Real-time content-aware texturing for deformable surfaces

Charalampos Koniaris
University of Bath
ck317@bath.ac.uk

Darren Cosker
University of Bath
dpc@cs.bath.ac.uk

Xiaosong Yang
University of Bournemouth
xyang@bournemouth.ac.uk

Kenny Mitchell
Disney Research
kmitchell@disneyresearch.com

Iain Matthews
Disney Research
iainm@disneyresearch.com

ABSTRACT

Animation of models often introduces distortions to their parameterisation, as these are typically optimised for a single frame. The net effect is that under deformation, the mapped features, i.e. UV texture maps, bump maps or displacement maps, may appear to stretch or scale in an undesirable way. Ideally, what we would like is for the appearance of such features to remain feasible given any underlying deformation.

In this paper we introduce a real-time technique that reduces such distortions based on a distortion control (rigidity) map. In two versions of our proposed technique, the parameter space is warped in either an axis or a non-axis aligned manner based on the minimisation of a non-linear distortion metric. This in turn is solved using a highly optimised hybrid CPU-GPU strategy. The result is real-time dynamic content-aware texturing that reduces distortions in a controlled way. The technique can be applied to reduce distortions in a variety of scenarios, including reusing a low geometric complexity animated sequence with a multitude of detail maps, dynamic procedurally defined features mapped on deformable geometry and animation authoring previews on texture-mapped models.

1. INTRODUCTION

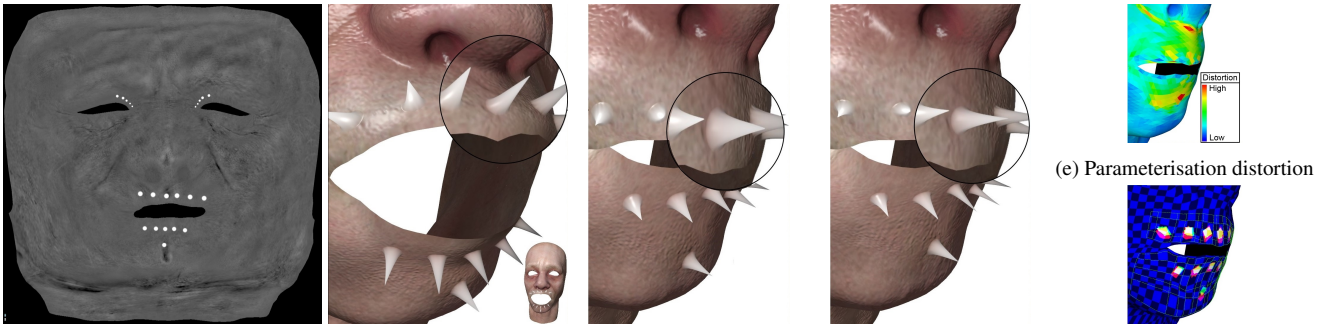
Texture mapping is the process of mapping detail to a surface using a parameterisation of the surface – the most common case being a 2D parameterisation of a 3D surface [7]. Some surface representations have natural parameterisations (e.g. NURBS), while others, such as polygonal meshes, require non-trivial methods to obtain such parameterisations. In the latter group, parameterisations are represented in the same way as vertices are: as piecewise-linear approximations to continuous functions. Naturally, dense discretisations of these functions provide higher-quality approximations. A metric for the quality of a parameterisation is the distortion introduced by the mapping [14]. Another source of distortion is the error introduced by the piecewise-linear approximation. Both of these sources of distortion depend on the parameterisation algorithm used, as well as the 3D surface discretisation.

An additional common source of distortion in the parameterisation is caused by animating the mesh (figure 2). As the mesh undergoes a non-Euclidean transform, the parameterisation ceases to be optimal. This manifests visually as mapped features on the deforming primitives appearing elastic. That can be desired in some cases (skin, rubber, etc) but not all. If specific parts of the mapped features need to appear rigid (as determined artistically or by some other means), this elastic behaviour can cause suspension of disbelief. Typical examples may include horns, armour elements and scales. To reduce this type of distortion, either the parameterisation needs to be regenerated, or the mesh needs to be manually edited (see section 2).

Contributions: We introduce a novel method to reduce distortions caused by the deformation of a parameterised surface in real-time. This allows a variety of texture mapped detail to be applied to an animated model without it undergoing visually undesirable behaviours – specified in a simple manner by an initial user input process.

Distortions are reduced over a pre-specified area in texture space, and the surface deformation or animation can be arbitrary and does not need to be known a priori. The distortion minimisation algorithm is guided by a user-supplied distortion control map of the specified region of interest (ROI). The distortion control map and region are supplied as a single preprocessing step along with the authored texture space information. The alternative to supplying control information in 2D texture space (as we propose here), is to include additional vertices into the geometry and manually ensure that their movements behave in an appropriate manner during animation. However, we argue that this approach is more complex and time consuming than our proposed approach – which is simply to highlight in 2D (e.g. using a standard paint package) the rigidity of regions based on brightness value. This and the selection of the ROI are the only user inputs required by our system (as well as artistic creation of the mesh and texture space detail). Multiple non-overlapping ROIs can be independently selected and re-parameterised.

For a given animation frame, we use our non-linear optimisation strategy to calculate a piecewise-linear warp that, when applied to the ROI's parameterisation, reduces distortions as a function of the current frame's surface deformation and the supplied rigidity information. We provide two variants of the algorithm that trade-off between performance and quality: an axis-aligned warp (requiring only a sparse set of lines as user input) that deforms a rectilinear grid mapped over the ROI along horizontal and vertical offsets, and a non-axis-aligned warp (requiring only a set of points as user in-



(a) Displacement map (b) Rest pose (c) Deformed pose (d) Deformed pose after re-parameterisation (e) Parameterisation distortion (f) Rigid features

Figure 1: Preserving texture space detail (in this example, spikes) given a coarse animated mesh (in this example, a face). Spikes are modelled as a displacement map (a) mapped on the rest pose (b). Deforming the face introduces distortion to the parameterisation (e), causing the skin and spikes to stretch (c). Our algorithm automatically corrects the parameterisation in real-time, so that only the spikes remain rigid (d). The parameterisation is corrected in regions of interest given distortion control maps, which specify features that need to remain undistorted, and overlaying and warping rectilinear grids (f).

put), where the grid may be arbitrarily deformed.

Our paper is organised as follows. In the following section (2) we outline related work and our contributions to the described problem. We then present an overview of our method (3) and continue with describing all its parts: *user input* and *grid partitioning* (4), *grid optimisation* (5) and *rendering* (6). A variation of the method that allows for arbitrary grid warping is described next (7), followed by a brief description of the hybrid CPU-GPU optimisation (8). We then present the results, discuss details and issues arising from the use of the method and finally conclude the paper.

2. RELATED WORK

The majority of existing mesh parameterisation algorithms that minimise parameterisation distortions apply such metrics to the whole domain, thus not taking into account the nature of the data being mapped [8, 20, 5]. However, there is a variety of relevant work which we now briefly cover and contrast with our proposed solution.

