[POSTER] Realtime Shape-from-Template: System and Applications

Toby Collins [*] Adrien Bartoli

ALCoV-ISIT, UMR 6284 CNRS Université d'Auvergne, Clermont-Ferrand, France

Abstract

An important yet unsolved problem in computer vision and Augmented Reality (AR) is to compute the 3D shape of nonrigid objects from live 2D videos. When the object's shape is provided in a rest pose, this is the Shape-from-Template (SfT) problem. Previous realtime SfT methods require simple, smooth templates, such as flat sheets of paper that are densely textured, and which deform in simple, smooth ways. We present a realtime SfT framework that handles generic template meshes, complex deformations and most of the difficulties present in real imaging conditions. Achieving this has required new, fast solutions to the two core sub-problems: robust registration and 3D shape inference. Registration is achieved with what we call Deformable Render-based Block Matching (DRBM): a highly-parallel solution which densely matches a time-varying render of the object to each video frame. We then combine matches from DRBM with physical deformation priors and perform shape inference, which is done by quickly solving a sparse linear system with a Geometric Multi-Grid (GMG)-based method. On a standard PC we achieve up to 21fps depending on the object. Source code will be released.

1 Introduction, Context and Contributions

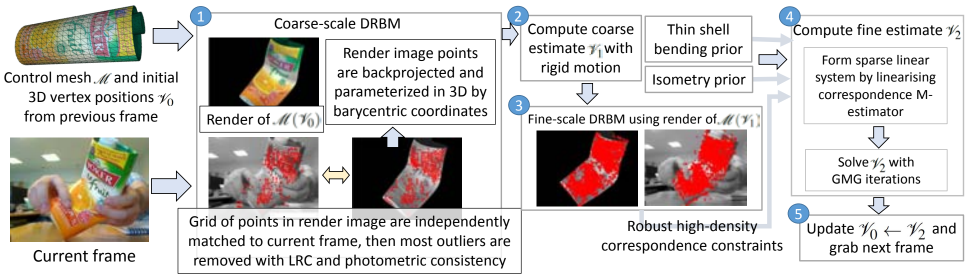
SfT is a crucial ingredient to enable 3D augmented reality with deformable objects and 2D video cameras. The goal of SfT is to compute the 3D shape of an object using a deformable 3D template [2, 10, 4, 8, 11]. Solving SfT is important for AR because it provides both the object's 3D shape in camera coordinates *and* dense registration. However, despite considerable research, a general, robust and realtime SfT system is not yet available. This has prevented many exciting applications emerging in AR and Human Computer Interaction (HCI). The SfT problem is very challenging because it involves solving for each frame two difficult, interdependent parts: these are *(a) registration*: computing high-density and robust motion constraints from the video frame and *(b) shape inference*: inferring the object's nonrigid 3D shape. Previous solutions to SfT use either *feature-based* [2, 10, 4, 8] or *direct* [7, 11] motion constraints. Feature-based constraints are the most common and take much inspiration from work with 2D templates [9]. These match salient keypoints from the template's texturemap and the video frame using *e.g.* SURF [3]. The main advantages are that they do not require an initial solution and are reasonably fast to compute. However they only provide sparse constraints for part *(b)* and do not work well for weakly textured objects, objects with repeating texture or blurred video frames. By contrast, direct motion constraints work by directly measuring the photometric agreement between the video frame and the deformed template. However because they are highly nonconvex constraints they are not that easy to work with. In [7, 11] they were linearised and the problem was solved with gradient-based optimisation. We refer to this strategy as *Direct Linearised Optimisation* (DLO). For each frame DLO solves the problem by alternation between parts *(a)* and *(b)* until a convergence criterion is reached. This requires an initial solution for each frame, which was done in [7, 11] by tracking the template frame-to-frame. Results were demonstrated on short videos with simple, flat surfaces with well-defined borders, and slow, smooth deformation, but were not close to realtime.

