Efficient Texture Mapping by Homogeneous Patch Discovery

R.Vikram Pratap Singh Anoop M.Namboodiri

Center for Visual Information Technology, IIIT Hyderabad, India.
vikrampratap.singh@research.iiit.ac.in, anoop@iiit.ac.in

ABSTRACT

Texture mapping algorithms use mesh parameterization methods to find an optimal map for the vertices of a 3D model in texture space. These techniques vary in the properties they try to optimize such as stretch and skewness of the texture when mapped onto the surface. While most of them do well in terms of quality, they tend to be computationally intensive for large mesh models, which limits their use in interactive applications. We propose a greedy alternative that is significantly faster than current algorithms and achieves comparable quality. We use a priority queue to store polygons and use tangential vectors to guide the texture over the surface. Our algorithm is simple to implement and can texture over a million polygons per second on a typical desktop. The algorithm does not impose any constraints on the mesh topology and we do not require the model to be cut into patches before texturing. Stretch and distortion measures are stable across models and are comparable to current algorithms. We also propose a method to generate self tile-able textures for use in conjunction with our texture mapping algorithm. We present qualitative and quantitative results in comparison with several other texture mapping algorithms. The efficiency and robustness of our algorithm makes it useful in interactive modeling applications and texture mapping large mesh models such as heritage monuments.

Keywords

Texture mapping, Mesh parameterization, Texture synthesis

1. INTRODUCTION

Any complex mesh model used in 3D graphics can be converted to a 2D planar sheet of vertices with appropriate stretching and cutting. Mesh parameterization is a bijective function $f : R3 \rightarrow R2$ that maps the 3D vertices onto a 2D plane. The parameterization should ideally be isometric, preserving the distances between vertices and the angles of the triangle. The goal of a mesh parameterization algorithm is to minimize the distortion of the triangle in terms

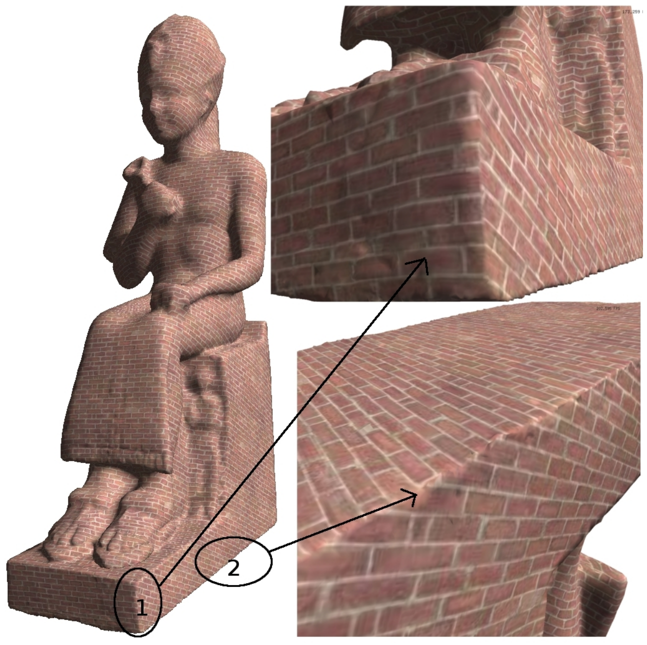


Figure 1: Texture patch edges that are pushed to the geometric edges of the 3D mesh.

of the angle (conformal maps) or the area (authalic maps) to reduce the amount of skewing or stretching of the texture. Such parameterizations have applications in texture mapping, re-meshing, morphing, mesh editing, mesh compression etc. We refer to mesh parameterization only in conjunction with texture mapping in all our further references. Unfortunately, except for synthetic surfaces that are inherently planar, it is difficult to flatten a 3D mesh without any distortion. The distortion is particularly high if the surface has got high curvature. We can reduce the distortion by cutting the mesh into patches at points of high curvature. However, increasing the number of cuts would introduce more discontinuities in the pasted texture, leading to deterioration in the perceived quality. A possible approach to minimize the visual impact of the cuts is to constrain them to edges of the 3D model (see Figure 1). A good mesh parameterization algorithm must optimize between the size of the texture patch and the amount of distortion within the patch. There are three types of mesh parameterization approaches: Planar, Spherical and Hierarchical.

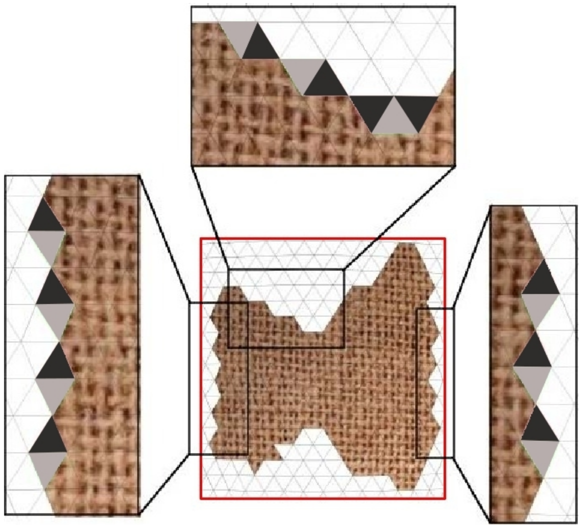


Figure 2: **Middle image shows a partially textured mesh model. Enlarged portions of the image show faces of the mesh at the boundary of the texture, labeled in brown and gray colors alternately. Among these faces, if we texture any brown colored face, neighboring gray colored faces will have all the three vertices textured and vice versa.**