Sander et al. [18] minimise a signal-stretch metric that allows reduction of distortions of any vector-valued function (the signal) defined over the domain. The metric is non-linear and the process requires a few minutes per model. While the technique is content-aware, it does not take into account temporal coherence of the parameterisation, which is important when considering a deforming mesh.

Sheffer and De Sturler [19] overlay a 2D uniform Cartesian grid on the texture parameterisation domain (as a 2D triangle mesh) and warp the grid so that the warped parameterisation minimises edge length distortions. While this technique uses an overlaid grid to warp the parameterisation, it is not content-aware and so ignores features which should remain rigid or fixed (i.e. non-sliding features). Our technique is also approximately three orders of magnitude faster, allowing the parameterisation warping to run in real-time (see section 9).

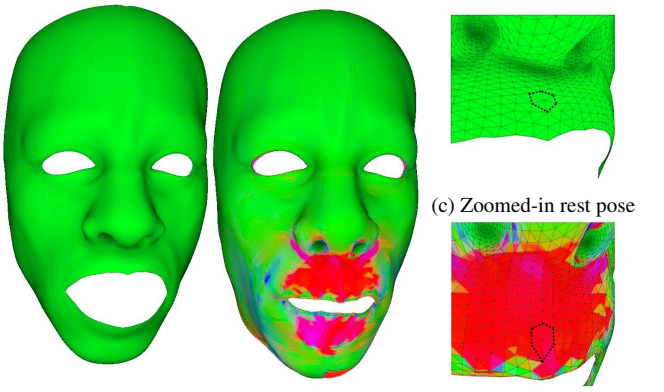
Ptex [4], by Burley and Lacewell, eliminates the need for explicit parameterisation by using the natural one afforded by subdivision surface quad-faces and providing anisotropic filtering between faces. While this eliminates many of the explicit parameterisation issues, such as distortions and seams, animated meshes still pose a problem, as the individual quads still deform and distortions are reintroduced.

If the parameterisation needs to remain constant, mesh deformation techniques can be employed to calculate a further constrained deformation, so that parameterisation distortions remain low. Such techniques [3, 21, 22] focus on editing complex meshes in a physically plausible and aesthetically pleasing way, while preserving geometric details. Barycentric coordinates are also used for mesh deformation, by deforming complex meshes using simple control cages [11, 10, 13, 2, 9]. While these techniques are generally efficient in calculating a new plausible mesh pose given a few control transformations, they do not focus on cases where detail in contained in the texture space and animation in a coarser fine mesh (typical for a e.g. video game or a real time graphics engine). Still, a number of techniques have been developed that approach mesh deformation from a content-sensitive point of view and as such, we briefly discuss them below.

Popa et al. [17] approach content-aware deformation by introducing local bending and shearing stiffnesses as factors in how a mesh deforms. Given such material information, and transformations for a number of anchor triangles, they calculate the deformation of the mesh as a weighted sum or blend of the anchor transformations. The material information is user- or data-driven, providing additional control on how parts of the mesh deform when editing it. As the anchor transformations are required to be a combination of rotations and uniform scales, the space of supported deformations is restricted.

Kraevoy et al. [12] focus on protecting vulnerable parts of a complex model under global non-uniform scaling. They define a vulnerability map on a volumetric grid that encloses the object, and transform the grid while respecting this map. While they estimate vulnerability based on *slippage* and *normal curvature*, the map can be user-driven. The technique focuses only on a very special deformation case (non-uniform scaling transform), so it's not applicable to more complex deformations.

Yang et al. [23] simulate skin sliding by remeshing the surface



(a) Rest pose (b) Deformed pose (c) Zoomed-in rest pose (d) Zoomed-in deformed pose

Figure 2: Texture space distortion in given animation of a mesh. Deforming the rest pose (a) introduces parameterisation distortions as elements of the geometric mesh change size and shape (c,d) while the texture space mesh remains constant. Deformation of texture space relative to the chosen rest pose (b) is marked with blue (u axis), red (v axis) and purple (both axes). Note that such distortion can be desired if the mapped material needs to behave in an elastic way (skin, rubber, etc).

based on resampling of the its parameter space. They use the *Force Density Method* (FDM) to construct embeddings of original and deformed patches into their parameter domains. As the technique deforms the actual geometry and force densities are specified on edges, so the deformed patch needs to be highly tessellated and the result is dependent on the triangulation, which reduces the flexibility and applicability of the method.

Our re-parameterisation method is in the same spirit as content-aware image retargetting [6, 16] but is generalized to handle arbitrary surface deformations in 3D space. Additionally, such techniques do not take into account sliding of features or temporal coherence of solutions.

In production and in practice, in order to reduce texture space distortions in sensitive regions of an animated mesh, artists need to manually add vertices, tightly bounding the rigid area and making sure that it does not distort under deformation. For example, for rigged models, the regions around joints are the most prone to distortions, so additional vertices may be placed. When the deformation is known, additional vertices can be placed appropriately so that deformation is spread to areas that do not contain any salient rigid features. The problem remains when the deformation is unknown or varying so much that adding and manually animating vertices becomes impractical. Procedurally generated detail, static or animated, provides an even greater challenge as the location of the additional vertices cannot be easily determined.

While previous methods focus on global texture space preservation, require complex models or focus on preserving details in specific mesh areas, our method provides distortion control of texture space deformations in flexible user-specified areas with real time computation. To the best of our knowledge, we are the first to present such a flexible method, allowing any detail representable in texture space (colour, bump/displacement maps, etc.) to be controlled in terms of its distortion given a lower detail mesh. In addition, our method requires no pre-training (other than creation of the simple distortion map) or prior knowledge of the underlying animation.

3. OVERVIEW

Our method is applicable to all parameterised, deformable surfaces, and takes into account salient, rigid features of a given static or animated detail map. The overall process (figure 3) can be summarised by the following steps:

1. **User input and grid partitioning.** A rectangular region (mapped to the unit square) is initially selected from the 2D parameterisation domain; this is the region that the algorithm will process. The rigidities associated with the ROI are provided at this stage as a greyscale texture map. We partition the ROI domain to a rectilinear grid. The grid is created by focusing grid cells around similar distortion control weights (section 4).
2. **Per-frame grid warping.** The rigidities and grid line coordinates are used as an input to our non-linear optimisation scheme which minimises texture-space deformation energy (section 5).
3. **Per-frame rendering.** After the deformed grid is calculated, we can render the surface using the adjusted parameterisation (section 6).

3.1 Notation

To aid clarity of exposition in the following sections, we outline our notation here before describing our technical solution. The ROI is expressed in 3D object space as $O(s,t)$ (rest pose) and $D(s,t)$ (deformed pose), where (s,t) are parametric coordinates on the unit square. The corresponding region in the 2D parameterisation is expressed as $T(s,t)$. The distortion control map is expressed as $R(s,t)$, containing continuous values in the range $[0..1]$ (non-rigid to rigid). Partial derivatives for any function $X(s,t)$ are written as $X'_s(s,t)$ and $X'_t(s,t)$. The dimensions of the distortion control map that we use are $W \times H$ and the dimensions of the rectilinear grid are $M \times N$ (horizontal and vertical lines). Unless noted otherwise explicitly, we will be using zero-based indexing. The 2D unit domain axes are specified as $\hat{\mathbf{s}}$ (horizontal) and $\hat{\mathbf{t}}$ (vertical). The 1D solutions for each axis are represented as $\mathbf{s}$ and $\mathbf{t}$, and have lengths of M and N respectively, while for the 2D variant, both solutions have dimensions of $M \times N$ each. As the formulas for the calculation of both axes are in many cases similar, we mainly will present formulas for a single axis and describe extrapolation to the other axis.

