Matchmaker: Constructing Constrained Texture Maps

Vladislav Kraevoy
vlady@cs.technion.ac.il

Alla Sheffer
sheffa@cs.technion.ac.il

Craig Gotsman
gotsman@cs.technion.ac.il

Computer Science Department, Technion-Israel Institute of Technology

Figure 1: Constrained texture mapping. Dots indicate constrained vertices and their positions.

Abstract

Texture mapping enhances the visual realism of 3D models by adding fine details. To achieve the best results, it is often necessary to force a correspondence between some of the details of the texture and the features of the model.

The most common method for mapping texture onto 3D meshes is to use a planar parameterization of the mesh. This, however, does not reflect any special correspondence between the mesh geometry and the texture. The *Matchmaker* algorithm presented here forces user-defined feature correspondence for planar parameterization of meshes. This is achieved by adding positional constraints to the planar parameterization. *Matchmaker* allows users to introduce scores of constraints while maintaining a valid one-to-one mapping between the embedding and the 3D surface. *Matchmaker*'s constraint mechanism can be used for other applications requiring parameterization besides texture mapping, such as morphing and remeshing.

Matchmaker begins with an unconstrained planar embedding of the 3D mesh generated by conventional methods. It moves the constrained vertices to the required positions by matching a triangulation of these positions to a triangulation of the planar mesh formed by paths between constrained vertices. The matching triangulations are used to generate a new parameterization that satisfies the constraints while minimizing the deviation from the original 3D geometry.

Keywords: Triangle meshes, Texture Mapping, Parameterization

1. Introduction

Texture mapping is a ubiquitous tool in computer graphics that easily adds the appearance of detail to otherwise bland object surfaces. The most common way to define texture on 3D surface meshes is by parameterizing the mesh over a planar domain, providing a one-to-one mapping between the parameter domain and the surface. To obtain visually pleasing results, the metric distortion introduced by the mapping should be kept to minimum. An additional requirement, often necessary, is the enforcement of feature correspondence between the texture and the 3D surface. For example, when applying face texture to a head model, the eyes must be placed in the sockets, the mouth should correspond to the model's mouth, and so on. Constraints also help hide texture seams resulting from parameterization discontinuities [Gu et al. 2002; Sheffer and Hart 2002].

A similar need may arise when pasting parts of one geometry onto another [Biermann 2002], or building a correspondence between two 3D surfaces by aligning the features on their parameterizations [Alexa 2000].

1.1 Previous Work

The problem of generating a planar parameterization for a 3D mesh surface has received a lot of attention over the past several years [Floater 1997; Lévy and Mallet 1998; Hurdal et al. 1999; Hormann and Greiner 1999; Sheffer and de Sturler 2000; Haker et al. 2000; Sander et al. 2001; Lévy et al. 2002; Desbrun et al. 2002]. All the methods provide a parameterization by embedding the 3D mesh in the plane and using the embedding function together with barycentric coordinates within each mesh triangle to define a piecewise-affine mapping. Some of the methods guarantee the validity of the resulting parameterization (i.e. that it is one-to-one) [Floater 1997; Hurdal et al. 1999; Hormann and Greiner 1999; Sheffer and de Sturler 2000; Sander et al. 2001] by ensuring that the embedding does not contain overlapping triangles. All the methods strive to minimize the distortion of the metric structures occurring during the mapping, using different distortion metrics. Many authors [Eck et al. 1995; Hurdal et al. 1999; Sheffer and de Sturler 2000; Haker et al. 2000; Lévy et al. 2002; Desbrun et al. 2002] minimize different variants of a conformality met-

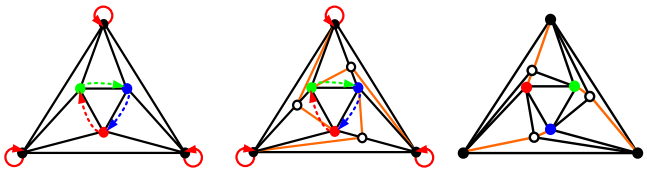
ric. Sander et al. [2001] minimize a stretch metric, solving a highly non-linear minimization problem. One property which affects the level of the distortion is the shape of the parameter domain's boundary. Some methods [Floater 1997; Sander et al. 2001; Eck et al. 1995; Lévy and Maillot 1998] require the domain boundary to be a convex polygon, thereby increasing the resulting distortion. More recent works [Hormann and Greiner 1999; Sheffer and de Sturler 2000; Lévy et al. 2002; Desbrun et al. 2002; Sander et al. 2002] eliminate this constraint, computing the boundary as part of the minimization procedure. In this way they are able to achieve significantly lower levels of distortion. None of the methods provides a straightforward solution to enforcing positional constraints on the vertices of the planar embedding.

Several researchers introduced methods which satisfy positional constraints approximately or "softly". Guenter et al. [1998] used a learning theory approach to build the position correspondence, satisfying a set of soft constraints. His method is not guaranteed to find a solution. Lévy and Mallet [1998] allowed user-defined matching of iso-parametric curves. In a later work Lévy [2001] suggested a method to incorporate soft positional constraints into the formulation of the parameterization problem. His algorithm satisfies the constraints in the least-squares sense. The method works well for a small number of constraints, but can fail for large sets of constraints, resulting in an invalid parameterization. Soft constraints may give acceptable results for feature matching. However, for other applications such as hiding texture discontinuities along seams in the parameterization, hard constraints must be used to achieve perfect alignment of the texture along the seams.

