What you seam is what you get

Vallet Bruno, Bruno Lévy

What you seam is what you get: automatic and interactive UV unwrapping

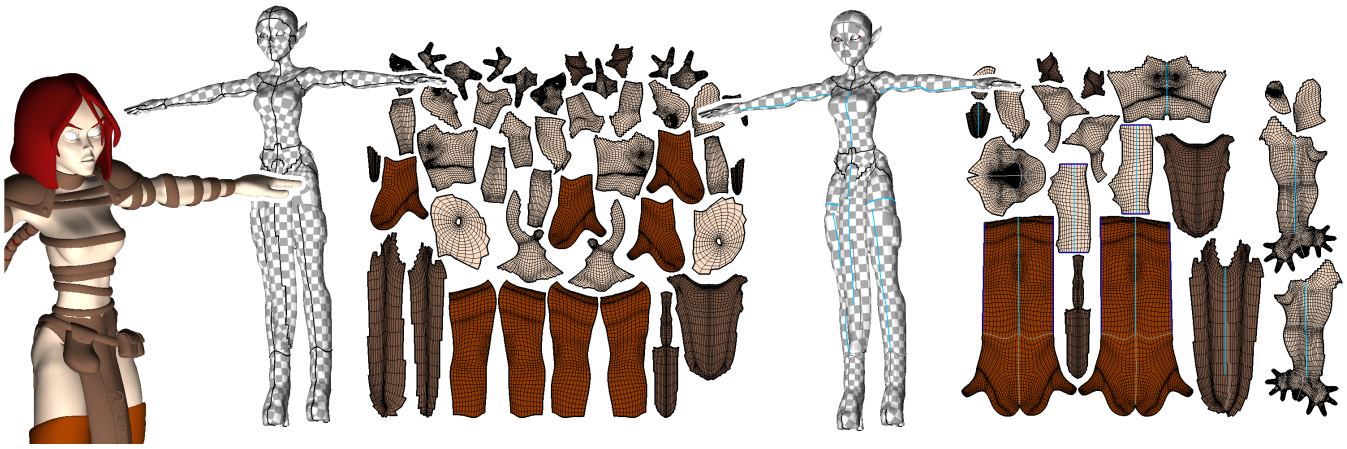


Figure 1: *Left: from a meshed model, our system automatically proposes an initial set of seams (black lines) and a valid texture mapping that the user starts with. Right: the user can interactively improve the mapping, by sewing charts and constraining seams (blue lines). As shown in the video, each user interaction is systematically echoed with instant visual feedback.*

Abstract

3D paint systems opened the door to new texturing tools, directly operating on 3D objects. However, although time and effort was devoted to mesh parameterization, UV unwrapping is still known to be a tedious and time-consuming process in Computer Graphics production. We think that this is mainly due to the lack of well-adapted segmentation method. To make UV unwrapping easier, we propose a new system, based on three components :

- A novel spectral segmentation method that proposes reasonable initial seams to the user;
- Several tools to edit and constrain the seams. During editing, a parameterization is interactively updated, allowing for direct feedback. Our interactive constrained parameterization method is based on simple (yet original) modifications of the ABF++ method, that make it behave as an interactive constraint solver;
- A method to map the two halves of symmetric objects to the same texels in UV space, thus halving texture memory requirements for symmetric objects.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture; I.3.4 [Computer Graphics]: Graphics Utilities—Paint systems; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms;

Keywords: segmentation, parameterization, spectral geometry

1 Introduction

3D paint systems, introduced in the pioneering work of Hanrahan and Haeberli [1990] permitted to enhance the visual richness of Computer Graphics objects by texturing them. Behind the scene, generating a texture atlas from a 3D mesh is vital for texturing applications. It consists in putting the object in correspondence with (a subset of) $\mathbb{R}^2$ in the following way :

- *(1) Segmentation:* the object is cut into one or more charts;
- *(2) Parameterization:* each chart in 3-space is put into one to one correspondence with a subset of $\mathbb{R}^2$;
- *(3) Packing:* the charts are arranged in texture space to minimize storage requirements.

Time and effort was devoted to the problem of mesh parameterization (see e.g. SIGGRAPH course notes [Hormann et al. 2007] for an overview). Such mesh parameterization techniques are now well known and broadly used in the Computer Graphics industry. Recently, formalizing the relations between deformations and curvature lead to both efficient and provably correct methods [Ben-Chen et al. 2008].

However, texture atlas generation - that graphists call ”UV unwrapping” - is still mentioned as a difficult task, that requires a great deal of user interaction, as can be seen in the tutorials and videos of popular softwares [DeepUV ; Unfold3D b; BodyPaint]. Most of the user time is taken by the tedious task of manually defining seams and cutting the object [Unfold3D a], also called “edge marking”.

A natural idea would be to use one of the numerous available segmentation methods instead. However, segmentation methods rarely achieve results comparable to how an artist would cut a model. Thus, artists usually prefer to manually cut the object, then parameterize, and iterate while the result is not satisfactory, which can be very time consuming.

The requirements of a U,V mapping are dictated by the way artists texture-map models. They use a mixture of 3D paint systems and 2D image editing. For instance, working in 2D makes it possible to use regular patterns and cutting/pasting existing 2D images. Therefore, during the editing process, artists continuously go back and forth between 3D and 2D space, using different softwares. To “know where they are” in 2D, they superimpose a stencil with a 2D projection of the mesh. For this reason, it is important to have a segmentation that yields a 2D space that is meaningful for the user.

Such a structure-aware U,V mapping is shown in Figure 2. Note also in the figure that symmetric features can be mapped to a single image in texture space (for instance the two legs and the two arms

Figure 2: *A low-polygon mesh and the associated texture (from Cubix studios), that was painted in 2D. Note how important parts (legs, hands, face) are kept. Note also that the legs (and also the two arms) are mapped to a single image, thus reducing texture storage.*

and the two halves of the skirt). For this example, the U,V mapping was carefully designed by an artist, which is reasonable for such a low-detailed mesh. In this paper, we propose a set of tools to facilitate creating such U,V mappings for more detailed model, by automating the process as much as possible. The requirements of a structure-aware U,V mapping (structure preservation and features alignment) involve semantic information. In general, it is not possible to automatically infer the semantics from the data.

Therefore, our algorithm first computes a valid mapping, that the user can interactively refine and improve. The design of our interactve tools is guided by the following two principles :

- *Sewing* is easier than *cutting*. Cutting involves complex user interaction [Unfold3D a], whereas sewing requires a single click;
- *continuous and systematic feedback* on each individual user action is important. "Black box" systems that separate seam-cutting from unfolding are problematic since they involve long error-and-trial loops in the process. This can be avoided by providing the user with direct feedback while editing the seams.