The future of SfT probably lies with direct constraints since they provide far denser motion constraints than features. But is DLO the best strategy for achieving robust realtime results? DLO has strong similarities with solving optic flow using gradient-based optimisation, and shares its well-known problems. Firstly it has a limited convergence basin and can become trapped in a local minimum. A coarse-to-fine blurred image pyramid is usually used to try to increase the basin, however this leads to other problems, because it tends to lose motion information at small image structures and at high frequency textures, and works poorly for low-contrast textures. The second problem is knowing where to apply the constraints. Ideally they should only apply at points which are visible, however this is difficult to enforce, particularly with strong occlusions and a poor initial solution. The third problem is speed. For each frame, DLO requires alternation between parts *(a)* and *(b)*, which would be expensive for complex objects with large deformation spaces.

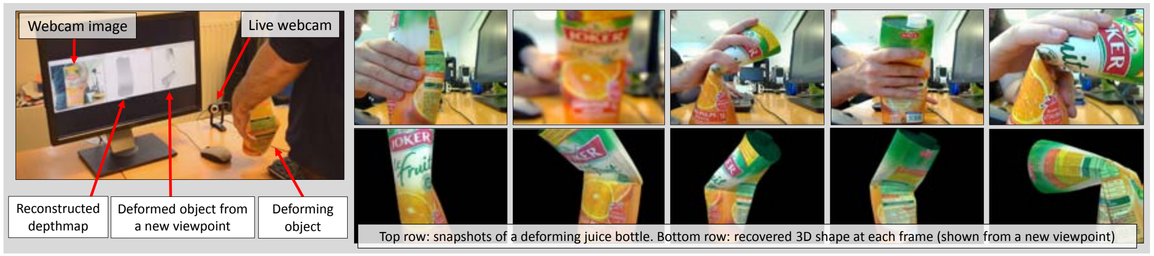
We propose a different approach to direct constraints in SfT that does not have the main disadvantages of DLO. We call this DRBM. DRBM is inspired by block-matching based optic flow and patch-matching [1], and does *not* involve image linearisation. DRBM solves part *(a)* by rendering an initial estimate of the template's shape and then *independently matching* a dense set of small windows in the render image with the video frame. A window's match is found by a quasi-exhaustive search over 2D displacements for the one yielding the best photometric agreement. This process is highly parallelisable and runs in realtime on modern GPUs. Once finished, outlier matches are filtered using two important criteria: Left-Right Consistency (LRC) and photometric agreement. These filters are powerful and normally remove most incorrect matches. The remaining matches are then used as *correspondence constraints* for solving part *(b)*. There are four main reasons why DRBM is advantageous over DLO. Firstly, DRBM *registers* points on the template to the video frame, whereas DLO only achieves registration if it correctly converges, which is not guaranteed since it is a local optimisation. Secondly, after running DRBM we can solve part *(b)* without needing to iteratively recompute motion constraints, and this makes our system fast. Thirdly, DRBM does not need a smoothing pyramid so it handles small image structures and high frequency texture without problems. Fourthly DRBM handles occlusions naturally, because occluded windows in the render image do not tend to have matches that pass both filters.

After running DRBM we solve part *(b)* using a sub-sampled *control mesh* of up to several thousand vertices. Once done the full resolution mesh can be interpolated quickly using *e.g.* linear blend skinning, which is highly parallelisable. Skinning has not been applied before in SfT, yet is important to facilitate realtime speeds for detailed templates. Our system boils down to solving one sparse linear least squares problem for each frame, with $3N$ unknowns (three for each of the N control mesh vertices). However solving this in realtime is still nontrivial. We have found factorisation-based solvers are too expensive for more than a few hundred vertices, and

