Efficient computation of constrained parameterizations on parallel platforms[☆]

Theodoros Athanasiadis[a], Georgios Zioupos[b], Ioannis Fudos[a,*]

[a]Department of Computer Science, University of Ioannina, GR45110 Ioannina, Greece
[b]Department of Mathematics, Aristotle University of Thessaloniki, GR54124 Thessaloniki, Greece

Abstract

Constrained isometric planar parameterizations are central to a broad spectrum of applications. In this work, we present a non linear solver developed on OpenCL that is efficiently parallelizable on modern massively parallel architectures. We establish how parameterization relates to mesh smoothing and show how to efficiently and robustly solve the planar mesh parameterization problem with constraints. Furthermore, we demonstrate the applicability of our approach to real-time cut-and-paste editing and interactive mesh deformation.

1. Introduction

The purpose of mesh parameterization is to obtain a piecewise linear map, associating each face of the mesh with a surface patch on the parameterization domain. The parameterization domain is the surface that the mesh is parameterized on. Since the geometric shape of the parameterization surface will typically be different than the shape of the original mesh, angle and area distortion is introduced. Maps that minimize the angular distortion are called *conformal*, maps that minimize area distortion are called *authalic*, and maps that minimize distance distortion are called *isometric*. In this work, we deal with constrained *isometric* planar parameterizations. These maps are central to a broad spectrum of applications such as texture mapping, mesh completion, morphing and deformation transfer.

An important goal of parameterization is to obtain bijective (invertible) maps. The bijectivity of the map guarantees that there is no triangle flipping or overlapping. This is an important guarantee for certain applications, especially in the presence of user defined constraints on the vertices. On a planar parameterization domain a map may exhibit local or global bijectivity. Local bijectivity is achieved when there are no local triangle flips in the local neighborhoods of the mesh, whereas global bijectivity is achieved when there is no global mesh overlapping. Generally, global bijectivity is harder to achieve. Nevertheless, for most applications local bijectivity is sufficient.

The existing planar parameterization methods can be classified into two categories (for an extensive survey see [1]) : (i) methods that solve only linear systems, for example [2],[3],[4] and (ii) methods that use some kind of non-linear optimization. Typical methods of the former category, especially the earlier ones, have no guarantee for local or global bijectivity and usually offer inferior results as compared to the latter. Nevertheless, they are usually very fast and can be useful even as an initial solution for non-linear approaches. For example, in [5] although the energy minimized is non linear, a linear system is solved to obtain an initial parameterization of the mesh on the plane.

Amongst the latter category, several methods use some form of constrained or unconstrained non-linear optimization. These methods either reformulate the problem (resulting in non linearity) [6],[7] or directly minimize a non linear energy term [5],[8]. An indicative example is the work of [6] where the parametrization problem is reformulated in terms of angles subject to a set of constraints that ensure planarity and triangle validity of the final parameterization. Another example is the work of [7] where the authors use a set of vertices of the mesh called *cone singularities* to absorb the Gaussian curvature. This idea was further extended in [9] where the authors first determine automatically the location and the target curvatures of the singularities. They then proceed by solving a discrete Poisson equation on the mesh vertices to compute edge lengths and compute

[☆]The work in this paper was partially supported by a Heraclitus II grant through the operational programme "Education and Lifelong Learning" which is co-financed by Greece and the European Union through the European Social Fund

[*]fudos@cs.uoi.gr

the final embedding using a linear least squares methodology based on the computed edge lengths. A related work is [10] based also on *cone singularities* where a non linear solver is used to minimize the corresponding metric and compute the final parameterization.

For practical applications there is usually an additional requirement to accommodate user defined or automatically imposed constraints on the vertices of the parameterization. Generally, these constraints can be categorized into two groups: *soft constraints* that are approximately satisfied in the least squares sense and *hard constraints* that are precisely satisfied. Methods based on energy minimization can support soft constraints by adding a quadratic term to the energy function that measures the distance between the vertices and the desired location. Nevertheless, for linear approaches the additional term usually breaks the guarantees for bijectivity even for parameterizations on convex domains [1]. Hard constraints are even more difficult to support. Some methods can be extended to enforce hard constraints by the use of *Lagrange* multipliers [3]. However, such methods do not guarantee parameterization bijectivity.

In this work, we deal with the problem of computing a bijective planar parameterization of a mesh, subject to hard constraints. Additionally, soft constraints can be trivially supported due to the formulation of the problem. More specifically, this paper makes the following technical contributions:

- Establishes the relation between mesh smoothing and parameterization techniques and derives a simplified formulation for the *isometric* parameterization problem.
- Presents an efficient parallel implementation of a non-linear solver along with a number of heuristics that speed up substantially the parallel realization on modern hardware.
- Presents an iterative topological untangling process that solves efficiently the constrained parameterization problem.
- Demonstrates the applicability of the parallel solver on realizing the feature cut-and-paste design paradigm.

The rest of the paper is organized as follows. Section 2 offers theoretical background for mesh smoothing and establishes how it is related to parameterization. Section 3 describes the core of our constrained parallel solver for isometric parameterizations. Section 4 presents an application of our solver on cut-and-paste design. Finally, Section 5 offers conclusions.

2. Isometric parameterization

2.1. Mesh Smoothing Preliminaries

Before explaining the connection between the parameterization and the smoothing problem, we define three element types: (i) the *physical* element which is obtained through a mapping, possibly with area and angle distortion, of an element of the original mesh on the parameterization space, (ii) the *reference* element which is constructed by placing one node at the origin and the other nodes at unit lengths along the cartesian axes, and (iii) the *ideal* element which depends on the desired properties of the final mesh (see [11], [12]).

Furthermore, we define two affine mappings. The first mapping from the *reference* element x_r to the *ideal* element x_i is defined as :

$$x_i = \mathbf{W} * x_r \tag{1}$$

where matrix $\mathbf{W}$ is the edge matrix of the *ideal* element. The second mapping from the *reference* element x_r to the *physical* element x is defined as :

$$x = \mathbf{A} * x_r + x_0 \tag{2}$$

where matrix $\mathbf{A}$ is the edge (Jacobian) matrix of the *physical* element and x_0 is the vector with the coordinates of the first vertex. The matrix $\mathbf{A}$ holds information about the volume, the area, and the orientation of the *physical* element while x_0 controls its translation.

Based on the above definitions the shape matrix from the *ideal* to the *physical* element was defined in [11] as:

$$\mathbf{S} = \mathbf{A}\mathbf{W}^{-1} \tag{3}$$

and the associated barrier *shape* quality metric (η_{shape}) : $\mathbb{R}^{n\times n} \to \mathbb{R}$ as :

$$\eta_{shape} = \frac{||\mathbf{S}||_{\mathbf{F}}^2}{n \det(\mathbf{S})^{2/n}} \tag{4}$$

where for surface and volume meshes n is 2 and 3 respectively. The above metric can be used in an optimization process as an objective function to minimize over the vertices to obtain an optimal mesh. This quality metric assumes that each element has positive and non-zero determinants and consequently non-zero local area or volume. Furthermore, the barrier form is used to enforce positive Jacobian determinants to prevent folding. The mappings are depicted in Figure 1.

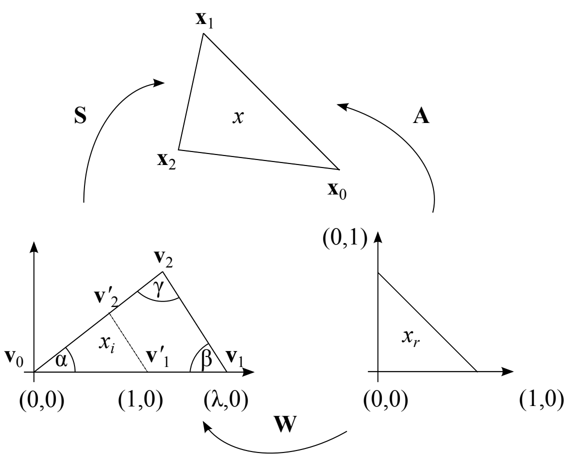


Figure 1: Ideal triangle $\triangle v_0 v_1 v_2$ and its similar triangle $\triangle v'_0 v'_1 v'_2$ on $\mathbb{R}^2$ along with the corresponding mappings.