Contributions:

Based on these two guiding principles, we propose a semi-automatic "UV unwrapping system" with the following features :

- We introduce a novel spectral segmentation algorithm, that provides the user with a reasonable initial set of seams and parameterization. To our knowledge, this is the first approach that uses higher-order spectral cuts. This allows to detect good seam candidates (Section 3). We also prove that our set of seams contains all the axes of symmetry of the mesh (Appendix) ;
- When the detected seam is an axis of symmetry, our method can parameterize the two halves onto a single zone in U,V space, thus halving storage requirements ;
- We provide the user with interactive editing and constraint tools to improve this initial segmentation. We describe an original way of using the ABF++ algorithm, with two simple modifications that make it act as a constraint solver. (Section 4).

Before entering the heart of the matter, we summarize the previous works and give an overview of the system.

2 System overview

2.1 Previous works

Many automatic atlas generators and 3D paint systems have been proposed in the past years. Pedersen [1996] developed a "patchino-based" texturing method. His approach uses existing images that can be translated, rotated and deformed on implicit surfaces. In the same spirit, a constrained texture mapping for polygonal meshes was proposed [Lévy 2001]. More recently, a system [Schmidt et al. 2006] based on exponential maps was proposed to texture-map manifolds with multiple patchinos. These approaches are well adapted to texture-map a model with existing images, but do not integrate well in the modeling pipeline if new textures need to be created, since the representations that they use are non-standard.

Other approaches were proposed to assign (u,v) coordinates to the vertices of the mesh. Therefore, since they use a standard representation, these approaches can easily cooperate with other tools. In *interactive texture mapping* [Maillot et al. 1993] much effort was put into interactivity, but overlaps and high deformations often occur. The "model pelting" system [Piponi and Borshukov 2000] proposes a pragmatic approach for Catmull-Clark surfaces. It was implemented in several commercial products (e.g., 3D Studio). However, it may fail for complicated models. The texture atlas generator of LSCM [Lévy et al. 2002] achieves better parameterization but is highly dependent on the quality of the segmentation and on the surface curvature. Finally, Iso-charts [Zhou et al. 2004] is probably currently the best compromise and is now part of DirectX 10, but still lacks user interaction.

Our system is based on a new spectral segmentation algorithm combined with an interactive parameterization method. We will show how this achieves good interactivity and robustness altogether.

2.2 Initial segmentation

The first step of our system consists in computing reasonable initial seams, that the user can start with. Based on the idea that it is easier to remove a seam than to create a new one, our system creates more seams than necessary, and lets the user remove the undesired ones. Most of the so-constructed seams are natural, and resemble what an artist would have made. Our new spectral segmentation method is presented in Section 3.

2.3 Tools

Starting from this segmentation, the charts are parameterized (with ABF++) and packed in texture space, providing a texture atlas. In contrast with previous non-interactive methods where this atlas is the final result, we only consider it as a starting point for an interactive process. To improve this initial mapping, we provide the user with the following tools :

- *Seam editing:* Since the initial segmentation contains in general too many seams, the user can delete seams (and conversely create new ones if need be, although this is seldom the case);
- *Sew/Unsew (Figure 3):* The user can choose to keep only parts of a seam by sewing/unsewing it with a "zipper" metaphor. As shown in the figure, this can be used both to group related parts and to untangle overlaps;
- *Boundary angle constraint (Figure 4):* An alternative way of untangling an overlap is to constrain an angle on the boundary.
- *Straightening (Figure 5):* When the user selects a sequence of edges with this tool, all the selected edges will be aligned. Seams that cross at a vertex can also be constrained to be orthogonal;

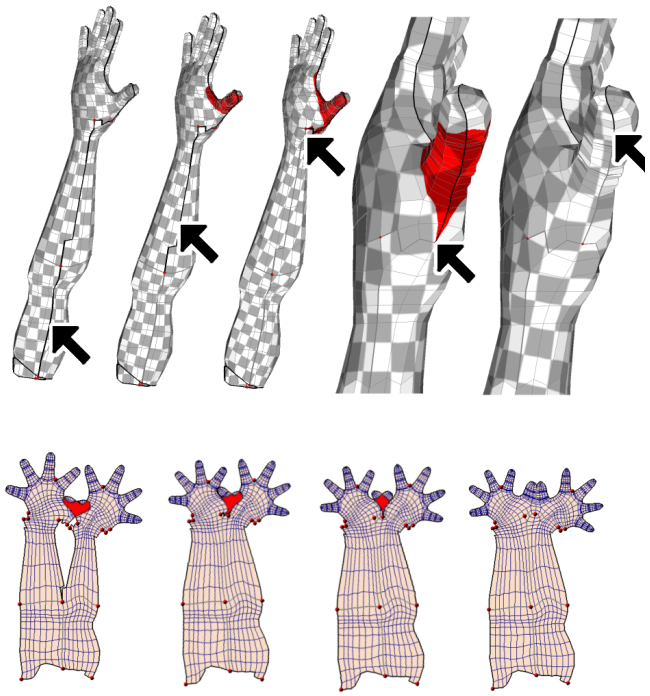


Figure 3: *The user can interactively sew seams. The red zone displays the overlaps (the two thumbs cross in UV space). With a "zipper" metaphor, the user can glue the two thumbs and resolve the overlaps. Editing can be done in 3D (top) and/or in UV space (bottom).*

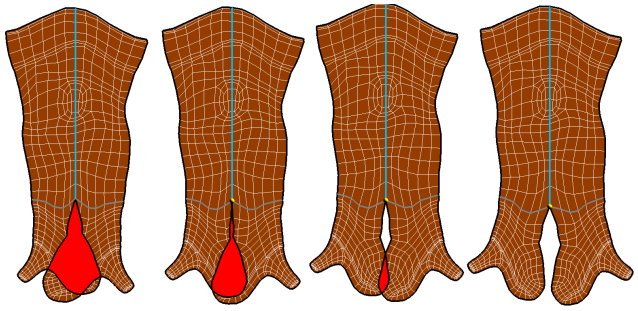


Figure 4: *To untangle overlap, besides sewing the sides of the overlap, the user can also install an angle constraint (yellow dot). By moving the mouse left and right, the user changes the angle at the yellow dot, and the parameterization is updated accordingly.*

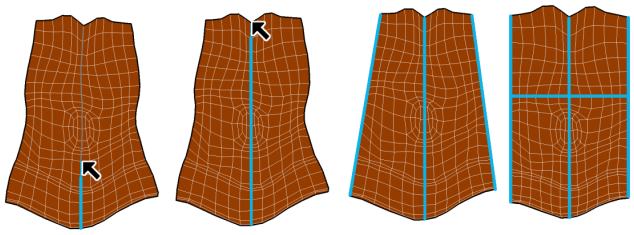


Figure 5: *Seams can be constrained to be straight in UV space. They can be also constrained to cross with right angles. This feature is useful for texturing models with regular patterns, by ensuring that the pattern will line-up.*

These tools can be applied either in the 3D view or in UV (texture) space. After each individual action, the parameterization is updated, and visual feedback is given to the user. In particular, overlaps are displayed in the 3D window, as red zones (see Figure 3). This is done on the GPU, by using the stencil buffer. We will now describe in detail the two main parts of our system (segmentation and editing).