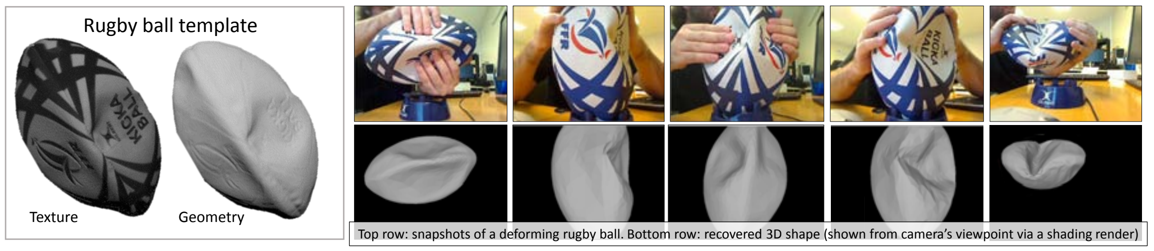
[*]This work has received funding from the EU FP7 ERC research grant 307483 FLEXABLE.


Control mesh $\mathscr{M}$ and initial 3D vertex positions $\mathscr{V}_0$ from previous frame
Current frame
1
Coarse-scale DRBM
Render of $\mathscr{M}(\mathscr{V}_0)$
Render image points are backprojected and parameterized in 3D by barycentric coordinates
Grid of points in render image are independently matched to current frame, then most outliers are removed with LRC and photometric consistency
2
Compute coarse estimate $\mathscr{V}_1$ with rigid motion
Thin shell bending prior
Isometry prior
3
Fine-scale DRBM using render of $\mathscr{M}(\mathscr{V}_1)$
Robust high-density correspondence constraints
4
Compute fine estimate $\mathscr{V}_2$
Form sparse linear system by linearising correspondence M-estimator
Solve $\mathscr{V}_2$ with GMG iterations
5
Update $\mathscr{V}_0 \leftarrow \mathscr{V}_2$ and grab next frame

	Juice bottle	Rugby ball	Bending paper
N	868	1502	432
Average processing time (ms)			
1	10.2	11.2	10.2
2	5.3	5.1	4.8
3	22.1	24.5	19.1
4	24.1	39.9	8.3
other	7.3	8.6	5.0
Total	69.0	89.3	47.4
fps	14.5	11.2	21.1



Top row: snapshots of a deforming juice bottle. Bottom row: recovered 3D shape at each frame (shown from a new viewpoint)



Top row: snapshots of a deforming rugby ball. Bottom row: recovered 3D shape (shown from camera's viewpoint via a shading render)

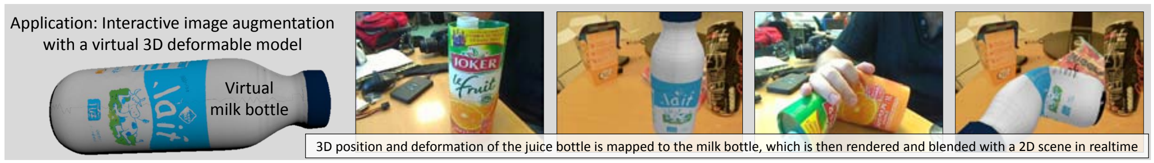

Application: Interactive image augmentation with a virtual 3D deformable model
Virtual milk bottle
3D position and deformation of the juice bottle is mapped to the milk bottle, which is then rendered and blended with a 2D scene in realtime

Figure 1: Summary of proposed system for solving SfT in realtime (top part), and snapshots of results with two different deformable objects (bottom part). In the last row we show an example AR application using our system.

without expensive preconditioning Conjugate Gradient (CG)-based solvers require too many iterations. One possible direction is to reduce the problem's dimensionality using linear bases from the stiffness matrix [8]. However to handle complex deformation such as surface creasing we have found many bases are needed (*e.g.* 200 or more), and the resulting dense linear system becomes too expensive to solve. Our answer is to solve with a GMG. GMG has not been applied to SfT before, yet it is probably the most suitable method to efficiently resolve complex deformations. GMG is iterative, but has the distinct advantage over CG in that the number of iterations required for convergence is constant in N [15].