Desbrun et al. [2002] use Lagrange multipliers to add "hard" constraints to the parameterization formulation. Similarly to Lévy [2001], the method works well for a small number of constraints, but can generate foldovers when the number increases.

Alexa [2000] addressed a related problem of embedding a closed genus-0 3D mesh on a sphere with constraints. The motivation for this work was morphing of 3D meshes and the constraints were used to align features between two meshes. However, the heuristic solution he suggested does not guarantee a solution, even if one exists.

Providing a valid mesh embedding that exactly satisfies a set of positional constraints is extremely difficult. Indeed, it is easy to show that for a given mesh connectivity not every set of constraints can be satisfied (see Figure 2 (a)). At the same time, an embedding can always be found if the mesh is enriched by adding a number of additional *Steiner* vertices [Pach and Wenger 1998] (Figures 2 (b) and (c)). Regrettably, finding the minimal number of vertices that have to be added is NP-Hard [Pach and Wenger 1998]. Eckstein et al. [2001] introduced a method that enforces "hard" (exact) positional constraints by deforming an existing embedding while adding a number of Steiner vertices. This method theoretically can handle large sets of constraints but is extremely complicated. Relying on a multiresolution construction, it is difficult to implement and not very robust. The examples presented in that paper are rather simple, so it is not obvious how the method scales to more complicated inputs.



(a) (b) (c)

Figure 2: A mesh and a set of constraints which cannot be satisfied without Steiner vertices. (a) Original connectivity. Arrows indicate constrained locations. (b) Connectivity with extra Steiner vertices (white circles) and connecting edges (orange). (c) Mesh satisfying the constraints.

1.2 *Matchmaker* overview

In this paper we introduce a new method, called *Matchmaker*, which enforces positional constraints in a parameterization. The method satisfies the constraints exactly while preserving as much as possible the metrics (angles, distances) of the original 3D geometry.

The input to *Matchmaker* is an existing planar embedding of the 3D mesh. The boundary of the planar domain can have any shape. Therefore free-boundary methods such as [Sheffer and de Sturler 2000; Lévy et al. 2002] can be used to generate the initial embedding, resulting in significantly lower parameterization distortion. To parameterize closed surfaces or surfaces with high genus we cut the surface into a single patch using the method of Sheffer and Hart [2002]. *Matchmaker* maps the fixed vertices to the user-defined positions as follows. First it matches a triangulation of the positions to a triangulation of the planar mesh formed by paths between constrained vertices. Then, the matching triangulations are used to generate a new parameterization that satisfies the constraints while minimizing the metric distortion. Similarly to Eckstein et al. [2001], when necessary, we add a small number of Steiner vertices.

Matchmaker can handle a large number of constraints, including those that require large deformation of the planar input mesh. The algorithm enforces these constraints while adding only a small number of Steiner vertices to the 3D mesh. It also minimizes the distortion, introduced by adding the constraints, to an acceptable level. (The effect of large distortion on a parameterization can be seen in Figure 3 (h), which shows an intermediate, unsmoothed mapping.) Last but not least, *Matchmaker* is very efficient and simple to implement. It does not require any additional user interaction beyond specification of the constraints.

The rest of the paper is organized as follows. Section 2 defines the terminology used throughout the paper. Section 3 provides an overview of the algorithm. Section 4 describes in detail the main step of the algorithm – the construction of the matching positions and mesh triangulations. Section 5 demonstrates the application of the method on several examples and discusses some implementation details. The last section (Section 6) summarizes the work and described areas of future research.

2. Definitions

To define the matching procedure that is at the heart of the *Matchmaker* algorithm, we introduce the following notations. A *triangulation* $T(P)$ of a set P of points in 2D is a partition of

the plane by a maximal number of non-intersecting segments between points in P. It is easy to see that all the bounded regions in the partition must be triangles [de Berg et al. 2000].

A *mesh triangulation* $T_M(W)$ of a planar mesh M induced by a set of vertices W is a partition of M into patches where the boundary of each patch is defined by three edge paths in M between vertices in W. A mesh triangulation is *valid* **iff**

- The paths are distinct and do not intersect except at the shared end vertices in W.
- The patches do not overlap.
- Each patch is a single, manifold connected component.

Consider a set of vertices W in a mesh and a corresponding set of positions P in the plane, s.t. $|W|=|P|$. A triangulation $T(P)$ of P and a mesh triangulation $T_M(W)$ induced by W are said to *match* **iff**

- $T_M(W)$ is valid.
- Each edge (p_i,p_j) in $T(P)$ corresponds to a path (w_i,w_j) in $T_M(W)$ and vice-versa.

3. The *Matchmaker* Algorithm

The input to *Matchmaker* consists of a planar mesh M, a set of constrained mesh vertices V_C, and a matching set P_C of the user-defined 2D positions of those vertices. The main use of the planar mesh is to generate the optimal virtual boundary to minimize the anticipated parameterization distortion.

The stages of the algorithm are listed below. They are described in more detail in the following sections. Figure 3 demonstrates the implementation of the algorithm on a small example.