Planar mesh parameterization: Parameterization of a planar mesh surface can be defined as bijective mapping of the mesh vertices onto a 2D plane, such that there is no overlapping and distortion of the triangles. Surfaces with the topology of a disk can follow such parameterization. A closed surface like a sphere (surfaces with zero genus) cannot be flattened to a sheet. So the mesh has to be cut into patches that are homeomorphic to a disk and then parameterized independently to map them to a 2D plane. A surface with high genus would require large number of cuts. Fixed boundary approaches flatten the boundary of the patch to a predetermined simple convex polygon like a square or circle. Here the optimization function minimizes the distortion of the triangles with the constraints of fixed boundary vertices.

Floater [5] takes such an approach and uses barycentric coordinate system to represent an interior vertex property by interpolating properties from its neighboring vertices. This type of parameterization typically generates significantly more distortion than free boundary techniques.

Yoshizawa *et al.* [22] starts with the method described in [5] and uses an iterative approach to decrease the error at each step. The ABF [15] algorithm by Sheffer *et al.* uses the Lagrangian function for optimizing the distortion by enforcing constraints on angles. Later, [16] and [23] improved on the speed of this approach. LSCM [8] uses least squares approximation of Cauchy-Riemann equations to produce quasi-conformal parameterization of the mesh. Ray *et al.* [12] proposes a hierarchical version of LSCM. Liu *et al.* [9] introduces ASAP and ARAP methods. While ASAP has results similar to LSCM [8], ARAP produces better parameterization results.

Spherical mesh Parameterization: Geometric models are often described by closed, genus-zero surfaces. For such models, the sphere is the most natural parameterization domain, since it does not require cutting the surface into disks. Here we parameterize a triangular mesh onto a unit sphere based on an optimality condition. This assignment should reduce the distortion induced into the spherical triangles and there should be no overlap. For complex and highly deformed surfaces, generating a low distortion and non overlapping parametrized spherical triangles is difficult. Gotsman *et al.* [6] introduced the concept of spherical parameterization. Praun *et al.* [11] and Saba *et al.* [13] further improved the method. Octahedral mesh parameterization was proposed by Praun and Hoppe *et al.* [11], where an octahedron is used instead of a sphere.

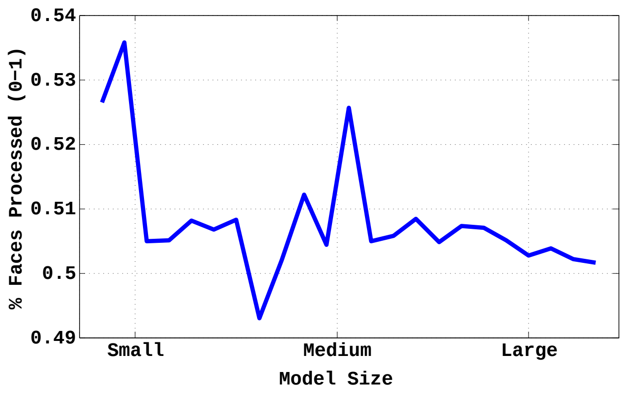


Figure 3: **Proportion of faces that are explicitly textured with increasing model size.**

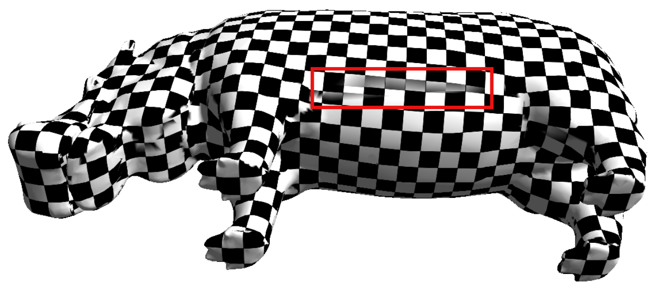
Hierarchical mesh parameterization: Hierarchical mesh parameterization is done by calculating parameterization over a coarser mesh consisting of a small number of triangles, which is then refined up the hierarchy. Lee *et al.* [7] and Sander *et al.* [14] use hierarchical mesh parameterization for texture mapping. Sander *et al.* [14] uses planarity and compactness heuristics for parameterization and texture packing.

Texture Synthesis: Texture synthesis is another area where parameterization finds its application. Traditional texture synthesis methods approach the problem in raster scan order, considering the boundary of already synthesized regions. In order to carry out synthesis on arbitrary surfaces, they need to be mapped to a planar surface using a mesh parameterization technique. Wei *et al.* [20], Ying *et al.* [21] and Praun *et al.* [10] use this approach to do texture synthesis directly on mesh models.