4. USER INPUT AND RECTILINEAR GRID PARTITIONING (STEP 1)

User input is provided in the form of a greyscale image map for distortion control and a ROI in the parameterisation space. As we remap the parameter space of the ROI, the shape of the region can be any shape that can be bijectively mapped to a rectangle. In our examples, we use scaled and rotated rectangles for their simplicity of converting between texture coordinates and (s,t) parametric coordinates (figure 4).

Optimisation performance for our algorithm depends on the resolution of the grid. Therefore, a grid with fewer lines will result in higher performance, as demonstrated in our results. Grid lines

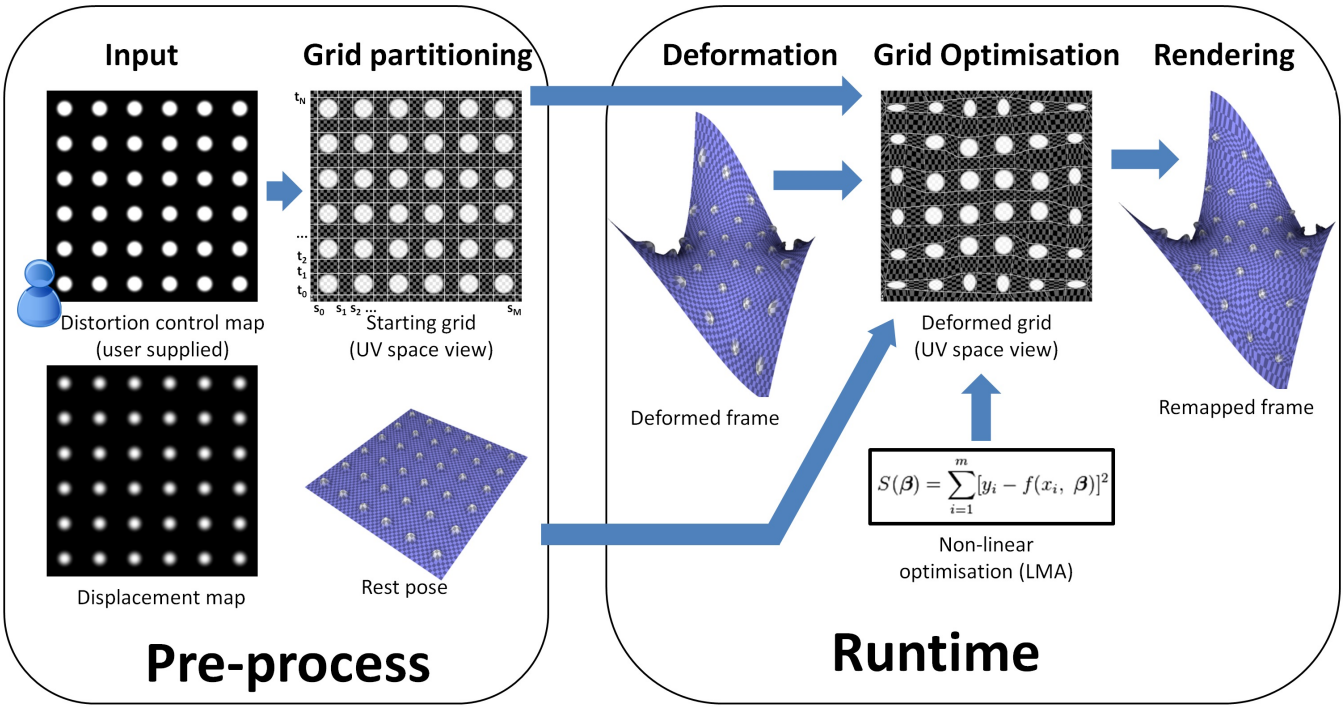


Figure 3: Method overview: Step 1: the starting grid is generated as a pre-process, given the distortion control map. The distortion control map is specified so that the bumps on the plane are preserved. When deforming, the original surface, deformed surface, distortion control map and starting grid are all used by the non-linear optimiser to calculate a new grid that minimizes a weighted distortion energy functional. The starting and warped grids are then used to warp the parameterisation function, which is used to sample the mapped surface detail when rendering.

should be chosen so that the resulting cells enclose as-similar-as-possible distortion control weights and rigid areas are enclosed in cells as tightly as possible. For relatively simple cases the grid line offsets can be automatically calculated with algorithm 1, otherwise it can be provided by a user.

This can be performed once as a preprocessing step, if the mapped detail (and thus the distortion control map) remains constant throughout deformation. Alternatively, if the distortion control map varies from frame to frame, the grid will either need to regenerate or be fine enough so that for any map similar values are still clustered together as tightly as possible.

5. GRID OPTIMISATION (STEP 2)

In this section we describe our approach for updating the texture space parametersation by minimising distortion energy given underlying mesh changes.

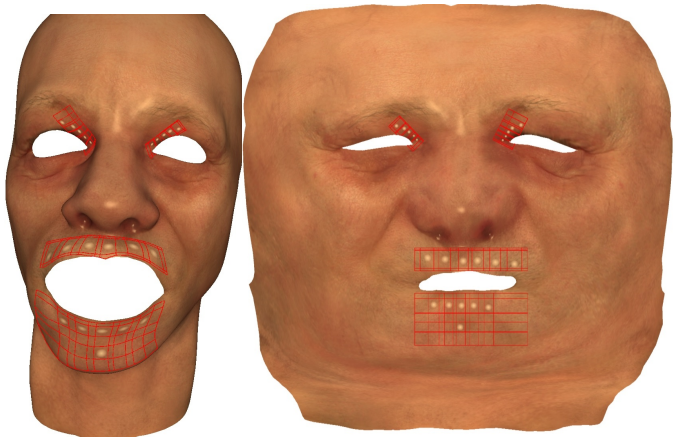
5.1 Energy Formulation

The distortion metric that we use is based on comparing the deformed remapped surface to the original in terms of stretch along the $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ directions. We define the total per-axis distortion energy as the sum of the individual per-cell, per-axis distortion energies. The horizontal energy is defined as:

$$E_s = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} E_{s_{ij}} \quad (1)$$

Data: R, W, H, e
Result: s,t
// Maximum per-column distortion control weight
for *each column* $j = 0 \rightarrow (W-1)$ **do**
  $r_j = \max_{i \in [0,H-1]} R(i,j)$;
end
// Calculate s lines
$s_0 \leftarrow 0$;
$idx \leftarrow 1$;
for *each column* $j = 0 \rightarrow (W-1)$ **do**
  if $\|r_j - r_{j+1}\| > e$ **then**
    if $r_j > r_{j+1}$ **then**
      $s_{idx} \leftarrow \frac{j}{W-1}$;
    else
      $s_{idx} \leftarrow \frac{j+1}{W-1}$;
    end
    $idx \leftarrow idx + 1$;
  end
end
$s_{idx} \leftarrow 1$;
// ...Similarly for t lines

