An Automatic 3D Texturing Framework

Rupert Paget
Swiss Federal Institute of Technology
Computer Vision Group
Zürich, Switzerland
rpaget@vision.ee.ethz.ch

Abstract

In this paper we address the need to automatically texture 3D objects. The currently accepted approach is to submit the 3D object to a sophisticated 3D modelling package. Then with a lot of user know-how, correctly cut and parameterise the mesh so as to minimise the overall distortion. In the presented framework, this is completely automated, even the texture discontinuities across the mesh seams are virtually eliminated. The framework also supports multiple textures over the mesh, plus overlays and bump mapping. We even revisit pixelwise texture synthesis, and show its superiority over conventional patch-based methods. Thereby, included in the framework is a texture synthesiser, not only for creating 2D texture, but also 3D volumetric texture for the purpose of filling the 3D object.

1. Introduction

In the world of medical simulators, there is a need to automatically texture organ models. In this paper we describe a particular framework to solve this problem. These models are parametrically generated [17] by which they can describe patient specific organs, or the development of an inconsistent growing pathology like a tumour. The basic premise of this work is that these models do not have a predefined shape, and are therefore not amenable to a one off mesh parameterisation by an expert, but vary depending on the 3D surface. We also wish to texture these models as realistically as possible for various scenarios. The scenarios may vary depending on age or pathology of the organ. To obtain the correct correspondence between scenario and texture, medical knowledge is required. Therefore we provide a simple user interface between a texture database of in-vivo images and the variable 3D model, by which an untrained user can automatically texture the object to their desire, without the need to use a graphical designer each time. The framework consists of an automatic mesh cutter, parameteriser, and a seam hider, to provide a realistic texture map. Plus a 3D block synthesiser is provided for texturing new surfaces that are exposed during a medical cutting simulation. The flow of the process in our framework is depicted in Figure 1. Obviously the same framework can be applied to other 3D models that require an easy to use interface for texturing. The framework is downloadable from [10].

2. Previous work

There has been a lot of work on applying a 2D texture to a 3D surface. Some early approaches applied direct texture synthesis to the 3D surface [19, 20]. These approaches used the Markov random field method of texture synthesis [3, 11, 18] which requires a spatial neighbourhood function. To perform the synthesis over the 3D surface the spatial neighbourhood function is warped over the surface to approximate the texture distortion by the surface. However the approximations used are generally only good for short range neighbourhood functions, and quickly break down for anything but the most basic surfaces. There is also never a one-to-one correspondence between the warped neighbourhood and the surface lattice, therefore the neighbourhood pixel values need to be interpolated. This means synthesis algorithms like [1] which preserve high frequency information can not be used.

An alternative approach is to paste patches of texture directly onto the surface [7, 9, 12]. These methods perform very well and arguably better than the synthesis on surface methods. They do suffer from a lot of little texture discontinuity seams (except for [9]), but these are largely eliminated through boundary matching or alpha blending, or a combination of both. However these methods need mesh resampling to correctly support the desired texture scale, which can lead to arbitrary distortion. Mesh cutting is also dealt with in an inconsistent manner by relying on the synthesis algorithm itself. Then with any affine transformation or replacement of the texture over the surface, the texture

