Continuity Mapping for Multi-Chart Textures

Francisco González* Gustavo Patow†
Geometry and Graphics Group, Universitat de Girona, Spain

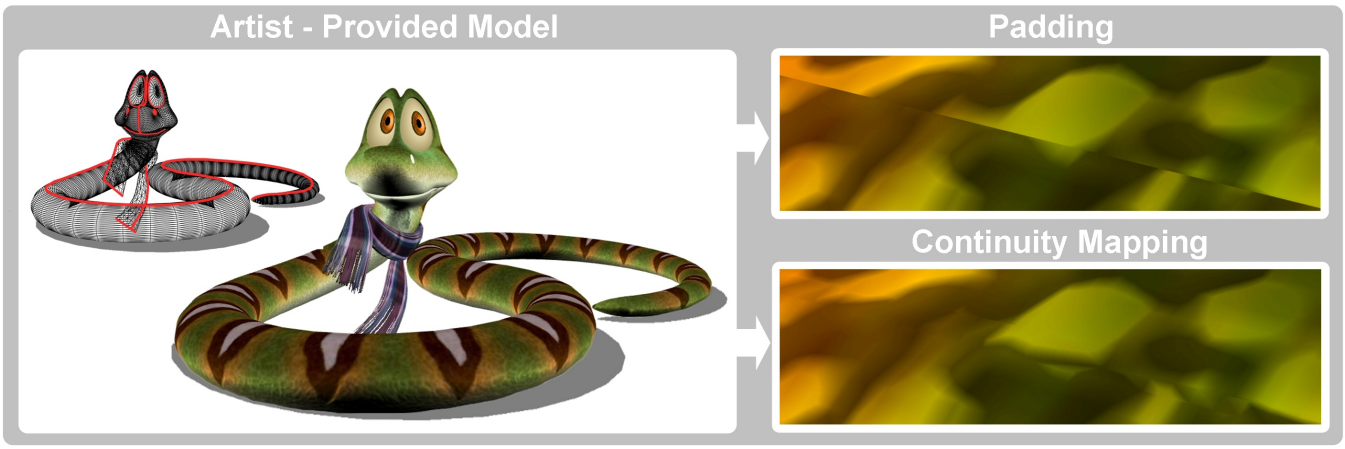

Figure 1: *Filtering with* Continuity Mapping *(Snake from www.daz3d.com, 25448 triangles, atlas* 700×3200*). In spite of the artist's fine-tuning, seams are visible with padding alone (top).* Continuity Mapping *makes any multi-chart parameterization seamless (bottom).*

Abstract

It is well known that multi-chart parameterizations introduce seams over meshes, causing serious problems for applications like texture filtering, relief mapping and simulations in the texture domain. Here we present two techniques, collectively known as *Continuity Mapping*, that together make any multi-chart parameterization seamless: *Traveler's Map* is used for solving the spatial discontinuities of multi-chart parameterizations in texture space thanks to a bidirectional mapping between areas outside the charts and the corresponding areas inside; and *Sewing the Seams* addresses the sampling mismatch at chart boundaries using a set of stitching triangles that are not true geometry, but merely evaluated on a per-fragment basis to perform consistent linear interpolation between non-adjacent texel values. *Continuity Mapping* does *not* require any modification of the artist-provided textures or models, it is fully automatic, and achieves continuity with small memory and computational costs.

1 Introduction

Multi-chart parameterization methods are widely used both to achieve low distortion texturing and to parameterize topologically complex models. When texturing, the standard approach is to perform a bilinear interpolation over a regular grid of texels defined on each chart, which can be at completely different regions in texture space. The problem is that these regular texel grids do not match up at chart boundaries (Figure 2) and bilinear signals defined over two adjacent charts will not be continuous across the boundary, which causes discontinuities in the texturing function and visible artifacts that artists usually hide by a tuning process (see Figure 1). This problem is aggravated when automatic parameterizations are used, as they result in an increased number of seams that are not manually fine tuned by artists. This situation causes problems for several application domains such as texture filtering or displacement/height maps, where it is often referred to as *watertight texture sampling*. This problem can be slightly abated by increasing texture resolution, but artifacts can still be seen at *any* resolution. In contrast, our method works correctly even with low-resolution texture atlases.

Another possible solution is extending the chart and filling the neighboring area with texture values from corresponding neighboring charts (padding). However, this only serves to reduce the visibility of the seams, which are still visible at shorter distances. In fact, pre-filtering techniques are most likely to fail at chart boundaries, mainly because of the mismatch in texel frequencies at both sides of a seam in texture space. Actually, discontinuities are intrinsic to the sampling and bilinear reconstruction process, so any automatic amendment technique can at most reduce the effects of the different sampling frequencies at the chart boundaries, but never make them disappear completely.

Our solution, *Continuity Mapping*, is a combination of two independent, but related techniques: *Traveler's Map*, which solves the spatial discontinuity, and *Sewing the Seams*, which solves the sampling mismatch with a set of virtual texture-space triangles that connect texels in *different* charts. Thus, *Continuity Mapping* fixes the problem by defining a continuous reconstruction across the adjacent charts, with *small* computational and memory costs.

Contributions: They can be summarized as follows

1. With *Continuity Mapping*, the texturing function becomes continuous, which is a fundamental condition for correct fil-

*e-mail:gonzalez@ima.udg.edu
†e-mail:dagush@ima.udg.edu

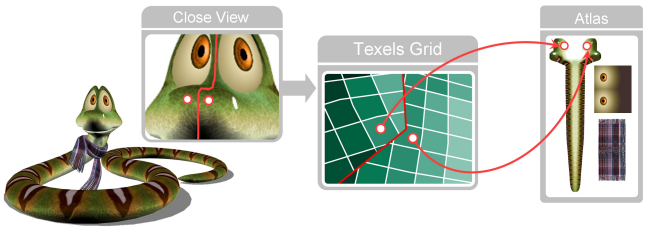

Figure 2: *Motivation. Fragments from different sides of a seam are parameterized onto disjoint areas in texture space (different orientation and stretching), leading to discontinuities in close-up views.*

tering. As a consequence, we avoid artifacts and make the parameterization seamless.