Algorithm 1: Generating the rectilinear grid from a distortion control map R of size $W \times H$ with values in $[0,1]$ given a threshold e that separates



(a) Object space (b) Texture space

Figure 4: Four ROIs and starting grids in object space (a) and texture space (b) enclosing rigid points (shown in white) using the mesh of figure 1.

where the per-cell energy is defined as:

$$E_{s_{ij}} = \int_{t=t_i}^{t_{i+1}} \int_{s=s_i}^{s_{i+1}} R(s,t)F_s(s,t)dsdt \tag{2}$$

where $F_s(s,t)$ calculates stretch at a point (s,t) as the squared weighted difference of the lengths of the original and deformed/remapped geometric partial derivative along the $\hat{\mathbf{s}}$ direction at that point:

$$F_s(s,t) = (|f'(s)|\left\|D'_s(f(s),g(t))\right\|_2 - \left\|O'_s(s,t)\right\|_2)^2 \tag{3}$$

where f, g are the linear functions that remap s and t for the given cell. Similarly for the vertical energy:

$$F_t(s,t) = (|g'(t)|\left\|D'_t(f(s),g(t))\right\|_2 - \left\|O'_t(s,t)\right\|_2)^2 \tag{4}$$

It can be seen that movement of vertical lines does not result in any horizontal energy change and vice versa. Derivation of the energy can be found in appendix A.

5.2 Constraints

We wish to calculate axis-aligned grid lines that minimise the above distortion energy while satisfying the following requirements: a) no line fold-overs b) smooth line changes from frame to frame, and c) allow "seamless" solutions for looping animations. We also prefer local minima in our solution as this results in minimal sliding of features across texture space. These requirements can be formulated as boundary constraints in our optimisation, as they are compatible with local solutions – which in turn allows for faster optimisation than searching for a global solution. Below, we show constraints for the $\hat{\mathbf{s}}$ axis only, and refer to previous (known) and current (unknown) solutions as $\mathbf{s}^k$ and $\mathbf{s}^{k+1}$ respectively.

Fold-over Constraint Fold-over constraints prevent discontinuities in the remapping and are easily enforced by bounding a line between the midpoints of the segments between the line and its adjacent neighbours:

$$\frac{s_{i-1}^k + s_i^k}{2} < s_i^{k+1} < \frac{s_{i+1}^k + s_i^k}{2} \tag{5}$$

Smooth Line Constraint Smooth line changes can simply be enforced by restricting the movement of a line in a solution to a maximum offset o:

$$s_i^k - o < s_i^{k+1} < s_i^k + o \tag{6}$$

Looping Animation Constraint The above constraints result in local solutions, so the local minima requirement is satisfied. For a looping animation consisting of K frames we add the following constraint for the j-th frame:

$$d = \frac{K - |2(j+1) - K + 1| - 1}{2}$$
$$s_i^0 - do < s_i^{k+1} < s_i^0 + do$$

where s^0 is the solution for the first (or last) frame. This constraint effectively shifts the bounds so that the first and last solutions are matching, and solutions in between vary smoothly.

As the intersection of all these ranges might be $\emptyset$, we need to define a behaviour that gives precedence to one constraint over another. Given constraints of descending priority C_h, C_l, a merged constraint can be calculated as follows:

$$F(C_h, C_l) = \begin{cases} C_h, & \text{if } (C_l \cap C_h = \emptyset) \text{ or } (C_l \supseteq C_h) \\ C_l, & \text{if } C_l \subseteq C_h \\ C_h \setminus (C_h \cap C_l), & \text{otherwise} \end{cases}$$

Now, given the bound constraints for foldovers (C_F), smooth changes (C_S) and looping behaviour (C_L) we define the final bounds as $F(C_L, F(C_F, C_S))$ that give priority first to looping, then foldovers and finally smooth changes (looping is foldover-free).

5.3 Non-Linear Optimisation

Our overall optimisation is based on minimising the horizontal and vertical cell distortion energy integrals in eq. 2. We optimise this term using the Levenberg-Marquardt algorithm [15], as this also allows us to efficiently calculate local minima subject to the previously described bound constraints. The unknowns vector is the aggregation of all horizontal and vertical lines except the boundary ones. The starting point is the calculated solution from the nearest frame, as we want solutions to be as local as possible to achieve temporal coherence.

6. RENDERING (STEP 3)

There are two options for applying the new optimised parameterisation when rendering the distortion-corrected output. Given the

original and deformed grid lines we may either alter the geometry or the texture coordinates. Let $F_s(\mathbf{s}) = \mathbf{s}'$ and $F_t(\mathbf{t}) = \mathbf{t}'$ be the piecewise-linear functions that remap the original to the optimised grid. As both are strictly monotonic, they can be easily inverted (F_s^{-1}, F_t^{-1}).

To alter the geometry, we use the texture coordinates $T(s,t)$ with the remapped deformed geometry $D(F_s(s), F_t(t))$. Similarly, to alter the texture coordinates, we use the deformed geometry $D(s,t)$ with the inversely remapped texture coordinates $T(F_s^{-1}(s), F_t^{-1}(t))$.

Such a remapping is very efficient, but there are trade-offs to using any of the two methods above. If the geometry is altered, the ROI needs to be densely discretised (or dynamically tessellated in the GPU) and $D(s,t)$ needs to be known for the entire ROI. If the texture coordinates are altered, $T(s,t)$ needs to be known for the entire ROI. As displacement mapping requires the geometry to be modified, the best overall option is to alter the geometry.

7. NON AXIS-ALIGNED GRID WARPING

We now describe how our proposed solution can be extended to allow non-axis-aligned grid warping. Given the per-cell bilinear warping functions $S_b(s,t)$, $T_b(s,t)$ and defining $H_s(s,t) = D(S_b(s,t), T_b(s,t$ equations 3 and 4 become:

$$F_s(s,t) = (\left\|H'_s(s,t)\right\|_2 - \left\|O'_s(s,t)\right\|_2)^2 \quad (7)$$

$$F_t(s,t) = (\left\|H'_t(s,t)\right\|_2 - \left\|O'_t(s,t)\right\|_2)^2 \quad (8)$$

where

$$H'_s(s,t) = S'_{b_s} D'_s(S_b(s,t), T_b(s,t)) + T'_{b_s} D'_t(S_b(s,t), T_b(s,t))$$
$$H'_t(s,t) = S'_{b_t} D'_s(S_b(s,t), T_b(s,t)) + T'_{b_t} D'_t(S_b(s,t), T_b(s,t))$$

It is straightforward to modify the boundary constraints to take into account the increased number of neighbours per point (four instead of two). The efficiency of the error calculations is reduced, as instead of directly sampling the partial derivative lengths (eq. 3, 4) we only need to sample the partial derivative vectors, scale them and calculate their norms (eq. 7, 8). The Jacobian calculation process is identical, but in this case each per-axis cell energy is affected by all four adjacent points. An example of non-axis-aligned distortion correction can be seen in figure 3.