The only existing realtime SfT systems are feature-based methods [6, 8], however these require simple, smooth templates, such as flat sheets of paper that are densely textured, which deform in simple, smooth ways, and they suffer from the problems with feature-based constraints as discussed above. By contrast, our system uses high-density direct constraints and can handle many of the real-world challenges of SfT. These include *(i)* non-smooth fully-3D template meshes, *(ii)* partial views, caused by external occlusions, self-occlusions or when the object moves beyond the field-of-view, *(iii)* strong lighting changes, *(iv)* cluttered scenes, *(v)* rapid camera and/or object motion, *(vi)* optical and/or motion blur, *(vii)* repeating texture, *(viii)* complex, high frequency deformations and *(ix)* high-frequency textures. No previous SfT method can handle all the above (realtime or otherwise). We also present examples of tracking completely round the back of deforming objects, which has not been seen before in SfT. We refer the reader to the additional videos which shows this, and the above challenges being handled.

2 Methodology

2.1 Overview

We solve SfT with a frame-to-frame tracking process outlined in Fig. 1 (top section). We model the object's surface with a triangulated control mesh $\mathscr{M}$ with triangles $\mathscr{F} \overset{\text{def}}{=} \{\mathbf{f}_1, \mathbf{f}_2, \dots \mathbf{f}_F\}$ and vertices $\mathscr{V} \overset{\text{def}}{=} \{\mathbf{v}_1, \mathbf{v}_2, \dots \mathbf{v}_N\}$. We use $\mathscr{V}_{rest}$ to denote the known vertex positions of the control mesh in a rest pose. We assume the object does not tear as it deforms (*i.e.* its topology is preserved). Therefore $\mathscr{F}$ is fixed and only $\mathscr{V}$ varies. We use $\mathscr{M}(\mathscr{V})$ to denote the deformed control mesh with its vertices set to $\mathscr{V}$. We use $\mathscr{V}_0$ to denote the initial estimate of $\mathscr{V}$ in 3D camera coordinates for a given frame. We initialise $\mathscr{V}_0$ at the start of the video by rigidly transforming $\mathscr{V}_{rest}$ to a canonical pose in front of the camera. We then render $\mathscr{M}(\mathscr{V}_0)$ with OpenGL and overlay the render with the live video. The user moves the object to roughly align it with the render, and once done notifies the system and tracking begins.

We handle rapid motion using a two-stage coarse-to-fine strategy. The coarse estimate $\mathscr{V}_1$ resolves fast changes of the object's rigid pose, which occurs frequently for objects held in the hand, or when the camera is hand-operated. This is computed by running DRBM at a coarse scale with a render of $\mathscr{M}(\mathscr{V}_0)$ then rigidly transforming $\mathscr{V}_0$ to agree with matches from DRBM. The fine estimate $\mathscr{V}_2$ is computed by running DRBM at a fine scale with a render of $\mathscr{M}(\mathscr{V}_1)$. This produces denser matches which we use to resolve precise deformation. One cannot compute $\mathscr{V}_2$ using only the matches because computing 3D deformation from 2D motion is very ill-posed. We handle this by introducing deformation priors which favour smooth solutions that do not stretch or shrink the surface much. We combine all the constraints into an energy function that is solved with a GMG method. The next frame is then acquired, we make the update $\mathscr{V}_0 \leftarrow \mathscr{V}_2$ and the process is repeated.

2.2 Deformable Render-based Block Matching

Rendering. In DRBM we render two types of images. The first is a greyscale image of the object, which is a texturemapped render from the camera's viewpoint, illuminated with ambient light and of size $W \times H$ where W and H denote the width and height of the video frame. The alpha channel of the render image stores the render mask, which holds a value of 1 if the pixel is on the template and 0 otherwise (we refer to these as foreground and background pixels respectively). The second image is a *barycentric coordinate image*, which is of size $W \times H \times 3$. This stores, for each foreground pixel, scalars $(j \in [1,F], b_1, b_2)$. The scalar j is the face in $\mathscr{F}$ that intersects the perspective optical ray passing through the pixel. The scalars b_1, b_2 give the barycentric coordinates of the intersection point with $0 \leq b_{\{1,2\}} \leq 1$ and $b_1 + b_2 \leq 1$. The pixel's 3D position $\mathbf{q} \in \mathbb{R}^3$ is then given in terms of the three vertices of $\mathbf{f}_j$ by $\mathbf{q} = b_1 \mathbf{v}_{\mathbf{f}_j(1)} + b_2 \mathbf{v}_{\mathbf{f}_j(2)} + (1 - b_1 - b_2)\mathbf{v}_{\mathbf{f}_j(3)}$. We use this to build a correspondence constraint for pixels that are matched by DRBM.