3 Segmentation

3.1 Previous works

The problem of segmentation is generally ill-posed, so it has received a broad variety of treatments. Segmentation papers specialized for parameterization include Seamster [Sheffer and Hart 2002] and D-charts [Julius et al. 2005]. Seamster generates seams running through regions of high curvature and minimum visibility, and D-charts creates near developable charts with some criteria on their shape. Other segmentation techniques (see [Attene et al. 2006] for a comparative study) try to extract object features. In [Katz and Tal 2003], the segmentation relies on the extraction of feature points and cores.

A hierarchical segmentation is performed in [Katz et al. 2005] using a fuzzy clustering. The segmentation of [Cohen-Steiner et al. 2004], relies on building clusters as planar as possible. A recursive segmentation approach is proposed in[Zhang and Liu 2005] based on part saliency.

Segmentations based on the eigenvectors of a Laplacian or affinity matrix are called spectral and are reviewed in Zhang's recent EG star [Zhang et al. 2007]. Spectral approaches usually rely on embedding the mesh using the first eigenvectors, and K means-clustering [Liu and Zhang 2004] or contour analysis [Liu and Zhang 2007] is performed in that space. [Rustamov 2007] defines a spectral embedding that makes the segmentation pose-invariant. Namely, they first transform the mesh into the *GPS* coordinates (Global Point Signature) defined by :

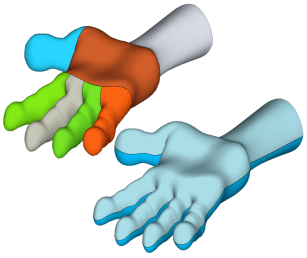
$$GPS(\mathbf{p}_i) = \left(\frac{1}{\lambda_1}^2 v_1^i, \frac{1}{\lambda_2}^2 v_2^i, \ldots \frac{1}{\lambda_n}^2 v_n^i \right)$$

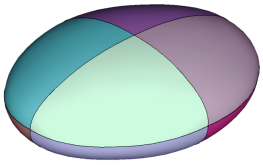
where λ_k denotes the k^{th} eigenvalue of the Laplacian, and where v_k denotes the k^{th} eigenvector (and v_k^i is the component associated with $\mathbf{p}_i$). This also corresponds to the Manifold Harmonics transform [Levy 2006; Vallet and Levy 2008]. As recalled in [Levy 2006], the first GPS coordinate is constant and associated with the eigenvalue $\lambda_1 = 0$, since the Laplacian operator vanishes for constant functions. The second GPS coordinate is called the Fielder vector. It is well known in spectral graph theory that the Fielder vector is a good candidate for partitioning a graph.

3.2 Higher-order spectral cuts

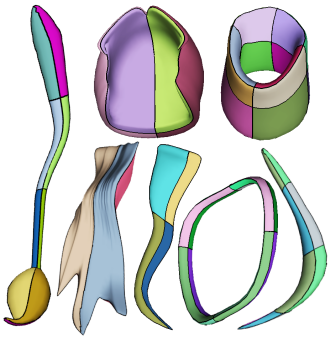
In terms of signal processing, the GPS coordinates correspond to the *whitening transform*, that decorrelates the component of a signal [Comon 1994]. In other words, each GPS coordinate has a minimum covariance with the GPS coordinates of lower order. We already know that the Fielder vector yields good spectral cuts. Since the higher-order eigenvectors are decorelated from the previous ones, they are likely to contain the remaining meaningful geometric information. More specifically, we show that the set of spectral cuts yielded by the eigenvectors contains all the symmetry axes of the mesh (see Appendix A). Motivated by this result, we propose in the next section to select the seam among the spectral cuts yielded by all the higher-order eigenvectors.

Our goal is to find a good initial segmentation that the user can start with. If we consider for instance a hand, most existing segmentation algorithms separate the fingers from the palm. This is what would be expected if the goal was to recover semantic information. In our case, this would generate too many charts. We rather want to split the hand into two halves, that will be easy to unwrap. Moreover, the user will more easily recognize the shape of the hand in 2D-space. In other word, what we want to detect is the "equator" of the object. To split the hand this way, i.e. to detect where the thickness of the object is, a (wrong) idea that comes naturally to mind would be to use principal components analysis.





Principal components analysis (PCA) approximates the shape of the object by an ellipsoid, and computes the axes of this ellipsoid. They are obtained as the eigenvectors and eigenvalues of a certain matrix (the covariance matrix). The shortest axis (associated with the smallest eigenvalue) corresponds to the "equator". Unfortunately, this only works for straight objects. Methods based on fitting planes (e.g., VSA [Cohen-Steiner et al. 2004]) suffer from the same limitation (see Figure 7).



Therefore, we need a definition of the "equator" that also works for curved geometry. To do so, our idea is to use spectral segmentation, i.e., in a certain sense, PCA in a curvilinear space, that follows the shape of the object. Spectral segmentation is a classic tool, see [Levy 2006] and [Zhang et al. 2007] for a complete survey of this family of methods. One of these methods [Rustamov 2007] uses the eigenfunctions of the Laplace operator to define a "pose-invariant" embedding, then use VSA in this space. Our idea is different. As shown in the inset, what we want to do is to find the "curvilinear version" of the cuts detected by PCA, since this detects the "equator" of the object, which is suitable to texture mapping. The next subsection explains how this can be done. Moreover, we will show that this also detects the symmetries of the mesh.

3.3 Spectral seams generation

We first need to briefly recall the definition of a graph Laplacian. Let (V,E,F) be the Vertices, Edges and Facets sets of a structured polygonal mesh, and $G=(F^*,E^*)$ its dual graph (graph of facets). The Laplacian matrix L of G is defined as $L=A-D$ where A is the adjacency matrix of the G and D is a diagonal matrix where D_{ii} is the valence of vertex i (i^{th} row sum of A). We can efficiently compute the eigenpairs (v_i,λ_i) of L following the numerical method in [Vallet and Levy 2008] (and replacing Laplace-Beltrami with the graph Laplacian). To each eigenvector v_i corresponds a bipartition

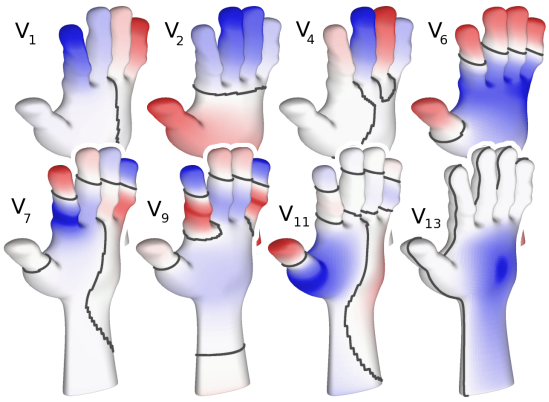


Figure 6: *Facet-graph Laplacian eigenvectors. The eigenvector v_{13} corresponds to the "equator". The undesired oscillatory ones can be filtered out by analyzing their nodal sets (red and blue zones).*

of the set F of facets into $F_i^+=\{f\in F|v_i(f)\geq 0\}$ and $F_i^-=\{f\in F|v_i(f)<0\}$. Thus we will call "seam" related to v_i the set of edges $E_i^0=\{e^*(f,f')\in E^*|v_i(f)v_i(f')<0\}$ where $e^*(f,f')$ is the dual edge between facets f and f'.