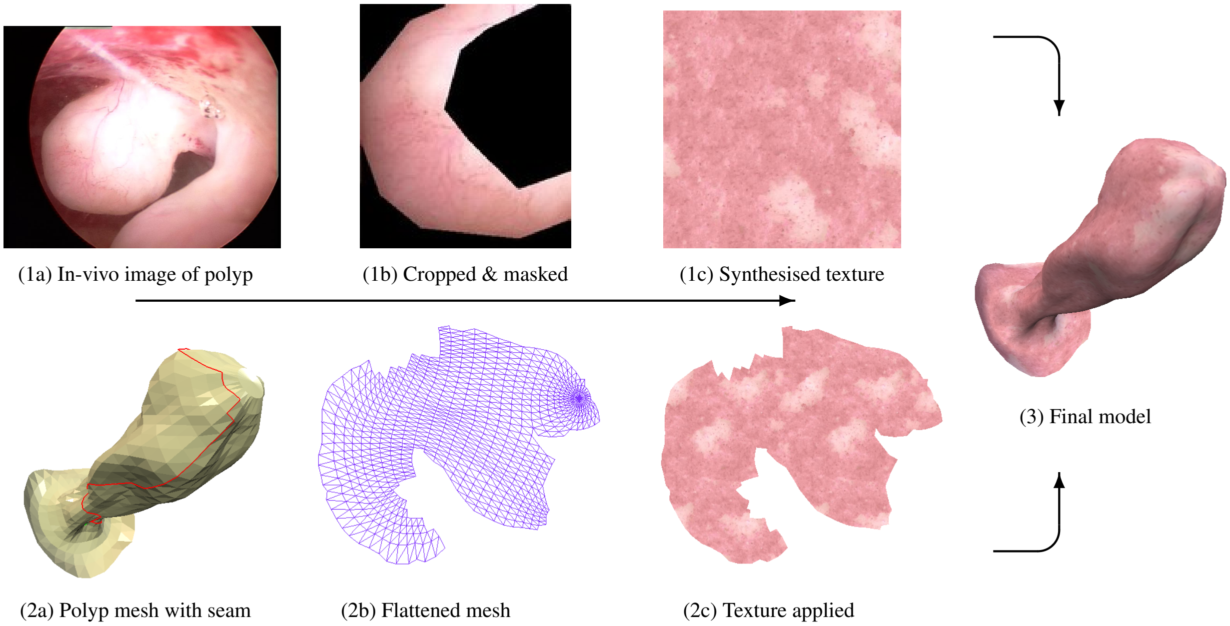


Figure 1. Model texturing process.

patches need to be recalculated, which is a hindrance to interactivity.

With the recent advances in mesh parameterisation, we have chosen a simple approach. Cut and parameterise the whole 3D surface mesh to a 2D plane. Use the inverse 2D to 3D surface mapping to project a tileable texture onto the 3D surface. Then use a similar notion as in [7] to perform precalculated alpha blending across the mesh seams. All affine transformations of the texture over the 3D surface are then just simple matrix multiplications which can easily be done interactively in OpenGL.

3. Texturing framework

3.1. In-vivo data acquisition

While in some application areas it is possible to completely synthesise artificial textures, this is not a viable approach in surgical simulation. A ground truth covering several aspects of visual appearance of organs has to be obtained, which serves as input to the texture generation chain. For this reason, 10 hours of in-vivo video taken during various hysteroscopies were acquired. Based on this material, a texture database was created. The images were selected to represent different structures in the uterine cavity. High resolution views of characteristic tissues were chosen, which were optimally perpendicular to the surface without a strong spotlight effect. We obtained 69 different samples that contained usable textures of various types of tissue. These were then cropped and masked to obtain just the desired surface regions.

3.2. Texture synthesis

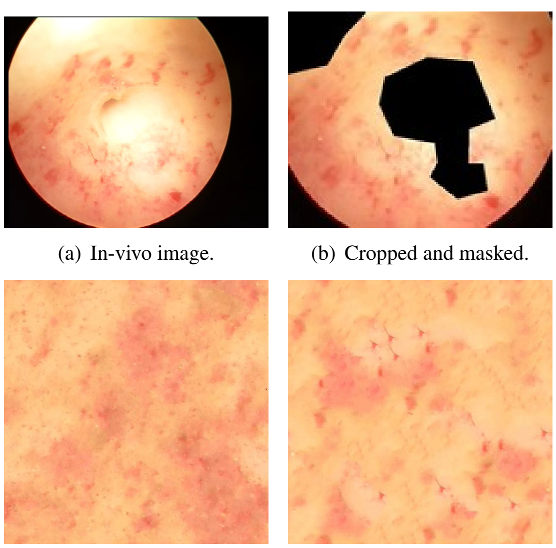
Arbitrary mapping of the in-vivo textures to a 3D surface requires the textures to be tileable. This means that the textures need to be resynthesised. Existing texture synthesis algorithms can be basically sorted into two categories. Pixel-based approaches [3,11,18] and patch-based methods [4,8]. We decided to use the former, synthesising the required textures with a fast nonparametric pixelwise texture synthesis (FNPTS) algorithm. It is a modified version of the multi-scale texture synthesis algorithm presented in [11] sped up with a neighbourhood searching scheme similar to the one proposed in [1]. The Manhattan distance summed over the N neighbours was chosen as the neighbourhood cost function:

$$\text{Cost} = \sum_{i}^{N} \begin{cases} |s_i - r_i| \frac{n_i}{M} & \text{if } s_i \text{ is defined} \\ \text{Max} \frac{n_i}{M} & \text{otherwise} \end{cases} \quad (1)$$

where $s_i \in S$ are pixels in source image (s_i may be undefined if it is outside the image, or is masked), and $r_i \in R$ are pixels in the output image. The factor n_i/M equals the number of times the output pixel has been iterated divided by the proposed number of iterations per pixel. If s_i is undefined, then a cost, equal the maximum possible pixel cost, is added to the overall neighbourhood cost.