Matching. DRBM works by quantising the render image into a grid of small square windows of width w that are placed g pixels apart, then searching for the discrete 2D displacement $\Delta x \in [-d,d]$, $\Delta y \in [-d,d]$ that transforms the window to the video frame with the highest photometric similarity. We find the displacement with exhaustive search, which is extremely parallelisable on the GPU, with methods to prune the search space. Various similarity metrics can be used and we have found Normalised Sum of Absolute Differences (NSAD) works well. To correctly handle the boundary of the render, for each window we compute the NSAD error only with foreground pixels, which makes it invariant to background image structures. To increase speed we eliminate windows that have a very low intensity standard deviation, because they do not generally produce reliable correspondences. We use a conservative threshold of 5. We use two weak image invariants to prune the search space. Because all windows in the render have some image structure, the true displacement should be one which also has some image structure. Therefore we prune all displacements whose windows in the video frame have an intensity standard deviation lower than 5. The second invariant uses the fact that a correctly matched windows should have similar central moments.

Processing outliers. Matches from DRBM will contain outliers, and we detect the majority of these with two simple but powerful filters. Any remaining outliers are handled effectively using a robust correspondence cost function in part (*b*). First we perform Left-Right Consistency (LRC), which is a well-known tool in block matching that can eliminate many outliers (we use a LRC threshold of $w/8$ pixels). From the remaining matches we use the x84 rejection rule to eliminate matches with high NSAD compared to the population median. This is fast to compute and works well because the threshold automatically adapts to the imaging conditions. For example, if there is motion blur, the NSAD of correct matches increases, and x84 reduces the outlier rejection threshold accordingly.

Coarse and fine levels. As discussed above, we compute DRBM matches at coarse and fine levels. The purpose of the coarse level is to determine rapid changes of the object's rigid pose. This means we require fewer matches than the fine stage because we are resolving only 6 degrees of freedom. However we need to handle larger differences between the render image and video frame. We handle rapid within-plane translation using a larger d. Rapid depth and out-of-plane rotations (*i.e.* tilts) are not too problematic for block matching, and handled using a smaller w. We handle rapid within-plane rotation by matching the render image to some rotated versions of the video frame. Of these, the displacement with lowest NSAD is taken as the match. We use three rotations: $+30°$, $0°$ and $-30°$. In the fine stage, since large rotations have been accounted for by the coarse stage, we match only to the (unrotated) video frame. The default parameters for the coarse and fine stages are image resolution (W,H): $((160,120),(320,240))$ pixels, window size (w): $(13,9)$ pixels, grid spacing (g): $(6,3)$ pixels and maximal search range (d): $(50,25)$ pixels respectively.

2.3 DRBM Correspondence Constraints and Solving $\mathscr{V}_1$

Correspondence constraints. Suppose that we have a DRBM match whose window is centred at $\mathbf{u} \in \mathbf{R}^2$ in the render image and has a displacement $\Delta\mathbf{u} \in [-d,d]^2$. If the barycentric image at $\mathbf{u}$ is (m, b_1, b_2), then we have a correspondence constraint for the the m^{th} face. This can be expressed with a linear equation: $f(\mathscr{V}) \overset{\text{def}}{=} [\mathbf{q}_x(\mathscr{V}), \mathbf{q}_y(\mathscr{V})]^\top - \mathbf{q}_z(\mathscr{V})\tilde{\mathbf{u}} = \mathbf{0}$, where $\tilde{\mathbf{u}}$ denotes the 2D position of $\mathbf{u} + \Delta\mathbf{u}$ in normalised pixel coordinates. Because matches are noisy and may still contain outliers, we constrain $\mathscr{V}$ using f with a robust energy term:

$$E_{DRBM}(\mathscr{V}) \overset{\text{def}}{=} \frac{1}{C}\sum_{c=1}^{C} \rho\left(f(\mathscr{V})\right), \ \rho(\mathbf{x}) \overset{\text{def}}{=} \sqrt{\|\mathbf{x}\|^2 + \varepsilon^2} \qquad (1)$$

where C denotes the number of matches, ρ is the pseudo $L1$ M-estimator and ε is a small constant (we use $\varepsilon = 10^{-2}$).

Solving $\mathscr{V}_1$. Recall that the transform from $\mathscr{V}_0$ to $\mathscr{V}_1$ is the rigid transform $\mathbf{M}$ which accounts for global motion of the object from the previous to the current frame. We solve this using the coarse-level matches with $\arg\min_{\mathbf{M} \in SE_3} E_{DRBM}(\mathbf{M}(\mathscr{V}_0))$, where $\mathbf{M}(\mathscr{V}_0)$ rigidly transforms $\mathscr{V}_0$ by $\mathbf{M}$. This is solved quickly with a few Gauss-Newton iterations (we use at most 5 iterations).

2.4 Solving $\mathscr{V}_2$ with Deformation Priors and GMG

We solve $\mathscr{V}_2$ by combining DRBM constraints with two deformation priors: isometry and thin-shell bending. This is done by minimising the following energy:

$$E(\mathscr{V}_2) \overset{\text{def}}{=} E_{DRBM}(\mathscr{V}_2) + \lambda_{iso} E_{iso}(\mathscr{V}_2) + \lambda_{shell} E_{shell}(\mathscr{V}_2) \qquad (2)$$

where $E_{iso} \in \mathbb{R}^+$ and $E_{shell} \in \mathbb{R}^+$ are energies from the isometric and thin-shell bending energy priors, and E_{DRBM} is built using the fine-level correspondences. The scalars λ_{iso} and λ_{shell} are weight terms that govern the relative influence of the priors.

Deformation priors. The isometric energy we use encourages each of the mesh triangles to transform quasi-rigidly. This is based on [12], however unlike [12] we apply the constraint on a per-face basis, rather than for each vertex 1-ring neighbourhood. The benefit is to decouple isometric energy from bending energy (for example many surfaces can bend considerably more than they can stretch, and we want to model this). It also makes updating local rotations far faster (see below), since we do not need the Singular Value Decomposition. The isometric energy is defined as follows:

$$\begin{aligned} E_{ISO}(\mathscr{V}) &\overset{\text{def}}{=} \textstyle\sum_{j=1}^{F} w_j E'_{ISO}(\mathbf{f}_j(1), \mathbf{f}_j(2), \mathbf{f}_j(3)) \\ E'_{ISO}(a,b,c) &\overset{\text{def}}{=} \min_{\mathbf{R} \in SO_3, \mathbf{t} \in \mathbb{R}^3} \left\| \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} \mathbf{v}_a^{rest} \ \mathbf{v}_b^{rest} \ \mathbf{v}_c^{rest} \\ \mathbf{1}^\top \end{bmatrix} - [\mathbf{v}_a \, \mathbf{v}_b \, \mathbf{v}_c] \right\|_2^2 \end{aligned} \qquad (3)$$