We prove the following theorem (see Appendix) : Given a mesh (V,E,F), the set of seams related to the eigenvectors v_i of the dual graph Laplacian L contains all the axes of symmetry of the mesh.

At this point, by similarity with PCA, one may think that we will use the first three eigenvectors of L. However, there are some complications in our (curvilinear) case : we need to compute more than just 3 eigenvectors. Figure 6 shows that one needs to compute up to the 13^{th} eigenvector to find the eigencut that corresponds to the "equator" of this hand. This can be explained as follows :the meaning of orthogonality is simple in Euclidian space. The three eigenvectors of the covariance matrix used by PCA are orthogonal, and yield orthogonal seams. In contrast, the eigenvectors of L have several means of being orthogonal. In addition, they can also wind (like sine waves). Fortunately, the unwanted oscillatory eigenvectors can be filtered-out, by noticing that they generate non-connected F_i^+ and F_i^- domains. They are redundant with eigen-

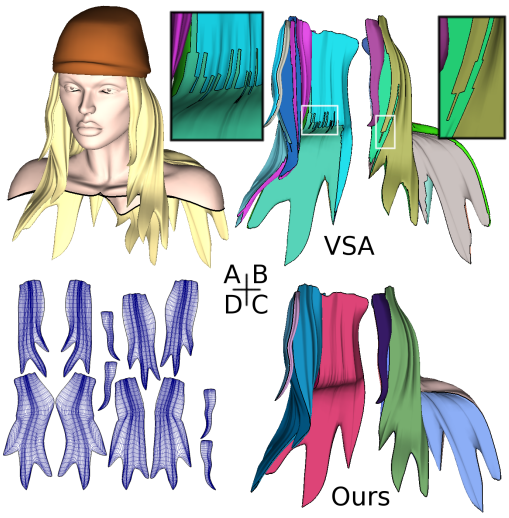


Figure 7: *Segmentation comparison: A: initial model; B: VSA generates jaggies; C: our segmentation is more natural and cleaner; D: resulting parameterization.*

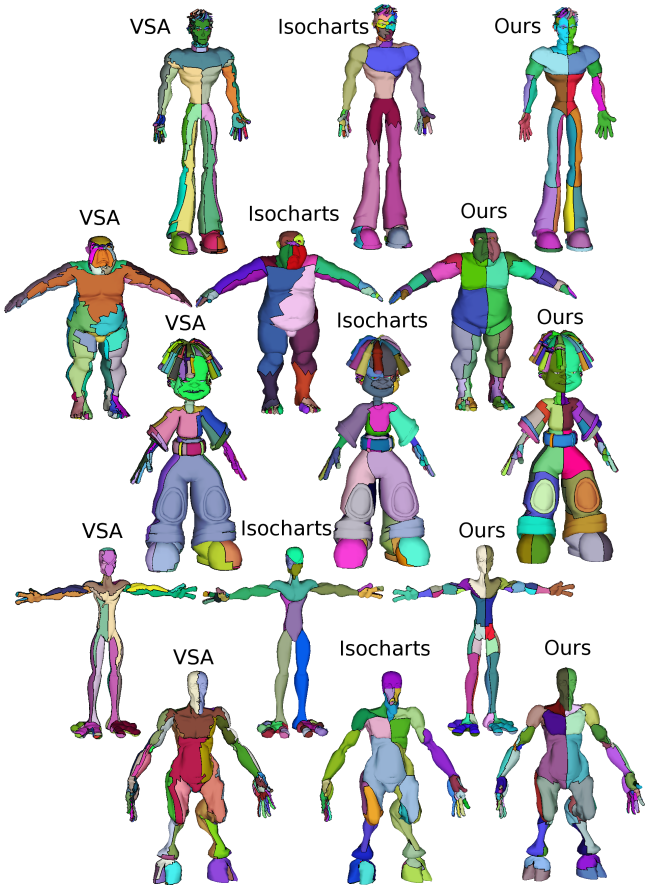


Figure 8: *Compared segmentation results (more of them in companion document).*

vectors with lower eigenvalues. More precisely, they correspond to oscillations in a direction that was already found. In contrast, if F_i^+ and F_i^- are connected, the eigenvector corresponds to a new direction, independent of the previous ones.

Therefore, only the eigenvectors that yield connected nodal domains F_i^+ and F_i^- are used for the segmentation. With this criterion, the selected eigenvectors correspond to the main extents of the surface, and can be seen as curvilinear principal directions. This point of view is strengthened by recent results [Jones et al. 2007] that prove that such principal directions always exist on geodesic disks of the surface. The choice for this criterion was also guided by the result on symmetry that we prove in Appendix A. We show that all the seams that correspond to antisymmetric eigenvectors contain the axis of symmetry. In many case, the first one corresponds to this axis and is found by our method.

We compute 30 eigenvectors. Among all the valid spectral cuts, we select the "equator", i.e., the one that has maximal length, and subdivide the mesh along its zero-set. The process is then recursively iterated on F_i^+ and F_i^- until a valid parameterization is obtained on each part. We compared our method with VSA and Isocharts on a database of 36 models (see companion document, some of them are shown in Figure 8). As can be seen, our method generates a structure-preserving segmentation. Global and local symmetries are also often captured. However, as compared to Isocharts, our method sometimes tends to oversegment (arms on the 3rd and 4th rows). As far as timing is concerned, our method never took more than 4 min. 30 to segment a model (Pentium M, 2 GHz).

4 Interactive Constrained Parameterization

Once the initial seams have been computed as explained in the previous section, the user can chose to sew, unsew and straighten them, and can set some angle constrains on the border. During the editing process, the parameterization is updated. This provides the user with direct feedback on the distortion (visualized by a checkerboard) and overlaps (visualized by red zones). This step relies on parameterization with user constraints. A good survey on parameterization can be found in [Hormann et al. 2007]. We chose to use the ABF++ [Sheffer et al. 2005] parameterization technique which offers a good compromise between angle and area distortions.

4.1 Angle Based Flattening review

The parameterization method is based on ABF++, as it already has all the machinery to implement the new constraints corresponding to the tools. We briefly recall the ABF method then explain how the constraints can be taken into account. In ABF, the variables are not the vertices' texture coordinates but angles α_t^v in texture space (angle of corner v in triangle t). Starting from the angles β_t^v measured on the 3D mesh, optimal angles ϕ_t^v are defined as:

$$\phi_t^v = \frac{2\pi\beta_t^v}{\sum_t \beta_t^v} \text{ if } v \text{ is interior, else } \beta_t^v$$

Angular deformation of the parameterization is then minimized through the energy:

$$E_{ABF} = \sum_t \sum_{v\in t} (\alpha_t^v - \phi_t^v)^2 \tag{1}$$

ABF minimizes E_{ABF} under constraints

$$\forall t\, g_1(t) = \sum_{v\in t} \alpha_t^v - \pi = 0 \tag{2}$$

$$\forall v\, g_2(v) = \sum_{t|v\in t} \alpha_t^v - 2\pi = 0 \tag{3}$$

$$\forall v\, g_3(v) = \prod_{t=(v,v',v'')} \sin(\alpha_t^{v'})/\sin(\alpha_t^{v''}) - 1 = 0 \tag{4}$$

which are enforced through Lagrange multipliers and ensure that the α's define a valid planar embedding. The constrained minimizer is found by an iterative Newton solver, and the texture coordinates can then be reconstructed by least squares minimization (not detailed here).