The cost function is calculated over a subset of sampling pixels from the source image. In contrast to [1], the subset we sample from are all pixels with a neighbour of the same colour as the respective neighbour of output pixel being iterated. The pixel that returns the lowest neighbourhood cost is chosen as the one to transfer its colour to the respective pixel in the synthetic image. The iteration count is then incremented for that output pixel. Once all output pixels have reached their proposed number of iterations, the synthesis process stops. The site visitation sequence across the output pixels can be in any random order as described in [11].

Compared to patch-based algorithms (e.g. [4, 8]), our pixel-based approach produces more stochastic variations in the synthetic texture, while providing comparable fidelity. Finally, due to its pixel-based strategy, our method can easily cope with image masking (a high desirability when dealing with low quality source textures). Using FNPTS, we were able to produce a variety of different large scale tileable tissue textures from our database. A small example can be seen in Figure 2.



(a) In-vivo image. (b) Cropped and masked.

(c) Synthesised texture I. (d) Synthesised texture II.

Figure 2. Synthesis of in-vivo textures

3.3. Texture block synthesis

As part of the texturing framework, we also created a 3D texture synthesis algorithm. It basically works the same as FNPTS, but the output is a 3D block of texture. The only alteration to the algorithm is that the 2D output neighbourhood is supplemented with two extra neighbourhoods of equal size, so that there exists one 2D output neighbourhood in each of the three axis planes with in the 3D block. To assign a colour to an output pixel, the neighbourhood cost function is calculated separately for each axis plane. The input pixel that minimises the sum of these cost functions is the one that is chosen to colour the output pixel.

3D block texturing is used in the texturing framework to texture newly exposed surfaces during a cutting procedure within the surgical simulator. This is convenient, as no 2D parameterisation of the surface is required, and therefore can be textured in real time. One may wonder why this technique is not used for texturing the 3D surface of organs. There are two major restrictions in synthesising a 3D texture block. One: the source texture has to be isotropic. There are 2 axes for the source texture, but 3 for the 3D block. Therefore at least one plane in the 3D block will correspond to a source texture with equally labelled axes. Two: the source texture also needs to be homogeneous, such that any cut through the 3D block of texture will represent a stochastic version of the 2D source texture. This means that this method will fail for say a source texture of circles with equal radii. The expected 3D texture block would be a block of spheres, in which case any cut through the block would generate circles of variable radii. However, as all but the largest circles are not present with in the source texture, the synthesis method will fail. Another disadvantage with 3D block texturing is the greater texture memory required for an equivalent 3D surface texture resolution.

3.4. Mesh parameterisation

Mesh parameterisation is carried out by first cutting the 3D mesh so that it can be ”unfolded” onto a 2D plane. When deciding how to cut the mesh, one first needs to choose a parameterisation method. Basically mesh parameterisers fall into two categories [5]; ones that require a fixed convex boundary, and those that do not. The ones that allow for a free boundary typically provide reduced distortion and require fewer seams over the mesh. Of these types there are basically two that at least partially guarantee that no triangle flips will occur. There is the angle based flattening (ABF) [14,15], and then there is the least squares conformal maps (LSCM) [6], but as discussed in [14], ABF performs a lot better.

3.4.1. Mesh cutting

In this work we have chosen to use the Seamster method [16], from which the surface is separated along existing edges. This process is carried out according to two quality metrics. Firstly, a visibility measure for mesh edges is calculated. This is done, by rendering the scene orthographically from different viewpoints and recording how many times each element is visible. Secondly, a distortion measure is calculated at each mesh node by estimating the Gaussian curvature.