2. The method works completely in texture space and it uses the original artist's designed model with a multi-chart texture, and without modifying the artists' contents at all.

3. *Continuity Mapping* does *not* require any re-parameterization of the artist designed model, nor the use of non-accurate texturing operations like texture transfers. This avoids the usual blurring and other similar problems that result from the mismatch of the original and the target texture resolutions.

4. It is a GPU-friendly technique that can be evaluated completely in runtime using only single-pass fragment shaders, with very low computational and memory costs.

2 Related Work

Continuity Mapping is strongly related to texturing parameterization, as it intends to solve the problem of continuity in texture space [Floater and Hormann 2005]. In order to achieve this, several approaches have been proposed.

Geometry Images [Gu et al. 2002] unwrap an entire mesh into a single chart, creating parameterizations with greater distortion and less uniform sampling than can be achieved with multiple local charts, particularly for surfaces of high genus. Yao and Lee [2008], Carr et al. [2006] and Sander et al. [2003] present extensions to parameterize them into multiple charts, and Purnomo et al. [2004] describe a new type of seamless quadrilateral chart-based atlas. There, the neighboring chart's texels were copied into a one-pixel gutter on the boundary of the original chart. This work was the extension of [Carr and Hart 2002], where mip-mappable atlases were created with one chart per triangle, also using a gutter to sample across seams. Continuity Mapping supports wider filter widths than these approaches while keeping a gutter of only a couple of pixels at the chart boundary. Octree Textures [Benson and Davis 2002] and Tiletrees [Lefebvre and Dachsbacher 2007] are two approaches to texture continuity that encode textures in octree-like data structures around the model. Mesh Colors [Yuksel et al. 2008] is a technique that stores colors on vertices defined over the mesh, with the parameterization defined *directly* by the mesh itself. If an artist provides a "traditional" model with a texture atlas (the most common scenario), a texture transfer process must be applied, and any of the mentioned techniques will suffer from the blurring and loss of detail resulting from that transfer, no matter how many samples are used for the evaluation. Also, seams in the original textures could be transferred as artifacts. Continuity Mapping would be a good solution for this problem, and to the best of our knowledge, it is the first and only seamless texture mapping technique that doesn't rely on reparameterization.

Sheffer and Hart [2002] present a method that finds places to put seams, without eliminating them. Sander et al. [2003] use an atlas to map the piecewise surface onto charts of arbitrary shape and average values to reduce seam visibility. Also, Kraevoy et al. [2003] and Zhou et al. [2005] present methods that employ parameterization constraints to hide seams. More recently, Castano [2008] suggested using patch ownership to assign consistent displacement along seams for subdivision surfaces. Due to the different scaling and orientation of the charts, this still produces seams, although they are less noticeable. On the contrary, *Continuity Mapping* completely eliminates seam visibility, with just a slightly increase in the computational and memory costs.

In lapped textures [Praun et al. 2000] the perceptibility of seams is reduced by applying alpha-blending at the edges of the pasted texture patches. Losasso and Hoppe [2004] used textures as a height field and blended heights between mip-map levels. In a similar approach presented by De Toledo et al. [2008], neighboring charts must share boundaries with each other, resulting in some overlapping between them. However, some undesirable artifacts appear on regions with strong curvature, and self-shadows are very difficult to compute. None of these techniques completely eliminates the seams from the parameterizations.

One of the techniques presented in this work, the *Traveler's Map*, can be considered as a generalization of the Indirection Maps presented by Lefebvre and Hoppe [2006a] as a structure to synthesize a texture over a discontinuous atlas. They can only be used for simple simulations in texture space and do not provide any solution for the seam visibility problem in 3D space. Both techniques work in a similar way to the atlas transition functions defined by Grimm and Hughes [1995], but they differ in that they are defined in the surrounding area outside the charts.

3 Overview

Given a 3D textured model that has already been parameterized with any multi-chart technique, in a pre-processing stage we build both *Traveler's Map* and *Sewing the Seams* data structures to eliminate texture discontinuities later at runtime.

- **Traveler's Map** defines a correspondence in texture space, which allows any point (and direction) outside a chart to be related to the corresponding point inside. This information is encoded in texels *surrounding* the artist-provided charts, without modifying the artist's content.

- **Sewing the Seams** uses the information created by *Traveler's Map* to generate a thin border of interpolating triangles between charts to consistently filter texture values across the seams. Note that these triangles are only created in texture space, without altering the original 3D mesh. This is a great enhancement over the texture zippering techniques described, for instance, by Castano [2008] or Sander et al. [2003].

4 Traveler's Map

As mentioned before, the main objective of *Traveler's Map* is to solve spatial discontinuities in texture space introduced by multi-chart parameterizations. In the following subsections we explain how to build and use them.

4.1 Construction

Given a multi-chart parameterized 3D model, *Traveler's Map* first checks for the 3D seam edges, which are the set of edges belonging to the chart boundaries of the mesh, by simply looking for the

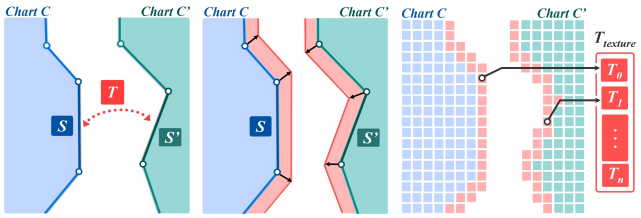

Figure 3: *Traveler's Map definitions. Seams s and s' are paired using transformation T (left). We create a security border (middle) around each chart storing references to the respective T (right).*

edges in 3D that are parameterized to different positions in texture space. In fact, each seam edge has two unique instances, s and s', in texture space and we pair them through a transformation matrix T (see Figure 3, left). This matrix transforms the points from s to s' by translating, rotating and scaling them longitudinally. For each seam edge in texture space, we store its transformation in a pair of 1D textures (float RGB32), called *Transformation Textures*.