8. HYBRID CPU-GPU OPTIMISATION

We use the CPU implementation of the Levenberg-Marquardt optimiser from the ALGLIB library [1] and provide an objective function that calculates the squared errors and the Jacobian on the GPU. We now outline how the components of the objective function – described in section 5.1 – are handled. We give details for the axis-aligned version of the algorithm, as the non-axis-aligned version is similar.

The O'_s and O'_t functions are precalculated and stored in a texture during the pre-processing stage after the selection of the ROI. At the start of the optimisation, D'_s and D'_t are also calculated and stored in a texture.

The distortion energy integral in the objective function is calculated using a *DirectCompute* shader. $(M-1)\times(N-1)$ thread groups are dispatched (one per cell) and in each group a $K \times L$ grid of threads

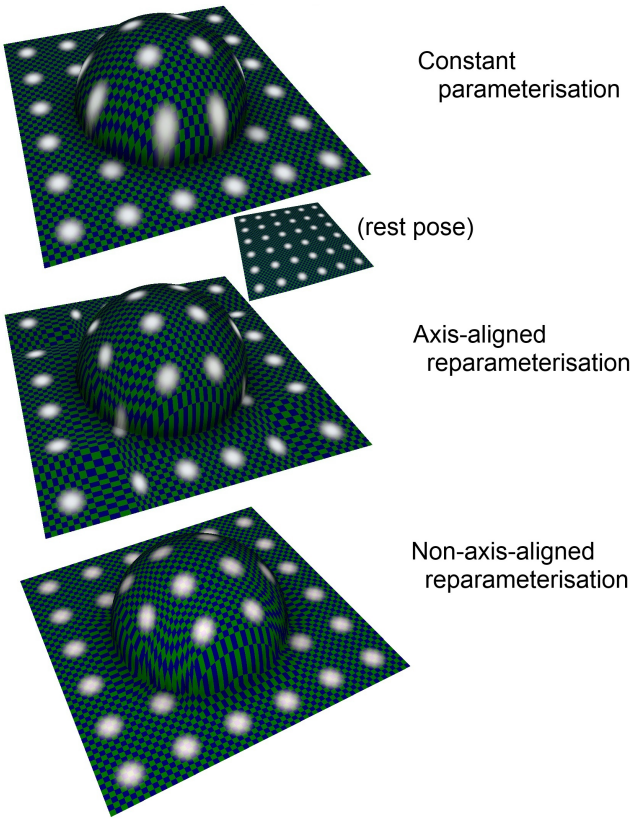


Figure 5: The original mesh is shown minimised in the top-middle. The deformation introduces creases on the mesh (top, middle and bottom). Normal texture mapping results in significant stretch of detail on the bump (top). Our axis-aligned solution reduces the stretch on features on the bump, but introduces other artifacts (middle). The optimizer converges to a suboptimal solution where some features are located on the crease. Also, due to the axis-aligned nature of the process, varying deformation across a strip results in distortions of features in undeformed areas. This can be observed here as compression of features outside the bump as a side-effect of stretch reduction of features on the bump. The non-axis-aligned version of our algorithm correctly preserves the features (bottom).

is executed (max size 32×32). The thread grid for a cell is used to calculate the integral: the cell is uniformly split to $K \times L$ sub-cells and the two integrals $E_{s_{ij}}$ and $E_{t_{ij}}$ are evaluated using the midpoint method. The sub-cell results are summed using a GPU reduction operator [24] and are read back in the CPU. All required calculations are two texture fetches, adds and multiplies, so the shader evaluation is very efficient.

The Jacobian is calculated numerically in the same shader using finite differences. As the warp is piecewise-linear, the energy for each cell is affected only by those adjacent to the cell grid lines. More specifically, $E_{s_{ij}}$ is only affected by changes in s_j and s_{j+1} and similarly $E_{t_{ij}}$ is only affected by changes in t_i and t_{i+1}. As a result, the cost of calculating the Jacobian is only approximately four times more than a single error calculation, as it requires a little more than a total of five evaluations of the error function.

In a cell $([s_0,s_1],[t_0,t_1])$, given a very small offset h the subcell error is additionally calculated four more times for four modified

cells: $([s_0+h,s_1],[t_0,t_1])$, $([s_0,s_1-h],[t_0,t_1])$, $([s_0,s_1],[t_0+h,t_1])$, $([s_0,s_1],[t_0,t_1-h])$.

The integral calculation can efficiently handle the additional cell coordinates, as f and g (and their derivatives), being linear in nature, can be analytically adjusted for the new interval.

9. RESULTS

We validated the algorithm on a real dataset as well as procedurally defined geometry. The real data set consists of face animation data from motion capture: 615 frames for sequence "face90" and 1373 frames for sequence "face49" – each containing $8,820$ vertices and $17,216$ triangles (shown in figures 1, 2, 4 and 7). The procedural examples (figures 3, 5 and 9) contain 100 frames, 16,384 vertices and $32,258$ triangles. For each sequence we manually authored distortion control maps. The face animation data contains localised deformation, resulting in more subtle distortion control results as can be seen from the corresponding figures. In our distortion control maps we used (bounding) circles to represent areas where we wished to minimise distortion. However, note that the shape of distortion control features can be arbitrary (defined by the user), and is not limited to circles.

The optimisation process is non-linear and the time required for the calculation of a solution for a given frame depends on the number of unknowns, the complexity of deformation as well as the optimisation parameters (i.e. derivative step size, stopping condition tolerances, iterations). Optimisation and remapping times for our test sequences are shown in table 1. These were measured using a 2.66GHz Xeon CPU with 24 GB RAM and an NVIDIA GeForce GTX 580 with 4GB VRAM. The ALGLIB optimiser settings are selected for a high quality/low-error solve, i.e. all tolerances are set to 0 and the maximum iterations are set to 100.

Even though the optimisation algorithm can be used in real-time on a similar hardware configuration, the results can also be precomputed and efficiently stored for use on less powerful hardware. The storage cost needed for a single ROI is $(M+N-4)$ floats, multiplied by the number of frames. So, 100 frames of animation for a moderately partitioned region (e.g. 10×10) would require about 6 KB. Similarly, storage for a non-axis-aligned grid would be 25 KB. As such, the small storage costs makes the technique ideal for use in low GPU bandwidth hardware.

10. DISCUSSION

In this section we discuss various issues and capabilities of the proposed algorithm.

10.1 Grid Lines and Filtering

To avoid texture filtering artefacts, when calculating the initial rectilinear grid, we must ensure that when separating a low-rigidity from a high-rigidity cell, the separating line must move a few pixels towards the low-rigidity one, as otherwise texture filtering can cause stretching of rigid detail. An additional consequence is that using hardware trilinear filtering can also cause artifacts, as sampling from low resolution mipmaps can result in rigid detail bleeding in a non-rigid (and potentially highly deforming) neighbouring area.

10.2 Edge Discontinuities

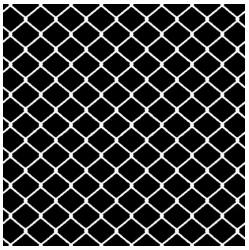


Figure 6: An example of a distortion control map that cannot be used successfully with the algorithm.