1. **Virtual Boundary:** First, the mesh is embedded in a bounding rectangle and the region between the mesh boundary and the rectangle is triangulated. This generates a new mesh M^* which consists of the triangles of M and the new triangles (Section 3.1). We fix the vertices of the virtual boundary in their current locations, adding them to V_C and P_C, respectively.
2. **Matching**: This is the heart of the algorithm. The matching procedure finds a triangulation $T(P_C)$ of the fixed points P_C and a matching triangulation $T_{M^*}(V_C)$ of the mesh M^* (Section 4). The matching adds Steiner vertices to the mesh if it fails to compute the matching triangulations without them.
3. **Embedding**: Given the matching triangulations, each mesh triangle in $T_{M^*}(V_F)$ is mapped to the corresponding triangle in $T(P_F)$ (Section 3.2). After the mapping we obtain a provably valid embedding of the original mesh which satisfies the constraints (Figure 3 (g)).
4. **Smoothing**: The resulting embedding is smoothed, keeping the fixed vertices in place, in order to reflect the geometry of the 3D model as much as possible (Section 3.3).
5. **Post-Processing:** To reduce the number of redundant Steiner vertices, those that can be removed without violating the validity of the mapping are removed from the mesh (Section 3.4)

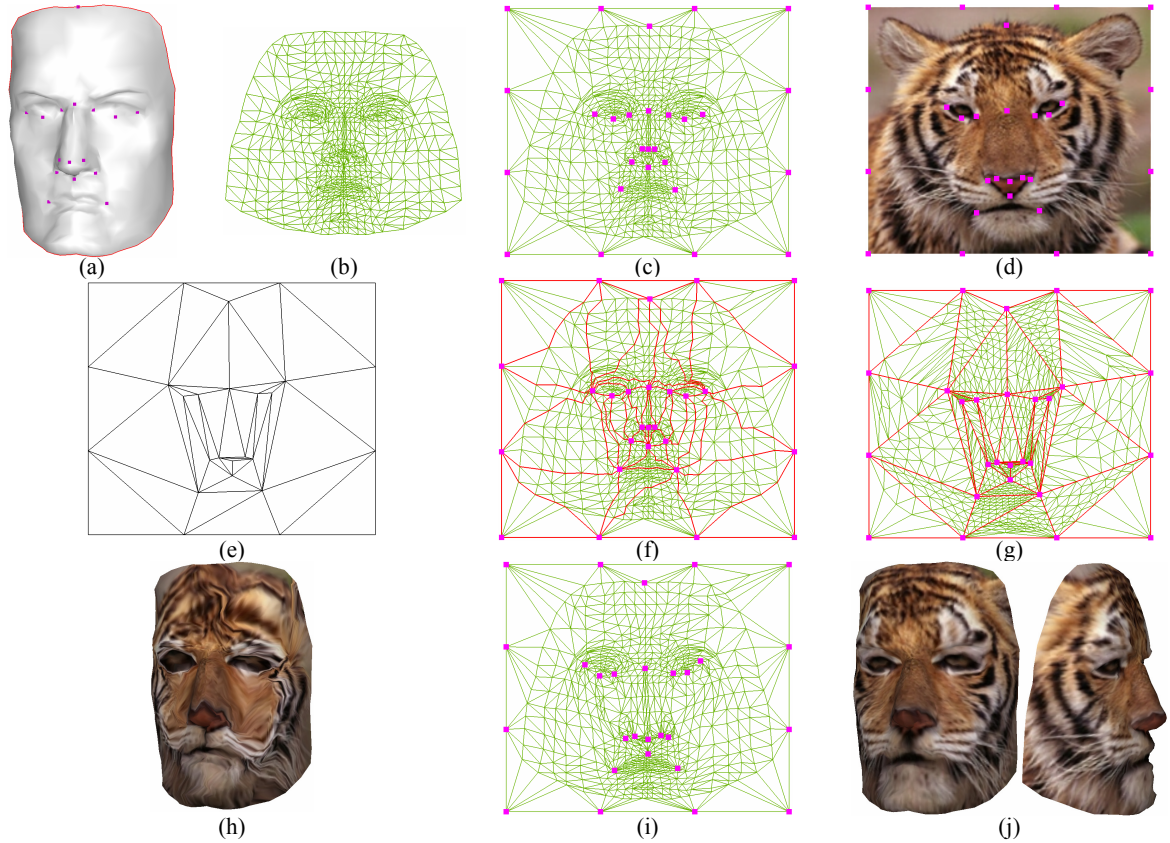


Figure 3: Mapping a tiger face onto a human. (a) 3D model with V_C vertices. (b) Unconstrained parameterization. (c) Parameterization with virtual boundary. (d) Texture with P_C points. (e) Triangulation of P_C points. (f) Matching triangulation $T_{M^*}(V_C)$ of M*. No Steiner vertices are required. (g) Triangle embedding. (h) Textured model after embedding. (i) Mesh after constrained smoothing. (j) Resulting textured model.

The result of the algorithm is a valid embedding that satisfies the constraints while closely preserving the metrics of the unconstrained parameterization.

We now describe each step in more detail.