Then, as shown in Figure 3, middle, we create a security border a couple of pixels wide around each chart by drawing quads that extend 2D seam edges to the exterior of the chart. For every seam edge in a chart, we consider its exterior perpendicular 2D vector as the 2D normal. The average 2D normal at a vertex is computed as the average of the 2D normals at the neighboring edges. The quads are built using the original seam edge, the two averaged 2D normals at each seam vertex, and the segment that closes the quad.

We render to texture these quads, and for each rendered texel, we store a reference to the respective entry in the *Transformation Textures*. It is important to note that this extra information is stored in the empty spaces between charts without modifying the artist-provided texture. As illustrated in Figure 3, right, this implies that the separation between the charts should be a few texels wide as happens with padding techniques or Indirection Maps [Lefebvre and Hoppe 2006a].

Continuity Mapping needs a texture that stores the *Traveler's Map*. As the security borders are kept in the empty space between the artist-provided charts, we merged both textures (artist and *Traveler's*) into a single one (8 bits RGBA), thus considerably reducing memory requirements. We also added a 1-bit mask in the alpha channel to determine if a point lies inside/outside a chart. *Transformation Textures* are kept separate.

4.2 Usage

Using a *Traveler's Map* is quite easy. Whenever a point with texture coordinates (u, v) has to be evaluated, we query the combined artist-*Traveler's* texture to know if it lies outside a chart, but on the security border. If it does, then the corresponding transformation T is retrieved from the *Transformation Textures* and the point is transformed with $(u', v') = T \cdot (u, v)$ to a point inside the chart. The values at the coordinates (u', v') are then used to fetch the correct values for the given point. Hence, if evaluation is required outside a chart, only two extra texture fetches are needed.

5 Sewing the Seams

As the name suggests, the function of this technique is to sew (zipper) the seams together by generating for each chart a thin border of filtering triangles in texture space. These triangles are then used to correctly interpolate and filter texture values at chart boundaries where there is a mismatch of resolutions and orientations.

5.1 Construction

The construction process for *Sewing the Seams* consists of three steps: identification of *trustworthy* texels, construction of the *Shared Triangulation*, and the construction of the *Non-Shared Triangulation*. Figure 4 illustrates this process.

Trustworthy Texel Identification: Traditional bilinear filtering is performed in the GPU by interpolating the four nearest neighboring texel centers, but here special care must be taken if one of these centers is "outside" the chart, as its value is considered to be untrustworthy. Thus, a trustworthy texel center is defined as one being "inside" the projected chart line in texture space, with one of its eight neighbors being "outside". We can think of these texel centers as the representation of the true boundary of the artist-defined content inside the charts, that, in the authors' point of view, should be strictly preserved (see Figure 4(b)).

Once trustworthy texels have been identified, we create segments that join them. This can be done easily since almost all trustworthy texels can be 4-connected with neighboring trustworthy texels (see Figure 4(b)). If a texel center cannot be connected (e.g. because of a very acute angle between two seam edges), it is provided anyway as an independent point for the triangulation.

Next, we create an association between trustworthy texels and seam edges. A trustworthy texel will be associated with the seam edges lying on the square formed by its eight neighboring texel centers, as they would affect the bilinear interpolation. Trustworthy texels (and segments) associated with only one seam edge will be used as input for the *Shared Triangulation*, while trustworthy texels associated with two or more seam edges will be taken into account during the *Non-Shared Triangulation* step. The reason to distinguish between two triangulations according to the texels involved is simple: trustworthy texels shared by more than one seam edge have more than one matrix T to choose from (one for each seam edge), so we cannot use them independently, or even an average matrix, as the triangulations would not match when back-projected in 3D. That is, the triangulation cannot be shared.

Shared Triangulation Construction: As said before, in this step we build the triangulation using only the trustworthy texel centers shared by one seam edge. So, for every seam edge s in texture space, we use *Traveler's Map* to transform the associated texel centers (and segments) from the twin seam edge s' to the outside of s (see Figure 4(c)). Then, to guarantee valid and nice (not skinny) triangulations and, at the same time, avoid the generation of undesirable interior-chart triangles, we use a constrained Delaunay triangulation of a planar straight line graph (PSLG) [Shewchuk 1996]. To build the PSLG, we use the previously mentioned vertices and segments and close it with the first texel center, starting from the extremes, that generates a closing segment that does not intersect with an already existing segment (see Figure 4(d), bottom row). The texel centers and segments discarded by this iterative process (like the dotted segment from Figure 4(d)) are treated later in the section on *Non-Shared Triangulation*.

Due to the shared nature of this triangulation, the twin seam edge s' will use the same set of triangles created for s but will employ the corresponding transformation T from *Traveler's Map* (see Figure 4(d), top row).

Non-Shared Triangulation Construction: Before creating the *Non-Shared Triangulation*, we compute the two intersections between the *Shared Triangulations* near a seam vertex (red dot in Figure 4(e)) and the seam edges. We are interested in the intersections closest to the seam vertices, which create two new vertices called *Intersection Vertices* (orange dots in Figure 4(e)). Then, we build