Remapping of axis-aligned lines results in seams on the edges of the selected ROI, as the parameterisation there will be discontinuous. We alleviate this with the following steps: a) ensure the cells of the rectilinear grid that are adjacent to the boundary contain non-rigid data and b) at the boundary cells, linearly blend the original and optimised parameterisation so that when approaching a vertical edge the $\mathbf{s}'$ solution blends to $\mathbf{s}$ and when approaching a horizontal edge the $\mathbf{t}'$ solution blends to $\mathbf{t}$.

10.3 ROIs and Chart Boundaries

If a selected ROI contains a texture space region which is unused, the resulting distortion control weight and partial derivative length textures will contain the texture initialisation values, which should be zero. In this case, the calculated energy values (eq. 3 and 4) in those areas are 0 and do not affect the rest of the process.

10.4 Failure Cases

There are cases where the algorithm fails, such as when distortion on the deformed surface varies significantly across a horizontal or vertical strip (figure 6). Additionally, high-frequency deformation (e.g. by introducing creases on previously smooth areas) will result in slower convergence or lower quality results, especially for the axis-aligned version of the algorithm (figure 5).

11. CONCLUSION AND FUTURE WORK

In this paper we presented a technique to re-parameterise regions in texture space, such that important rigid features mapped on these regions are preserved when the surface deforms. The proposed algorithm requires minimal user interaction and exhibits fast computation and runtime evaluation as well as very low storage requirements.

The technique can be applied to reduce elastic distortions on a variety of scenarios where highly detailed rigid features are represented on a map, abstracted from the underlying low-complexity deforming geometry they are mapped on. In modeling packages, artists can preview animations with detail mapped in a content-aware way, without manually altering the model geometry to achieve similar results. Real-time animated rigid detail on deformable objects also becomes a possibility, whereas previously it would require a significant amount of work from artists. Precomputed re-mappings for canned animations can enhance visual quality by reducing the rigid detail distortion on deformable surfaces in low-power hardware. In general, control cages/sparse meshes with static or dynamic detail are ideal candidates for use with this method, as the geometry remains unchanged and as a result it can be shared with more detail maps, requiring only pre-calculated parameterisation corrections for ROIs for each detail map.

Example	Solver(msec)	ObjFunc (msec)	Evals (Jac)	Unknowns	Render(ms)
aa-face90-umouth	9.85	8.49	18(4)	14	1(1.35)
aa-face90-lmouth	10.17	8.93	16(3)	14	1(1.5)
aa-face49-leye	22.26	20.7	23(6)	12	1(1.3)
aa-face49-reye	26.34	24.39	28(8)	12	1(1.3)
aa-face49-all	68.62	62.51	85(21)	(14,14,12,12)	1.1(1.8)
aa-saddle	26.97	22.3	25(8)	24	2.8(5)
nonaa-face90-umouth	22.66	13.96	19(6)	48	1(1.35)
nonaa-face90-lmouth	54.84	23.16	20(7)	80	1(1.5)
nonaa-saddle	741.8	108.4	30(13)	288	2.8(5)
aa-worm	8.87	13.41	17(3)	38	3.2(5.2)
nonaa-worm	3078.15	425.170	27(15)	192	3.2(5.2)

Table 1: Per-frame timings using our real and synthetic test data. The example "aa-face49-all" uses combined times for all four ROIs of the face49 animation sequence. Each distortion control map is associated with a number of unknowns for the non-linear minimizer, which is $M+N-4$ for the axis-aligned version (aa-) and $(M-2)\times(N-2)\times 2$ for the non-axis-aligned version (nonaa-). The solver times correspond to the times required for the whole optimisation (CPU-GPU). The ObjFunc times correspond to the times required for all calculations of the objective function in the GPU, with and without Jacobian calculation, including the transfers to the CPU. The Evals column shows the average number of objective function evaluations per solved frame, while the number in the parentheses shows the average number or required Jacobian evaluations per solved frame. The rendering times correspond to close-up views of highly tessellated geometry using the modified and original parameterisations (numbers inside and outside parentheses respectively). Rendering times for the modified parameterisation include rendering the parameterisation to a texture and sampling it from the normal shader used for the mesh. It can be seen that even though the objective function calculation times scale well with the number of the unknowns, the rest of the optimisation does not (especially for the non-axis-aligned variant) so for larger problem sizes the performance deteriorates quickly. These examples have not been optimised for performance, as all tolerances are zero.

The piecewise-linear warp is $C0$-continuous on the grid edges, so we would like to modify the algorithm so that the remapping is at least $C1$-continuous in the domain of the ROI, resulting in a higher-quality re-parameterisation. We would also like to extend the algorithm to handle any ROI shape and perform content-aware warp of the volumetric space (thick shell over the surface), so that detail of any complexity can be preserved under deformation.

12. REFERENCES

[1] ALGLIB. `www.alglib.net`.

[2] M. Ben-Chen, O. Weber, and C. Gotsman. Variational harmonic maps for space deformation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 34. ACM, 2009.

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

[4] B. Burley and D. Lacewell. Ptex: Per-face texture mapping for production rendering. In *Computer Graphics Forum*, volume 27, pages 1155–1164. Wiley Online Library, 2008.

[5] M. Floater and K. Hormann. Surface parameterization: a tutorial and survey. *Advances in multiresolution for geometric modelling*, pages 157–186, 2005.

[6] R. Gal, O. Sorkine, and D. Cohen-Or. Feature-aware texturing. In *Proceedings of Eurographics Symposium on Rendering*, pages 297–303, 2006.

[7] P. Heckbert. Survey of texture mapping. *Computer Graphics and Applications, IEEE*, 6(11):56–67, 1986.

[8] K. Hormann, B. Lévy, A. Sheffer, et al. Mesh parameterization: Theory and practice. *SIGGRAPH Course Notes*, 2007.

[9] A. Jacobson, I. Baran, J. Popovic, and O. Sorkine. Bounded biharmonic weights for real-time deformation. *SIGGRAPH'11: ACM SIGGRAPH 2011 Papers*, 2011.

[10] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. *ACM Transactions on Graphics (TOG)*, 26(3):71, 2007.

[11] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 561–566. ACM, 2005.

[12] V. Kraevoy, A. Sheffer, A. Shamir, and D. Cohen-Or. Non-homogeneous resizing of complex models. In *ACM Transactions on Graphics (TOG)*, volume 27, page 111. ACM, 2008.

[13] Y. Lipman, D. Levin, and D. Cohen-Or. Green coordinates. In *ACM Transactions on Graphics (TOG)*, volume 27, page 78. ACM, 2008.

[14] J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 27–34. ACM, 1993.

[15] J. More. The levenberg-marquardt algorithm: implementation and theory. *Numerical analysis*, pages 105–116, 1978.

[16] D. Panozzo, O. Weber, and O. Sorkine. Robust image retargeting via axis-aligned deformation. In *Computer Graphics Forum*, volume 31, pages 229–236. Wiley Online Library, 2012.

[17] T. Popa, D. Julius, and A. Sheffer. Material-aware mesh deformations. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*, pages 22–22. IEEE, 2006.

[18] P. Sander, S. Gortler, J. Snyder, and H. Hoppe. Signal-specialized parametrization. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 87–98. Eurographics Association, 2002.