4.2 Tools implementation

First, since our surface is a polygon mesh (represented by a halfedge data structure), we (virtually) triangulate it. Then an initial atlas is computed using the regular ABF++ method. Once this first atlas is computed, a new parameterization is computed each time a tool is applied. Seam creation/removing and sewing/unsewing tools correspond to the regular setting of ABF++. Each time one of this tool is used, the list of variables, energy and constraints are updated accordingly. To speed-up the sewing operation, we store in each border halfedge an additional pointer refering to the opposite halfedge on the other side of the seam. Therefore, no geometric search is required to sew a seam, only simple pointer operations are done.

The other tools that allow specifying constraints can be implemented by simple modifications of ABF++'s g_2 constraints.

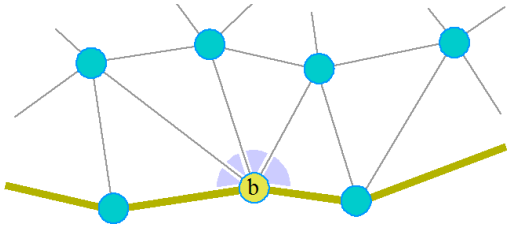


Figure 9: *Angle-on-border tool: The border angle at vertex b is constrained through the sum of angles incident to b (purple).*

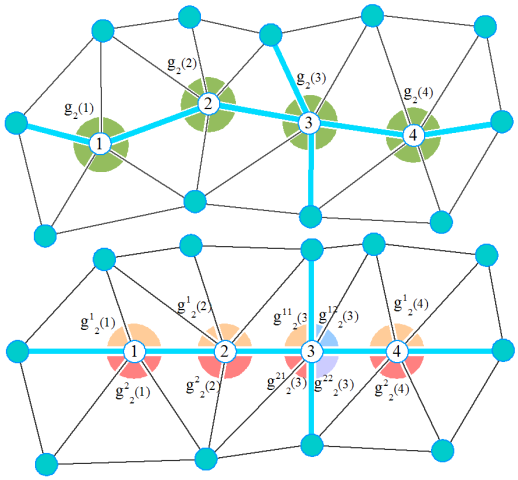


Figure 10: *Straightening tool: Blue edges are aligned by splitting the constraint g_2 into 2 constraints g_2^1 and g_2^2. If two straightening lines cross, g_2^1 and g_2^2 are split again into 4 constraints g_2^{11}, g_2^{12}, g_2^{21} and g_2^{22}*

Angle-on-border constraint : To enforce a constraint on the angle at a border vertex b (see Figure 9), we constrain the sum of angles incident to v_b by creating an additional g_2 constraint.

Straightening constraints : For the straightening tool, the alignment of two successive edges is enforced by splitting g_2 into two constraints (see Figure 10) :

$$g_2^1(v) = \sum_{t \in S_1(v)} \alpha_t^v - \pi = 0 \quad (5)$$

$$g_2^2(v) = \sum_{t \in S_2(v)} \alpha_t^v - \pi = 0 \quad (6)$$

where $S_1(v)$ and $S_2(v)$ is the partition of the set of triangles incident to v into two sets by the constrained line. Since the new constraints enforce the old ones (g_2^1 and $g_2^2 \Rightarrow g_2$), we still got a valid mapping (that in addition satisfies the straight seams). If a second constrained line crosses the first one, g_2^1 and g_2^2 are split again into four constraints (see Figure 10) :

$$g_2^{nm}(v) = \sum_{t \in S_{nm}(v)} \alpha_t^v - \pi/2 = 0$$

where $S_{n1}(v)$ and $S_{n2}(v)$ denote the partition of $S_n(v)$ into two sets by the new straightening line.

Each time the user moves the mouse (to sew an edge, or to change the constrained angle), two iterations of ABF++ are applied to update the result. It generally suffices to get a good result. When the user releases the mouse, the regular stopping criterion is used instead. Figure 11 and the companion video shows that complex

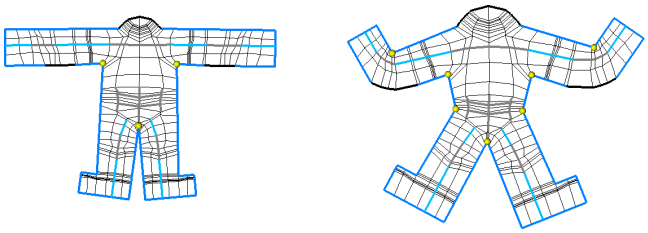


Figure 11: *With our two simple modifications, ABF++ takes a network of geometric constraints into account. Complex relations between constrained angles (yellow dots) and straight seams (blue edges) are satisfied.*

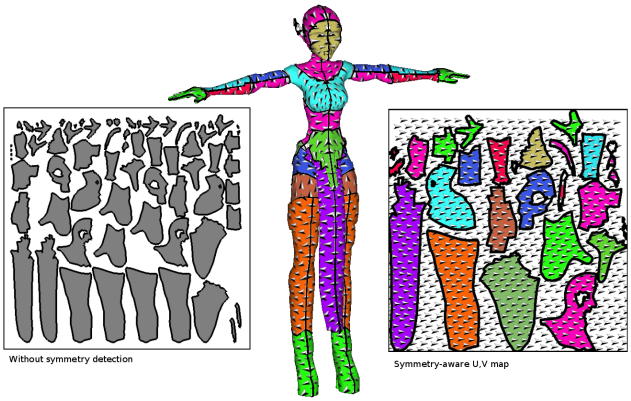


Figure 12: *Example of symmetry-aware U,V unwrapping. Left: the map obtained without symmetry detection. Right: the symmetry-aware unwrapping. Note for instance that the four halves of the legs are mapped onto a single location in U,V space (in orange).*

networks of constraints can be taken into account, which is not that surprising since in a certain sense, the regular ABF++ already considers the mesh as a network of constraints. However, to our knowledge, this modification and special use of ABF++ are original.