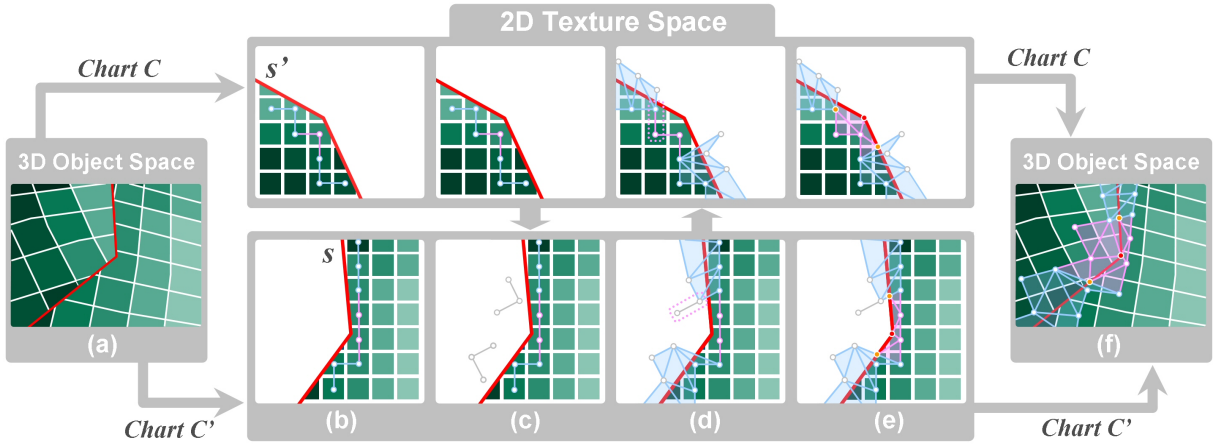

Figure 4: *Construction of* Sewing the Seams. *(a) Charts and seams in 3D. (b) Identify and join the trustworthy texel centers in every chart. (c) With* Traveler's Map*, transform the centers to the outside of the corresponding twin seam on the other chart. (d-e) Triangulate both the interior and the exterior centers, taking special care with corners where two seam edges meet. (f) The triangulation mapped back in 3D.*

another constrained Delaunay triangulation of a PSLG with the vertices and segments discarded from the *Shared Triangulation*, the intersection vertices, the edges from the intersection vertices to the respective seam vertices, and the vertices and segments involving texels shared by more than one seam edge. Although *Non-Shared Triangulations* share the seam edges to guarantee continuity, they are built independently for each chart (see Figure 4(e)).

It may be noted that the *Sewing* triangulation works for any number of charts meeting in a single seam vertex, without requiring any special consideration, as shown in Figure 5. Also, cases where there is no possibility of building a *Shared Triangulation* (e.g. when all trustworthy texels in a chart are shared by more than one seam edge), using the algorithm described above, a *Non-Shared Triangulation* can be built without any problem. The algorithm works even with one-texel sized charts, as long as they have at least one trustworthy texel center. A chart that does not cover a texel center is a problematic case, even for the graphics hardware itself.

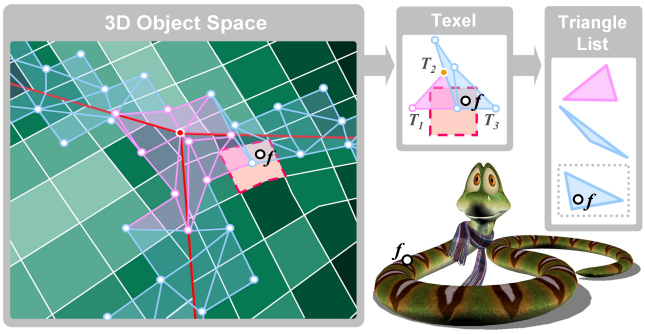

Figure 5: Sewing the Seams *usage and data structures. Fragment f queries the list of triangles associated with the texel (T_1, T_2 and T_3), and point f is found in triangle T_3.*

5.2 Storage details

For the *Sewing the Seams* technique, in the artist's texture covered by the triangulations, we need to store in every texel a list of all sewing triangles that overlap with it (see Figure 5). In these texels we store a reference to a texture called *Sewing Indexes Texture* (RGB8), which stores lists of triangle identifiers. These identifiers point to the triangles stored in a third texture, called the *Sewing Triangles Texture* (float RGB32). As trustworthy centers have an artist-defined color, we move the color information to the first entry in the lists in the *Sewing Indexes Texture*, followed by the actual list of triangle identifiers.

As all our one or two texel-wide triangulations mainly use texel centers as triangle vertices, texels are usually only half covered by the triangles. To avoid unnecessary evaluations, we subdivide our texels into four sub-texel quadrants, and store four bits in the empty channels of the combined artist's Texture telling the shader if there are triangles in that quadrant. In this manner, we avoid about 50% of the evaluations, achieving a significant increase in performance.

Attribute information (e.g. color) for the triangle vertices is not stored directly in our textures. Rather, we store the texture coordinates instead and use them as vertex coordinates and to retrieve attribute information. This facilitates the use of dynamic content in textures, like simulations or animations in texture space. For trustworthy texel centers, we simply store their texture coordinates and, in the case of centers belonging to the shared triangulation, the seam ID to retrieve the respective matrix T for transformations between charts. For vertices that are not texel centers (intersection and seam vertices), as they do not have artist-defined attributes, we take their values from two nearby trustworthy texels. So, we just only store the coordinates and weights of these texel centers. For intersection vertices these weights come from the linear interpolation to the edge on the *Shared Triangulation*, while for seam vertices the two closest texel centers are chosen.

It is noteworthy to observe that the *Sewing the Seams* technique uses only the transformation matrices created for *Traveler's Map*, and not the security border required for the same. Hence, the spacing between the charts can be smaller than the spacing when *Traveler's Map* is used alone.

5.3 Filtering with Sewing the Seams

If a fragment to evaluate falls in a texel which contains only color information (an alpha value different than 0), or if its corresponding quadrant has no sewing information, we evaluate it with a typical bilinear interpolation. Otherwise, we access both *Sewing* textures and search for the triangle that contains the fragment coordinates. The final color is obtained by a simple barycentric coordinate inter-

polation of the attributes associated with the respective texel centers (e.g. the artist-provided texture) (see Algorithm 1). If needed, other sampling-based interpolation schemes, like anisotropic filtering or wider kernels, can be implemented quite easily from these data structures.