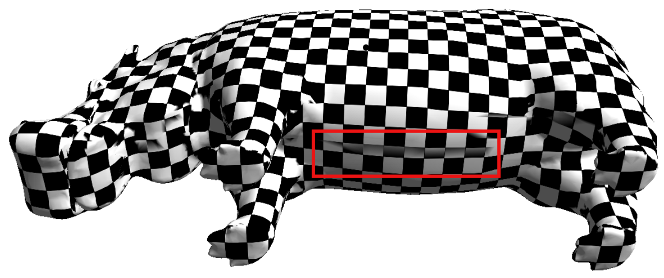
All the above methods minimize some criterion function to define a bijective function $f : R3 \rightarrow R2$ that maps the 3D vertices onto a 2D plane. Our algorithm avoids an explicit criterion optimization and tries to minimizes the stretch of the texture as it is grown over the 3D mesh.

2. PATCH DISCOVERY MAPPING

We present the algorithm for triangulated meshes, although it can be extended to quad or any polygonal meshes. We start by texturing a random face, by dropping it into the texture plane of the face, calculated from the orientation



(a)



(b)

Figure 4: The seam that is formed on the hippo's side in (a) can be moved to a less visible position at the bottom by manually marking a seam line using our tool, resulting in (b).

vector and bi-normal vector of the orientation vector and normal vector of the face. We then proceed to texture the triangles around the initial one, expanding the texture by incrementally pushing the texture boundary. The boundary of textured region is pushed in the direction of homogeneous orientations, stopping at faces with large change in orientation vectors. The process is implemented efficiently using a priority queue for candidate faces to be textured, where we use *age* of the face to strike balance between homogeneity of the region and compactness of the boundary. Our algorithm processes the mesh in terms of faces starting with a randomly selected face, which is a triangle in our discussions. The initial triangle ΔABC, is textured by assigning any one of its vertices to a random texture coordinate, say A is assigned the texture coordinates of the center of the texture space.[1] Now vertices B and C are projected onto the texture plane of the current face and the vectors $\overrightarrow{ab}$ and $\overrightarrow{ac}$ are calculated. Texture coordinates of B (b_x, b_y) are calculated as follows.

$$b_x = (a_x + \overrightarrow{ab}_x) \tag{1}$$

$$b_y = (a_y + \overrightarrow{ac}_y) \tag{2}$$

Similarly, we calculate texture coordinates for the vertex C. We have now textured the first triangle in a direction consistent with that of its orientation vector. Now we spread this texture over to the adjacent faces in the direction of orientation vectors. We can compute coherent orientation fields using a variety of methods, including Tangent vector

[1]Upper case letters refer to geometric space and lower case letters refer to texture space

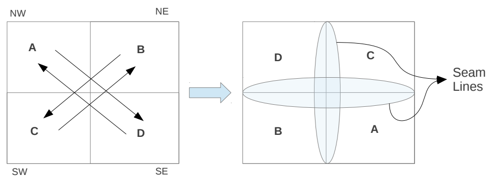


Figure 5: Quadrant swapping for creating the tileable textures and resulting seam-lines.

fields suggested by Fisher *et al.* [4]. As we are interested in increasing execution speed of texture mapping algorithm, we have used a simpler approach to compute the orientation field of the mesh model. We compute the principal direction of stretch of the mesh model from PCA of the 3D vertices. The orientation vector of each triangle is taken as the projection of the principal direction onto the triangle. For each face, the sum of angle differences made by the current face orientation vector with the orientation vectors of neighboring faces are precomputed and stored. From now onwards, we refer to this value as the *age* of the face as the reason for this would seem obvious during the further reading. The neighboring faces of the textured face are pushed into a priority queue using the *age* as priority (larger age indicates lower priority). We now pick the face from the top of the heap for texturing. The reader may note that for a single connected mesh, any face picked out of the queue will have either two or three of its vertices textured. If there are n disconnected components, there will be $n-1$ faces while processing which have zero vertices textured.

Faces with three textured vertices in the queue should be given high priority to be textured first. If all three of its vertices are textured, then there is nothing else to do, but declare the face as textured and remove it from the queue, while pushing its neighboring faces into the priority queue. This step helps us in maintaining the scale of the texture, while avoiding any stretch. In practice, about 50% (see Figure 3) of the faces are textured in this way. This happens because if a face has two of its neighboring faces already textured, then it will have all its three vertices textured. This observation can be clearly seen in Figure 2. If the triangle to be textured has two of its vertices assigned, then we project both the vertices onto the texture plane of current face. Lets assume that we are texturing the ΔABC where A, B and C are its vertices. Vertices A, B are textured and C is not. The texture coordinates of the third vertex $C(c_x, c_y)$ are computed as below.

$$c_x = \frac{(a_x + \overrightarrow{ac}_x) + (b_x + \overrightarrow{bc}_x)}{2} \tag{3}$$

$$c_y = \frac{(a_y + \overrightarrow{ac}_y) + (b_y + \overrightarrow{bc}_y)}{2} \tag{4}$$

where a_x, a_y are texture coordinates of vertex A and b_x, b_y are texture coordinates of vertex B. The overall process is summarized in the Algorithm 1. In equations (3) and (4), we average the texture coordinates suggested by the two vertices. If the texture coordinates suggested by the vertices differ beyond a value *errorThreshold*, we do not process the