2.2. Shape matrix construction for conformal parameterization

As noted in the previous section the definition of the *ideal* element depends on the desired properties of the final mesh. Therefore to preserve the angles of a triangle of the original mesh, we define on the parameterization space an *ideal* triangle $\triangle v_0 v_1 v_2$ with the same angles. Moreover for reasons that will become apparent, we define its similar triangle $\triangle v'_0 v'_1 v'_2$ with base $||v'_0 v'_1|| = 1$ and $v'_0 = v_0$ (see Figure 1) where:

$$\frac{||v'_0 v'_1||}{||v_0 v_1||} = \lambda, \quad \lambda > 0 \tag{5}$$

with the use of basic trigonometry on $\triangle v'_0 v'_1 v'_2$ we may further define the coordinates of the point v'_2 as :

$$v'_{2x} = \frac{\cot\hat{\alpha}}{\cot\hat{\alpha} + \cot\hat{\beta}} \quad , \quad v'_{2y} = \frac{1}{\cot\hat{\alpha} + \cot\hat{\beta}} \tag{6}$$

Therefore, from (5) and the definition of **W** :

$$\mathbf{W} = \begin{bmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{bmatrix} = \frac{1}{\lambda}\begin{bmatrix} 1 & v'_{2x} \\ 0 & v'_{2y} \end{bmatrix} = \frac{1}{\lambda}\mathbf{W}' \tag{7}$$

and from equations (6) and (7) :

$$\mathbf{W}^{-1} = \lambda \mathbf{W}'^{-1} \tag{8}$$

$$= \lambda \begin{bmatrix} 1 & -\frac{v'_{2x}}{v'_{2y}} \\ 0 & \frac{1}{v'_{2y}} \end{bmatrix} = \lambda \begin{bmatrix} 1 & -\cot\hat{\alpha} \\ 0 & \cot\hat{\alpha} + \cot\hat{\beta} \end{bmatrix} \tag{9}$$

it follows that :

$$\mathbf{S} = \mathbf{A}\mathbf{W^{-1}} = \mathbf{A}(\lambda \mathbf{W}'^{-1}) = \lambda \mathbf{A}\mathbf{W}'^{-1} = \lambda \mathbf{S}' \tag{10}$$

Matrices W' and S' are defined on the triangle $\triangle v'_0 v'_1 v'_2$. Moreover, using (10) the barrier *shape* metric (4) is defined as :

$$\eta_{shape} = \frac{||\mathbf{S}||^2_{\mathbf{F}}}{\det(\mathbf{S})} \overset{(10)}{=} \frac{||\lambda \mathbf{S}'||^2_{\mathbf{F}}}{\det(\lambda \mathbf{S}')}$$

$$= \frac{\lambda^2 ||\mathbf{S}'||^2_{\mathbf{F}}}{\lambda^2 \det(\mathbf{S}')} = \frac{||\mathbf{S}'||^2_{\mathbf{F}}}{\det(\mathbf{S}')} = \eta'_{shape} \tag{11}$$

showing that the barrier *shape* quality metric is scale invariant. Therefore, to obtain angle preserving parameterizations, only the angles of the triangles are required to compute the above matrices.

2.3. Connection with MIPS energy

Hormann and Greiner's MIPS method [5] was the first parameterization method that supported free boundaries and aimed at computing a parameterization that minimized the *Dirichlet energy per parameter-space area*. Since this energy is minimal for conformal mappings this gives parameterizations that are "as conformal as possible". To show how it relates to the *shape* quality metric that targets angle preserving mappings we start with the definition of **A** :

$$\mathbf{A} = [\vec{v_1} - \vec{v_0}, \vec{v_2} - \vec{v_0}] = \begin{bmatrix} v_{1x} - v_{0x} & v_{2x} - v_{0x} \\ v_{1y} - v_{0y} & v_{2y} - v_{0y} \end{bmatrix} \tag{12}$$

and the shape matrix $\mathbf{S}'$:

$$\mathbf{S}' = \mathbf{A}\mathbf{W}'^{-1} \overset{(12)}{=} [\vec{v_1} - \vec{v_0}, \vec{v_2} - \vec{v_0}]\mathbf{W}'^{-1}$$

$$\overset{(9)}{=} [\vec{v_1} - \vec{v_0}, \vec{v_2} - \vec{v_0}] \begin{bmatrix} 1 & -\cot\hat{\alpha} \\ 0 & \cot\hat{\alpha} + \cot\hat{\beta} \end{bmatrix}$$

$$= [\vec{v_1} - \vec{v_0}, (\vec{v_2} - \vec{v_1})\cot\hat{\alpha} + (\vec{v_2} - \vec{v_0})\cot\hat{\beta}] \tag{13}$$

Furthermore, we have for $\det(\mathbf{S}')$:

$$\det(\mathbf{S}') \overset{(8)}{=} \det(\mathbf{A}\mathbf{W}'^{-1})$$

$$= \det(\mathbf{A})\det(\mathbf{W}'^{-1})$$

$$\overset{(9)}{=} (\cot\hat{\alpha} + \cot\hat{\beta})\det(\mathbf{A}) \tag{14}$$

Finally, it can be shown (see Appendix A) that :

$$||\mathbf{S}'||^2_{\mathbf{F}} = (\cot\hat{\alpha} + \cot\hat{\beta})(c^2\cot\hat{\gamma} + a^2\cot\hat{\alpha} + b^2\cot\hat{\beta}) \tag{15}$$

Using (11),(14) and (15) we get :

$$\eta'_{shape} = \frac{1}{2}\frac{a^2\cot\hat{\alpha} + b^2\cot\hat{\beta} + c^2\cot\hat{\gamma}}{\det(\mathbf{A})} \tag{16}$$

From (16), we derive that the barrier shape quality metric is equal to the half of the MIPS energy [5].

2.4. *Isometric parameterization*

To obtain an area-preserving parameterization the area of each triangle on the parameterization space should tend to match its original area (E) on the mesh : $\det(A) \to 2E$. To measure this deviation a usual metric is [13]:

$$\frac{2E}{\det(\mathbf{A})} + \frac{\det(\mathbf{A})}{2E} \tag{17}$$

Therefore, a metric for the area preservation can be defined as:

$$\eta_{area} \overset{(17,14)}{=} \frac{2E(\cot\hat{\alpha} + \cot\hat{\beta})}{\det(\mathbf{S}')} + \frac{\det(\mathbf{S}')}{2E(\cot\hat{\alpha} + \cot\hat{\beta})}$$
$$\overset{(10)}{=} \det(\mathbf{S}) + \frac{1}{\det \mathbf{S}} \tag{18}$$

where we define for each *ideal* triangle $\frac{1}{\lambda} = \sqrt{2E(\cot\hat{\alpha} + \cot\hat{\beta})}$. Unlike the shape metric, η_{area} is not scale invariant. More specifically, the scale factor of each *ideal* triangle defines the desired area on the final parameterization. By combining the two metrics η_{shape} for shape preservation and η_{area} for area preservation, we get the simplified combined metric that targets isometric parameterizations :

$$\eta_{isometric} = \eta_{shape} \cdot \eta_{area} = ||\mathbf{S}|| + \frac{||\mathbf{S}||}{\det(\mathbf{S})^2} \tag{19}$$

Therefore, to define the $\mathbf{S}$ for each triangle i we need three scaling factors $[\lambda_i, \lambda_i \cot\hat{\alpha}_i, \lambda_i \cot\hat{\beta}_i]$ computed from the original mesh.

3. Constrained isometric parameterizations

There are several alternatives in minimizing the non linear metric for isometric parameterizations. For example, in the works of [5] and [12] a non-linear solver is used in which each node is individually optimized based on the objective function. However, it is not always feasible to efficienlty parallelize such an approach due to the arising data dependencies. For this reason, we have opted to use a preconditioned conjugate gradient approach and optimize all nodes simultaneously.

Conjugate gradient methods comprise a class of algorithms for unconstrained optimization. These methods have very low memory requirements and have linear convergence for most problems. An advantage of this class of algorithms is that only the objective function value and the gradient of the objective function are used during the optimization phase. Therefore, they do not require knowledge of the sparsity structure of the Hessian and are suitable for large scale optimization.

