. Remaining distortion could be corrected using operators in curvilinear coordinates [Sta03].

Texture Synthesis. A seamless parameterization is useful for synthesizing textures which are continuous over the surface. We generate wrinkles on the Stanford Bunny (figure 9) using the method of Witkin & Kass [WK91]. While the metric tensor corrects for element area and shape, we find distortion nevertheless affects the result further motivating our efforts to generate low-distortion chart images.

Geodesics. Solving the Eikonal PDE $|\nabla T| = 1$ after setting $T = 0$ at a single source vertex yields a distance field over the surface, shown in Figure 10. Geodesic paths from any vertex to this source vertex can then be found by a simple downhill flow on T. The fast marching method [Set96] gives an approximate solution to the Eikonal equation on a regular grid. RMCGIs enable a variation on the fast marching method that better preserves the circular shape of the front over the mesh [NK02] using a fragment shader kernel operating on triangles constructed from nearby fragments. Extraordinary vertices confound this approach, so their distance values are simply averaged from their neighbors.

Quad Meshes. Geometry images [GGH02, SWG*03] implicitly encode a regularly tessellated mesh in a texture by storing one vertex position in each texel and constructing edges between adjacent vertices. Figure 9 shows a quad mesh reconstructed from a rectangular-patched representation of the bunny and rendered as a Catmull-Clark surface. The closeup shows how we reconstruct geometry near extraordinary vertices in the patch arrangement: each texel adjacent to an extraordinary vertex becomes a vertex of an n-gon. (These n-gons can be further tessellated into quads and triangles if desired.) Note that the set of quads we form here is unrelated to the arrangement of rectangular charts formed by the atlas, which itself may contain T-junctions.

Smoothing. We implement feature-preserving smoothing with bilateral filtering [TM98]. Seamless smoothing along chart boundaries is handled by the boundary copy, yielding simpler implementation compared to other parameter-space smoothing methods [ST04]. Extraordinary vertices are handled by averaging among the same texels used to construct the n-gon in quad mesh reconstruction. The 1.4M triangle Dragon in figure 8 was smoothed at a rate of approximately seven smoothing iterations per second.

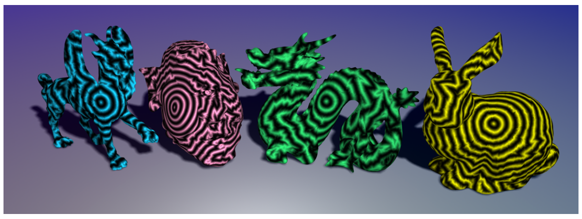


Figure 10: *Isocontours of approximately equal geodesic distance, computed with a regular kernel in a fragment shader using the fast marching method. (Left to right: Feline, Jerry, Dragon, Bunny.)*

8. Conclusion

We have demonstrated a novel method for generating low distortion rectangular charts on manifold surfaces. These charts provide good utilization of graphics hardware designed for processing with texture images, and the low distortion mapping provides a vehicle for graphics hardware to process a surface signal. We have demonstrated the usefulness of this map with the implementations of various surface PDE applications as image operators on the rectangular chart images. Such operations would incur greater distortion artifacts if performed on a single geometry image [GGH02] and would be more difficult to implement correctly and efficiently on the non-square charts of multi-chart geometry images [SWG*03]. Compromises of C^0 continuity and somewhat increased storage are thus justified by the efficiency and simplicity of the surface chart image data structure, resulting in a tool for added effects and realism in GPU-accelerated graphics.

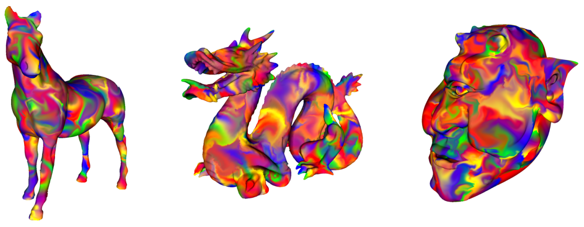


Figure 11: *Results of user interaction with real-time fluid flow evaluated over chart images by the GPU.*

References

[BMRJ04] Boier-Martin I., Rushmeier H., Jin J.: Parameterization of triangle meshes over quadrilateral domains. In *Proc. Symp. on Geom. Proc.* (2004), pp. 197–208.

[BSCO01] Bertalmio M., Sapiro G., Cheng L.-T., Osher S.: Variational problems and pdes on implicit surfaces. In *Proc. Workshop on Variational and Level Set Methods in Computer Vision* (2001), pp. 186–193.

[BX03] Bajaj C. L., Xu G.: Anisotropic diffusion of surfaces and functions on surfaces. *TOG 22* (2003), 4–32.

[CH02] Carr N. A., Hart J. C.: Meshed atlases for real-time procedural solid texturing. *TOG 21*, 2 (2002), 106–131.

[CH04] Carr N. A., Hart J. C.: Painting detail. *TOG 23*, 3 (Aug. 2004). (Proc. SIGGRAPH).

[CM04] Chan H. H., Markov I. L.: Practical slicing and non-slicing block-packing without simulated annealing. In *Proc. GLSVLSI* (2004), pp. 282–287.