3.1 Virtual Boundary Construction

The boundary of the planar input mesh M can have any shape. If they are not user-fixed, the boundary vertices should move freely following the constraints, reducing the mapping distortion. However, we need a fixed boundary surrounding the entire parametric domain in order to perform the embedding (Section 3.2). Therefore, *Matchmaker* adds a rectangular virtual boundary to the mesh. The region between the true and virtual boundaries is triangulated using a constrained Delaunay triangulation [Shewchuk]. The boundary construction can handle input meshes with several connected components (e.g. Figure 9). This enables *Matchmaker* to apply constraints to entire parameterization atlases, not only to individual pieces. No previous method has been able to do this. From here on the algorithm operates on the mesh M^* which contains both the triangles of M and the newly generated ones. The number of virtual boundary vertices is roughly proportional to the number of vertices on the boundary of the unconstrained mesh. The vertices of the rectangular boundary together with their current positions are added to the constrained vertex set V_C. Figure 3 (b) and (c) show a planar mesh before and after the virtual boundary was added.

3.2 Embedding

Once the matching, to be described in Section 4, is complete, we obtain a triangulation $T(P_C)$ of the fixed points P_C and a matching mesh triangulation $T_{M^*}(V_C)$ of M^*. Each triangle T_i in $T(P_C)$ corresponds to a triangular patch S_i of $T_{M^*}(V_C)$. The vertices of each triangle T_i are three positions (p_k,p_l,p_m) of three constrained vertices (v_k,v_l,v_m). The patch S_i is bounded by the paths between these three vertices (v_k,v_l),(v_l,v_m), and $(v_m.v_k)$. The mesh is embedded into the triangulation as follows.

1. Place the vertices v_i in V_C at the corresponding positions p_i.
2. For each path (v_i,v_j) in the mesh triangulation, place the path vertices at equal distances along the space triangulation edge (p_i,p_j).
3. For each patch S_i, use barycentric parametrization [Tutte 1963] to place its interior vertices inside the triangle T_i. The barycentric mapping of a mesh into a convex domain is known to produce a valid embedding.

The result of the procedure is a valid embedding of the mesh M^* which satisfies the positional constraints (Figure 3 (g)). The new mesh is referred to as M_{new} throughout the rest of the paper.

3.3 Constrained Smoothing

The embedding generated in the previous step is provably valid. However, the shape of the mesh M_{new} is very different from that of M^*. As a result, using M_{new} as a parameter domain for the original 3D surface will introduce very high metric distortion (Figure 3 (h)). The smoothing procedure is aimed at restoring the metrics of the original mesh by re-parameterizing M_{new} using a variant of the conformal (harmonic) mapping [Eck et al. 1995]. The weights for the mapping are derived either from the unconstrained mesh M^* or the original 3D surface.

The mapping algorithm is modified to maintain the constrained vertices in place and to prevent foldovers. We use an iterative procedure, in which we repeatedly compute the new positions p_{new} of the unconstrained mesh vertices v in $V \setminus V_C$ from the positions of their neighboring vertices. After a position is computed, we test whether the triangles incident on the vertex v remain correctly oriented. If any triangle folds over, the vertex position is left unchanged. A quite similar constrained procedure was used by Sander et al. [2001]. This test guarantees that the resulting mesh remains valid, namely has no foldovers. After smoothing is complete, the resulting mesh (Figure 3 (i)) is very similar in shape to the unconstrained input mesh (Figure 3 (c)).

3.4 Post-Processing

In the post-processing step we finalize the parameterization of the original 3D surface using M_{new}. In order to use the smoothed mesh M_{new} as the parameter domain for the 3D surface it needs to have the same connectivity as the 3D mesh. The first step towards this is to discard the virtual boundary triangles that are no longer needed. If the matching procedure did not add Steiner vertices to the planar mesh, the connectivity of M^* and the 3D mesh will now be identical and the goal will have been achieved. If Steiner vertices were added, they need to be added to the 3D mesh as well. To minimize the number of changes in the 3D mesh connectivity, the algorithm removes all unnecessary Steiner vertices from M_{new}. Thus, only a fraction of the vertices added by the matching process actually remains in the 3D mesh. The algorithm scans the Steiner vertices, testing if they can be removed. A vertex can be removed **iff** the mesh does not fold over after the removal and the original connectivity is preserved. Figure 6 shows a part of the head model (Figure 9) embedding before and after the removal of Steiner vertices.

After post-processing, *Matchmaker* has achieved the goal of generating a valid, low-distortion parameterization of the 3D surface which satisfies the user-defined constraints.

4. Matching

The matching procedure is the heart of *Matchmaker*. This procedure generates a triangulation $T(P_C)$ of the positions P_C and a matching triangulation $T_{M^*}(V_C)$ of the mesh M^*. Once a valid match is found, the algorithm can proceed with the embedding stage (Section 3.2). Therefore, finding a valid match is equivalent to finding an embedding of the mesh that satisfies the constraints. There are combinations of mesh connectivity and constraints for which a match does not exist (Figure 2). However, these combinations can always be resolved by adding Steiner vertices to the mesh (Figure 5). Determining if a match exists (with no additional vertices) is equivalent to finding an embedding. Hence, it is an NP-hard problem, as is the question of how many Steiner vertices must be added to generate a match [Pach and Wenger 1998]. Therefore, to compute a match in reasonable time, we developed a greedy algorithm that introduces Steiner vertices when it fails to find a match without them. Using Steiner vertices the algorithm is guaran-

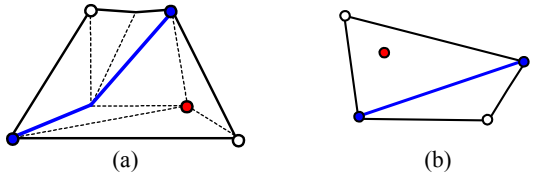
teed to find a match. Examples of matching triangulations are shown in Figure 3 (e) and (f), above and Figure 9 (d) and (e), below.