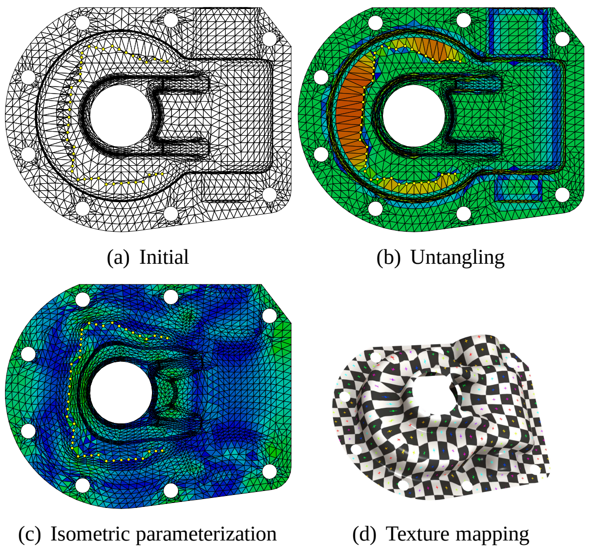


Figure 2: Constrained parameterization of the Casting model [14]. Area deformation is also depicted (blue and red colors correspond to high and low distorted areas respectively).

Another important advantage of these algorithms is that it is possible to implement them with only BLAS-1 operations. There operations can be implemented very efficiently on modern hardware [16].

To obtain an initial solution to the parameterization problem we may use one of the established linear parameterization approaches. This can be easily done by using the standard parameterization techniques based on barycentric coordinates [17],[18]. The boundary vertices are mapped to the boundary vertices of a convex polygon with the same number of vertices and in the same order. Then, the interior vertices are placed in such a way that each vertex is the centroid of its neigh-

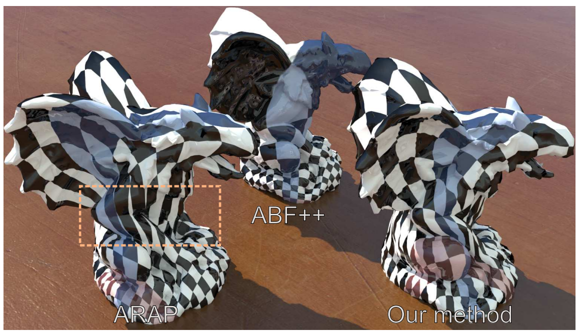


Figure 5: Comparison of parameterization results for the Gargoyle model. The ARAP parameterization is non bijective in the highlighted area.

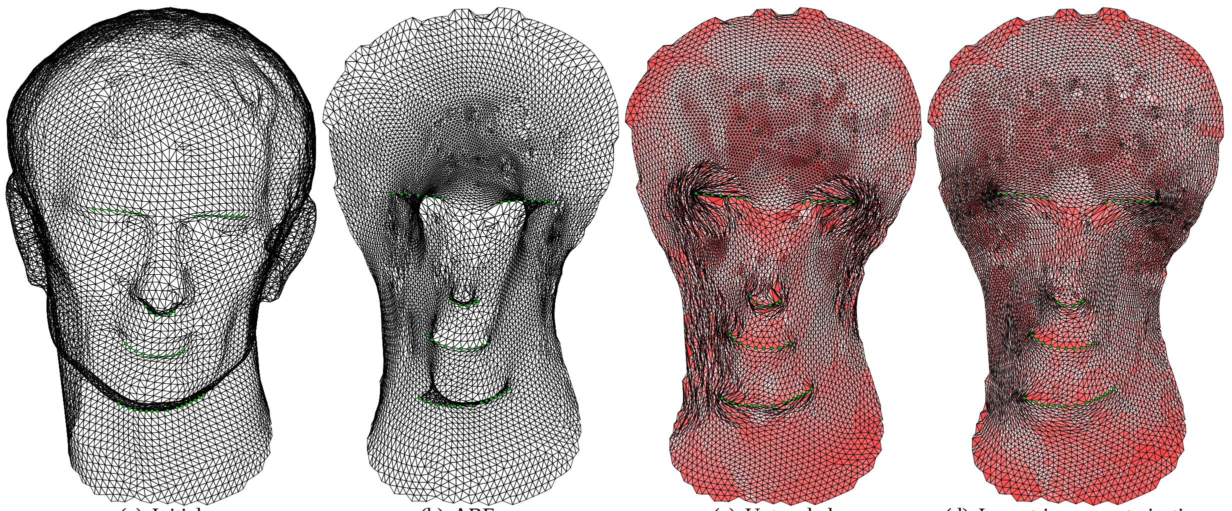


Figure 3: Constrained parameterization comparison. The constraints are : (i) the outer boundary and (ii) a set of internal nodes around the eyes and the nose area.

boring vertices. Following this approach, there are two issues to take into account : (1) the shape of the boundary polygon and (2) how to map the boundary vertices to the polygon. For the boundary polygon usual choices are the unit circle and the unit quad whereas for the mapping usual approaches are parameterization methods such as the *chord length* or *centripetal* parameterization.

In this process, an important problem is that for most practical examples the boundary vertices do not form a convex polygon and therefore the obtained parameterizations may not be bijective. Another issue is that if we use this parameterization as an initial solution for conjugate gradient this can be very far from the optimal solution resulting in slow convergence. A better alternative for computing an initial solution in our approach, is to use a parameterization technique such as [2],[3], or even [6] that minimize angular distortion. Naturally, in the presence of additional internal constraints and non convex boundaries the resulting mapping is not expected to be bijective. For example, as demonstrated in Figure 3(b), constraining a set of internal vertices and using the ABF++ method [6] results in a final parameterization that is not bijective.

3.1. Preconditioning

Non linear CG can be preconditioned by choosing an appropriate positive definite preconditioner matrix. Any matrix that approximates $\nabla^2 f(x^*)^{-1}$ is a good preconditioner for nonlinear functions. Therefore, a reasonable choice for such a matrix is the inverse of the diagonal of the Hessian matrix. Nevertheless, if x is far from a local minimum, the diagonal of the Hessian may not be positive-definite. Another possible CG preconditioning strategy is to compute an approximation to $\nabla^2 f(x^*)^{-1}$ generated by a quasi-Newton update formula of the Broyden family [19], [20]:

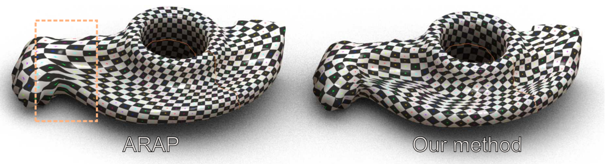


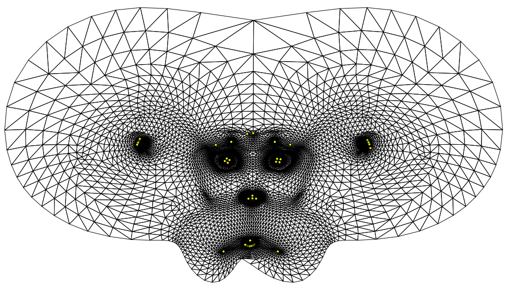
Figure 6: Comparison between the ARAP parameterization and our solver result. The highlighted region is not locally bijective.

$$H_{k+1} = v_k^T H_k v_k + \rho_k s_k s_k^T \tag{20}$$

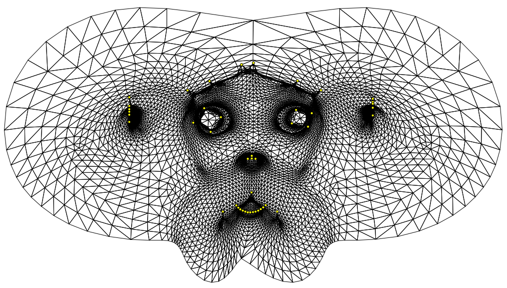
where $\rho_k = 1/y_k^T s_k$, and

$$v_k = I - \rho_k y_k s_k^T \tag{21}$$

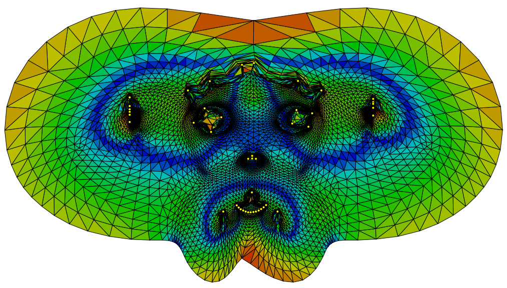
here, $s_k = x_{k+1} - x_k$, $g_k = \nabla f(x_k)^T$, and $y_k = g_{k+1} - g_k$. The above mentioned approach is known as Limited-memory BFGS (L-BFGS) and is useful for solving large problems since this method maintains a simple and compact approximation of the Hessian matrices. Therefore, it is suitable when the Hessian is dense or the second derivatives are costly to compute. A modified ver-