The term $w_j \in \mathbb{R}^+$ is a per-triangle weight, which is set to the triangle's area. The terms $\mathbf{R}$ and $\mathbf{t}$ give the least-squares rigid transform that maps the triangle from the rest pose to 3D camera coordinates according to its three vertices. This is a non-convex energy because of the condition $\mathbf{R} \in SO_3$. We turn it into a convex quadratic energy using the first-order approximation of $\mathbf{R}$: $\mathbf{R} \approx (\mathbf{I}_3 + \text{skew}(\mathbf{r}))\mathbf{R}_0$, where $\mathbf{R}_0$ is an approximation of $\mathbf{R}$, $\mathbf{r} \in \mathbb{R}^3$ is a rotation vector and $\text{skew}(\mathbf{r})$ computes a 3×3 skew-symmetric matrix from $\mathbf{r}$. We set $\mathbf{R}_0$ using the best-fitting rotation that maps the triangle from the rest pose to $\mathscr{V}_1$. Because a triangle's vertices are coplanar, this can be computed analytically. The approximation of $\mathbf{R}$ is good for small changes in rotation up to ≈ 30 degrees, which is hardly ever exceeded in practice. We substitute $\mathbf{R} \leftarrow (\mathbf{I}_3 + \text{skew}(\mathbf{r}))\mathbf{R}_0$ in Eq.(3), which makes the minimisation problem in Eq. (3) linear least squares in $\mathbf{r}$ and $\mathbf{t}$. It is straightforward to show that $\mathbf{r}$ and $\mathbf{t}$ can be eliminated to make E_{ISO} convex and quadratic in $\mathscr{V}$.

The thin-shell bending energy penalises high changes in the template's curvature. We base this on the discrete form in [13]. This works by taking the four vertices from every pair of neighbouring triangles and measuring the degree that their motion deviates from affine motion. This is attractive because it is a quadratic convex constraint. However [13] requires adding a virtual vertex to each triangle, which increases the number of unknowns by $3F$. We have modified this to eliminate requiring additional vertices, which makes E_{shell} convex and quadratic in $\mathscr{V}$.

Solution. We linearise ρ about $\mathscr{V}_1$ which makes all terms in Eq. (2) convex and quadratic in the unknowns. We solve this globally using the associated normal equations, which is a sparse linear system, with a two-level GMG [15]. To do this we requires a *restriction matrix* $\mathbf{G}$ that transfers a vector field on $\mathscr{V}$ to a vector field on a lower-dimensional approximation of $\mathscr{V}$. We also require an *interpolation matrix* $\mathbf{H}$ that transfers vectors from the approximation back to $\mathscr{V}$. We make the restriction matrix using the main modes of variation of $\mathscr{V}$ induced by the thin-shell bending matrix $\mathbf{B}$. This gives $\mathbf{G}$ as the k eigenvectors of $\mathbf{B}$ with the k smallest eigenvalues. The interpolation matrix $\mathbf{H}$ is then given by $\mathbf{G}^\top$. The best choice of k depends on how smooth we expect the object's deformations to be. For objects that can strongly crease we have found $k = 150$ to be a good choice, whereas for objects that only deform smoothly k can be reduced. We solve the system's smooth level using a fast factorisation-based solver (Eigen's LDLT solver), and solve its fine level with a small number of Gauss-Seidel iterations (we use 5). We terminate GMG after 3 v-cycles.

3 Results and Realtime AR

3.1 Implementation

Our system is implemented in multithreaded C++, OpenGL and CUDA. 3D templates are stored as standard Wavefront .obj files. Currently we only use the GPU for rendering, DRBM and linear blend skinning. We use the Eigen libraries for sparse and dense linear algebra on the CPU and OpenCV for video acquisition, camera calibration, user interaction, and 2D visualisation.