Algorithm 1 Proposed Algorithm

```
Input: Mesh model
Output: Texture coordinates assigned to vertices
$Factor := 1/(number\ of faces)$;
while priority queue is not empty do
    Pop the face with least $age$ from the priority queue,
    giving higher priority to faces with 3 vertices textured.
    if $error > errorThreshold$ then
        Enqueue face again with $age$=$age + LargeVal$.
    else
        Process the face as explained in Section 2.
        Enqueue neighbors with $age = age + Factor$.
        $Factor := Factor + 1/(numberoffaces)$;
    end if
end while
```

face but push it back onto the queue with its *age* increased by a $largeVal$. Value of the variable $largeVal$ can be set to anything that will push the face to the end of the queue. This avoids the processing of such faces and hence propagation of distorted features is avoided. If the suggested texture coordinates are within the $errorThreshold$, we assign the average of those texture coordinates and push the neighboring faces in the priority queue. After processing, we decrease the *age* of all the faces in the priority queue by some $Factor$. We took this $Factor$ to be $1/(number\ of faces)$. By doing so we wish to process those faces that have stayed in the queue for long time and hence they are given high priority to be textured next. This helps in smoothly developing the texture over the surface without leaving any holes.

Decrease in the length of the texture patch boundary can be observed, as there is a potential possibility that these small holes left untextured may grow into bigger holes, causing an increase in the texture patch boundary. Operations on the priority queue are the core part of our algorithm. In order to avoid decreasing the *age* of all faces in the queue(time complexity $O(n)$), we increase the age of the faces that are being pushed on the priority queue incremented by $Factor$. The value of $Factor$ keeps incrementing in every iteration by a value $1/(number\ of faces)$. Instead if we increment by a value larger than $1/(number\ of faces)$ say $100/(number\ of faces)$, the algorithm will lose its capacity to form texture patches at geometric edges as the new faces introduced into the priority queue will always be inserted at the last and hence acts like a normal queue. Flow of texture will be in a spiral fashion around the starting face. This will introduce a lot of texture distortion, but will decrease the execution time.

Decrease in execution time depends on two things: 1)Insertion into the priority queue and 2)Deletion from the priority queue. In the above case with large increment in $Factor$ value, faces are inserted always at the back of the priority queue, as it will make only one comparison at the end of the heap and gets inserted there. Deletion includes deleting the top of the heap and a re-heap of the priority queue which is of $O(logn)$ complexity. On a whole the time required for execution will decrease. The value of the increment is critical to the proper functioning of our algorithm. $errorThreshold$ used in the Algorithm 1 is initially set to 25 $pixels$ and if all the faces left in the priority queue are giving a error more than $errorThreshold$, we increment it by 10 $pixels$ and restart the algorithm. This would avoid a potential infinite loop in the algorithm.

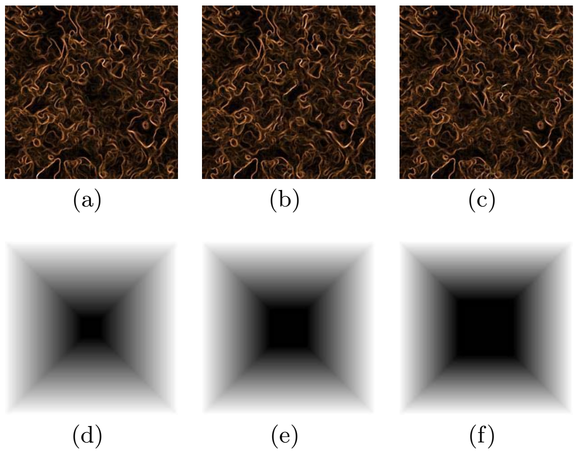


Figure 6: Different weight maps produce different textures. (a)-(d), (b)-(e) and (c)-(f) are texture and weight map pairs. The rectangular perimeter region of all the images are same, which is a desired quality.

2.1 Salient Features

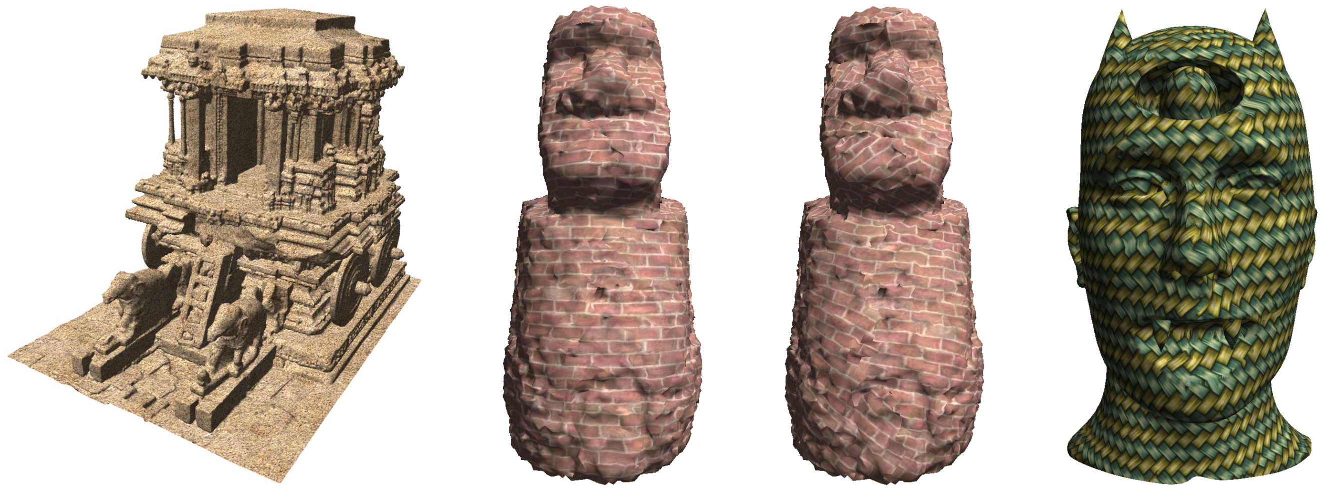
1. Most striking feature is the speed of the algorithm (see Figure 7(a)). Vertices are textured by considering only the other two vertices and the orientation vector of the face. Only half of the faces are actually processed, saving much time (see Figure 3).