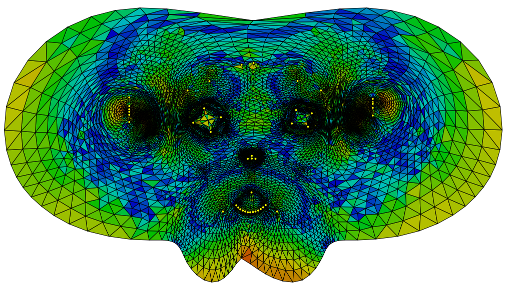
(a) Initial



(b) ABF++



(c) Untangled



(d) Isometric parameterization

Figure 4: Constrained parameterization of the Suzanne model [15]. The constrained nodes consist of the outer boundary (86 nodes) and 45 internal nodes (Figure a) that are translated to new positions (Figure b).

sion of H_k is stored implicitly, by storing a certain number (m) of the vector pairs (s_i, y_i) that are used in (20) and (21). Figure 7 demonstrates the convergence of the BFGS preconditioned nonlinear CG, with Hestenes and Stiefel [21] update formula and different choices of m. The basic Hessian matrix H_k^0 plays an important role in

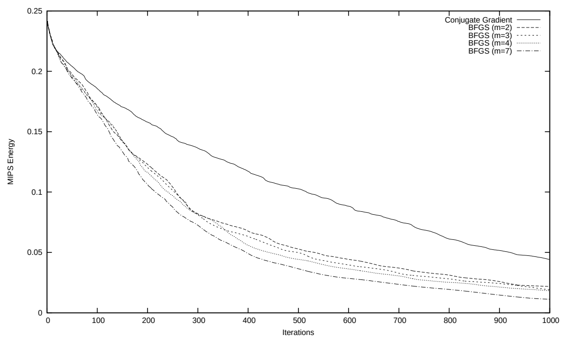


Figure 7: Comparison of the Conjugate Gradient convergence with and without preconditioning.

the performance and the robustness of the algorithm. A popular choice is to set $H_k^0 = \gamma_k I$ where γ_k is a scaling factor. The scale factor generally is used as an estimate of the size of the true Hessian matrix to ensure that the search direction is well scaled, and as a result the full step length can be accepted (usually if it satisfies the *Wolfe* conditions [22]). Moreover, the scaling prohibits the eigenvalues of the approximate Hessian from becoming large. There are several approaches in the literature for computing γ_k, such as those presented in [23],[20] and [24]. In practice, we found the scaling factor proposed by [24] to perform the best.

3.2. Parallel implementation and results

As an API for our implementation, we have used OpenCL 1.1. The core of our solver is the L-BFGS method described in [19] and the software is available at `http://www.cs.uoi.gr/~fudos/smi2013.html`. Algorithm 1 summarizes the basic steps of the solver. To maximize the performance of our implementation,

Algorithm 1 Preconditioned Conjugate Gradient

```
1: $d_0 = -H_0 g_0$
2: $\beta_k = \frac{y_{k-1}^T g_k}{y_{k-1}^T d_{k-1}}$
3: $d_k = -H_k g_k + \beta_k d_{k-1}$
4: $x_{k+1} = x_k + a_k d_k$
```

we have considered a number of factors. Two important considerations in modern GPUs are: (i) the efficient memory usage so as to achieve maximum memory bandwidth and (ii) tuning the instruction usage so as to achieve the maximum instruction throughput [25]. Therefore, we have used a number of heuristics to optimize the parallel performance of our non linear solver such as :

- Reduction of the divergence of the parallel kernels.
- Coalesced memory operations.
- Exact line searches.

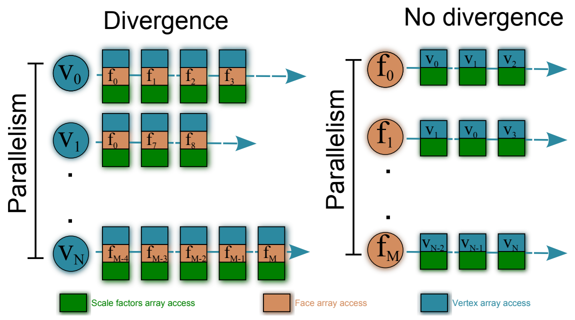


Figure 8: The vector ∇f can be computed in parallel for each vertex with indirect memory access and execution divergence (left) or for each face with more coalesced memory transactions and without divergence (right).

The costlier step in the solver is the computation of the gradient ∇f. The gradient can be computed per vertex, where each computation should access the adjacency list of the vertex and the corresponding scaling factors of the adjacent faces. If we use this straightforward approach to parallelize the computation, two problems arise on massively parallel platforms: random memory access patterns and divergence due to the different adjacency lists. In our implementation, we follow a two step approach to tackle these problems. First, we use an auxiliary buffer where we store the contribution of each face to its adjacent vertex gradients and then we add up the corresponding partial contributions to obtain the final gradient vector. This process is an efficient way to parallelize the computation on the faces (see Figure 8). Each face accesses only its three scaling factors and its vertices; consequently there are fewer levels of indirection while most of the read accesses are coalesced. The summation step still requires to randomly access the auxiliary buffer to compute the final gradient, but the most computationally intensive first part of the gradient computation is accelerated substantially.

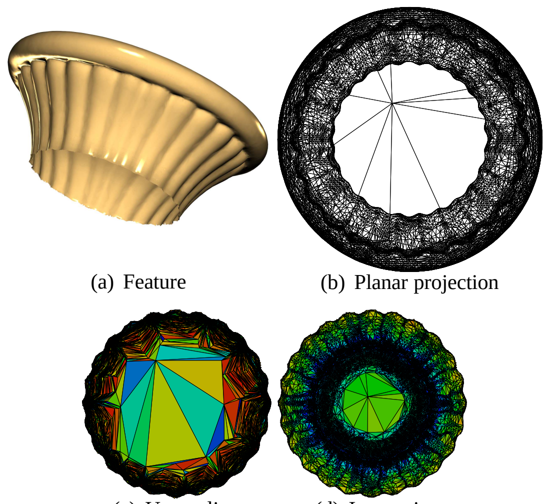
The other major performance issue in the performance of conjugate gradient methods is the line search, which requires sufficient accuracy to ensure that the search direction yields a descent. The line search is typically performed in two stages: a bracketing phase where we constrain the desirable step length, and an interpolation phase that computes the step length within this bracketed interval. Ideally, we would like to find the global minimizer for f. However, this approach usually requires many evaluations of the objective function f and possibly the gradient ∇f. Therefore, there is a trade-off between the accuracy of the line search and the computational cost. A common approach is to perform inexact line searches until some accuracy criteria are met. Common criteria are the *Wolfe* conditions [22].

The usual "strong" *Wolfe* condition requires an extra evaluation of the gradient of the function. This approach performs well for applications running on platforms with only a few threads like CPUs. On the contrary, the evaluation of the gradient on massively parallel platforms is relatively expensive. Even though we improved the speed of this computation with the previously described approach, due to the complexity of the gradients the evaluation of ∇f on such platforms is still an order of magnitude slower than the evaluation of f. Thus, in many cases it is more efficient to carry out more function evaluations instead of using the "strong" *Wolfe* condition to stop the line search process prematurely. A related issue is the scaling of the Hessian matrix, the scaling ensures that the search direction is well-scaled and that the full step ($a_k = 1$) is accepted in most iterations.

Having made the above observations, we have opted for a hybrid approach. First we scale the Hessian and check the strong *Wolfe* condition with the full step. If the step is rejected, we perform an exact line search using the derivative free *Brent* method [26] having as a limit the square of the hardware double accuracy. This is also the accuracy limit for the specific line search method. Since the *Brent* method needs a bracketing triplet (x_a, x_b, x_c), satisfying $f(x_b) < f(x_a)$ and $f(x_b) < f(x_c)$, in the case of the isometric metric, we take into account the discontinuity of the function when an element becomes inverted. This discontinuity occurs along the search direction d_k using a negative step. To avoid this case, and assuming that the line search begins with a bijective mapping, we find a low bound ϵ where the function is defined. To compute ϵ, we start from a small value and follow a procedure similar to the backtracking line search approach. This approach worked reliably in our experiments; the initial bracketing phase typically costs three or four function evaluations whereas the *Brent* line search method usually costs less than ten function evaluations.

Figures 5 and 6 illustrate the parameterization of two meshes from [14] with our solver and show a comparison with the ARAP method that also targets isometric parameterizations [4]. In both the cases the ARAP method failed to produce bijective parameterizations.