[19] A. Sheffer and E. De Sturler. Smoothing an overlay grid to minimize linear distortion in texture mapping. *ACM Transactions on Graphics (TOG)*, 21(4):874–890, 2002.

[20] A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications. *Foundations and Trends® in*

Computer Graphics and Vision, 2(2):105–171, 2006.

[21] O. Sorkine. Laplacian mesh processing. In *Eurographics State-of-the-Art Report*, pages 53–70, 2005.

[22] W. Xu and K. Zhou. Gradient domain mesh deformation - a survey. *Journal of computer science and technology*, 24(1):6–18, 2009.

[23] X. Yang, R. Southern, and J. Zhang. Fast simulation of skin sliding. *Computer Animation and Virtual Worlds*, 20(2-3):333–342, 2009.

[24] E. Young. Directcompute optimisations and best practices. In *GPU Technology Conference*, 2010.

APPENDIX

A. DERIVATION OF ENERGY FOR AXIS-ALIGNED DEFORMATION

We want identical lengths for a small segment (s_0, s_1), where the geometry function can be considered as piecewise-linear.

$$\left\|D(s'_1) - D(s'_0)\right\| = \|O(s_1) - O(s_0)\|$$

Because $s' = f(s) = cs + d$, the above can be further rewritten as:

$$\|D(f(s_1)) - D(f(s_0))\| = \|O(s_1) - O(s_0)\|$$

The function $f(s)$ can be also written as follows:

$$f(s) = s'_0 + \frac{s - s_0}{s_1 - s_0}(s'_1 - s'_0)$$

and its derivative in this case is:

$$f'(s) = \frac{s'_1 - s'_0}{s_1 - s_0}$$

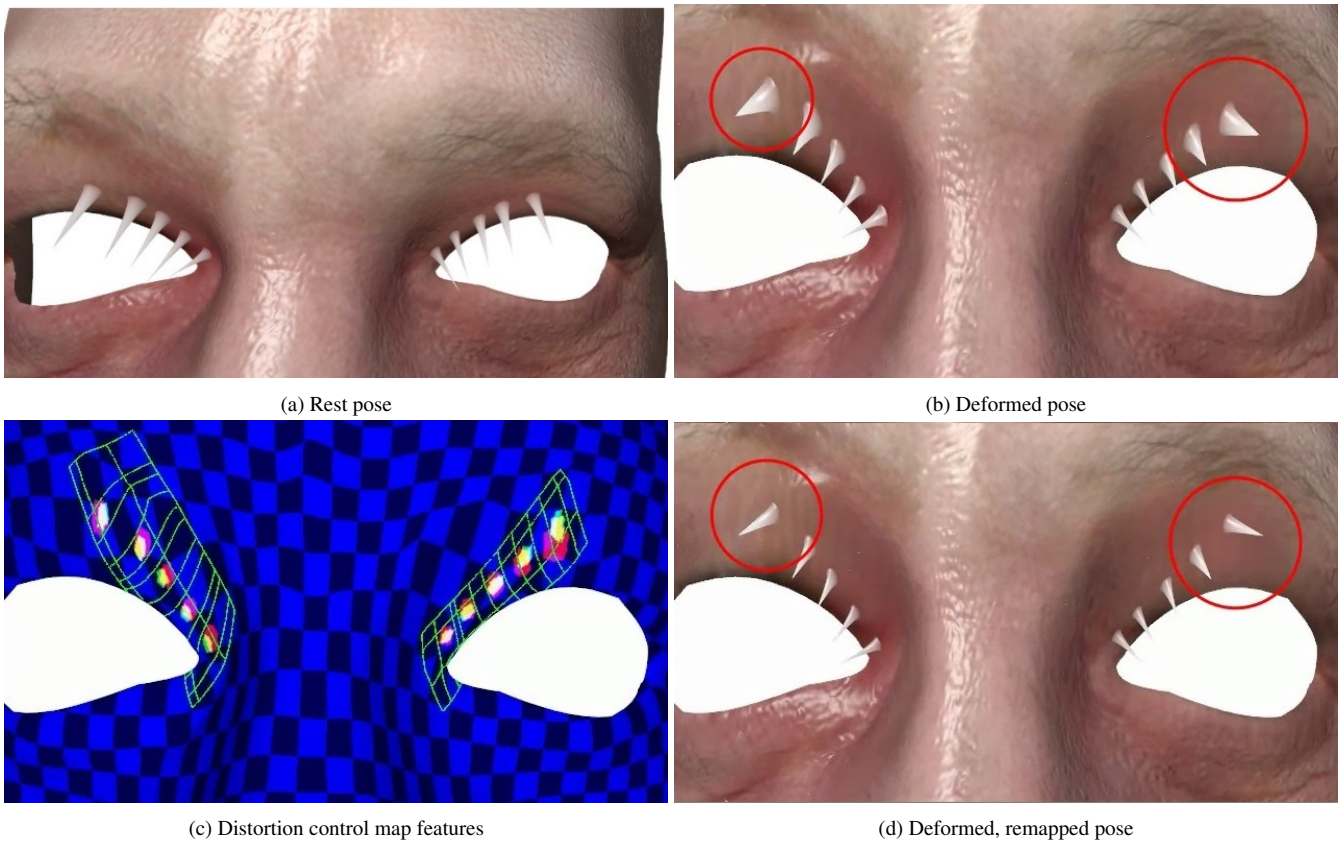
Divide by $s_1 - s_0$ to be able to convert it to derivatives:

$$\frac{\|D(f(s_1)) - D(f(s_0))\|}{s_1 - s_0} = \frac{\|O(s_1) - O(s_0)\|}{s_1 - s_0}$$

Rewriting $H(x) = D(f(x))$ leads to:

$$\begin{aligned}
\frac{\|H(s_1) - H(s_0)\|}{s_1 - s_0} &= \frac{\|O(s_1) - O(s_0)\|}{s_1 - s_0} && \Rightarrow \\
\left\|H'(s)\right\| &= \left\|O'(s)\right\| && \Rightarrow \\
|f'(x)|\left\|D'(f(x))\right\| &= \left\|O'(s)\right\| \\
|f'(x)|\left\|D'(f(x))\right\| - \left\|O'(s)\right\| &= 0
\end{aligned}$$

and this leads to the equations 3 and 4.