4.3 Symmetry-aware parameterization

As explained in Section 3 and proved in the Appendix, our set of spectral seams contains all the axes of symmetry of the mesh. Therefore, it is possible to detect symmetries and take them into account to reduce texture storage requirements. Each time a seam is cut by the algorithm, we check the graph isomorphism between both halves. In general, graph isomorphism belongs to NP, but in our case, symmetry can be checked in linear time by traversing both halves from two halfedges that were previously connected. During this traversal, pointers that connect each vertex to its symmetric vertex are stored. When symmetry is detected, both halves are mapped to a single zone of U,V space. The algorithm is recursively applied to one of the halves, and the other half copies the texture coordinates from the first one using the pointers computed at the previous step. The result is demonstrated in Figure 12 (result obtained automatically, without any user intervention). Note the similarity with the carefully designed mapping in Figure 2 in the introduction. Clearly this mechanism can be deactivated if the user wants to apply a non-symmetric texture onto a symmetric part of the mesh.

Results

Figure 13 shows a model that is difficult to unwrap with traditional tools. Indeed, many parts of the model are hidden by the legs, antennas, etc... which makes it difficult for the user to mark edges (the user would need to click on surface zones that are not visi-

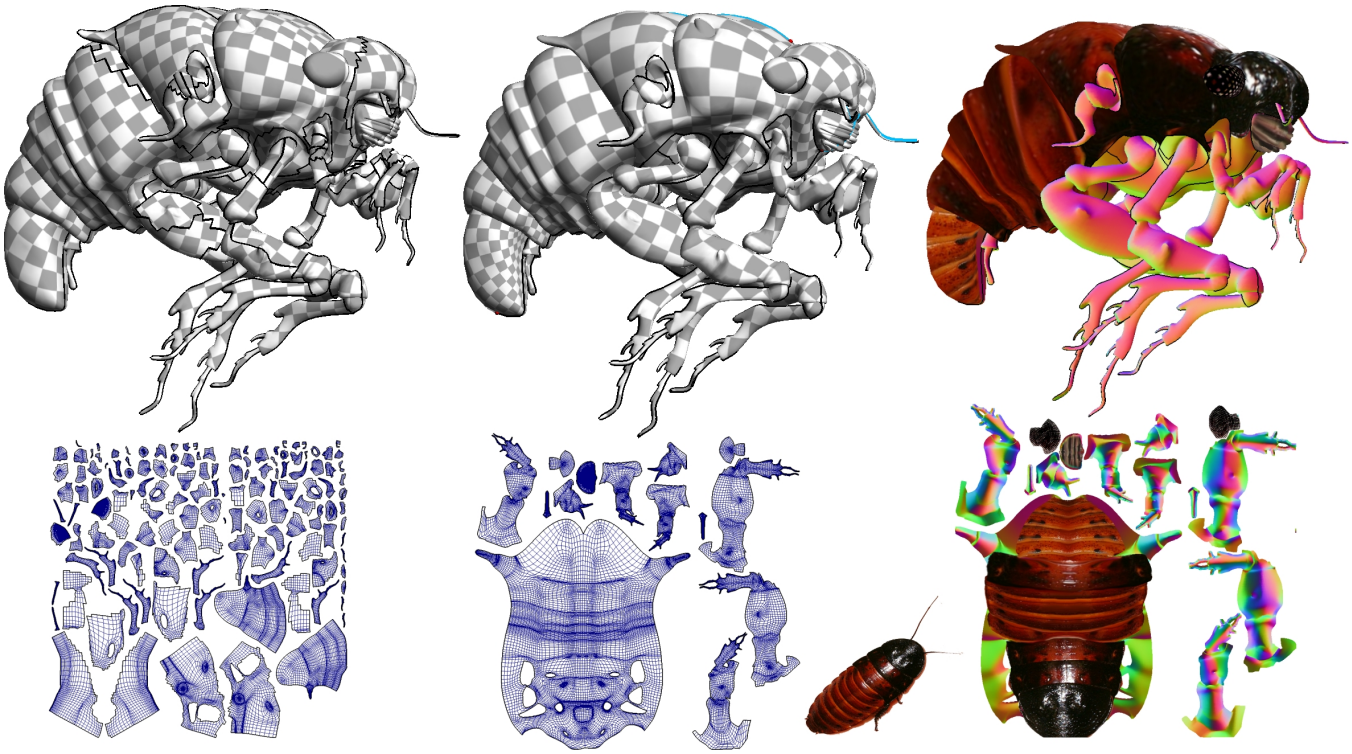


Figure 13: *UV-unwrapping a challenging model. This surface is a topological sphere. The legs make it impossible to select seams with traditional tools. Left: our system automatically constructs an initial texture atlas. Center: this unwrapping is obtained by editing the seams in 2D and in 3D. Right: model being textured, texture can be edited, copy-pasted, mirror-cloned in 2D (session time: 10 minutes).*

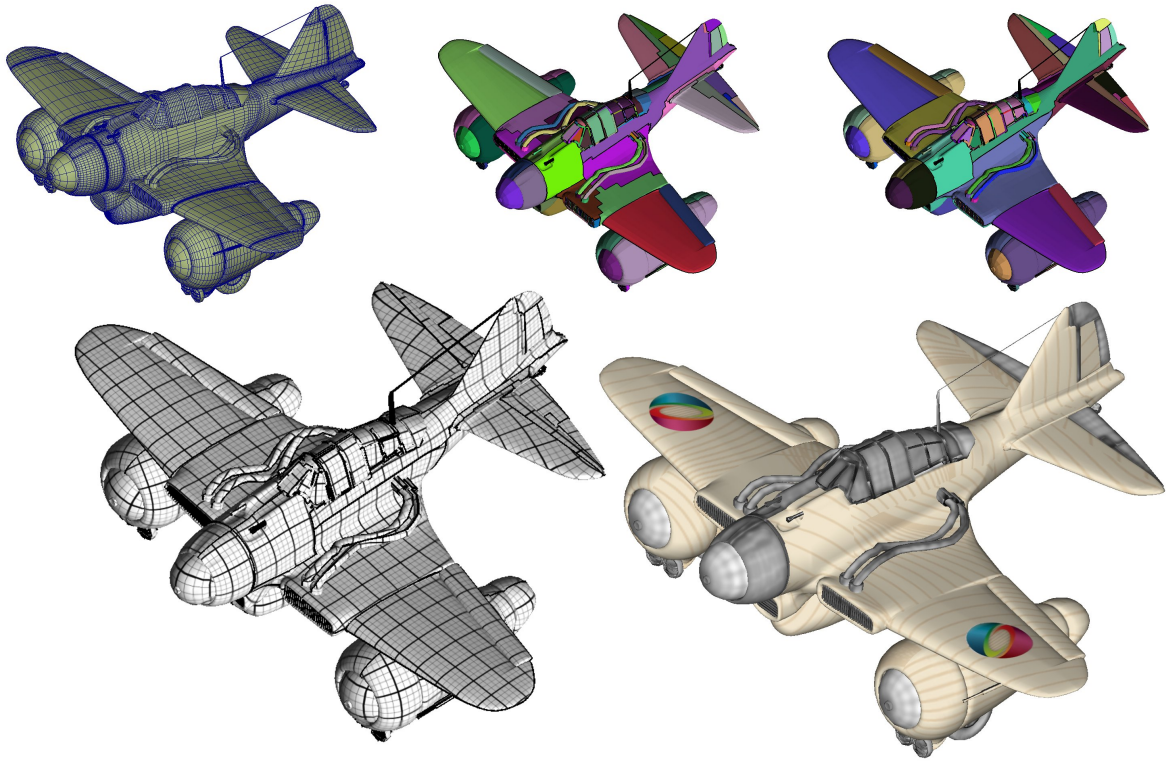


Figure 14: *UV-unwrapping a plane. Top row shows initial mesh, automatically-obtained segmentation and user-refined segmentation (session time: 3 minutes). Bottom row shows parameterization and textured model.*