Furthermore, we have obtained performance results minimizing the isometric metric and using an NVIDIA Tesla c2070 and an Intel i7-2600k processor. Table 1 illustrates the scalability of the solver while running a fixed number of iterations with different level of detail whereas Table 2 and Table 3 provide a comparison with the publicly available parameterization software of [4] in terms of running times and parameterization quality.



(a) Feature (b) Planar projection

(c) Untangling (d) Isometric

Figure 9: Constrained boundary parameterization (a) Original feature (b) Planar projection (not bijective) (c) Untangled mesh (with area deformation) (d) Further optimized isometric parameterization (also showing area deformation).

3.3. *Parameterization with hard constraints*

Computing a parameterization with positional constraints is a difficult research problem. Traditionally, planar parameterization methods support such constraints. For example Levy et al [2] incorporates soft and hard constraints in the linear system formed while Desbrun et al [3] use Lagrange multipliers to add positional constraints. However in the presence of a lot of constraints, these methods can fail to compute a bijective parameterization even if such a parameterization is known to exist. The inherent problem is that the bijectivity of the resulting map is based on certain properties of the energy minimized. Thus, adding linear terms in the system formed for soft constraints or removing variables for hard constraints can break theoretical guarantees for bijectivity. More recent methods methods such as [6] also support positional constraints but bijective parameterizations are not guaranteed (Figures 3,4). Other methods proceed by adding Steiner vertices to ensure the bijectivity [27] or compute a parameterization by partitioning the mesh into patches that are then parameterized while maintaining the smoothness and continuity between them [28].

When the boundary vertices are fixed, the constrained parameterization problem is equivalent to mesh untangling. Existing methods for untangling employ geometric or optimization-based approaches. For example in [29] the authors seek to maximize the minimum element area and formulate the mesh untangling problem as a series of local linear programming problems. Knupp et al [30] optimize a global function that measures the difference between the absolute and signed element area. The latter requires a custom solver and a modified line search approach since the gradient of that function is not continuous.

Another approach to handle folded meshes is to modify the shape quality metric η_{shape} so as to incorporate an untangling process as suggested by [31]. Having established the connection between the parameterization and the shape metric based smoothing, we may use this method with the appropriate isometric equations of section 2 and solve the resulting convex problem. In our experiments this approach worked reasonably well for small to medium sized meshes. Nevertheless, it further requires the solution of a series of non linear problems. More importantly, this objective function can take extreme values on large meshes and consequently it is very difficult to optimize it effectively. For the above reasons we follow a different two step approach. First, we consider the boundary of the parameterization along the constraints imposed fixed and we treat the constrained parameterization problem as an untangling problem which is solved with simple topological operations. Afterwards, we proceed by optimizing the untangled map with the non linear solver in order to improve the quality of the parameterization. For such an approach to work in the case of free boundaries a good initial solution should be available. In this case of free boundaries we use the method of [6] to obtain this solution.

To derive the topological operator per vertex we formulate the untangling problem as a minimization problem on the inverted elements:

$$min\, f(x) = -\sum_{i}^{n} \det(\mathbf{A}) \tag{22}$$

For the above function if we compute the gradients of a triangle with vertices v_0,v_1,v_2 and edges e_0,e_1,e_2 with respect to vertex v_0 we obtain:

$$\det A = (e_{0y} \cdot e_{2x} - e_{0x} \cdot e_{2y}) \Rightarrow$$

$$-\frac{\partial \det(A)}{\partial v_{0x}} = -(v_{2y} - v_{1y}) = -e_{1y} \quad (23)$$

$$-\frac{\partial \det(A)}{\partial v_{0y}} = (v_{2x} - v_{1x}) = e_{1x} \quad (24)$$

This means that the gradient of the negative triangle area with respect to one of its vertices is equal to the opposite triangle edge rotated $\frac{\pi}{2}$ clockwise about the triangle normal. If we sum all the gradients of the adjacent triangles this gives a decent direction and therefore we can move all the vertices along the corresponding lines:

$$v_i^{k+1} = v_i^k + \delta \sum_{j \in N_i} CW_{\frac{\pi}{2}}(e_{opposite}) \quad (25)$$

where N_i is the set of adjacent inverted triangles around vertex i. Unfortunately, if we only apply the above operator there are numerical instability issues. The problem is directly related to the invariance of the area deformation under shears [13]. A shear does not change the area of a triangle and therefore extremely sheared triangles may occur during the minimization process. The problem becomes severe when the total area of the triangles around a vertex is very small and the corresponding valley has almost collapsed. In that case the edge lengths may become arbitrarily large especially if we use a bad initial solution. For this reason, we added a correction term that also reduces the shearing of a triangle when the area of that triangle drops below a certain threshold. This correction term is a vector perpendicular to the gradient of the area and the orientation depends on the angles of the triangle as illustrated in Figure 10. The shear direction is perpendicular to the gradient and the addition of this term keeps the decent property of the total operator. The addition of this term topologically "bends" the decent direction of the vertex towards the centroid of the valley resulting in reduced edge lengths.

Algorithm 2 summarizes the basic steps of the untangling process. where the factors are defined based on a user defined minimum area β as :

$$factor_j = \begin{cases} 1 - \frac{|\det(A_j)|}{\beta} & , -\beta \leq \det(A_j) \leq \beta \\ 0 & , otherwise \end{cases} \quad (26)$$

This way we move all the vertices along the decent direction according to a δ value without performing an exact line search. We have avoided exact line searches for two reasons. First, the gradient computations are affordable, unlike the non linear functions of the previous section, and therefore it is better to perform more gradient than function evaluations. Second, in this way we only need local topological operations that are more robust arithmetically. The latter is important especially for modern GPUs since it allows us to use only single precision operations that reduce the memory bandwidth requirements and are much faster than the corresponding double precision operations. Figures 2,3,9,4 and 14 illustrate some untangling results that are used as an intermediate step for computing isometric parameterizations. The running time of the untangling process depends on the number of constraint vertices and the initial solution. For example, for the mesh of Figure 3 with 13095 faces and 241 constrained vertices the time for the untangling process was 100ms on a Tesla c2070. In all the experiments performed, the running time is primarily affected by the non linear optimization step that follows the untangling process. For typical meshes of up to 50k triangles, with hundreds of constraints, the average time for untangling was less than a second.

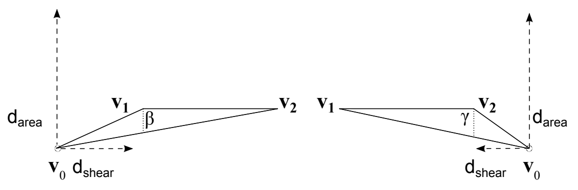


Figure 10: Shear operator for two different triangle configurations. The shear operator is parallel to the opposite edge and the direction depends on which of the two opposite angles is greater than $\frac{\pi}{2}$. If both angles are acute then no shear operator is applied.

Algorithm 2 Untangling Process

```
1: $\delta \leftarrow \frac{1}{2}$
2: for $k = 0$ to convergence or $\delta < \epsilon$ do
3:     for $i = 0$ to $N$ do
4:         $d_{area} = \sum_{j \in N_i} CW_{\frac{\pi}{2}}(e_{opposite})$
5:         $d_{shear} = \sum_{j \in N_i} factor_j \cdot shear(e_{opposite})$
6:         $v_i^{k+1} = v_i^k + \delta(d_{area} + d_{shear})$
7:     end for
8:     if $\sum |\det(\mathbf{A}^{k+1})| \geq \sum |\det(\mathbf{A}^k)|$ then
9:         $\delta \leftarrow \frac{\delta}{2}$
10:    end if
11:    $k = k + 1$
12: end for
```

4. An application: mesh editing through cut-and-paste

In the context of mesh editing, the cut-and-paste paradigm extracts a characteristic feature from a source surface and copies it on a target surface. The user usually selects a surface region which has two parts: the base surface and the detail surface. The base surface is a connected subset of the original surface and the detail surface is used as a feature to be copied/pasted. The goal is to replace the detail part of the target surface with the detail part of the source surface. The key question is how to transfer correctly the details from the source surface to the target. The target base surface may be altered as well to achieve smooth blending.

The detail surface can be stored either as a height-field or a parametric volume map [32]. The drawback of the height-field representation is that usually general features may be thick or contain overhangs and they can not be properly represented. To paste the detail surface to a target model, the corresponding vertices of the target model are moved based on the detail map. In 3D, a smooth attachment of boundaries between the pasted feature and its base is sometimes required. One possible way to resolve this issue is to perform a union operation between the two models and then apply a blending function along the boundaries of the features [33]. However, blending functions for arbitrary meshes is a difficult problem to solve efficiently and robustly. Snap-paste [34] suggests an iterative algorithm for aligning the feature and the base surface, by positioning and deforming both surfaces. However they do not avoid the need for remeshing.