2. A priority queue based patch discovery algorithm that grows into homogeneous regions, as the priority of a face is inversely proportional to the orientation distortion at the face.

3. The patch discovery process tends to terminate at faces of large variations in normals resulting in patches getting bounded by edges of the model at places where the seam is less visible. This improves the perceptual quality of texturing (see Figure 1) and helps to reduce the stretch. This property of our algorithm results in cutting the mesh into patches. However, unlike most mesh parameterization methods this is an inherent part of our algorithm. Sorkine *et al.* [17] also proposes a mesh parameterization technique that does texture mapping and mesh cutting simultaneously.

4. Our algorithm does not have any restriction on the topology as it incrementally textures the surface and cuts it at appropriate places.

5. Increase in complexity of the mesh topology has a negligible effect on the run time of the algorithm and its texture quality(see Figure 7(b) and 7(c)). Other mesh parameterization algorithms like ABF++[16] drastically increase their execution speed and texture boundary length with increase in complexity of mesh topology. This suggests that our algorithm is robust to complex surface topologies.



(a) stonechariot

(b) Our result

(c) ABF++ result [16]

(d) Mannequin-devil

Figure 7: (a) Stone chariot, a heritage monument having $1,586,181$ faces was textured in 3.28 sec. (b) and (c) show a noisy mesh model having 20k faces was texture in 0.012 sec by our algorithm and in 7.21 sec by ABF++[16]. (d) shows our texturing result on a complex model.

6. Texture patches follow the orientation vectors and remain in coherence with neighboring texture patches making the seam less obvious. This phenomenon can be observed in the Figure7(b) and 7(c).

2.2 Constraining patch seams

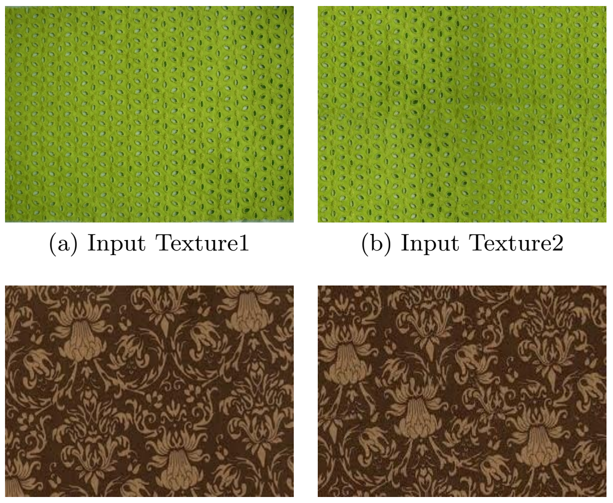
In some situations, it will be quite useful if we can decide where the patch seams occurs. Taking advantage of the local nature of patch growing in our algorithm, we can specify where the texture patch seams are formed on the mesh model. To achieve this, our texturing tool allows us to select the faces where we want the seam to appear and while texturing, the algorithm treats those faces similar to those with high variation in orientation, pushing it back onto the queue for later processing. This effectively transfers the seams onto these regions even if they are smooth in nature. Figure 4 shows a typical example, where the original method created a seam on the side of the hippo model, where it is quite visible. By selecting faces at the bottom of the hippo, the seam was pushed to the bottom, making it less visible.

3. CREATING SELF-TILEABLE TEXTURES

Use of mesh parameterization for texture mapping assumes that the texture on to which the mesh is parameterized is large enough to cover the complete model. If not, the texture patch is repeated to make it large. This will result in periodic discontinuities unless the patches are self-tileable: i.e., it should satisfy the property that the south edge of the texture should be seamlessly tileable with north edge and similarly east edge with west edge. In simple terms, we wish to make our texture toroidal in nature. Our approach to acquire this property is similar to the procedure to create Wang tiles [1]. We start by dividing the input texture image into four equal pieces of 2X2. The north-west piece is then swapped with south-east piece and similarly swap north-east piece with south-west piece. As seen in Figure 5, we now have a texture patch that is tileable on all edges, but with possible seams in the interior portions. We use texture synthesis and the same image for searching neighborhood, to make the center portion smooth and continuous. A weight map is used to keep the tileable edges from changing during the synthesis process. We use the image quilting method proposed by Efros *et al.* [3] to correct the center portion. We can also generate variants of the texture patch with the same tileable edges by changing the weight map as seen in Figure 6. Some examples of our tileable patch generation algorithm are shown in Figure 8.