The nodes that contribute significantly to the total mesh distortion are selected. These nodes are then connected by minimally visible seams. This problem is known to be NP-Complete [16]. However a viable solution can be obtained with the approximate minimal Steiner tree (MST) algorithm [13]. A number of alternatives exists in calculating the edge cost. We chose to supplement the calculation with a curvature cost as described in [6], giving final cost for each edge as:

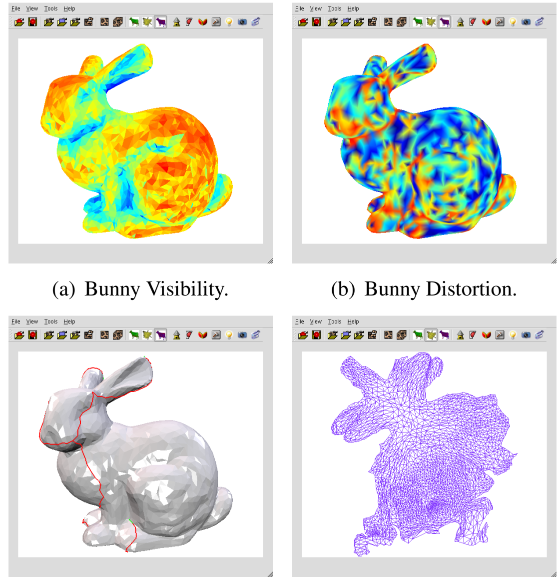
$$e_{cost} = |e| e_{visibility} (1 - e_{curvature})$$

This helps to place cuts along edges of high curvature, in which texture artefacts are less visible. Note though, the curvature measure is not the same as the distortion measure.

3.4.2. Angle based flattening

After cutting, the mesh is ready for flattening and texture parameterisation. We used the angle based flattening [15] to flatten the mesh to a 2D domain. The algorithm minimises the sum of relative square distances between the angles in the original mesh and that of a flattened mesh. A set of constraints guarantees the validity of the flat mesh. The resulting constrained minimisation problem is formulated using Lagrange multipliers and solved using Newton-Raphson method. At each iteration a large sparse linear system needs to be solved for which we used the direct solver SuperLU [2]. We also implemented the matrix variation as described in [21]. This creates a simpler matrix, but can become ill-conditioned [14]. Employing line searching and backtracing to the Newton-Raphson method does well in circumventing this problem.

After solving for the 2D planer angles, vertices need to be extrapolated to a 2D plane. The traditional method of growing the mesh by using the angles to find the 3rd vertex in each triangle can quickly become unstable from a lack of numerical precision. Therefore we used the global optimiser according to [14]. For each triangle a complex angle can be calculated which can be quickly and efficiently used to calculate globally optimal positions for the vertices. An example of this process for a mesh of the Stanford Bunny is shown in Figure 3.



(a) Bunny Visibility. (b) Bunny Distortion.

(c) Bunny Cut. (d) Bunny Flattened.

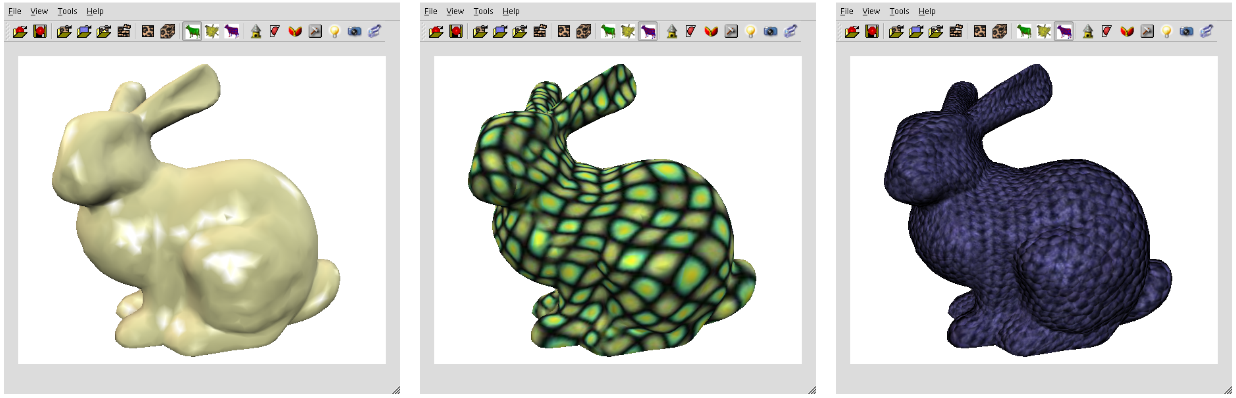
Figure 3. Seamster mesh parameterisation.