Algorithm 1 filteringWithSewingTheSeams

```
 1: value = artistTex[texCoord]
 2: if isAColor(value) then
 3:     return getInterpolatedValue(value, artistTex, texCoord)
 4: else
 5:     intersectionFound = False
 6:     qFlags = value.z //read the 4 quadrant flags
 7:     if currentQuadrantHasSewingInfo(qFlags,texCoord) then
 8:         triangleListId = value.xy
 9:         texelCenterColor = sewingIdsTex[triangleListId]
10:         numTriangles = sewingIdsTex[++triangleListId]
11:         while numTriangles> 0 & !intersectionFound do
12:             triangleListId++
13:             intersectionFound = pointInTriangle(sewingTrisTex,
                                        triangleListId, texCoord)
14:             numTriangles– –
15:         end while
16:     end if
17:     if intersectionFound then
18:         return computeTriangleColor(sewingTrisTex,
                        triangleListId, texCoord)
19:     else
20:         return getInterpolatedValue(value, artistTex, texCoord)
21:     end if
22: end if
```

6 Mip Mapping and Shader LoD

Continuity Mapping allows seamless mip-mapping over multi-chart textures. First, we construct the *Continuity Mapping* pyramid repeating the procedure described in Section 3 for different resolutions. As the results in Section 8 demonstrate, the construction time required is short, making this procedure quite practical. Then, in run-time, given the sewing triangles at each mip-map level, interpolation is computed at two texture resolutions and then interpolated between them. Each sample for each level may come from either a barycentric (linear) interpolation on a filtering triangle (evaluated with attributes from the textures themselves), or a standard bilinear interpolation on the texel grid. Obviously, the cost is the sum of these evaluations plus the interpolation of the final values.

Although *Continuity Mapping* works for every distance, we shift to regular textures at medium distances, when the use of *Continuity Mapping* is no longer noticeable. It is important to note that, even for large distances where we use a regular texture, if there is an animation or we are computing a continuous simulation in texture space, a *Traveler's Map* should at least be built to guarantee the continuity of the simulation in all mip-map levels.

7 Applications

Seamless Texture Filtering. This is one of the most straightforward applications of *Continuity Mapping*. As can be seen in Figures 1 and 6, we show two models: the snake (25448 triangles) and the Neptune (80000 triangles). The first one has been carefully parameterized by an artist to explicitly hide the seams, while the second one has been parameterized with Iso-charts [Zhou et al. 2004]. Also, observe how the snake model is made by only four charts (1422 seam edges) and the Neptune model is parameterized into 100 charts (7218 seam edges). The last two columns show the corresponding textures applied to the models using the padding technique and the *Sewing the Seams* solution, respectively. As can be seen, seams are still visible at close range despite of the fine-tuning the artist did over the parameterization. Also, *Sewing the Seams* shows its stability working with atlases like the one used for the Neptune mesh, which has a large number of charts, including several small ones, even containing seams at the subtexel level.

Continuous simulations. Another interesting application of *Continuity Mapping* is in the area of continuous simulations like 2D fluid simulations [Stam 2003], droplets [Kaneda et al. 1993] and Reaction Diffusion [Witkin and Kass 1991]. In general these simulations are usually done onto regular mappings, as did Carr et al. [2006], to simplify texel neighborhood computations. Not only is this restrictive in nature, it also introduces a strong texture distortion that is not good for numerical simulations involving derivative computations [Witkin and Kass 1991]. We use the *Traveler's Map* to evaluate simulation properties when the methods require sampling outside a chart (see Figure 7). In that case, we use the associated matrix to compute a texel inside the chart sharing the common seam edge. For rendering, we use the full *Continuity Mapping* technique, including *Sewing the Seams* for correct texture filtering in addition to bump mapping. Note that a correct bump mapping requires the computation of the normal field and its interpolation through the seams, which can also be done with our technique.

Multi-Chart Relief Mapping. Another application of *Continuity Mapping* is Relief Mapping [Policarpo et al. 2005] over multi-chart textures. Up to now, general objects have been taken into account only over simple continuous parameterizations [Chen and Chang 2008] or by solving only for the silhouettes [Oliveira and Policarpo 2005], but the problem of using Relief Mapping in combination with a multi-chart parameterization remains unsolved. With *Continuity Mapping*, seamless multi-chart relief mapping is possible (see Figure 8). Figure 9 shows that it is possible to sample outside a chart while searching for the intersection point. Since there are security borders surrounding the charts, if we sample on a security border, we use the *Traveler's Map* to retrieve the 2D matrix that will transform the point on the *outside* of one chart to the *inside* of the corresponding chart where the search should continue. It is important to note that we not only transform the point, but also the travelling direction, so the search for the intersection point can continue without any issues on the other chart. In fact, the *Traveler's Map* gets its name from this functionality – if you are lost while travelling the multi-chart parameterization, simply use the *Traveler's Map* to find the direction. When the intersection search process requires sampling in areas very close to the seams, we use *Sewing the Seams* to correctly interpolate height and color values in this area. The only extra precaution that needs to be taken is adjusting the maximum step length to be smaller than the security border width. However, if the step is made larger, the algorithm can backtrack until it finds the security border when empty space is found instead.

Also, as demonstrated by Figure 8, *Continuity Mapping* proves to be more accurate than padding (blue artifacts are fragments evaluated outside the charts) and Indirection Maps [Lefebvre and Hoppe 2006a], which is not surprising considering that our approach provides sub-texel accuracy transformations and also preserves directions.