3.1 Aperiodic Tiling

In the above algorithm we consider only one tileable tex-



(a) Input Texture1

(b) Input Texture2

(c) Output Texture1

(d) Output Texture2

Figure 8: (a), (b) are input images and (c), (d) are corresponding output images which are self tileable.

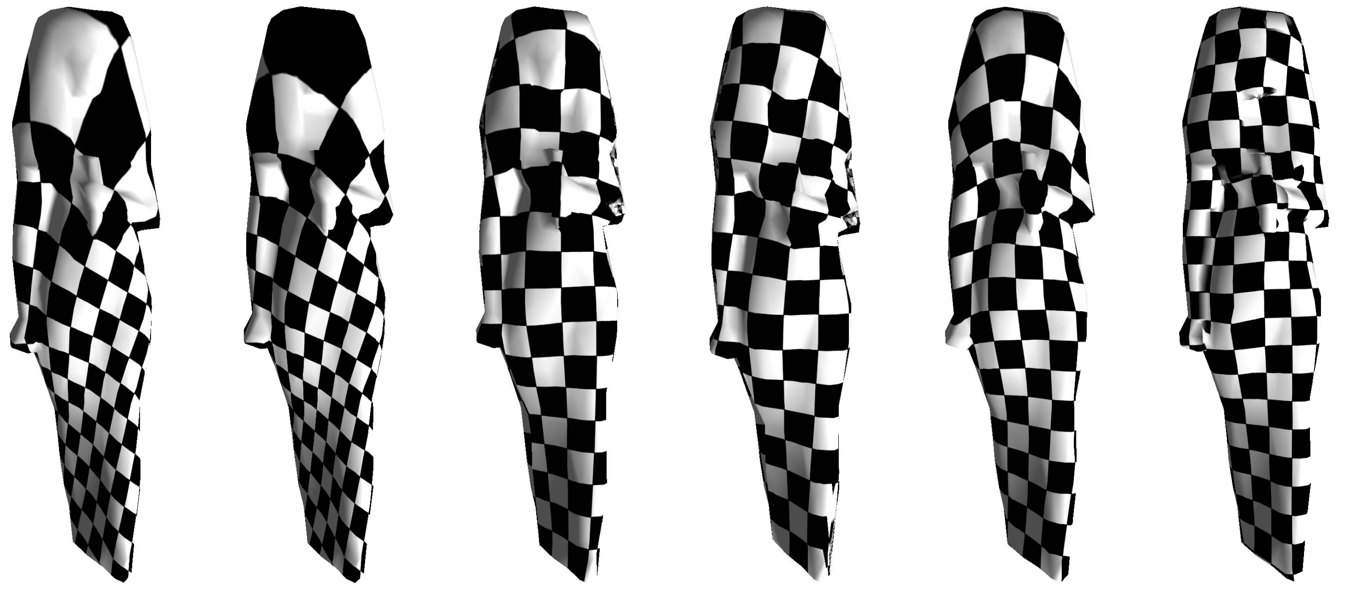


Figure 9: Results of various algorithms on Isis model. The proposed algorithm (f) preserves scale much better than state-of-the-art methods while limiting distortion.

ture, which can introduce periodicity in the texture pasted. Jos Stam *et al.* [18] proposed a substitution rule to generate an infinite array of tiles that tile seamlessly with other tiles based on the color coding given for the edges. We can use a simplified approach for this by synthesizing more than one self-tileable texture, which can be produced by the same algorithm as mentioned, by using different weight maps. This produces different texture as shown in Figure 6. These variants may be used interchangeably to create aperiodic tiling during texture mapping.

4. RESULTS AND DISCUSSION

Table 1 shows the time taken and *errorThreshold* values of various mesh models, calculated on a Intel Core 2 Duo CPU E4600 @ 2.40 GHz and our algorithm has been

Serial	Model	Faces	Time (sec)	Error
1	Buddha	100,000	0.1298	55
2	Gargoyle	20,000	0.0138	95
3	Bunny	69,451	0.0628	35
4	Heptoroid	573,440	0.7830	25
5	Pegasus	127,099	0.1343	465
6	Maxplank	98,260	0.0812	25
7	Fertility	483,226	0.5678	25
8	Dragon	100,000	0.1284	45
9	Laurana	499,998	0.6355	155
10	Hand	23,186	0.0128	45
11	Horse	96,966	0.1057	125

Table 1: Execution time with models of various sizes. Error(in pixels) denotes the largest shear of triangle in pixels during texture mapping. These mesh models are shown in Figure 10.