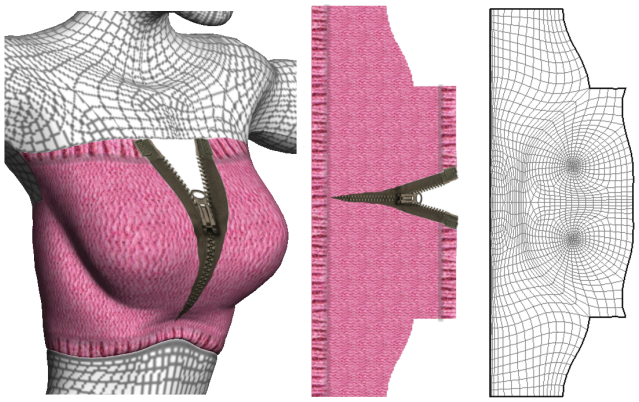


Figure 15: *An example of U,V unwrapping and texturing obtained with our method. Our structure-aware unwrapping and straightness constraints facilitate using repetitive patterns (wool) and placing structured elements (zipper).*

ble). In contrast, our tool automatically constructs an initial UV-unwrapping that is correct, and that permits initiating user interaction. Note that all the important seams (for instance the axis of symmetry, and the seams that split the legs into two halves) are detected by our initial segmentation. By selecting seams in both 3D and 2D, one obtains a good result in less than 10 minutes. Figure 14 shows another example. After the initial automatic segmentation was obtained (in less than 2 minutes), the user glued 8 seams (top row). The bottom row shows the parameterization and textured model (session time: 3 minutes). Figure 15 shows how our alignment constraints facilitates painting with patterns (wool) and elements (borders, zipper). Creating such a texture would not be possible with a 3D paint system.

Discussion and Future Works

We have introduced a semi-automatic system for UV unwrapping. Our new segmentation method automatically generates a reasonable initial mapping, that preserves the symmetry of the model, and that the user can iteratively refine using simple and intuitive tools. Our interactive parameterization is based on two simple modification of the now widely available ABF++ algorithm. Therefore, starting from an implementation of ABF++, our toolset can be easily integrated into existing 3D modeling packages. We provide in the supplemental material the C++ sources of a plugin for the open-source Graphite modeler.

Several ideas could make our method more automatic. For instance, using a visibility criterion similar to the one used by Seamster would help selecting the seams automatically. To reduce texture memory storage, besides symmetry detection, we have also experimented an extension of our framework that detects multiple instances of the same mesh by using their graph spectra (already computed by our segmentation) as a shape signature, as suggested in [Reuter et al. 2005].

Acknowledgments

Acknowledgments will be given in the final version.

References

ALLIEZ, P., COHEN-STEINER, D., DEVILLERS, O., LEVY, B., AND DESBRUN, M. 2003. Anisotropic polygonal remeshing. *ACM Transactions on Graphics (proceedings SIGGRAPH) 22*, 3 (July), 485–493.

ATTENE, M., KATZ, S., MORTARA, M., PATANE, G., SPAGNUOLO, M., AND A.TAL. 2006. Mesh segmentation - a comparative study. In *SMI*.

BEN-CHEN, M., GOTSMAN, C., AND BUNIN, G. 2008. Conformal flattening by curvature prescription and metric scaling. *Computer Graphics Forum (Proc. Eurographics) 27*, 2.

BODYPAINT. Bodypaint 3.5 relax uv enhancements. `http://www.cineversity.com/tutorials/lesson.asp?tid=925`.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. In *SIGGRAPH*.

COMON, P. 1994. Independent component analysis: a new concept? *Signal Processing 3*, 36.

DEEPUV. 3ds max + deepuv tutorial - uv mapping. `http://cg-india.com/tutorials/3dsmax_tutorials_uvmapping.html`.

HANRAHAN, P., AND HAEBERLI, P. 1990. Direct WYSIWYG painting and texturing on 3D shapes. In *SIGGRAPH 90 Conf. Proc.*, Addison Wesley, 215–223.

HORMANN, K., LEVY, B., AND SHEFFER, A. 2007. Mesh parameterization: Theory and practice. In *SIGGRAPH Course Notes*.

ISENBURG, M., GUMHOLD, S., AND GOTSMAN, C. 2001. Connectivity shapes. In *Visualization'01 Conference Proceedings*, 135–142.

JONES, P., MAGGIONI, M., AND SCHUL, R. 2007. Manifold parametrizations by eigenfunctions of the laplacian and heat kernels. In *Proc. Nat. Acad. Sci.*

JULIUS, D., KRAEVOY, V., AND SHEFFER, A. 2005. D-charts: Quasi-developable mesh segmentation. In *Computer Graphics Forum, Eurographics*.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, vol. 22, 954–961.

KATZ, S., LEIFMAN, G., AND TAL, A. 2005. Mesh segmentation using feature point and core extraction. In *The Visual Computer (Pacific Graphics)*, vol. 21, 649–658.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 362–371.

LÉVY, B. 2001. Constrained texture mapping for polygonal meshes. In *SIGGRAPH 01 Conf. Proc.*, ACM Press, 417–424.

LÉVY, B. 2005. Numerical methods for digital geometry processing. In *Israel Korea Bi-National Conference - Invited talk*.

LEVY, B. 2006. Laplace-beltrami eigenfunctions towards an algorithm that "understands" geometry. In *SMI*.

LIU, R., AND ZHANG, H. 2004. Segmentation of 3D meshes through spectral clustering. In *Proc. of Pacific Graphics*, 298–305.

LIU, R., AND ZHANG, H. 2007. Mesh segmentation via spectral embedding and contour analysis. *Computer Graphics Forum (Special Issue of Eurographics 2007) 26*, 385–394.

MAILLOT, J., YAHIA, H., AND VERROUST, A. 1993. Interactive texture mapping. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 27–34.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. Fibermesh: designing freeform surfaces with 3d curves. In *SIGGRAPH*, ACM, New York, NY, USA, 41.

PEDERSEN, H. K. 1996. A framework for interactive texturing on curved surfaces. In *SIGGRAPH*, ACM, New York, NY, USA, 295–302.

PIPONI, D., AND BORSHUKOV, G. 2000. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *SIGGRAPH*.

REUTER, M., WOLTER, F.-E., AND PEINECKE, N. 2005. Laplace-spectra as fingerprints for shape matching. In *SPM '05: Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, ACM Press, New York, NY, USA, 101–106.

RUSTAMOV, R. M. 2007. Laplace-beltrami eigenfunctions for deformation invariant shape representation. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 225–233.