Other applications. Displacement Mapping and Caustics computations in the GPU are applications that can also benefit from the use of *Continuity Mapping*. On the one hand, displacement mapping using multi-chart textures introduces undesirable holes in the mesh along the seams, producing much more visible artifacts that are harder to hide than texturing artifacts [Castano 2008]. Here, *Continuity Mapping* can provide a continuous sampling over the

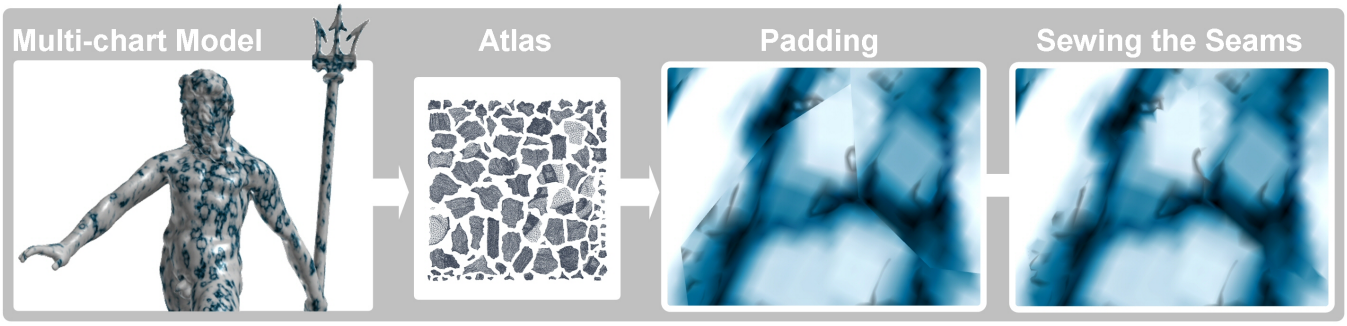

Figure 6: *Seamless Texture Filtering on the Neptune model (Atlas* 2048^2*). Third column: padding. Fourth column:* Sewing the Seams*.*

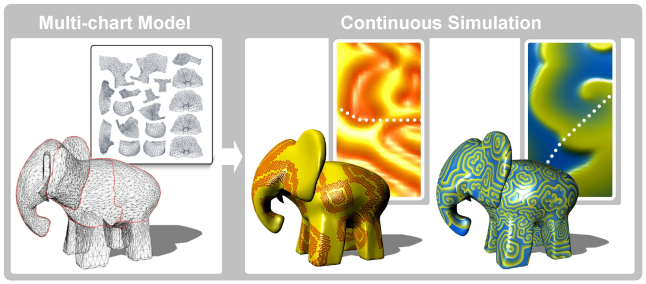

Figure 7: *A parameterized elephant model (13402 triangles) and different results for continuous reaction-diffusion simulations. Seams are represented by the dotted lines in the close views.*

seam regions avoiding the aforementioned geometry cracks. On the other hand, real-time caustics (for example, Photon Mapping in texture space) require photon hit filtering. This is usually achieved by blurring hits using splatting, but photon splats may fall on a chart border, resulting in energy being lost due to part of the photon being splatted outside a chart [Szirmay-Kalos et al. 2008]. Yet again, *Continuity Mapping* can prove to be a valuable tool as it accumulates energy at the correct texture charts.

8 Results and Discussion

Preprocessing times for *Continuity Mapping* are small, ranging from 22 seconds for the bunny model (5058 triangles) up to about two minutes for the Neptune model (80000 triangles). Also, in the latter example we can see that *Continuity Mapping* behaves correctly with large meshes with subtexel seam edges, as they are naturally included in the *Non-Shared Triangulation*. The only requirement is that the charts must have at least one trustworthy texel center. Also, just like the padding technique, there must be a few pixels of separation between charts, between non-topologically-adjacent edges of the same chart or between consecutive edges with very acute angles, for the *Traveler's Map* to work. However, in all our experiments we have not come across any noticeable problems in these cases. Some applications might require extremely high precision for the case of subtexel edges in the *Traveler's Map*, which cannot guarantee which edge will provide the matrix for that texel. This can be solved by keeping, for every texel in the border, a list of all quads that project onto it, as done in the *Sewing the Seams* technique, but we considered this to be unnecessary and inefficient, not having observed any noticeable artifact in any of our experiments.

The increase in memory consumption of an artist texture goes from 4MB for a 1024^2 resolution to 36MB for a 3072^2 resolution, while the memory consumption for *Continuity Mapping* goes from 1.3MB for a 1024^2 texture resolution to a mere 3.46MB for a 3072^2 texture resolution. A quick look at Figure 10 reveals that *Continuity Mapping* data structures grow in the order of $O(n)$ as the texture resolution grows in the order of $O(n^2)$. The reason for this is simple: these data structures follow the chart boundaries, which are basically 1D, embedded in a 2D space. The model used in the graph is the bunny from Figure 8 parameterized with LSCM [Lévy et al. 2002]. This storage efficiency can be compared to Perfect Spatial Hashing (which requires unconstrained access) [Lefebvre and Hoppe 2006b] because the *Traveler's Map* is not sparse. Furthermore, the textures for *Sewing the Seams* are equivalent to a Binary Image (embedded in the artist's texture) and a Hash Table texture, but without the need for an extra Offset texture.

The evaluation cost for *Continuity Mapping* varies from case to case, but the results can be generalized into two main categories: the cases where only the *Traveler's Map* is used, and the cases where both *Traveler's Map* and *Sewing the Seams* are used. While in the former scenario the cost is independent of atlas complexity (as explained in Section 4.2), in the latter, using *Sewing the Seams* varies the cost in different situations. But in general, we obtained high frame rates in all our experiments, with more than 1150 fps in an Nvidia GeForce GTX 280, while the models with regular texture filtering were displayed at 2400 fps. The graph in the Figure 11 shows that for a given resolution, *Continuity Mapping* requires less computational power as the model on screen gets smaller. This is due to the fact there are fewer fragments to evaluate. The aberration on the left part of the graph is caused by the continuously varying fragment count (the ones requiring *Sewing the Seams*), since the model is only partially visible. Also, the *sewing* lists were observed to have, in general and on average, 3.5 entries.

It is important to note that matrix T (see Section 4) stores a simplistic edge-longitudinal stretching measure. In cases of extreme stretching in the direction orthogonal to the edge, information from one texel can be taken farther away from where it should be. This case may require storing a per texel Jacobian, which is more accurate, but would largely increase memory requirements, and in our experience this has not been necessary.