implemented for single core processors only. Table 2 shows the real strength of our algorithm. All the values in bold show the best in that particular column. Clearly in terms of time and stretch our algorithm does the best. The best nearest algorithm in terms of time is ARAP [9] and our algorithm is 25.145 faster on an average for the 4 mesh models in the Table 2. Texturing time reported does not include the calculation of orientation vectors and assume it to be a preprocessing step. Our algorithm is sensitive to orientation changes. Smoothing of the orientation vectors over neighboring faces can increase the texture quality sometimes. All the results presented in this paper are without smoothing of orientation vectors. Density of the mesh vertices do not affect the quality of the texture, as this is the requirement for some mesh parameterization algorithms. Despite being simple, the algorithm can manage to texture complex surfaces because of the heuristics used for choosing a face while texture mapping. If large values are choosen for *Factor* the algorithm loses its property of cutting the texture patches at geometric edges. But increasing the *Factor* value decreases the execution time. $Factor := 9/(nooffaces)$ strikes a good balance between texturing time and texture patch boundary length. On an average, 99.9% of faces are textured within the *errorThreshold* of 25 *pixels*. The scale used for texturing all the results is 800, which means for every square unit area in geometric space 800 square pixels of texture space was pasted. So, the stretch is bounded between 0.96 to 1.03 for every unit length vector in texture space. Sorkine *et al.* [17] also proposed a similar greedy method approach which approximately textures at a rate of $20k$ triangles/sec on a 2.40 GHz processor. Our method textures at 1 Million triangles/sec because of its salient feature as mentioned in Section 2.1. Finally, we present our results on a number of mesh models of varying complexity in Figure 10 to illustrate the robustness and consistency of our algorithm.

Method	Hand (Fig 10.10)			Mannequin-devil (Fig 7(d))			Isis (Fig 9(f))			Gargoyle (Fig 10.2)		
	Time	Stre	Distor	Time	Stre	Distor	Time	Stre	Distor	Time	Stre	Distor
Barycentri [19]	1.55	1.44	52.512	1.578	2.02	52.0	0.011	0.582	72.31	0.862	0.509	62.94
MeanValue [5]	2.28	1.369	17.94	2.63	1.80	31.55	0.016	0.52	32.92	1.08	0.48	29.08
Multiresol [2]	2.45	1.39	9.16	3.24	1.73	10.07	0.017	0.51	33.07	1.07	0.48	27.95
LSCM [8]	4.94	1.42	7.127	4.64	1.62	23.32	0.02	0.65	11.0	10.65	0.47	3.89
HLSCM [12]	170.35	0.38	1.32	169.63	0.207	**1.63**	2.09	0.248	**4.39**	261.56	0.29	**2.28**
ABF [15]	5.62	0.29	1.148	25.05	0.32	3.32	0.5	0.478	4.6	8.15	0.31	3.37
ABF++ [16]	4.57	**0.135**	**1.146**	15.91	0.322	2.44	0.51	0.48	4.57	5.59	0.31	3.37
ARAP [9]	0.328	0.40	96.80	0.39	0.39	71.69	0.016	0.226	19.07	0.343	0.20	26.87
Proposed	**0.013**	0.145	4.607	**0.0149**	**0.175**	4.534	**0.0003**	**0.152**	7.52	**0.015**	**0.1903**	8.289

Table 2: Time (sec), stretch (area ratio) and Distortion (angle difference in degrees) for various algorithms of four models of differing complexity.

5. CONCLUSION AND FUTURE WORK

Our approach is highly robust and efficient, while producing texture mappings that are similar in perceptual quality compared to recent mesh parameterization techniques that are multiple orders of magnitude slower. Quantitative measures of stretch and distortion also shows the approach to be among the best available. Our approach is also amenable to interactive modifications, which can be easily integrated into our patch discovery process. We are currently working on a local optimization function that smoothens the stretch tension over a specified local region. Thus the texture quality can be further improved, thereby avoiding the occasional glitches in the texture.

Acknowledgment

This work was partly supported by India Digital Heritage project of the Department of Science and Technology, Govt. of India. Polygonal models used were acquired from the Gatech Large Geometric Models Archive, and the Stanford 3D Scanning Repository.

6. REFERENCES

[1] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. SIGGRAPH, 2003.

[2] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. SIGGRAPH, 1995.

[3] A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. SIGGRAPH, 2001.

[4] M. Fisher, P. Schröder, M. Desbrun, and H. Hoppe. Design of tangent vector fields. SIGGRAPH, 2007.

[5] M. S. Floater. Mean value coordinates. *Computer Aided Geom. Design*, 2003.

[6] C. Gotsman, X. Gu, and A. Sheffer. Fundamentals of spherical parameterization for 3d meshes. In *ACM SIGGRAPH 2003.*

[7] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. SIGGRAPH, 1998.

[8] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. SIGGRAPH, 2002.

[9] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. J. Gortler. A local/global approach to mesh parameterization. In *Symp. on Geometry Processing*, 2008.

[10] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. SIGGRAPH, 2000.

[11] E. Praun and H. Hoppe. Spherical parametrization and remeshing. SIGGRAPH, 2003.

[12] N. Ray and B. Levy. Hierarchical least squares conformal map. In *Pacific Conference on Computer Graphics and Applications*, 2003.

[13] S. Saba, I. Yavneh, C. Gotsman, and A. Sheffer. Practical spherical embedding of manifold triangle meshes. SMI, 2005.

[14] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. SIGGRAPH, 2001.

