Atlas-Aware Laplacian Smoothing

Peter G. Sibley[*]
Brown University

Gabriel Taubin[†]
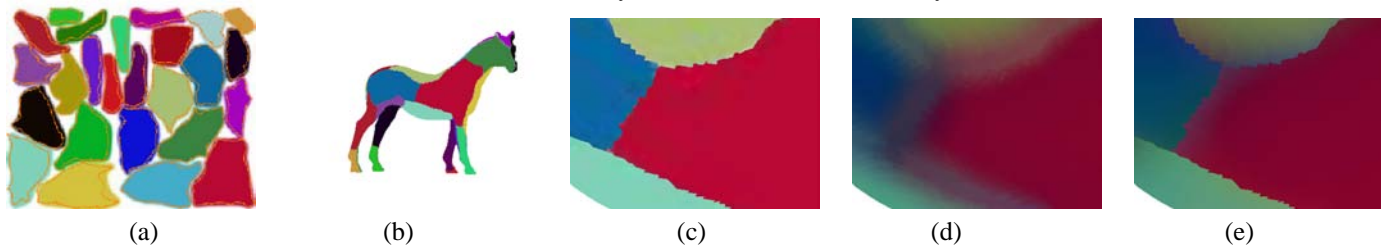Brown University

| (a) | (b) | (c) | (d) | (e) |

Figure 1: (a) A 256×256 texture map with chart boundaries of the mesh shown in orange. (b) A charted horse model with 97K faces. (c) Close up of the original textured model. (d) 125 iterations of atlas-aware Laplacian smoothing with $\lambda = 0.75$. (e) 125 iterations of standard Laplacian smoothing. Notice the boundary artifacts along the seams.

1 Introduction

We consider an image, a result of sampling over a finite two dimensional regular grid a vector valued function of two variables. There is a wealth of work in both theory and application of image processing operations. Our approach is to map into the parameter space then perform image processing operations there. Laplacian smoothing is the building block for defining linear filters, which can be described in terms of polynomials of the Laplacian operator. We focus on extending basic Laplacian smoothing to continuous vector valued functions irregularly sampled on multi-chart parameterized surfaces. Our goal is to efficiently perform Laplacian smoothing with out introducing seam artifacts. We present an algorithm for Laplacian smoothing over a texture atlas which takes into account the discontinuities imposed by the charts. Our general approach is based on the finite volume method, and the graph Laplacian used in geometric signal processing.

In image processing, the Laplacian of an image is obtained by convolving the image with the following 3×3 kernel:

$$L = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad . \qquad (1)$$

To perform Laplacian smoothing repeatedly sum the image and its Laplacian: $I^n = I^{n-1} + (\lambda L) * I^{n-1}$ for some $\lambda \in (0,1)$. This corresponds to evolving the diffusion equation: $\frac{\partial I}{\partial t} = \lambda \Delta I$. The simplicity and efficiency of this technique relies on the regular structure of the image.

Similar techniques have been applied for smoothing or denoising of large irregular polygonal meshes such as those created from isosurfaces of volumetric data. In geometric signal processing, the Laplacian operator for signals defined on vertices of a graph is defined as follows. Let $i^\star$ denote the 1-ring of the vertex i. The *Laplacian operator* is defined for a graph signal x with values in Euclidean space as

$$\Delta x_i = \sum_{j \in i^\star} w_{ij} (x_j - x_i), \qquad (2)$$

where edge weights w_{ij} are non-negative and sum to one over the vertex star. Laplacian smoothing is then $x^n = x^{n-1} + \lambda \Delta x^{n-1}$.

---

[*]e-mail: pgs@cs.brown.edu
[†]e-mail: taubin@brown.edu

[Taubin 1995] develops the notion of Fourier analysis on mesh signals using the Laplacian as defined above. Several authors expanded that work to include non-linear filters. More recently, [Gu et al. 2002; Sander et al. 2003] introduce geometry images and multi-chart variants, which are an alternate regular representation of a mesh with vertex positions packed into an image, and they apply compression techniques to the resulting models. This approach allows textures, normal maps, and geometry to be treated uniformly.

One interpretation of the Laplacian is an operator that diffuses information over all points in a domain. The finite volume method is a standard approach to construct approximate solutions of hyperbolic conservation laws [Versteeg and Malalasekera 1995]. The basic idea is to split to domain into small cells and enforce conservation by prescribing fluxes at the walls of each cell.

2 Algorithm

We first classify pixel of the texture as INTERIOR, SEAM or UN-USED, depending on whether or not the pixel is inside a chart, on a boundary of a chart or outside all charts. Fig. 2c shows two edges that correspond to a seam, and in orange the pixels that would be classified as SEAM pixels. Classifying pixels and storing seam adjacency information is performed once as a preprocessing step. For seam pixels we have to propagate or diffuse signal values across the seam. We evaluate the Laplacian in a two phase manner first pushing information out from the seam pixels across the seam border scaling the signal to be proportional to the cell interface length. Second, for interior pixels we evaluate the Laplacian using the regular 4-neighborhood structure, which does not need to be stored explicitly.

Consider a single seam, which corresponds to two separate edges of two charts. We have two tasks: determining the graph weights and determining the neighborhood structure of these irregularly shaped cells(Fig. 2b). For a cell with perimeter of g, and a side with length g_i, the weight for that interface is g_i/g. To determine the neighborhood structure, we use the fact that the seam and cell-wall intersections partition the two sides of the seam. We parameterize these intersections of the two edges with $\{t_i\}$ and $\{s_j\}$ for $t_i, s_j \in [0,1], t_i < t_{i+1}, s_j < s_{j+1}$. Cells k and l on opposite sides of the seam are neighbors if the corresponding interval $[t_k, t_{k+1}]$ and $[s_l, s_{l+1}]$ overlap. Using this criteria implies that the two edges of the seam are of the same length in the texture domain, in practice this is not always the case, so one of the edges of the seam is scaled.