Another approach [35],[36],[37] is the modification of differential coordinates instead of directly changing spatial coordinates. The mesh geometry is then implicitly modified after reconstructing the surface from the differential coordinates. This method has the advantage of reducing deformation artifacts that may appear after feature pasting. Nevertheless, sharp features are difficult to support. A tool for interactive cut-and-paste operations that uses the above approach is Mesh Mixer [38]. A distance preserving local parameterization is computed around the pasted area using approximate geodesic distances [39] and both the base surface and the feature are deformed using variational surface deformation techniques [37]. This method is very fast but suffers from some robustness issues as mentioned by the authors [39].

Existing cut-and-paste editing methods can be roughly categorized into two broad groups. The first group, uses mesh fusion to blend the source surface and target surface directly [40],[33]. The second group first extracts a base surface as a medium between the source surface and the target surface, and then transfers the details to the target surface via the base surface [32],[41],[42]. The former pays more attention to the smoothness of the boundaries at the joint of the source and target surfaces. The latter focuses on the global deformation of the source surface according to the target surface.

4.1. Our approach

In our approach, we have adapted tools and techniques from the differential coordinate and the mesh fusion scheme. We use an adaptive tessellation scheme that is built on top of the GPU tessellation unit. The tessellation is adaptive in the sense that only areas of interest are tessellated while the rest of the mesh remains untessellated. The feature area is parameterized with our non-linear solver and is subsequently stored in a 2D floating point texture. This texture is used in the tessellation evaluation shader to offset the base surface so as to create the feature in real time. The two main problems we address in this approach are: (i) how to support arbitrary features for pasting operations and (ii) how to smoothly blend the pasted feature and the base surface. Our approach is fast enough to support interactive operations.

The parameterization of the pasted feature should satisfy the following requirements to support arbitrary features:

- The boundary of the parameterization should be arbitrary. Convex boundary parameterizations are useful only for simple features and usually exhibit high distortion.

- The parameterization should be bijective and isometric. This allows us to store the feature in a 2D texture. The area and angular distortion of the parameterization should be minimal to avoid under sampling during the storage phase.

- Hard and soft constraints on the vertices should be handled robustly. This implies that the parameterization is bijective regardless of the constraints on the vertices (if such a parameterization exists).

All the above requirements can be satisfied by our non-linear solver. Therefore, provided that we have parameterized the base area using one of the established methods, we proceed by computing a constrained parameterization of the feature. More specifically, we fix the boundary along with other user defined internal points on corresponding positions of the base surface parameterization. To find the specific correspondences we may follow either an automatic or a user-driven approach. To automatically compute the constraints we project the feature boundary on the base surface and use the corresponding (s,t) parameterization coordinates of the base surface to constrain the feature parameterization. A limited user intervention may be necessary in cases of very complex boundaries. Aletrnatively, other methods can be used such as the snapping algorithm of [34].

Regarding the smooth attachment problem, a way to paste the feature without distorting its form is to treat the base surface as an elastic object and smoothly deform it to produce an appropriate area for pasting. The above observation led us to use a method that works very well in practice and produces intuitive results without deformin the feature surface. Our approach is based on the use of Radial basis functions (RBFs) [43],[44]. RBFs are a tool for interpolating data and are used to derive the displacement in any location in the space. RBF applications include mesh warping, medical imaging, and surface reconstruction [45],[46].

An RBF, s, is a function of the form:

$$s(y) = \sum_{i=1}^{N} w_i \phi(||y - x_i||) + p_m(y) \tag{27}$$

where ϕ is called the basic function, w_i is a scalar coefficient, $x_1, ..., x_N$ are the pairwise distinct control points of the RBF, and $p(x)$ is an m degree polynomial. Popular choices of basic function include the thin-plane and the polyharmonic splines. To compute the coefficients we need to solve a linear system of order equal to the number of RBF control points. The right part of the system is filled with the known values at the control points and the left part is filled with the values of the basis function between the control points and values that depend on the polynomial used. Once the coefficients have been calculated, any arbitrary point can be expressed through the function s (27). In our experiments we have observed that the polyharmonic and the Wendland [47] basic functions produce the most natural looking results. Furthermore, we use two sets of control points. The first set of stationary control points is positioned at the boundaries of the base region and fixes the boundaries of the domain. The coordinates of each control point are used to set the right part. The second set of moving control points is positioned at the boundaries of the pasted feature. More specifically, we project the boundary points of the pasted feature on the base surface and use the projected points as control points. To set the right part for this set of control points we use the corresponding coordinates from the boundary of the feature. Having established an initial correspondence between the boundary of the feature and the base surface, we may apply an additional transformation on the feature to better place it on top of the surface. This additional transformation only affects the right part of the linear system. Therefore, for reasons of efficiency we compute and store the LU factorization only once and perform a back substitution when this transformation matrix changes. This way, the user can interactively move and rotate the feature on the base surface.

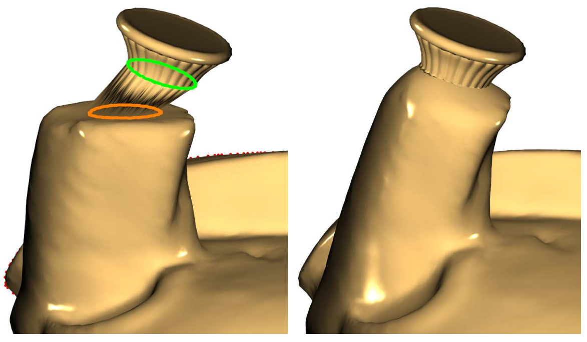

(a) Undeformed (b) RBF Interpolation

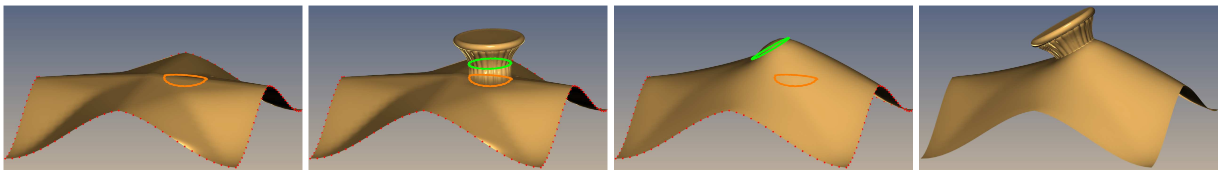
Figure 12: Base surface deformation.

After transforming the base surface with RBFs the feature is pasted with the use of the tessellation unit. More specifically, in the tessellation control shader we sample the feature texture to decide if the base surface needs tessellation. If the base surface is tessellated, we offset the new vertices using the feature texture and apply the additional transformation matrix (if there is one).

The process of RBF interpolation and pasting is illustrated in Figure 11 whereas Figures 12 and 13 illustrate the deformation result for different base surfaces. The reader is also referred to the supplementary material that accompanies this work and is also available at `http://www.cs.uoi.gr/~fudos/smi2013.html`.

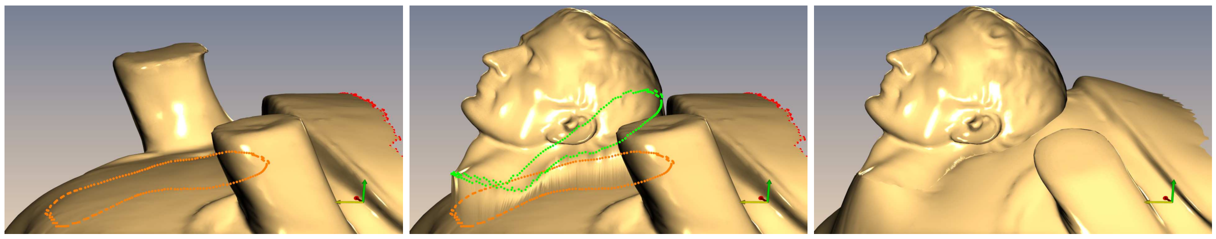
5. Conclusions

We presented an efficient parallel scheme to compute *isometric* parameterizations subject to soft and hard constraints. Our approach is based on establishing a theoretical connection between the well studied non linear mesh smoothing and the *isometric* parameterization. Using this scheme, we have successfully carried out a large number of experiments to validate the solver on parameterizing meshes up to one million triangles in a



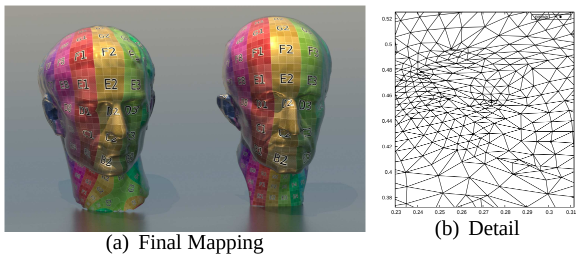
(a) (b) (c) (d)

Figure 11: Overview of the pasting process. (a) Base surface, fixed and movable control points are colored red and orange respectively. (b) Pasted feature, the boundary of the feature is green. (c) We may apply an additional transformation on the feature and deform the base surface with RBFs. (d) Final pasting result.