[CSAD04] Cohen-Steiner D., Alliez P., Desbrun M.: Variational shape approximation. *TOG 23*, 3 (2004). (Proc. SIGGRAPH).

[Flo97] Floater M. S.: Parametrization and smooth approximation of surface triangulations. *CAGD 14*, 4 (1997), 231–250.

[GGH02] Gu X., Gortler S. J., Hoppe H.: Geometry images. In *Proc. SIGGRAPH 2002* (2002), pp. 355–361.

[GLM96] Gottschalk S., Lin M. C., Manocha D.: OBBTree: A hierarchical structure for rapid interference detection. In *Proc. SIGGRAPH 96* (1996), pp. 171–180.

[Har04] Harris M.: *Fast Fluid Dynamics Simulation on the GPU*. Addison-Wesley, Mar. 2004, pp. 637–665.

[Hau01] Hausner A.: Simulating decorative mosaics. In *Proc. SIGGRAPH* (2001), pp. 573–580.

[JKS05] Julius D., Kraevoy V., Sheffer A.: D-charts: Quasi-developable mesh segmentation. In *Computer Graphics Forum, Proceedings of Eurographics 2005* (Dublin, Ireland, 2005), vol. 24, Eurographics, Blackwell, pp. 581–590.

[KLS03] Khodakovsky A., Litke N., Schröder P.: Globally smooth parameterizations with low distortion. *(Proc. SIGGRAPH) TOG 22*, 3 (2003), 350–357.

[LPRM02] Levy B., Petitjean S., Ray N., Maillot J.: Least squares conformal maps for automatic texture atlas generation. In *Proc. SIGGRAPH 2002* (2002), pp. 362–371.

[LWC05] Lui L., Wang Y., Chan T. F.: Solving pdes on manifold using global conformal parameterization. In *Proc. Variational, Geometric, and Level Set Methods in Computer Vision* (2005), pp. 307–319.

[NK02] Novotni M., Klein R.: Computing geodesic paths on triangle meshes. In *Proc. WSCG 02* (2002), pp. 341–348.

[PCK04] Purnomo B., Cohen J. D., Kumar S.: Seamless texture atlases. In *Proc. Symp. Geom. Proc.* (2004), pp. 67–76.

[RLL*05] Ray N., Li W. C., Levy B., Sheffer A., Alliez P.: Periodic global parameterization. In *Tech report* (2005).

[SCOGL02] Sorkine O., Cohen-Or D., Goldenthal R., Lischinski D.: Bounded-distortion piecewise mesh parameterization. *IEEE Visualization* (2002), 355–362.

[SdS00] Sheffer A., de Sturler E.: Surface parameterization for meshing by triangulation flattening. *Proc. Meshing Roundtable* (2000), 161–172.

[Set96] Sethian J.: A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci 93*, 4 (1996), 1591–1595.

[SH02] Sheffer A., Hart J. C.: Seamster: Inconspicuous low-distortion texture seam layout. *Proc. Visualization 2002* (2002), 291–298.

[SSGH01] Sander P. V., Snyder J., Gortler S. J., Hoppe H.: Texture mapping progressive meshes. In *Proc. SIGGRAPH 2001* (2001), pp. 409–416.

[ST04] Sibley P. G., Taubin G.: Atlas-aware laplacian smoothing. In *Poster: Proc. Visualization* (2004), p. 598.27.

[Sta99] Stam J.: Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 121–128.

[Sta03] Stam J.: Flows on surfaces of arbitrary topology. *(Proc. SIGGRAPH) TOG 22*, 3 (2003), 724–731.

[STK02] Shlafman S., Tal A., Katz S.: Metamorphosis of polyhedral surfaces using decomposition. *Computer Graphics Forum 21*, 3 (Sept. 2002).

[SWG*03] Sander P. V., Wood Z. J., Gortler S. J., Snyder J., Hoppe H.: Multi-chart geometry images. In *Proc. Sym. on Geom. Proc.* (2003), pp. 146–155.

[SY04a] Shi L., Yu Y.: Inviscid and incompressible fluid simulation on triangle meshes. In *Computer Animation and Virtual Worlds* (2004), pp. 173–181.

[SY04b] Shin Yoshizawa Alexander G. Belyaev H.-P. S.: A fast and simple stretch-minimizing mesh parameterization. In *Shape Modeling International* (2004), IEEE Computer Society, pp. 200–208.

[THCM04] Tarini M., Hormann K., Cignoni P., Montani C.: Polycube-maps. *TOG 23*, 3 (2004), 853–860.

[TM98] Tomasi C., Manduchi R.: Bilateral filtering for gray and color images. In *Proc. ICCV* (1998), p. 839.

[WK91] Witkin A., Kass M.: Reaction-diffusion textures. *(Proc. SIGGRAPH 91) Computer Graphics 25*, 3 (1991).

[ZMT05] Zhang E., Mischaikow K., Turk G.: Feature-based surface parameterization and texture mapping. *TOG 24*, 1 (2005), 1–27.

[ZSGS04] ZHOU K., SNYDER J., GUO B., SHUM H.-Y.: Isocharts: Stretch-driven mesh parametrization using spectral analysis. In *Proc. Symp. Geom. Proc.* (2004).