3.5. Texture blending

In parameterising the 3D surface mesh, it was cut and flattened. Then, a texture was applied to the flattened mesh, which thereafter was backprojected onto the 3D surface. With this method a texture discontinuity exists along the cut seam of the mesh. Although number, length and visibility of the seams have been minimised, these discontinuities still create unwanted visual artefacts. To reduce these artefacts, we apply an enhancement step based on alpha blending. This is done with duplicate triangles along the seams that contain the extrapolated texture coordinates from across the seam. Vertices directly on the seam are given an alpha value of 0.5, while the rest are incrementally decremented as the overlayed triangles progress away from the edge. Alpha blending is then easily performed in OpenGL by setting the material properties of each vertex of the overlayed triangle to the corresponding alpha value.

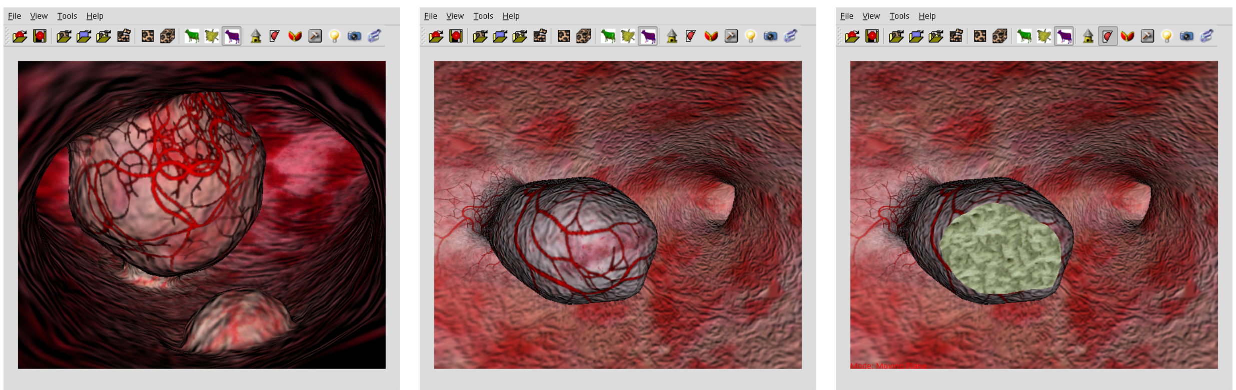
The discussion so far has only focused on texturing of a single mesh with a single texture. However, in our scene generation, a number of different structures exist, mainly healthy anatomy and neoplasms. In the specific case of hysteroscopy, we artificially grow myomas and polyps in the uterine cavity. Due to the differing genesis of these objects, they usual have different visual/textural appearance. In which case, we separate the mesh into its different facets, and process each one separately. Texture distortion is therefore minimised over each facet, and all that is required is blending between facets.

Blending between facets is a lot more straightforward than blending within an object. Basically we define our complete scene with a set of submeshes, where each submesh encapsulates the surface of one facet covered by just one texture. However for each submesh we also include an overlap region between meshes. These submeshes are then processed individually and texture is applied. The overlap region contains the necessary information to apply alpha



(a) Bunny. (b) Bunny textured. (c) Bunny textured with bump mapping.

Figure 4. The Stanford Bunny



(a) Myomas. (b) A fundus polyp. (c) Underneath fundus polyp.

Figure 5. Hysteroscopy simulator textured models

blending between objects. It is easy to make this overlap region quite large, which improves the blending across different textures.

4. Results

In Fig. 4 (a) the Stanford Bunny has been loaded. With just one click the bunny is cut, and parameterised and ready for texturing. Total time; less than one minute. In Fig. 4 (b) a tileable texture is simply loaded onto the bunny. Mouse interaction allows the texture to be scaled, rotated, or translated over the 3D surface of the bunny. The affine transforms are performed in realtime as they are simple 4x4 matrix operations within OpenGl's glMatrixMode(GL_TEXTURE). Even the alpha blending remains consistent through such operations. Fig. 4 (c) shows that bump mapping can also be handled within the framework.