A related problem of finding a mesh triangulation for a given triangulation of the fixed points was addressed by Praun et al. [2001]. While the paper does not explicitly mention the use of Steiner vertices, it seems the method might need a huge number of those to succeed and can generate extremely long paths.

4.1 Path Matching

In the first stage of the algorithm we search for matching paths/edges which can partition both the mesh and the 2D space into valid sub-regions. *Matchmaker* adds paths one by one, testing that the following conditions are satisfied after each path is added:

- Neither paths nor edges intersect, except at their end-vertices.
- The new path does not block necessary future paths. This problem can arise if the path partitions the mesh/space and generates one or two new sub-regions. The problem is shown in Figure 4, where a fixed vertex v_i and the corresponding position p_i are placed in different regions of the partition. The test must check that the vertices and positions are not separated into different sub-regions by the path.



(a) (b)

Figure 4: Illegal path (blue): (a) path in $T_{M^*}(V_C)$; (b) matching edge in $T(P_C)$. This path places the red constrained vertex and its position in different sub-regions.

- The sub-regions in the mesh and the space have the same (correct) orientation. Since both boundaries are planar polygons, this requires a simple polygon orientation test.

Paths are introduced until no more can be added. If at this stage we have a triangulation of the fixed positions, we are done. If not, the procedure described in Section 4.2 is applied.

```
Algorithm PathMatch
    M'=M*
    Compute S=set of shortest paths for each pair of vertices in
    V_C
    while S ≠ ∅
    begin
      s =S.RemoveShortest()
      if Legal(s)
      begin
        Add s to T_M*(V_C), add corresponding edge to T(P_C)
        Remove all the interior vertices of s from M'
        Update S – remove all paths (v_i,v_j) containing interior
        vertices of s and recompute them in M'\{s}
      end
    end
end
```

The function *Legal*(*s*) tests if the path *s* can be added to the mesh triangulation, namely whether it satisfies the conditions described above. Note that the paths in *S* are modified after each path insertion. Paths that intersect *s* are deleted and computed again inside $M'\setminus\{s\}$. Hence, *Legal* does not need to check path intersections.

When *PathMatch* terminates, no more paths can be added to $T(P_C)$ and $T_{M^*}(V_C)$ without violating some requirement. At this stage $T(P_C)$ defines a partition into polygons and $T_{M^*}(V_C)$ defines a matching partition of the mesh M^*. If all the polygons in $T(P_C)$ are triangles, we are done. Otherwise, a process that adds Steiner vertices to generate the missing paths is applied. In fact, if all the polygons are convex, further processing is not required since the embedding procedure (Section 3.2) is guaranteed to generate a valid mapping for any set of convex polygons, not just triangles.

4.2 Paths with Steiner vertices

At this stage $T(P_C)$ defines a partition of the domain into polygons. To use the partition for embedding, each polygon N in $T(P_C)$ needs to be triangulated. Even with Steiner vertices, not every triangulation of N has a matching mesh triangulation. The main problem that can arise is blocking of paths (Figure 4). The two stage procedure described below guarantees that a match is found.

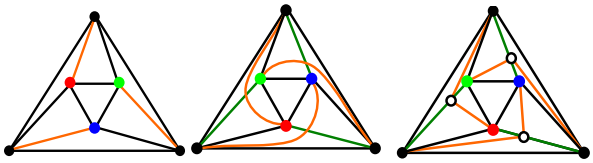
Blocking can happen only when the polygon N is not simply-connected, namely it has "holes". Hence, in the first stage *Matchmaker* converts each such polygon into a simply-connected one, by cutting along diagonals to form a single boundary loop.

```
Algorithm MakeSimple(N)
      while NumBoundaryLoop(N) > 1
        /* basic connectivity test */
      begin
        e = ComponentDiagonal(N)
        s = CreatePath(N,e)
      end
end
```

MakeSimple generates a simple polygon with a single boundary loop. Any polygon with multiple loops can be transformed into a simple polygon by adding diagonals which connect the loops and do not intersect the edges of the polygon [de Berg et al. 2000]. *ComponentDiagonal* finds such diagonal edges. *CreatePath* finds a matching path inside the matching mesh patch N_M. Since the edges do not split N into several polygons, such a path cannot introduce blocking. After *MakeSimple* terminates, the polygon can be triangulated using any standard algorithm such as *TriangulatePolygon*. By introducing triangulation edges and matching paths using the following recursive procedure, we guarantee that paths will not intersect (Figure 5).

```
Algorithm TriangulatePolygon(N)
      if size(N) == 3 /* triangle*/
        return
      e = TriangulationEdge(N, T(P_C))
      s = CreatePath(N,e)
      Add s to T_M*(V_C)
      split N into polygons N_1, N_2 using e
      TriangulatePolygon(N_1)
      TriangulatePolygon(N_2)
end
```

CreatePath$(N,(p_i,p_j))$ generates a path inside N_M from v_i to v_j. The sub-mesh N_M is connected; hence a path between any two vertices exists. Consider N'_M, which is the edge graph of N_M