(a) Base surface (b) Cut-and-paste without RBF (c) RBF Interpolation

Figure 13: Cut-and-paste example.



(a) Final Mapping (b) Detail

Figure 14: Mapping the Julius model to the plank model. We first parameterize the plank model (right) with ABF++ and we pin the vertices around the ears and eyes of the Julius model (left) at the corresponding positions of the plank parameterization. (a) The final unfolded parameterization is locally bijective (b) Detail of the final parameterization.

few seconds on a modern GPU. Finally, as an application of our solver, we have parameterized and stored free form features on floating point textures and then by exploiting the capabilities of the tessellation unit on modern GPUs we have supported cut and paste operations in real time.

A possible extension of our work would be the usage of a hierarchical decimation scheme similar to [5] or [6] to accelerate the convergence of the non-linear solver. Going a step further, we could exploit the connection between the parameterization and the smoothing problem to compute the *isometric* parameterization of volume meshes on 3D domains. By doing so, more properties of the original mesh could be preserved such as the volume preserving morphing and blending operations.

Another direction for future research is to provide a formal proof regarding regarding the convergence of the untangling process. This could be reached from connecting our process to standard gradient decent approaches and analysing the numerical instabilities that occur near the solution.

Appendix

$$\begin{aligned}
\|\mathbf{S}'\|_{\mathbf{F}}^2 = trace(\mathbf{S}'^T\mathbf{S}') &\overset{(13)}{=} (\vec{v_1} - \vec{v_0}) \cdot (\vec{v_1} - \vec{v_0}) + ((\vec{v_2} - \vec{v_1}) \cot\hat{\alpha} + \\
&(\vec{v_2} - \vec{v_0}) \cot\hat{\beta}) \cdot ((\vec{v_2} - \vec{v_1}) \cot\hat{\alpha} + (\vec{v_2} - \vec{v_0}) \cot\hat{\beta}) \\
&= \|\vec{v_1} - \vec{v_0}\|_2^2 + \|\vec{v_2} - \vec{v_1}\|_2^2 \cot\hat{\alpha}^2 + \|\vec{v_2} - \vec{v_0}\|_2^2 \cot\hat{\beta}^2 + \\
&2(\vec{v_2} - \vec{v_1}) \cdot (\vec{v_2} - \vec{v_0}) \cot\hat{\alpha} \cot\hat{\beta} \\
&\overset{(CL)}{=} \|\vec{v_1} - \vec{v_0}\|_2^2 + \|\vec{v_2} - \vec{v_1}\|_2^2 \cot\hat{\alpha}^2 + \|\vec{v_2} - \vec{v_0}\|_2^2 \cot\hat{\beta}^2 + \\
&(\|\vec{v_2} - \vec{v_1}\|_2^2 + \|\vec{v_2} - \vec{v_0}\|_2^2 - \|\vec{v_1} - \vec{v_0}\|_2^2) \cot\hat{\alpha} \cot\hat{\beta} \qquad (28)
\end{aligned}$$

where we use the *cosine law* (CL) on the last equation, if we further define the triangle edge lengths as :

$$a = \|\vec{v_2} - \vec{v_1}\|_2, b = \|\vec{v_2} - \vec{v_0}\|_2, c = \|\vec{v_1} - \vec{v_0}\|_2 \qquad (29)$$

Table 1: Numerical results for different levels of detail while running a fixed number of iterations (=1000) on the Blech mesh ($Area(rms)$ = 0.490457 and $Angular$ = 0.101195).

Level	# vertices	# faces	i7 (ms)	c2070 (ms)	Speedup	$Area(rms)_{1000}$	$Angular_{1000}$
Lod1	1815	3456	999	1072	0.93	0.114456	0.0058238
Lod2	7085	13824	1704	1196	1.42	0.114835	0.0052326
Lod3	27993	55296	4708	1744	2.69	0.142656	0.0069087
Lod4	111281	221184	18156	3322	5.46	0.319943	0.0464538
Lod5	443745	884736	73828	9795	7.53	0.452182	0.0741808

Table 2: Time comparison between our solver and ARAP

method	model	# vertices	# faces	bijectivity	iters	# evals f	time(s)
ARAP	Blech	27993	55296	yes	7	-	1.95
Solver(1000)	Blech	27993	55296	yes	1000	2633	1.65
Solver(1500)	Blech	27993	55296	yes	1500	3658	2.38
ARAP	gargoyle	24406	48672	no	4	-	1.08
solver(5000)	gargoyle	24406	48672	yes	5000	15570	10.16
solver(10000)	gargoyle	24406	48672	yes	10000	30551	19.88
ARAP	julius	209083	416286	yes	28	-	59.02
solver(2500)	julius	209083	416286	yes	2500	5583	14.71
solver(4000)	julius	209083	416286	yes	4000	8943	23.92

it follows that :

$$\begin{aligned}
\|\mathbf{S}'\|_{\mathbf{F}}^2 &\overset{(28),(29)}{=} (1-\cot\hat\alpha\cot\hat\beta)c^2 + (\cot\hat\alpha+\cot\hat\beta)(a^2\cot\hat\alpha + b^2\cot\hat\beta)\\
&= c^2\frac{\sin\hat\alpha\sin\hat\beta - \cos\hat\alpha\cos\hat\beta}{\sin\hat\alpha\sin\hat\beta} +\\
&\quad (a^2\cot\hat\alpha + b^2\cot\hat\beta)\frac{\cos\hat\alpha\sin\hat\beta + \sin\hat\alpha\cos\hat\beta}{\sin\hat\alpha\sin\hat\beta}\\
&= c^2\frac{-\cos(\hat\alpha+\hat\beta)}{\sin\hat\alpha\sin\hat\beta} + (a^2\cot\hat\alpha + b^2\cot\hat\beta)\frac{\sin(\hat\alpha+\hat\beta)}{\sin\hat\alpha\sin\hat\beta}\\
&= c^2\frac{-\cos(\pi-\hat\gamma)}{\sin\hat\alpha\sin\hat\beta} + (a^2\cot\hat\alpha + b^2\cot\hat\beta)\frac{\sin(\pi-\hat\gamma)}{\sin\hat\alpha\sin\hat\beta}\\
&= c^2\frac{\cos\hat\gamma}{\sin\hat\alpha\sin\hat\beta} + (a^2\cot\hat\alpha + b^2\cot\hat\beta)\frac{\sin\hat\gamma}{\sin\hat\alpha\sin\hat\beta}\\
&= \frac{\sin\hat\gamma}{\sin\hat\alpha\sin\hat\beta}(c^2\cot\hat\gamma + a^2\cot\hat\alpha + b^2\cot\hat\beta)\\
&= (\cot\hat\alpha+\cot\hat\beta)(c^2\cot\hat\gamma + a^2\cot\hat\alpha + b^2\cot\hat\beta) \qquad (30)
\end{aligned}$$

References

[1] K. Hormann, B. Lévy, A. Sheffer, Mesh parameterization: theory and practice, in: ACM SIGGRAPH 2007 courses, SIGGRAPH '07, ACM, New York, NY, USA, 2007.

[2] B. Lévy, S. Petitjean, N. Ray, J. Maillot, Least squares conformal maps for automatic texture atlas generation, ACM Trans. Graph. 21 (3) (2002) 362–371.

[3] M. Desbrum, M. Meyer, P. Alliez, Intristic parameterization of surface meshes, in: Eurographics Proceedings, 2002, pp. 209–218.

[4] L. Liu, L. Zhang, Y. Xu, C. Gotsman, S. J. Gortler, A local/global approach to mesh parameterization, in: Proceedings of the Symposium on Geometry Processing, SGP '08, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2008, pp. 1495–1504.