3.2 Video Snapshots, Timing and Applications to AR

In the supplementary video we show results on four different deformable objects being successfully tracked over thousands of frames. We used the same weights for all objects ($\lambda_{iso} = 0.60$ and $\lambda_{shell} = 0.30$). We show results with complex deformations and the ability of our method to handle the seven main challenges listed in section 1. Here we show some representative snapshots for two of the objects (Fig. 1). The first is a juice bottle made of plastic and liquid packaging board (320 mm tall). The control mesh was built from a CAD model ($N = 868$, $F = 1732$) that we texturemapped with Agisoft's Photoscan. This object is challenging because it can crease significantly as it deforms, which makes its deformation space very high dimensional and non-smooth. It also has little texture on its sides and repeated texture on its back. The second object is a rugby ball made of rubber. The control mesh was built using a David Scanner structured light system, with $N = 1502$, $F = 3000$. This object is challenging because its texture is symmetric and it has regions of little texture, and it certainly cannot be tackled with a feature-based method. For each object we show five video frames, and below each frame we show the recovered object's shape from our method. Timing information is given in Fig. 1 (top right). This is broken down into the four main processes and timed on a standard x64 desktop PC running Windows 7 with an Intel i7 3820 processor (4 cores), an NVidia GTX 780 graphics card and a Logitech Pro 9000 webcam. We also quantitatively evaluated the accuracy of our method using the bending paper dataset [14]. This is not an ideal dataset because measurements come from the Kinect sensor, which is accurate to within a few mm depending on the distance. We computed a mean depth error of 3.41mm over the sequence. This is similar to the most accurate method on that dataset [4] (see the comparison in [5]), however [4] is a non-realtime feature-based method that works for smooth flat surfaces.

We finish by applying our system to AR. The application is interactive augmentation of a 2D image with a virtual 3D deformable model, by physically manipulating a real object. This is desirable because it allows a user to augment the image without any expertise in computer graphics or CAD. The setup is shown in Fig. 1 (bottom row), where we use the physical juice bottle as a proxy for deforming a virtual plastic 3D milk bottle. This works by transferring in realtime the recovered 3D motion of the juice bottle's vertices to the milk bottle vertices, in a manner based on [13]. Fig. 1 shows the juice and milk bottles before and after the juice bottle was physically manipulated. Note that this is the first time this has been achieved with a monocular camera and fully-3D models.

4 Conclusion and Experimental Platform

We have presented a robust and carefully-designed framework for solving realtime SfT. We will release our source code to the community. This will provide computer vision and AR researchers with a fast working platform to test new ideas, solutions and applications. Our code is modular and allows different constraints to be easily swapped in or added to the energy function. There are several open challenges that remain. These include automatically relocalising the object when tracking is lost, handling untextured objects and handling objects that stretch significantly.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 2009.

[2] A. Bartoli, Y. Gerard, F. Chadebecq, T. Collins, and D. Pizarro. Shape-from-Template. *PAMI*, 2015.

[3] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded up robust features. In *ECCV*, 2006.

[4] F. Brunet, A. Bartoli, and R. I. Hartley. Monocular template-based 3D surface reconstruction: Convex inextensible and nonconvex isometric methods. *CVIU*, 2014.

[5] A. Chhatkuli, D. Pizarro, and A. Bartoli. Stable template-based isometric 3D reconstruction by linear least-squares. In *CVPR*, 2014.

[6] I. Leizea, H. Álvarez, and D. Borro. Real time non-rigid 3D surface tracking using particle filter. *CVIU*, 2015.

[7] A. Malti, A. Bartoli, and T. Collins. A pixel-based approach to template-based monocular 3D reconstruction of deformable surfaces. In *4DMod-ICCV*, 2011.

[8] J. Östlund, A. Varol, D. T. Ngo, and P. Fua. Laplacian meshes for monocular 3D shape recovery. In *ECCV*, 2012.

[9] J. Pilet, V. Lepetit, and P. Fua. Fast non-rigid surface detection, registration and realistic augmentation. *IJCV*, 2008.

[10] M. Salzmann and P. Fua. Linear local models for monocular reconstruction of deformable surfaces. *PAMI*, 2011.

[11] M. Salzmann, R. Urtasun, and P. Fua. Local deformation models for monocular 3D shape recovery. In *CVPR*, 2008.

[12] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Eurographics Symposium on Geometry Processing*, 2007.

[13] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 2004.

[14] A. Varol, M. Salzmann, P. Fua, and R. Urtasun. A constrained latent variable model. In *CVPR*, 2012.

[15] P. Wesseling. *An introduction to multigrid methods.* Wiley, 1992.