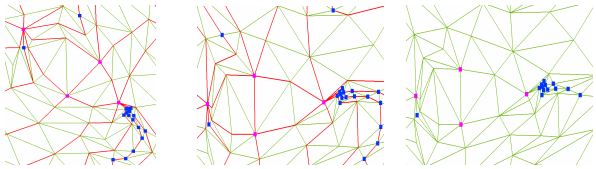


(a) (b) (c)

Figure 5: Match construction for the example in Figure 2. (a) $T(P_C)$ before (black) and after (orange) triangulation. (b) $T_{M^*}(V_C)$ before *TriangulatePolygon*. *CreatePath* generates the orange paths (c) by splitting the green edges (b).

without boundary vertices. N'_M will have several components only if N_M had edges whose two end vertices were on the boundary. If each such edge is split into two by adding a Steiner vertex, then N'_M will remain a single connected component. Hence, a path inside it, between v_i and v_j, will exist.

In practice we split only those edges which are needed by the path. An example of adding Steiner vertices is shown in Figure 6. Note for example the constrained vertex (purple) at the right eye corner. The number of paths emanating from the vertex exceeds its valence, hence Steiner vertices must be added to increase the latter.



(a) (b) (c)

Figure 6: Zoom in on the left eye area of Figure 9. (a) Mesh triangulation with Steiner vertices (blue). (b) Smoothed embedding. (c) Final embedding after removing redundant Steiner vertices.

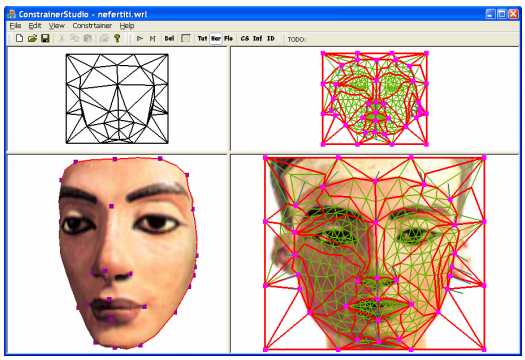


Figure 7: Matchmaker GUI.

5. Experimental Results

Matchmaker has been implemented in an interactive system, where the user can add and edit the constraints interactively. A GUI snapshot is shown in Figure 7. The session time per model varies, based on the number of constraints the user needs to add. It ranges from 10 minutes to half an hour for typical models. The *Matchmaker* algorithm runs in interactive time.

The use of *Matchmaker* for texture mapping is demonstrated on several examples of varying complexity. The first (Figure 3) is of about the same complexity as the examples in previously constrained mapping publications [Eckstein et al. 2001; Lévy 2001; Guenter et al. 1998]. Most of the other examples are an order of magnitude more difficult in terms of model complexity and number of constraints.

Figure 8 shows an example where constraints are used to hide a texture seam, which was added to reduce the mapping distortion [Sheffer and Hart 2002]. Texture discontinuities are removed by placing several constrained vertices along the seam. Note that constraining the seam vertices eliminates only the C^0 discontinuities. Additional constraints are needed to make the texture appear C^1 continuous (Figure 8 (d)). In order to stitch seams, the mapping must satisfy exact constraints; hence methods such as [Guenter et al 1998] or [Lévy 2001] cannot be used.

(a) (b) (c) (d)

Figure 8: Constraining vertices to eliminate (c) C^0 and (d) C^1 discontinuities along a texture seam.

Figure 9 shows the stages of texturing a head model (1,242 faces). The input parameterization atlas contains two charts of head halves. The image includes two mirrored profiles, defining the texture for the entire head. The constraints were used both to align features and to stitch textures along the boundary between the halves. To the best of our knowledge, *Matchmaker* is the first method capable of satisfying the constraints necessary to seamlessly texture an entire head from a single image. Note the huge difference between the initial embedding (b) and the final mesh (g) imposed by the constraints.

The Igea model (Figures 1 and 10) is another example of mapping two profiles to a model of a whole head (4,443 faces). The number of constraints (80) and the large displacement of the constrained vertices compared to the unconstrained map, make the matching a challenging problem. The matching procedure added 111 Steiner vertices to the mesh. After post-processing this number was reduced to 20.

The last and most complex example shows the texturing of a full body scan (7996 faces). The input includes a full body model [Cyberware], which was cut as a single chart (Figure 11 (b)) and two photographs of a standing person (front and back). The photos were aligned with the input mesh. There is a large discrepancy between the mesh and the image silhouettes, as indicated by the markers (Figure 11 (b)). Most of the 160 constraints were used to align the mesh and the silhouette and stitch the seams on the sides of the model. The matching added 1447 Steiner vertices. After post-processing, the number of vertices decreased to 110.

6. Conclusions

Matchmaker is a new and effective tool for adding positional constraints to texture maps. It provides a valid one-to-one mapping which satisfies user-defined constraints exactly, while closely preserving the metric structures of the original mesh. As demonstrated by the examples, it is able to successfully handle combinations of models and constraints which are an order of magnitude more difficult than those addressed by previous methods.

The triangle matching procedure which is at the heart of the algorithm does not rely on the planarity of the embedding. Therefore, we expect that *Matchmaker*, with some minor modifications, will work successfully for parameterizations on the sphere and other domains. Constrained embedding on the sphere is useful for morphing [Alexa 2000] or model recognition.