Another important consideration regarding *Continuity Mapping* is that it works entirely in texture space, so it strongly depends on the texture resolution and stretching. For instance, if the texture resolution is doubled, the resulting triangulation for *Sewing the Seams* would be much thinner in 3D space around the seams, and would in turn cause a smaller and smoother interpolation. On the other hand, if the texture is stretched more in one direction than the other, the resulting triangulation could result in skinnier triangles, and thus a lower quality triangulation. Also, if the scaling between two seam edges is too large, Continuity Maps will produce a smooth inter-

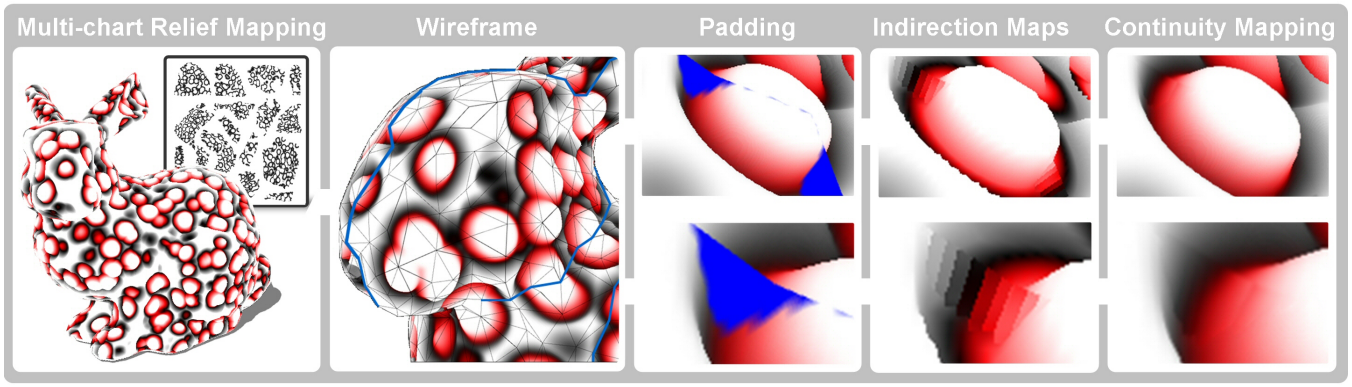

Figure 8: *Multi-chart Relief Mapping. Comparison between simple padding, Indirection Maps and Continuity Mapping.*

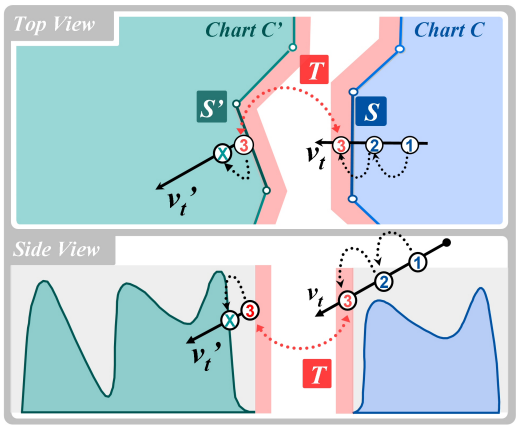

Figure 9: *Tracing a ray with Multi-Chart Relief Mapping.*

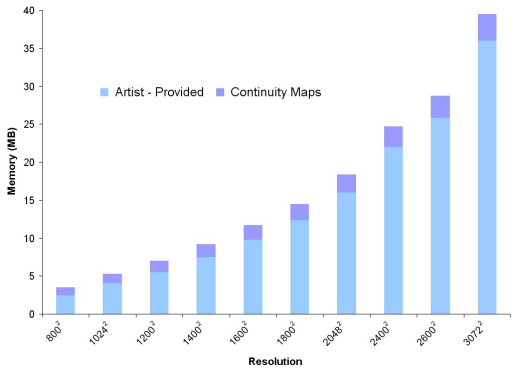

Figure 10: *Memory vs. texture resolution (bunny model).*

polation, but at a lower quality than when the edges have similar transformations. In our experiments, we did not observe any problems for scalings up to slightly more than 50%.

Continuity Mapping is limited when trying to smooth the seams of a texture with sharply varying features because it only works on a thin layer, up to two texels wide, across the seams. If the features on both sides of the seam do not match, *Continuity Mapping* will provide smooth continuity at short distances, but the seams will still be visible at medium and large distances because the features mismatch at larger scales, even when mip-maping is applied. This would probably require a smoothing pass with a large matrix, which could easily be computed with a *Traveler's Map*. In any case, solving a global texturing mismatch is not the objective of this article, but it does add continuity at the seams between charts without changing the original parameterization.

Sometimes artists use mirroring operations to build their meshes, resulting in the two halves of the object being mapped in the same area in texture space. The techniques presented so far cannot deal with this non-bijective parameterization, but this can be easily solved by mirroring the charts in texture space and restoring the bijectivity, which can be done automatically in a pre-process stage.

The explanations above refer to achieving correct interpolation of values/attributes across chart boundaries. Achieving continuity of higher order functions like successive derivatives (e.g. normal mapping) is simply a matter of using the functionality described above with that information. For instance, continuous normal mapping for a height field could be built in two simple passes: the first one computes a normal for each texel by using the height of its four immediate neighbors (using the full *Traveler's Map* at the chart boundaries), and the second pass would use *Sewing the Seams* to generate the continuous from the normal map.

Continuity Mapping is compatible with mesh deformations/animations, as it works completely in texture space without altering the original model geometry. Moreover, as seen in the Continuous Simulation, the information stored in the textures need not be static, and can be used with dynamic simulations or animations in texture space.

9 Conclusions

We have presented a technique that solves the continuity problems that arise in multi-chart parameterizations. The technique named *Continuity Mapping* is composed of two independent but related sub-techniques: *Traveler's Map* and *Sewing the Seams*. We also demonstrated the power of this technique by suggesting some applications that can be made seamless, which contrasts with the previous approaches.