[15] A. Sheffer and E. de Sturler. Parameterization of Faceted Surfaces for Meshing using Angle-Based Flattening. *Engineering with Computers*, 2001.

[16] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov. ABF++: fast and robust angle based flattening. *ACM Transactions on Graphics*, 2005.

[17] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski. Bounded-distortion piecewise mesh parameterization. In *Proc. Visualization*, 2002.

[18] J. Stam. Aperiodic texture mapping. Technical report, European Research Consortium for Informatics and Mathematics, 1997.

[19] W. T. Tutte. How to draw a graph. *Proc Lond Math Soc*, 13, 1963.

[20] L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. SIGGRAPH, 2001.

[21] L. Ying, A. Hertzmann, H. Biermann, and D. Zorin. Texture and shape synthesis on surfaces. In *Proc. Eurographics Workshop on Rendering Techniques*, 2001.

[22] S. Yoshizawa, A. Belyaev, and H.-P. Seidel. A fast and simple stretch-minimizing mesh parameterization. In *Shape Modeling and Applications*, 2004.

[23] R. Zayer, B. Lévy, and H.-P. Seidel. Linear angle based parameterization. ACM/EG Symposium on Geometry Processing, 2007.

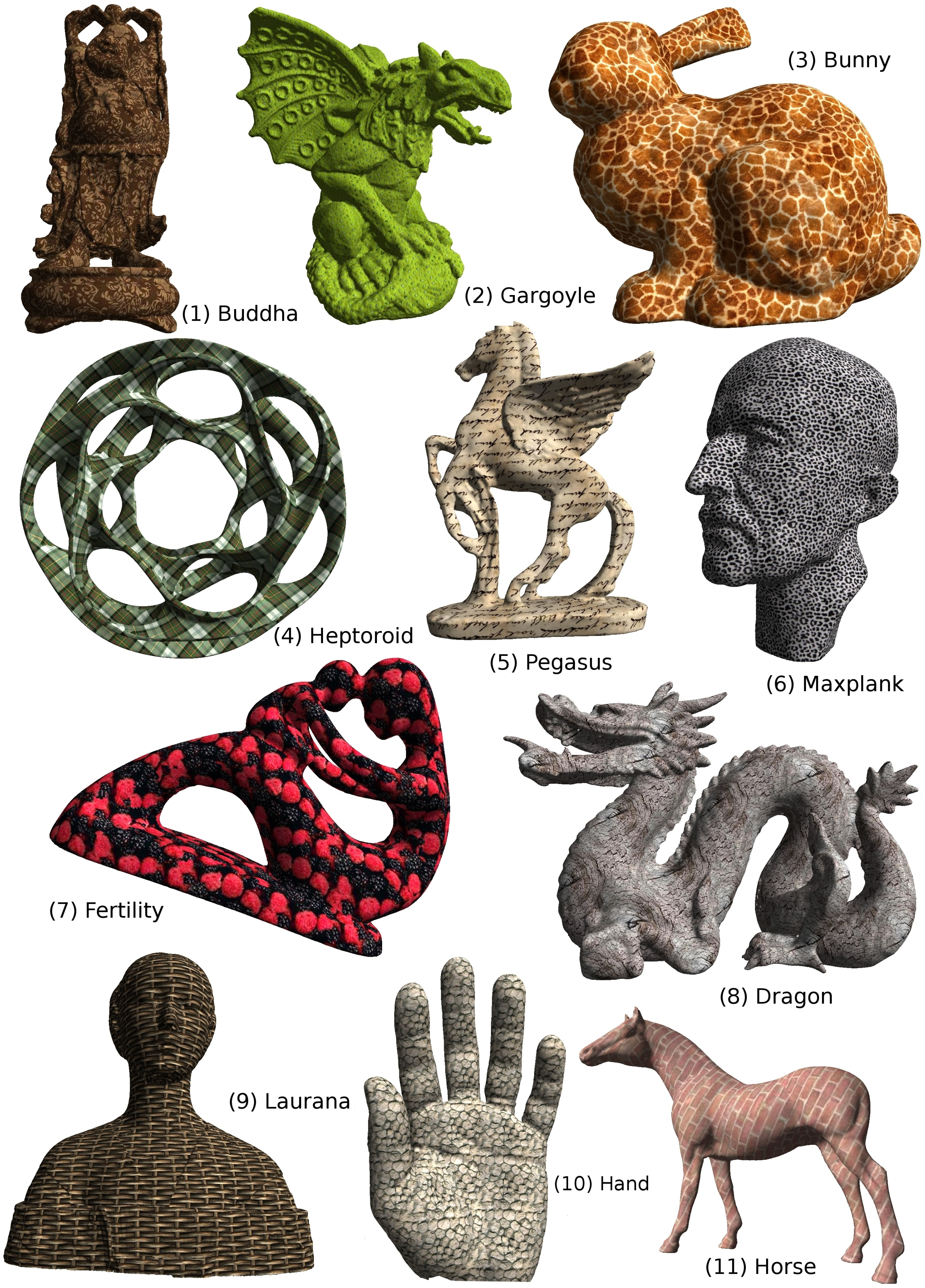


Figure 10: Results of texture mapping different mesh models of various complexities.