Currently, the user must add the constraints interactively, one by one. This can be quite tedious, especially for seam stitching where a large number of constraints must be added to achieve perfect alignment. An interesting problem for future research is the automatic placement of such constraints. Pattern mapping ideas [Soler 2001] could provide the optimal positions for the constrained vertices to be used by *Matchmaker*.

Acknowledgements

Thanks to Vitaly Surazhsky for the use of some of his software and helpful comments throughout this work. This research was partially funded by European grant HPRN-CT-1999-00117 (MINGLE), German-Israel Fund grant I-627-45.6/1999 and Israel Ministry of Science grant 01-01-01509.

References

ALEXA, M. 2000 Merging Polyhedral Shapes with Scattered Features, *The Visual Computer 16*, 26-37.

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O., Eds. 2000. *Computational Geometry*, 2nd ed. Springer.

BIERMANN, H., MARTIN, I., BERNARDINI, F., AND ZORIN, D. 2002. Cut-and-paste Editing of Multiresolution Surfaces. *ACM Transactions on Graphics*, *21*, 3, 312-321.

CYBERWARE INC., http://www.cyberware.com

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic Parametrizations of Surface Meshes. *In Proceedings of Eurographics 2002*, Blackwell Publishing, Saarbrucken, G. Drettakis and H.-P. Seidel, Eds., Computer Graphics forum, 21, 3, 210-218.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution Analysis of Arbitrary Meshes. In *Proceedings of ACM SIGGRAPH 1995*, Computer Graphics Proceedings, Annual Conference Proceedings, 173-182.

ECKSTEIN, I., SURAZHSKY, V., AND, GOTSMAN, C. 2001. Texture Mapping with Hard Constraints, *Computer Graphics Forum 20*, 3, 95-104.

FLOATER, M. S. 1997. Parameterization and Smooth Approximation of Surface Triangulation, *Computer Aided Geometric Design, 14*, 231-250.

GU, X., GORTLER, S., AND HOPPE, H. 2002. Geometry Images. *ACM Transactions on Graphics, 21*, 3, 355-361.

GUENTER, B., GRIM, C., WOOD, D., MALVAR, H., AND PIGHIN, F. 1998. Making Faces. In *Proceedings of ACM SIGGRAPH 1998*, Computer Graphics Proceedings, Annual Conference Proceedings, 55-66.

HAKER, S., ANGENENT, S., TANNENBAUM, A., KIKINIS, R., SAPIRO, G., AND HALLE, M. 2000. Conformal Surface Parameterization for Texture Mapping. *IEEE Transactions on Visualization and Computer Graphics*, *6*, 2, 181-189.

HORMANN, K., AND GREINER, G. 1999. MIPS - An Efficient Global Parametrization Method. In *Curve and Surface Design Conference Proceedings 1999*, 153-162.

HURDAL, M., BOWERS, P., STEPHENSON, K., SUMNERS, D., REHMS, I. K., SCHAPER, K., AND ROTTENBERG, D. 1999. Quasi-conformally Flat Mapping the Human Cerebellum. In *Proceedings of MICCAI'99*, volume 1679 of *Lecture Notes in Computer Science*, 279-286, Springer-Verlag.

LÉVY, B. Constrained Texture Mapping for Polygonal Meshes. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Proceedings, 417-424.

LÉVY, B., AND MALLET, J. L. 1998. Non-distorted Texture Mapping for Sheared Triangulated Meshes. In *Proceedings of ACM SIGGRAPH 1998*, Computer Graphics Proceedings, Annual Conference Proceedings, 343-352.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Transactions on Graphics, 21,* 3, 362-371.

MAILLOT, J., YAHIA, H., AND VERROUST, A. 1993. Interactive Texture Mapping. In *Proceedings of ACM SIGGRAPH 1993*, Computer Graphics Proceedings, Annual Conference Proceedings, 27-34.

PACH, J., AND WENGER, R. 1998. Embedding Planar Graphs with Fixed Vertex Locations. In *Proceedings of Graph Drawing '98*. Lecture Notes in Computer Science 1547, Springer-Verlag, 263-274.

PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. 2001. Consistent Mesh Parameterizations. In *Proceedings of ACM SIGGRAPH 2001*, E. Fiume, Ed.., Computer Graphics Proceedings, Annual Conference Proceedings, 179-184.

SANDER, P., GORTLER, S., SNYDER, J., AND HOPPE, H. 2002. Signal-specialized Parametrization. In *Proceedings of Eurographics Workshop on Rendering 2002*.

SANDER, P. V., SNYDER, J., GORTLER, S., AND HOPPE, H. 2001. Texture Mapping Progressive Meshes. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Proceedings, 409-416.

SHEFFER, A., AND DE STURLER, E. 2000. Surface Parameterization for Meshing by Triangulation Flattening. In *Proceedings of the 9th International Meshing Roundtable*, 161-172.

SHEFFER, A., AND HART, J. 2002. Seamster: Inconspicuous Low-Distortion Texture Seam Layout, *Proceedings of IEEE Visualization*, 291-298.

SHEWCHUK, J. R. Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator. http://www.cs.cmu.edu/~quake/triangle.html

SOLER, C., CANI, M. P., AND ANGELIDIS, A. 2002. Hierarchical Pattern Mapping. *ACM Transactions on Graphics, 21,* 3, 673-680.