Acknowledgements

We are grateful to I. García, S. Pattanaik, X. Pueyo, P. Brunet, S. Chawla, A. Forés, T. Paradinas, V. Gratorex, the GGG team and the anonymous reviewers (especially our primary) for all their valuable advice and help. This project was funded by grant TIN2007-67120

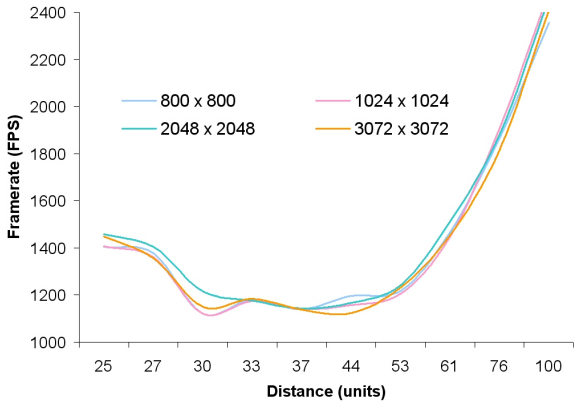

Figure 11: *Dependence of the frame-rate on the distance to the observer (bunny model, viewport:* 1024×768*).*

from the Spanish government.

References

BENSON, D., AND DAVIS, J. 2002. Octree textures. *ACM Trans. Graph. 21*, 3, 785–790.

CARR, N. A., AND HART, J. C. 2002. Meshed atlases for real-time procedural solid texturing. *ACM Trans. Graph. 21*, 2, 106–131.

CARR, N. A., HOBEROCK, J., CRANE, K., AND HART, J. C. 2006. Rectangular multi-chart geometry images. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, 181–190.

CASTANO, I., 2008. Next-generation rendering of subdivision surfaces. ACM SIGGRAPH 2008 presentations.

CHEN, Y.-C., AND CHANG, C.-F. 2008. A prism-free method for silhouette rendering in inverse displacement mapping. *Comput. Graph. Forum 27*, 7, 1929–1936.

DE TOLEDO, R., WANG, B., AND LEVY, B. 2008. Geometry textures and applications. *Comput. Graph. Forum 27*, 8, 2053–2065.

FLOATER, M. S., AND HORMANN, K. 2005. Surface parameterization: a tutorial and survey. In *Advances in multiresolution for geometric modelling*, N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds. Springer Verlag, 157–186.

GRIMM, C. M., AND HUGHES, J. F. 1995. Modeling surfaces of arbitrary topology using manifolds. In *SIGGRAPH '95 Proceedings*, 359–368.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. *ACM Trans. Graph. 21*, 3, 355–361.

KANEDA, K., KAGAWA, T., AND YAMASHITA, H. 1993. Animation of water droplets on a glass plate. In *Proceedings Computer Animation*, 177–189.

KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. 2003. Matchmaker: constructing constrained texture maps. *ACM Trans. Graph. 22*, 3, 326–333.

LEFEBVRE, S., AND DACHSBACHER, C. 2007. Tiletrees. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press.

LEFEBVRE, S., AND HOPPE, H. 2006. Appearance-space texture synthesis. *ACM Trans. Graph. 25*, 3, 541–548.

LEFEBVRE, S., AND HOPPE, H. 2006. Perfect spatial hashing. *ACM Trans. Graph. 25*, 3, 579–588.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3, 362–371.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph. 23*, 3, 769–776.

OLIVEIRA, M. M., AND POLICARPO, F. 2005. An efficient representation for surface details. Tech. Rep. RP-351, Federal University of Rio Grande do Sul - UFRGS.

POLICARPO, F., OLIVEIRA, M. M., AND JO A. L. D. C. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, 155–162.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, 465–470.

PURNOMO, B., COHEN, J. D., AND KUMAR, S. 2004. Seamless texture atlases. In *SGP '04: Proceedings of the 2004 Eurographics/SIGGRAPH symposium on Geometry Processing*, 65–74.

SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *SGP '03: Proceedings of the 2003 Eurographics/SIGGRAPH symposium on Geometry processing*, 146–155.

SHEFFER, A., AND HART, J. C. 2002. Seamster: Inconspicuous low-distortion texture seam layout. *Visualization Conference, IEEE*, 291–298.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, M. C. Lin and D. Manocha, Eds., vol. 1148 of *Lecture Notes in Computer Science*. 203–222.

STAM, J. 2003. Flows on surfaces of arbitrary topology. *ACM Trans. Graph. 22*, 3, 724–731.

SZIRMAY-KALOS, L., SZECSI, L., AND SBERT, M. 2008. Gpu-based techniques for global illumination effects. In *Synthesis Lectures on Computer Graphics and Animation*, B. Barsky, Ed. Morgan-Claypool.

WITKIN, A., AND KASS, M. 1991. Reaction-diffusion textures. In *SIGGRAPH '91 Proceedings*, 299–308.

YAO, C.-Y., AND LEE, T.-Y. 2008. Adaptive geometry image. *IEEE Trans. Visualization and Computer Graphics 14*, 4, 948–960.

YUKSEL, C., KEYSER, J., AND HOUSE, D. H. 2008. Mesh colors. Tech. Rep. tamu-cs-tr-2008-4-1, Department of Computer Science, Texas A&M University.

ZHOU, K., SYNDER, J., GUO, B., AND SHUM, H.-Y. 2004. Iso-charts: stretch-driven mesh parameterization using spectral analysis. In *SGP '04: Proceedings of the 2004 Eurographics/SIGGRAPH symposium on Geometry processing*, 45–54.

ZHOU, K., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM, H.-Y. 2005. TextureMontage: Seamless texturing of arbitrary surfaces from multiple images. *ACM Trans. Graph. 24*, 3, 1148–1155.