[5] K. Hormann, G. Greiner, MIPS: An efficient global parametrization method, in: P.-J. Laurent, P. Sablonnière, L. L. Schumaker (Eds.), Curve and Surface Design: Saint-Malo 1999, Innovations in Applied Mathematics, Vanderbilt University Press, Nashville, TN, 2000, pp. 153–162.

[6] A. Sheffer, B. Lévy, M. Mogilnitsky, A. Bogomyakov, Abf++: fast and robust angle based flattening, ACM Trans. Graph. 24 (2) (2005) 311–330.

[7] L. Kharevych, B. Springborn, P. Schröder, Discrete conformal mappings via circle patterns, ACM Trans. Graph. 25 (2) (2006) 412–438.

[8] P. V. Sander, J. Snyder, S. J. Gortler, H. Hoppe, Texture mapping progressive meshes, in: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH '01, ACM, New York, NY, USA, 2001, pp. 409–416.

[9] M. Ben-Chen, C. Gotsman, G. Bunin, Conformal flattening by curvature prescription and metric scaling, Computer Graphics Forum 27 (2) (2008) 449–458.

[10] B. Springborn, P. Schröder, U. Pinkall, Conformal equivalence of triangle meshes, ACM Trans. Graph. 27 (3) (2008) 77:1–77:11.

[11] P. M. Knupp, Algebraic mesh quality metrics, SIAM J. Sci. Comput. 23 (1) (2001) 193–218.

[12] P. M. Knupp, Introducing the target-matrix paradigm for mesh optimization via node-movement, in: S. Shontz (Ed.), Proceed-

Table 3: Comparison of quality between our solver and ARAP

Method	Model	L2(min)	L2(max)	L2(rms)	Area(min)	Area(max)	Area(rms)	Angular
ARAP	Blech	0.719205	1.35083	0.0812426	0.565892	2.01864	0.15528	0.005267
Solver(1000)	Blech	0.721571	1.39323	0.0628883	0.595902	4.40744	0.13110	0.008731
Solver(1500)	Blech	0.708662	1.48147	0.0573159	0.574745	4.47457	0.12180	0.005351
ARAP	Gargoyle	0.408068	7465.28	74.4895	0.000164	8.2565	1.03866	0.421756
Solver(5000)	Gargoyle	0.18212	25.9687	3.81626	0.005277	31.9465	1.78979	0.108743
Solver(10000)	Gargoyle	0.226868	16.4789	2.36491	0.010048	22.0854	1.63601	0.121859
ARAP	Julius	0.63778	56.1057	0.203055	0.00486726	4.269	0.34954	0.043286
Solver(2500)	Julius	0.575291	2.03919	0.155464	0.299203	8.559	0.32586	0.041716
Solver(4000)	Julius	0.618229	2.00369	0.130395	0.315212	6.409	0.28318	0.045691

ings of the 19th International Meshing Roundtable, Springer Berlin Heidelberg, 2010, pp. 67–83.

[13] P. Degener, J. Meseth, R. Klein, An adaptable surface parameterization method, in: In Proceedings of the 12th International Meshing Roundtable, 2003, pp. 201–213.

[14] Aim@shape, Aim@shape shape repository v4.0
URL `http://shapes.aimatshape.net`

[15] Blender, Blender Suite, Open Source Suite, http://www.blender.org (Blender Foundation).

[16] NVIDIA, Cublas (2012).
URL `http://developer.nvidia.com/cuda/cublas`

[17] W. T. Tutte, How to draw a graph, Proc. London Math. Soc 13 (1963) 743–768.

[18] M. S. Floater, Parametrization and smooth approximation of surface triangulations, Computer Aided Geometric Design 14 (1997) 231–250.

[19] J. Nocedal, Updating quasi-Newton matrices with limited storage, Mathematics of Computation 35 (1980) 773–782.

[20] D. C. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Mathematical Programming 45 (1989) 503–528.

[21] M. R. Hestenes, E. Stiefel, Methods of Conjugate Gradients for Solving Linear Systems, Journal of Research of the National Bureau of Standards 49 (6) (1952) 409–436.

[22] P. Wolfe, Convergence conditions for ascent methods, SIAM Review 11 (2) (1969) pp. 226–235.

[23] D. Shanno, K.-H. Phua, Matrix conditioning and nonlinear optimization, Mathematical Programming 14 (1978) 149–160.

[24] M. Al-Baali, Improved hessian approximations for the limited memory bfgs method, Numerical Algorithms 22 (1999) 99–112.

[25] NVIDIA, Opencl best practices guide (July 2009).

[26] R. Brent, Algorithms for Minimization Without Derivatives, Dover books on mathematics, Dover Publications, 2002.

[27] I. Eckstein, V. Surazhsky, C. Gotsman, Texture mapping with hard constraints, Computer Graphics Forum 20 (3) (2001) 95–104.

[28] V. Kraevoy, A. Sheffer, C. Gotsman, Matchmaker: constructing constrained texture maps, ACM Trans. Graph. 22 (3) (2003) 326–333.

[29] L. A. Freitag, P. Plassmann, Local optimization-based simplicial mesh untangling and improvement, International Journal for Numerical Methods in Engineering 49 (1-2) (2000) 109–125.

[30] P. M. Knupp, Matrix norms and the condition number: A general framework to improve mesh quality via node-movement, in: Eighth International Meshing Roundtable (Lake Tahoe, California, 1999, pp. 13–22.

[31] J. Escobar, E. Rodrguez, R. Montenegro, G. Montero, J. Gonzlez-Yuste, Simultaneous untangling and smoothing of tetrahedral meshes, Computer Methods in Applied Mechanics and Engineering 192 (25) (2003) 2775 – 2787.

[32] Y. Furukawa, H. Masuda, K. T. Miura, H. Yamato, Cut-and-paste editing based on constrained b-spline volume fitting, Computer Graphics International Conference 0 (2003) 222.

[33] K. Museth, D. E. Breen, R. T. Whitaker, A. H. Barr, Level set surface editing operators, ACM Trans. Graph. 21 (3) (2002) 330–338.

[34] A. Sharf, M. Blumenkrants, A. Shamir, D. Cohen-Or, Snappaste: an interactive technique for easy mesh composition, Vis. Comput. 22 (9) (2006) 835–844.

[35] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, H.-Y. Shum, Mesh editing with poisson-based gradient field manipulation, in: SIGGRAPH '04: ACM SIGGRAPH 2004 Papers, ACM, New York, NY, USA, 2004, pp. 644–651.

[36] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, H.-P. Seidel, Laplacian surface editing, in: SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, ACM, New York, NY, USA, 2004, pp. 175–184.

[37] M. Botsch, O. Sorkine, On linear variational surface deformation methods, IEEE Transactions on Visualization and Computer Graphics 14 (1) (2008) 213–230.

[38] R. Schmidt, K. Singh, meshmixer: an interface for rapid mesh composition, in: ACM SIGGRAPH 2010 Talks, SIGGRAPH '10, ACM, New York, NY, USA, 2010, pp. 6:1–6:1.

[39] R. Schmidt, C. Grimm, B. Wyvill, Interactive decal compositing with discrete exponential maps, ACM Trans. Graph 25 (2006) 605–613.

[40] T. Kanai, H. Suzuki, J. Mitani, F. Kimura, Interactive mesh fusion based on local 3d metamorphosis, in: Proceedings of the 1999 conference on Graphics interface '99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 148–156.

[41] H. Biermann, I. Martin, F. Bernardini, D. Zorin, Cut-and-paste editing of multiresolution surfaces, ACM Trans. Graph. 21 (3) (2002) 312–321.

[42] H. Fu, C. lan Tai, H. Zhang, Topology-free cutand-paste editing over meshes, in: In Geometric Modeling and Processing 2004, 2004, pp. 173–182.

[43] H. Wendland, Scattered Data Approximation, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2004.

[44] M. Buhmann, Radial Basis Functions: Theory and Implementations, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2003.

[45] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J.

McLennan, T. J. Mitchell, Smooth surface reconstruction from noisy range data, in: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, GRAPHITE '03, ACM, New York, NY, USA, 2003, pp. 119–ff.

[46] J. Carr, W. Fright, R. Beatson, Surface interpolation with radial basis functions for medical imaging, Medical Imaging, IEEE Transactions on 16 (1) (1997) 96 –107.

[47] H. Wendland, A. Mathematik, Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree, Advances in Computational Mathematics 4 (1995) 389–396.