Rather than store this adjacency information in a graph, we merge the two partitions then for every interval $[u_i, u_{i+1}]$ we compute the pixel locations (i, j) on either side and the interface length
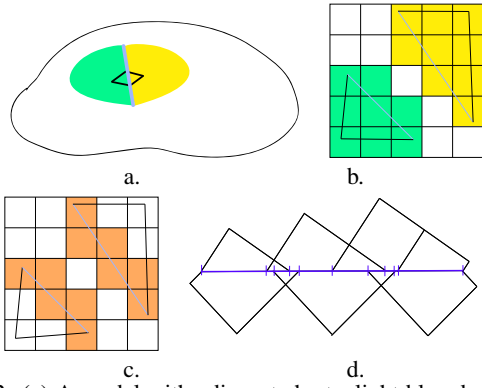
Figure 2: (a) A model with adjacent charts, light blue denotes the seam. (b) A 5×5 area in the texture map that shows two texture space triangles that correspond to adjacent triangles on the model. The two light blue edges correspond to the single seam on the model. (c) The pixels in orange denote classified as SEAM pixels. (d) The seam edge in texture space partitioned, shown with vertical bars, into *micro-edges*.

w_{ij}. We refer to these as *micro-edges*(Fig. 2d). For the other interfaces of a seam cell, we use the regular neighborhood structure, using weights proportional to the length of the interface. Both the micro-edges and the other seam cell interfaces are stored in a flat array of pixel correspondences and weights. The arrays for each seam are concatenated into an array M. After this preprocessing we have a buffer d that classifies the pixels, and M.

To perform a single step of Laplacian smoothing we proceed in two phases the first evaluates the Laplacian for the seam cells by pushing the data across the interfaces described by M. We use *wSum* as an accumulator of weights. b is our intermediate image buffer. First we define a function, FLOWACROSSINTERFACE to update the value of pixel i by pulling the value of pixel j.

FLOWACROSSINTERFACE$(i, j, w, wSum, b, t)$
1 $b[i] \leftarrow b[i] + w * t[j]$
2 $wSum[i] \leftarrow wSum[i] + w$
3 **return** $wSum, b$

We define a function PHASEONE which smooths the seam pixels.

PHASEONE(M, b, t, λ)
1   $wSum \leftarrow 0$
2   $P \leftarrow$ NIL
3   **for each** (i, j, w) **in** M
4   **do** $(wSum, b) \leftarrow$ FLOWACROSSINTERFACE$(i, j, w, wSum, b, t)$
5         $(wSum, b) \leftarrow$ FLOWACROSSINTERFACE$(j, i, w, wSum, b, t)$
6         $P \leftarrow P \cup i \cup j$
7   **for each** i **in** P
8   **do** $b[i] \leftarrow b[i]/wSum[i] * \lambda + (1.0 - \lambda) * t[i]$
9
10 **return** b

PHASETWO (d, b, t, λ)
1   **for** $i \leftarrow 1$ **to** LENGTH(t)
2   **do if** $d[i] = INTERIOR$
3   **then** $c \leftarrow (1 - \lambda) * t[i]$
4   **for** j **in** NEIGHBORS(i)
5   **do**
6                  $c \leftarrow c + \lambda/4 * t[j]$
7            $b[i] \leftarrow c$
8   **return** b

Finally, we define LAPLACIANSMOOTH for n steps; it simply iterates the above functions.

LAPLACIANSMOOTH(n, t, M, d, λ)
1   **for** $step \leftarrow 1$ **to** n
2   **do** $b \leftarrow 0$
3         $b \leftarrow$ PHASEONE(M, b, t, λ)
4         $b \leftarrow$ PHASETWO(d, b, t, λ)
5         $t \leftarrow b$
6   **return** t

One iteration of Laplacian smoothing is linear in the number of pixels. The only added space complexity is the classification buffer and array M, which is on the order of the number of seam pixels.

3   Results

We implemented the above algorithm in Java, and tested it on textured VRML models. The run time of regular Laplacian smoothing is comparable to the run time of the atlas-aware Laplacian smoothing. For the model shown in Fig. 1 with a 256×256 texture and 97K faces, 125 iterations of regular Laplacian smoothing took 11 seconds and atlas-aware Laplacian smoothing required 18 seconds.

Figure 3: (top left) Original texture. (top right) Original model. (bottom left) Atlas-aware Laplacian smoothed with parameters $\lambda = 0.25, n = 15$. (bottom right) Regular Laplacian smoothed texture using the same parameters. Notice the artifact at the hip.

4   Conclusions and Future Work

We are working on several improvements. The primary improvement is to take into account the distortion of the parameterization by using information from the first fundamental form, this would more accurately model diffusion on the surface while still operating in the parameter domain. Optimizing and moving the algorithm onto the GPU is another avenue of future work. The micro-edge data structure could be used for other PDE simulations over parameterized surfaces, where one must handle chart discontinuities.

References

GU, X., GORTLER, S., AND HOPPE, H. 2002. Geometry Images. *ACM Transactions on Graphics 21*, 3 (July), 355–361. Siggraph 2002, Conference Proceedings.

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

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Siggraph'95 Conference Proceedings*, 351–358.

VERSTEEG, H., AND MALALASEKERA, W. 1995. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Prentice-Hall.