SCHMIDT, R., GRIMM, C., AND WYVILL, B. 2006. Interactive decal compositing with discrete exponential maps. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, 605–613.

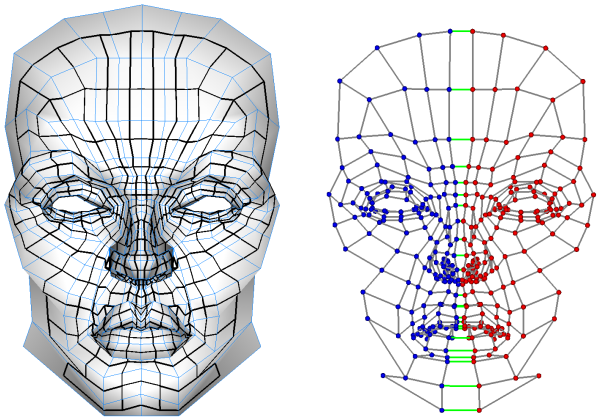


Figure 16: *Left: A quad mesh with its dual graph (black). Right: Exhibiting the graph symmetry. Edges of A' in gray, edges of A'' in green, vertices of V^1 in blue and vertices of V^2 in red.*

SHEFFER, A., AND HART, J. C. 2002. Seamster: Inconspicuous low-distortion texture seam layout. In *VIS '02: Proceedings of the conference on Visualization '02*, IEEE Computer Society, Washington, DC, USA.

SHEFFER, A., LÉVY, B., MOGILNITSKY, M., AND BOGOMYAKOV, A. 2005. ABF++ : Fast and robust angle based flattening. *ACM Transactions on Graphics - In print*.

UNFOLD3D. `http://www.polygonal-design.fr/e_unfold/videos/edgemarking_divx.avi`.

UNFOLD3D. Unfold3dmax tutorial. `http://www.polygonal-design.fr/e_unfold/3dmax.php`.

V. BORRELLI, F. CAZALS, J.-M. M., 2003. On the angular defect of triangulations and the pointwise approximation of curvatures.

VALLET, B., AND LEVY, B. 2008. Spectral geometry processing with manifold harmonics. In *Eurographics 2008*.

ZHANG, H., AND LIU, R. 2005. Mesh segmentation via recursive and visually salient spectral cuts. In *Proc. of Vision, Modeling, and Visualization*, 429–436.

ZHANG, H., VAN KAICK, O., AND DYER, R. 2007. Spectral methods for mesh processing and analysis. 1–22.

ZHOU, K., SYNDER, J., GUO, B., AND SHUM, H.-Y. 2004. Iso-charts: stretch-driven mesh parameterization using spectral analysis. In *SGP*.

A Spectral properties of symmetric graphs

In this Appendix, we study the properties of the facet graph of a symmetric mesh, in other words, the properties of symmetric graphs. We show that each axes of symmetry corresponds to an eigenvalue λ and an antisymmetric eigenvector v. By antisymmetric, we mean that for all pairs of symmetric facets (f, f'), we have $v(f) = -v(f')$. The zero-set, i.e. the set of segments that connect facets of opposite signs, corresponds to the axis of symmetry. We first give the definition related with symmetric graphs then we prove this result.

A.1 Symmetric graphs

We say that a graph G with $2n$ vertices is symmetric if there exists an ordering for its vertices such that its $(2n,2n)$ adjacency matrix can be written as :

$$A = \begin{pmatrix} A' & A'' \\ A'' & A' \end{pmatrix}$$

where A' and A'' are (n,n) symmetric matrices (see Figure 16). In other terms, the vertices set V of G can be split into two symmetric sets $V^1 = \{v_i \in V | 0 < i \leq n\}$ and $V^2 = \{v_i \in V | n < i \leq 2n\}$ such

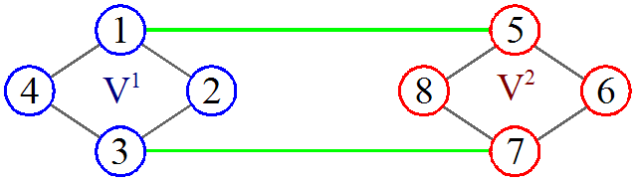


Figure 17: *A degenerate symmetric graph with $n = 4$*

that $v_i \in V^1$ and $v_{i+n} \in V^2$ will be called symmetrics. The condition on the adjacency matrix simply means that if two vertices are connected, their symmetrics are connected too. A' is the adjacency matrix within V^1 and V^2 and A'' contains the adjacencies between vertices of V^1 and V^2.

A.2 Graph Laplacian eigenvectors

Let D be the (n,n) diagonal matrix such that d_i is the valence of v_i (i^{th} row sum of $A'+A''$), then the graph Laplacian of G is given by :

$$L = \begin{pmatrix} A'-D & A'' \\ A'' & A'-D \end{pmatrix}$$

Let (f^s, λ^s) and (f^a, λ^a) be the eigenpairs of $A'+A''-D$ and $A'-A''-D$ (s and a stand for $s$ymmetric and $a$ntisymmetric). Then one can check that:

$$L\begin{pmatrix} f^s \\ f^s \end{pmatrix} = \begin{pmatrix} (A'-D)f^s + A''f^s \\ A''f^s + (A'-D)f^s \end{pmatrix} = \lambda^s \begin{pmatrix} f^s \\ f^s \end{pmatrix}$$

and

$$L\begin{pmatrix} f^a \\ -f^a \end{pmatrix} = \begin{pmatrix} (A'-D)f^a - A''f^a \\ A''f^a - (A'-D)f^a \end{pmatrix} = \lambda^a \begin{pmatrix} f^a \\ -f^a \end{pmatrix}$$

which proves that

$$\left(\begin{pmatrix} f^s \\ f^s \end{pmatrix}, \lambda^s\right) \quad \text{and} \quad \left(\begin{pmatrix} f^a \\ -f^a \end{pmatrix}, \lambda^a\right)$$

are eigenpairs of L. Thus there exists an eigen decomposition of L into half symmetric (same value on a vertex and its symmetric) and half antisymmetric (opposite value) eigenvectors. If the spectra of $A'+A''-D$ and $A'-A''-D$ do not overlap (no common eigenvalues), this eigen decomposition is unique. If they overlap, any combination of the corresponding symmetric and antisymmetric eigenvectors will also be an eigenvector. Thus in this degenerate case a solver might return eigenvectors which are neither symmetric nor antisymmetric. We never encountered such a degenerate symmetric graph on the real life models we experimented with. However we built an example with 8 vertices (see Figure 17) :

$$A' = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad A'' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

for which $(0,-1,1,0,...,0)^t$ is an eigenvector which is neither symmetric nor antisymmetric. Finally, if the graph has multiple symmetries (there are different decompositions of A in A' and A'') then the same arguments hold for each decomposition. Thus, there will exist an eigen decomposition of L where all eigenvectors will be either symmetric or antisymmetric relative to each graph symmetry, which will be unique iff the spectra of all $A'+A''-D$ and $A'-A''-D$ do not overlap.