The automatic 3D texturing framework was developed as a tool to allow surgeons to texture their own medical simulator scenes. In Fig. 5 (a) we show a textured scene for a hysteroscopy simulator. In this scene, each myoma is represented with its own texture. As can be seen, the texture of the myomas blends right into the texture of the uterus wall. No texture discontinuities can be seen. In Fig. 5 (b) we show an example of a polyp. The undersurface of the polyp is textured with 3D block texturing, as seen in Fig. 5 (c). This allows any cuts or interventions made into the polyp to be consistently textured. Figures 5 (a) and(b) also show the application of bump mapping as well as overlay textures (blood vessels).

5. Conclusions

We have presented a framework for automatically mapping 2D textures to a 3D surface. Other features are also available within the framework, including bump mapping, and the application of overlay textures. All that is required is a few simple and intuitive mouse clicks. The texture that is mapped to the surface needs to be tileable for which we also provide a fast nonparametric pixelwise texture synthesiser. The synthesiser is superior to conventional patch-based synthesisers, as it can easily handle masking of the source texture. Such a feature is advantageous when the source texture is of low synthesis quality (i.e. not completely of one texture, and is for example corrupted with reflections or oblique viewing angles). The same texture synthesiser was extended to 3D block texture synthesis of homogeneous isotropic textures for texturing newly presented surfaces exposed during cutting or intervention within the surgical simulator.

The presented texturing framework was developed for a surgical simulator, so that surgeons could easily apply their knowledge when designing the surgical scenes. However the framework is generic, and not limited to such tasks. Anyone wishing to texture their own models within a simple graphical user interface may simply download it from [10].

Acknowledgement

The author would like to thank all members of the hysteroscopy simulator team. This research has been supported by the NCCR Co-Me of the Swiss National Science Foundation.

References

[1] M. Ashikhmin. Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226, New York, NY, USA, 2001. ACM Press.

[2] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.

[3] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision*, volume 2, pages 1033–1038, Sept. 1999.

[4] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, New York, NY, USA, 2001. ACM Press.

[5] M. S. Floater and K. H. N. A. Dodgson. Surface parameterization: a tutorial and survey. In M. S. Floater and M. A. Sabin, editors, *In Advances in Multiresolution for Geometric Modelling*, pages 259–284. Springer, 2004.

[6] B. Levy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 362–371, New York, NY, USA, 2002. ACM Press.

[7] S. Magda and D. Kriegman. Fast texture synthesis on arbitrary meshes. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 82–89, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[8] A. Nealen and M. Alexa. Hybrid texture synthesis. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 97–105, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[9] F. Neyret and M.-P. Cani. Pattern-based texturing revisited. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 235–242, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[10] R. Paget. Texture synthesis and analysis. `http://www.texturesynthesis.com`, 2002.

[11] R. Paget and D. Longstaff. Texture synthesis via a noncausal nonparametric multiscale Markov random field. *IEEE Transactions on Image Processing*, 7(6):925–931, June 1998.

[12] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 465–470, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[13] A. Sheffer. Spanning tree seams for reducing parameterization distortion of triangulated surfaces. In *SMI '02: Proceedings of the Shape Modeling International 2002 (SMI'02)*, page 61, Washington, DC, USA, 2002. IEEE Computer Society.

[14] A. Sheffer, L. Bruno, M. Maxim, and B. Alexander. Abf++ : Fast and robust angle based flattening. *ACM Transactions on Graphics*, 2004.

[15] A. Sheffer and E. de Sturler. Parameterization of faceted surfaces for meshing using angle based flattening. *Engineering with Computers*, 17(3):326–337, 2000.

[16] A. Sheffer and J. C. Hart. Seamster: inconspicuous low-distortion texture seam layout. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 291–298, Washington, DC, USA, 2002. IEEE Computer Society.

[17] R. Sierra, M. Bajka, C. Karadogan, G. Szkely, and M. Harders. Coherent scene generation for surgical simulators. In *Medical Simulation: International Symposium*, pages 221 – 229, 2004.

[18] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH 2000, 27th International Conference on Computer Graphics and Interactive Techniques*, pages 479–488, 2000.

[19] L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 355–360, New York, NY, USA, 2001. ACM Press.

[20] L. Ying, A. Hertzmann, H. Biermann, and D. Zorin. Texture and shape synthesis on surfaces. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 301–312, London, UK, 2001. Springer-Verlag.

[21] R. Zayer, C. Rossl, and H.-P. Seidel. Variations on angle based flattening. In *Proceedings of Multiresolution in Geometric Modelling*, pages 285–296, 2003.