TUTTE, W. T. How to Draw a Graph, 1963, *Proceedings of the London Mathematical Society*, *13*, 743-768.

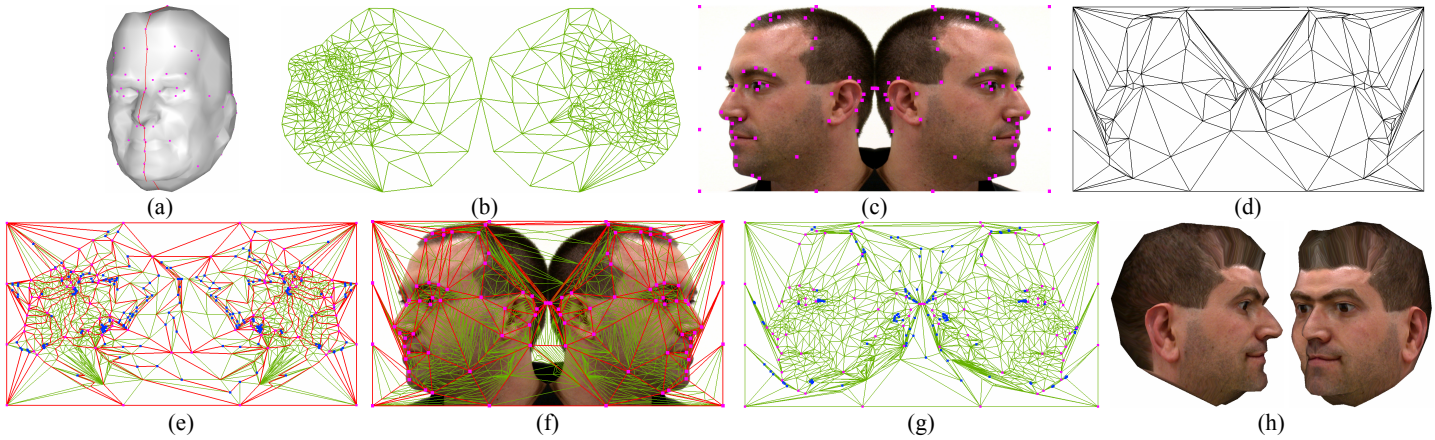


Figure 9: Texturing a head model: (a) 3D Model; (b) input texture atlas (two head halves); (c) texture with positions P_C; (d) space triangulation $T(P_C)$; (e) matching mesh triangulation $T_{M^*}(V_C)$, Steiner vertices shown in blue; (f) triangulation embedding; (g) final mesh M_{new}; (h) textured model.

Figure 10: Texturing the Igea data set (4,443 faces). Close to a hundred constrained vertices (red dots) were added to place features and stitch the central seam. The green lines in (b) indicate the constraints displacement compared to the input parameterization.

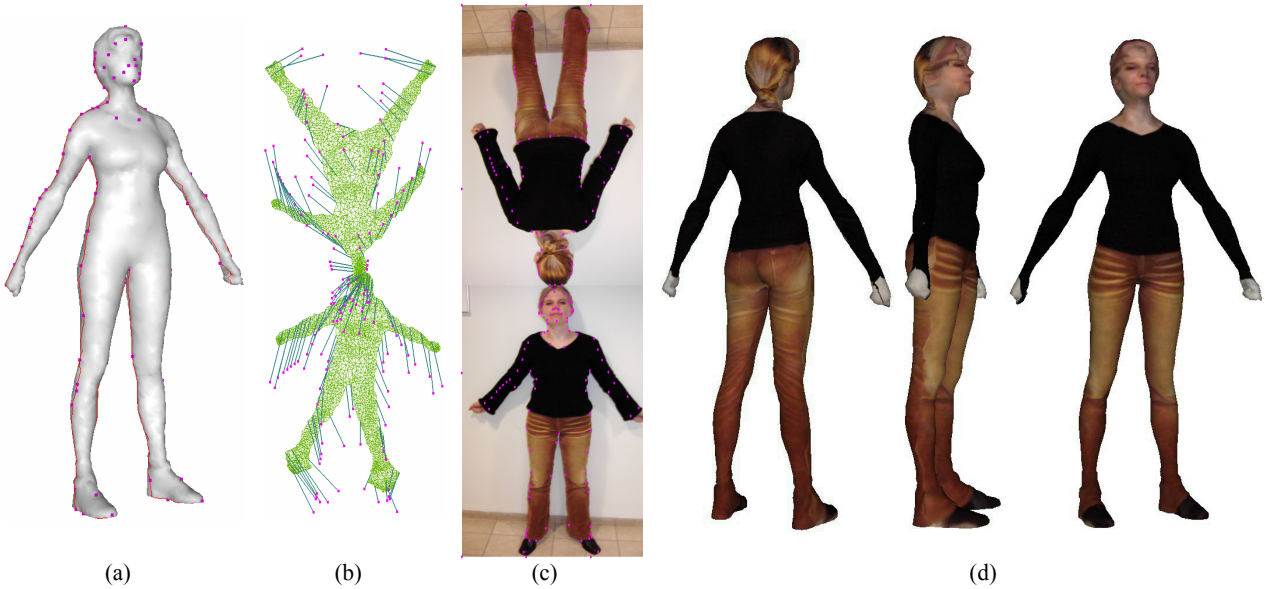


Figure 11: Texturing a full body scan (7,996 faces and 160 constraints).