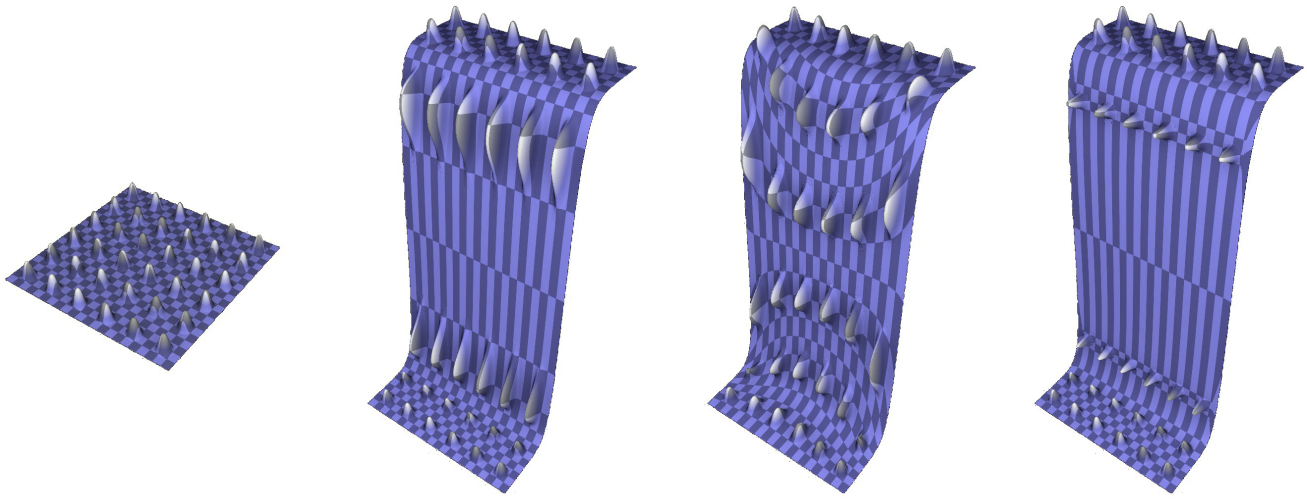
(a) Rest pose

(b) Deformed pose

(c) Distortion control map features

(d) Deformed, remapped pose

Figure 7: Mapped features near the eyes. Rest pose (b) and deformed frame (b). The features stretch under deformation. Our algorithm calculates a re-parameterisation that reduces distortions near the features (d). The remapped ROI rectangles and the features in the original and remapped parameterisation are shown in red and green respectively (c).



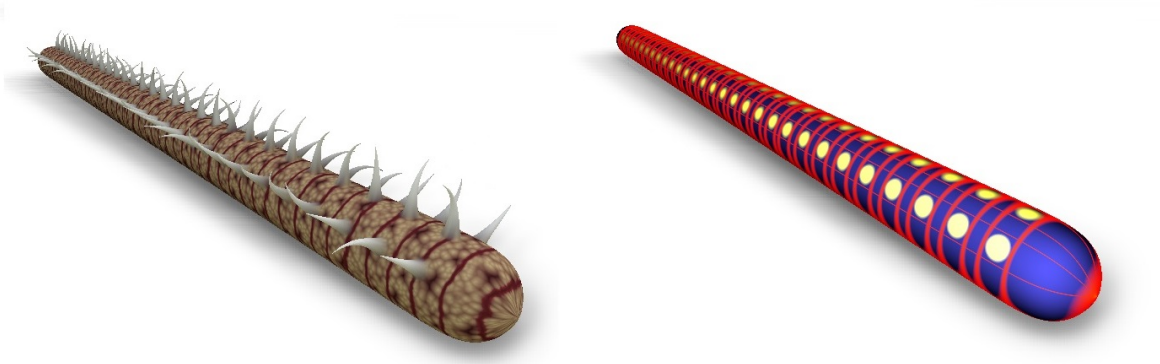
(a) Rest pose

(b) Deformed, using original parameterisation

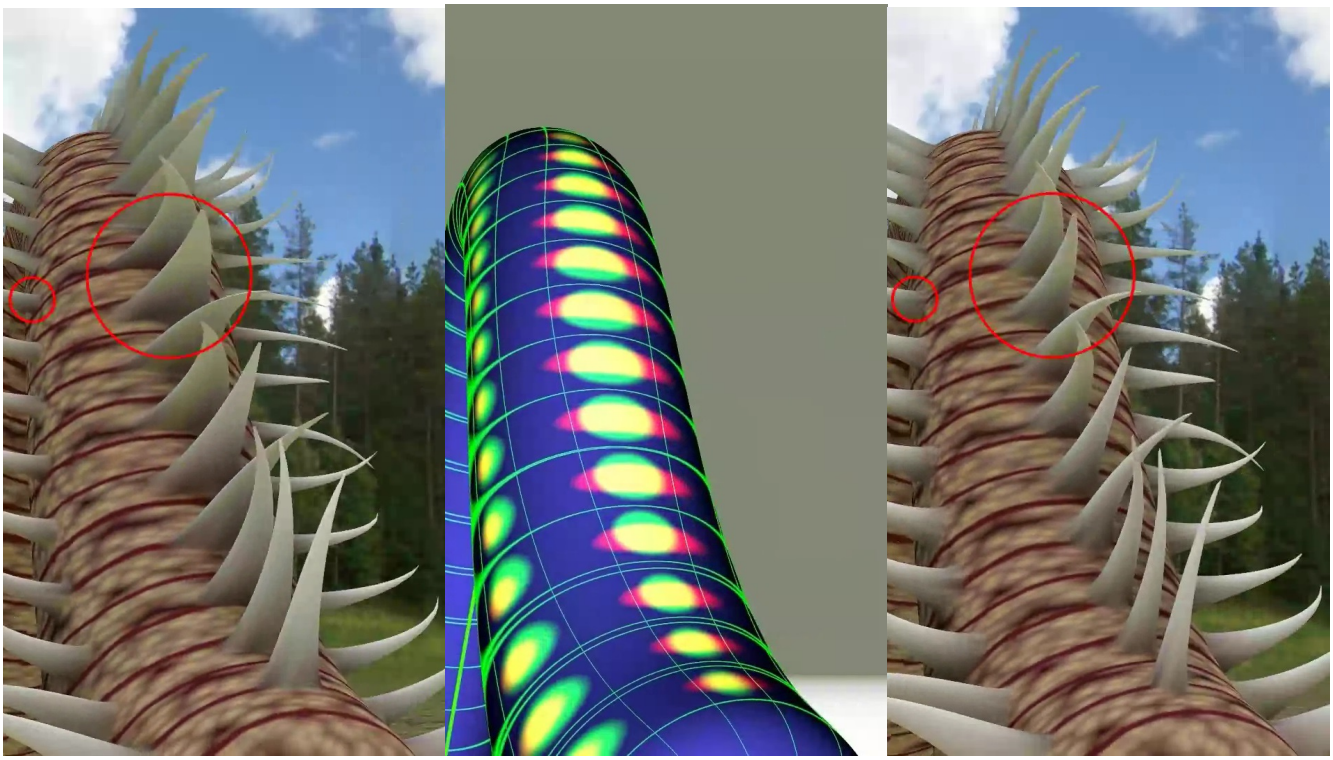
(c) Deformed, re-parameterised using CGAL's mean value coordinates

(d) Deformed, using content-aware re-parameterisation

Figure 8: Plane (a) deformation with mapped displacements. The constant parameterisation (b) stretches the features. Re-parameterising the mesh using an off-the-shelf method (c) does not preserve the features. Re-parameterising the mesh using our method (d) preserves the features and spreads the error in non-feature regions.



(a) Rest pose, normal (left) and distortion control map (right) views



(b) Deformed pose (c) Distortion control map features (d) Deformed, remapped pose

Figure 9: Worm example. The worm's horns are authored in the rest pose and are marked as rigid in the distortion control map (a). The deformed pose compresses the features near the centre of the worm (b). Using our method, the parameterisation is warped so that the parameterisation distortion near the features is spread to non-feature areas (d). The areas of interest in the original and content-aware parameterisation are shown in red and green respectively (c).